MD5 Collision Attack Lab

Table of Contents:

1 Abstract:

During this lab, I investigated how the MD5 hash function works and discovered how simple it is to generate collisions for this hash function. The first two assignments were experimental in nature, involving learning about the MD5 hash function and the software "md5collgen," which can generate MD5 collisions. I utilized this information in the final two jobs to construct two executable programs that accomplish quite different things yet had the same MD5 hash. This demonstrates that the MD5 hash algorithm is not immune to collisions (in fact, it allows even more powerful attacks than collision finding, since we can create collisions of meaningful files with a specific malicious purpose).

2 Introduction:

A hash function encrypts data using an algorithm and no key. Because there is no means to reverse the encryption, they are referred regarded as "one-way hash functions." A variable-length plaintext string is "hashed" into a (usually) fixed-length hash value (also known as a "message digest" or simply a "hash"). Hash functions are typically employed to ensure integrity: if a plaintext's hash changes, the plaintext has changed. Secure Hash Algorithm 1 (SHA-1), which generates a 160-bit hash, and Message Digest 5 (MD5), which generates a 128-bit hash, are two common older hash algorithms. Both MD5 and SHA-1 have flaws; newer options, such as SHA-2, are suggested.

Ronald Rivest invented the Message Digest algorithm 5 (MD5). It is the most extensively used hash algorithm in the MD family. Based on any input length, MD5 generates a 128-bit hash result. MD5 has grown in popularity over the years, although flaws have been revealed where collisions may be found in a reasonable length of time. MD6 is the most recent iteration of the MD family of hash algorithms, having been released in 2008.

A collision attack is the act of looking for collisions for a certain function. MD5 and SHA-1 are two of the most popular and widely used hash functions. MD5 and SHA-1, on the other hand, are subject to collision attacks based on differential cryptanalysis.

This lab report covers the following topics:

• One-way hash function

• The collision-resistance property

• Collision attacks

• MD5

3 Objective:

- The purpose of the attack in this lab is to comprehend the significance of collision attacks and observe firsthand the harm that may be inflicted if the collision-resistance characteristic of a commonly used one-way hash function is violated.

<u>Lab environment:</u> I'll utilize md5collgen, a binary that stands for "Fast MD5 Collision Generation."

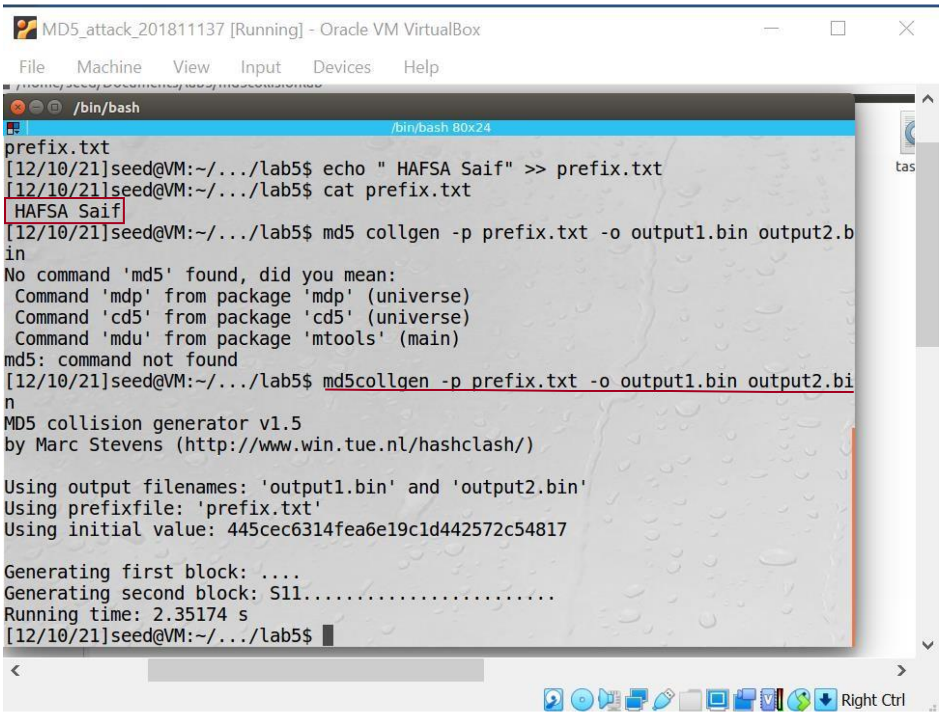
4 Tasks:

This lab required four particular tasks to be completed, and in the parts that follow, I summarize what I performed, what I saw, and what I learned from each task.

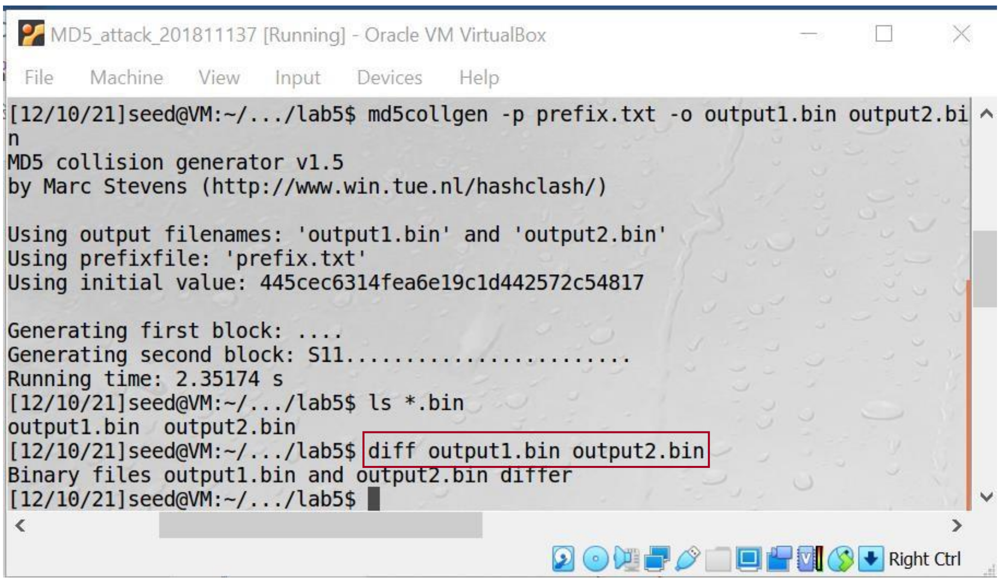
4.1 Task 1: Generating Two Different Files with the Same MD5 Hash.

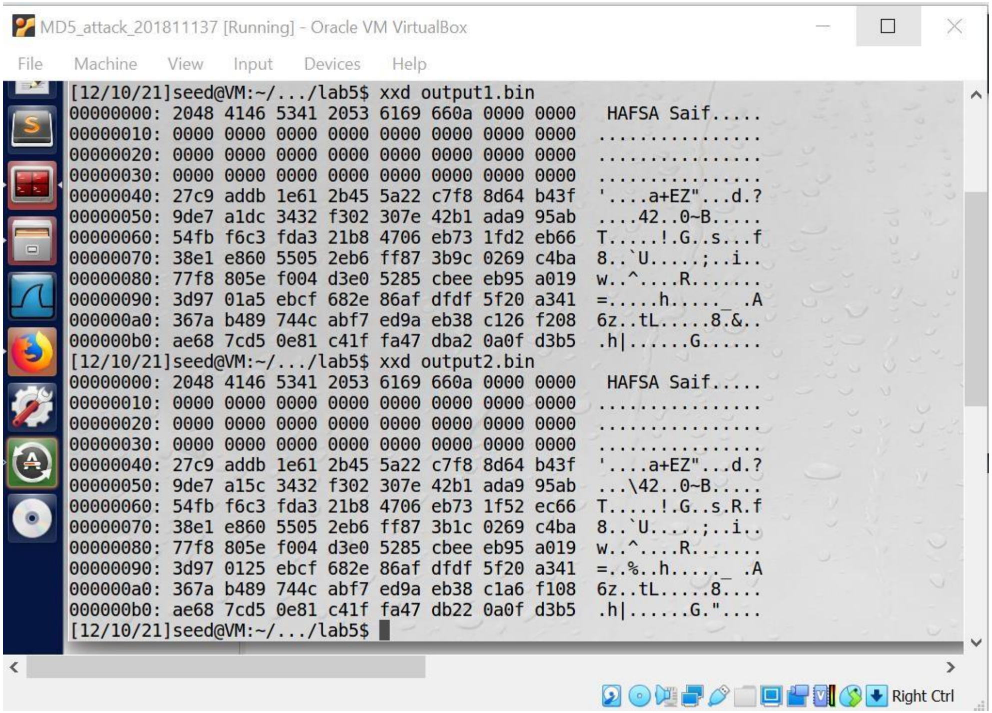
In task 1, I tried with the "md5collgen" tool to explore how to generate various files with the same MD5 hash. I specifically prepared a file called prefix.txt that only includes the string "HAFSA saif" and then executed the command:

md5collgen -p prefix.txt -o output1.bin output2.bin

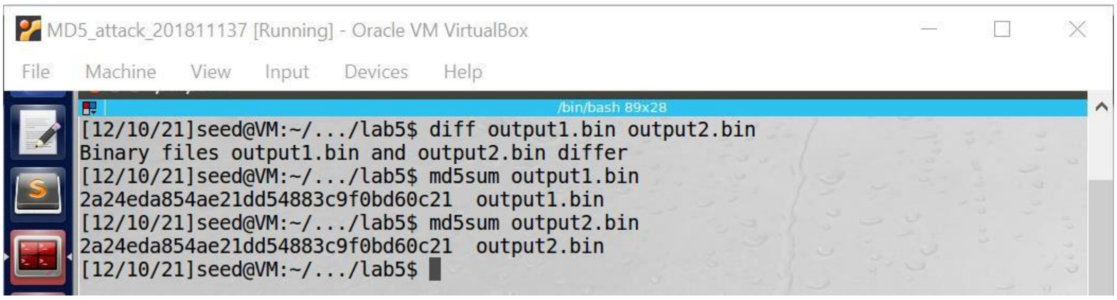


This resulted in two files that are distinct (as indicated by the diff command), but have the same md5 hash value (as determined by the md5sum command). The following screenshot depicts the process of executing diff and md5sum, as well as a hex dump of output1.bin generated by the "xxd" command:





According to the hex dump, the prefix.txt data was padded with zeros to get it up to 64 bytes, and then random-looking data was attached to that. The output2.bin file starts with the same 64 bytes (prefix and padding), but the subsequent bytes change somewhat.

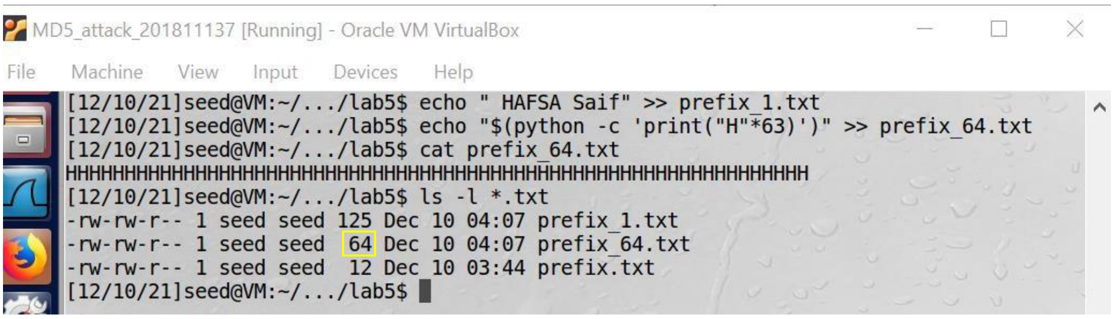


I produced a 92-byte prefix file and repeated the experiment to see what occurs with different prefix sizes. According to the hex dump, the padding bytes are utilized to make the prefix 128 bytes long. Then I carefully adjusted the prefix file to make it precisely 64 bytes long, and the md5collgen tool added no padding at all in this case. These tests produced all of the data needed to answer the lab write-queries. up's

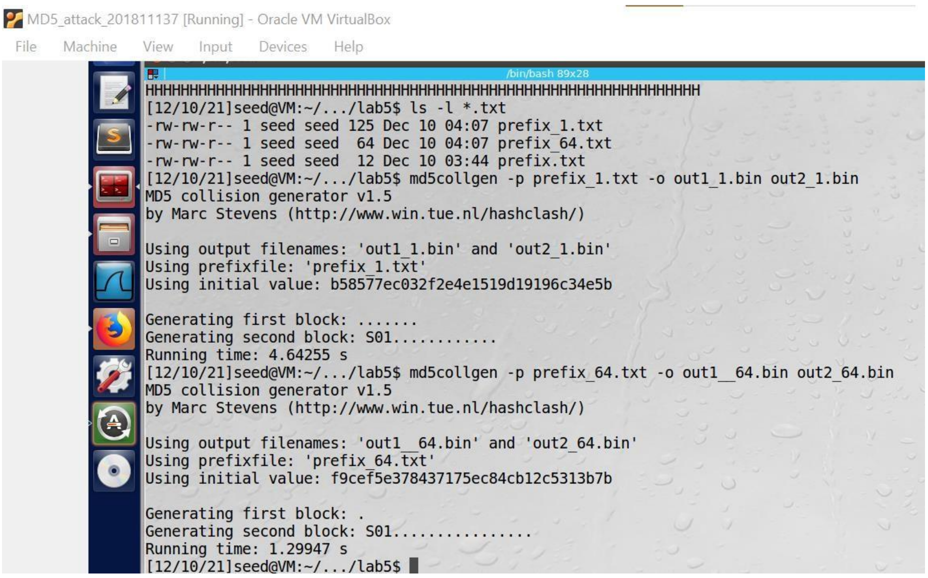
Discussion and Questions: For this task, the SEED lab asks three particular questions, which are answered below.

Quotation's 1: If the length of your prefix is not a multiple of 64, what is going to happen?

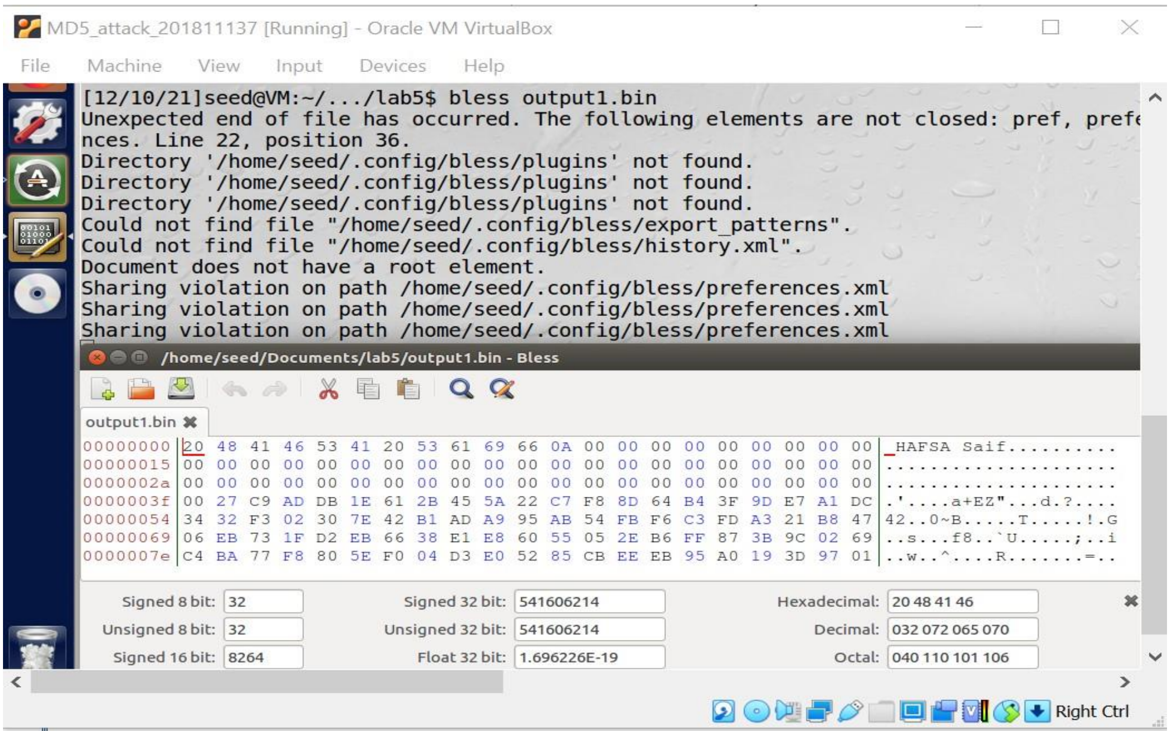
Answer: It will be padded with zeros or to be more specific the prefix was padded with zero bytes until the size was a multiple of 64.



To test this, I made a file called prefix 64.txt. Following the execution of md5collgen -p prefix 64.txt -o out1_64.bin out2_64.bin

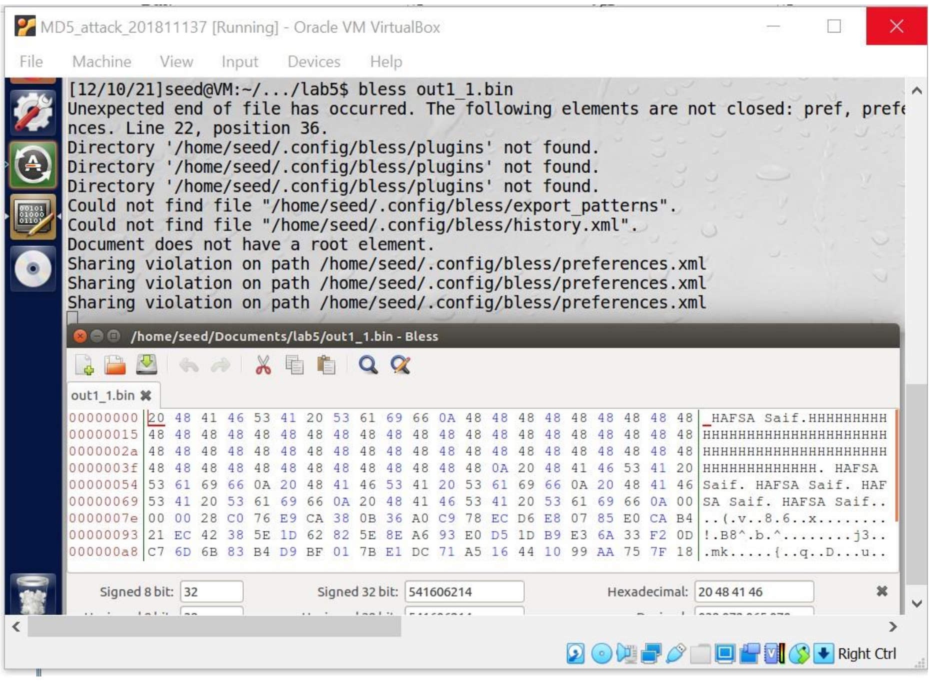


We can see from the output of bless output1.bin that it has been padded with zeros. This is due to MD5's processing of 64-byte blocks.



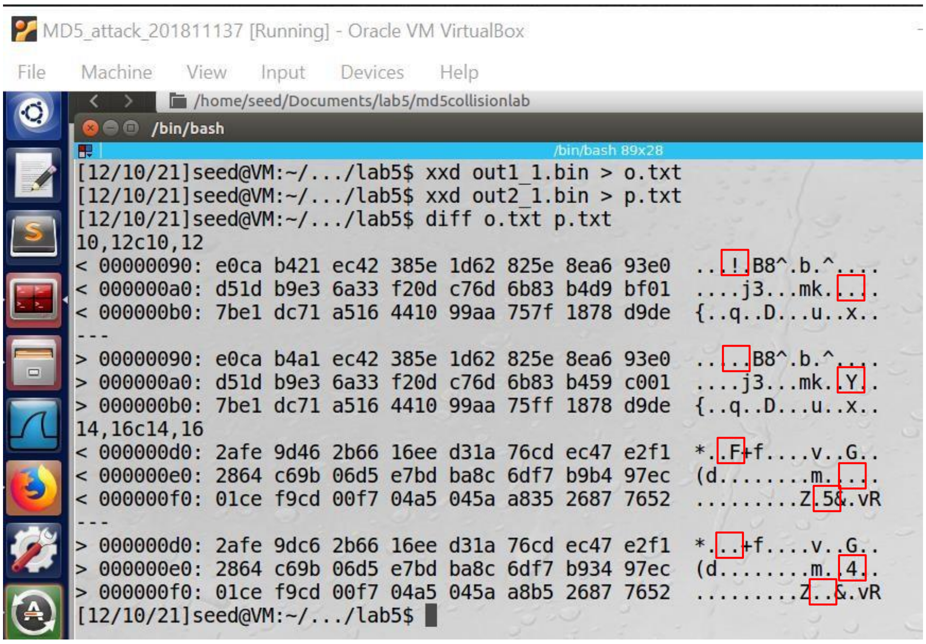
Question 2: Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

Answer: There are no zero-bytes inserted for padding with the 64-byte prefix. The prefix file is swiftly followed by the collision's random-looking data.



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

No, not all bytes are unique. In the prior situation, the bytes only change at places 10, 12, 14, and 16. After several attempts, it is discovered that these differences are not continuous.



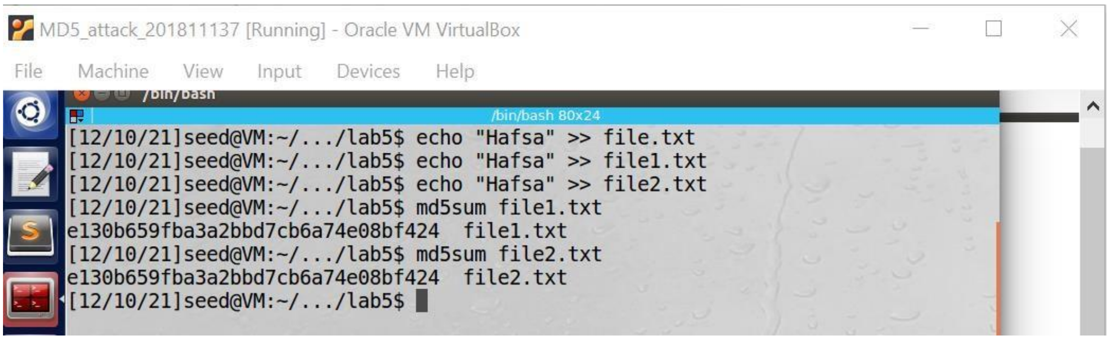
Finally, while 5 bytes have changed, the changes are minor: each byte that has changed has just a single bit difference between the outputs.

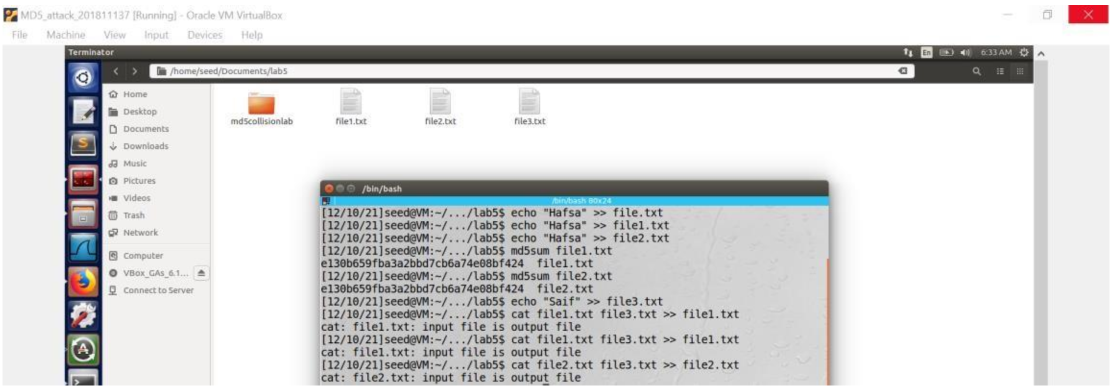
4.2 Task 2: Understanding MD5's Property.

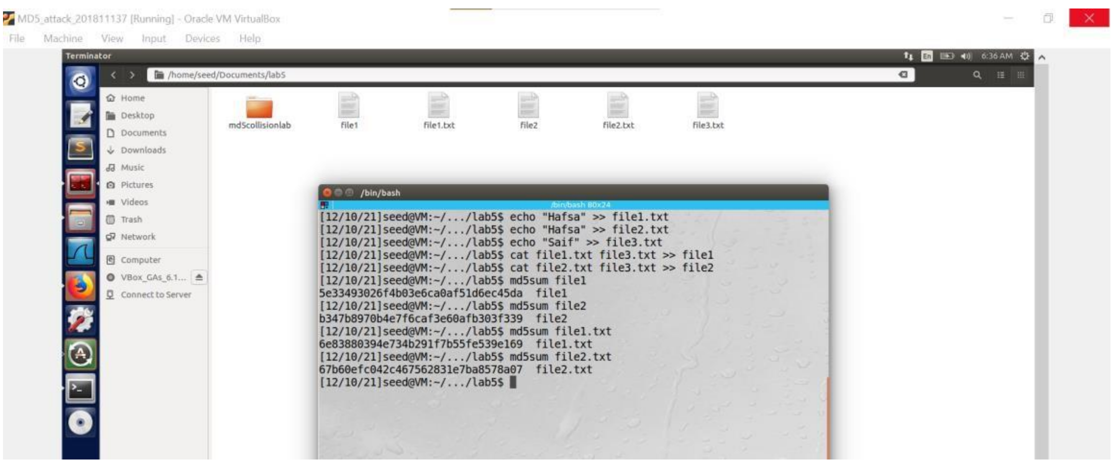
One interesting thing is that you may append the same file to the outputs of md5collgen and, while the MD5 hash changes, the resulting hashes will be same. This is due to the fact that many hash algorithms, such as SHA-1, SHA-2, and MD5, may be extended in length. The hash function keeps an internal state and processes the message in defined blocks by applying a compression function on the current state and block.

Because the MD5 hashes of both files were identical, it is fair to believe that the internal state after the method was performed was identical. Let us refer to this internal state as H. So, after appending a file, there will be a step in which the compression method is executed on H and the current block, which will be the same for both updated files. The MD5 hash for both files would be the same as a result of this.

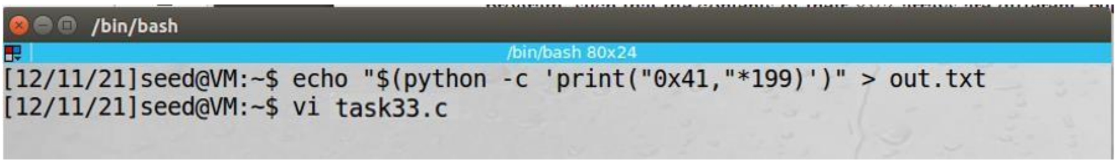
Here I was making 2 files



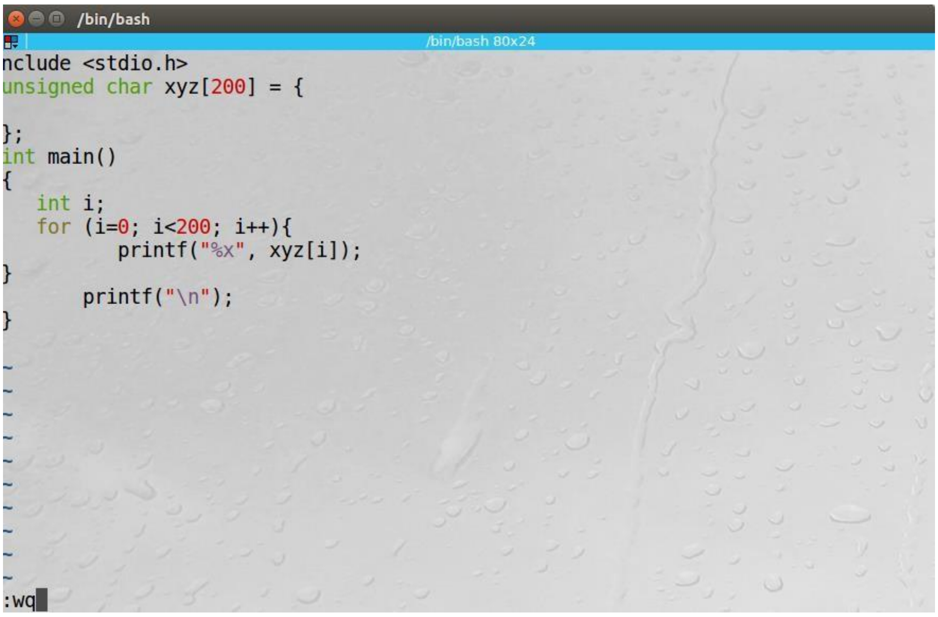


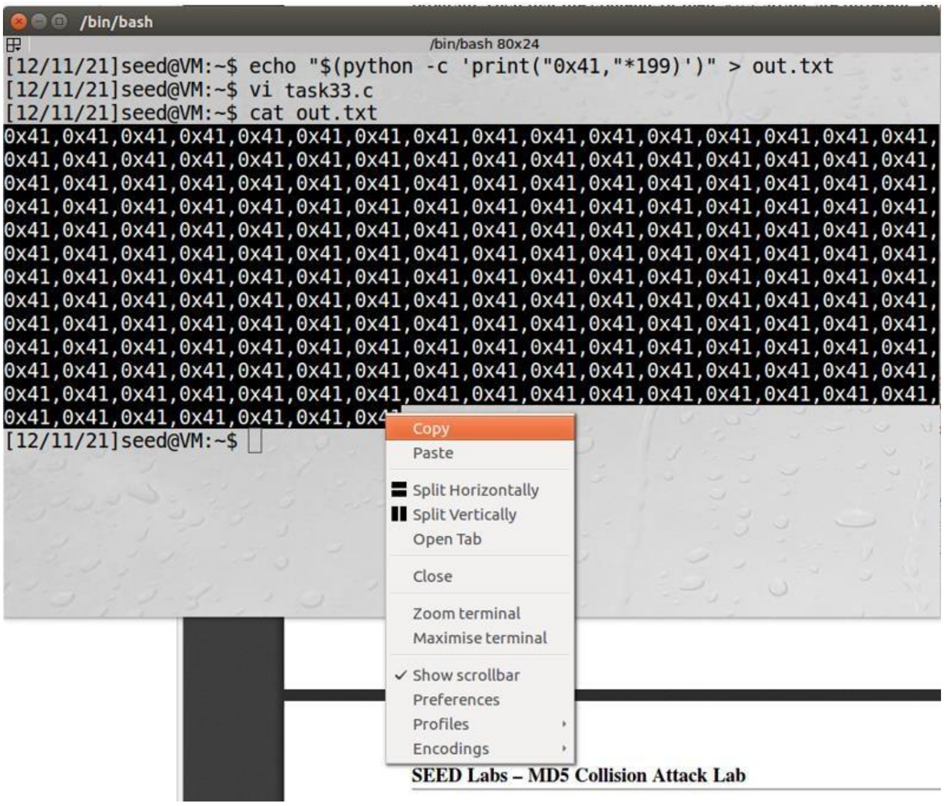


4.3 Task 3: Generating Two Executable Files with the Same MD5 Hash.



Vi task33.c to that command to open & write c program in terminal





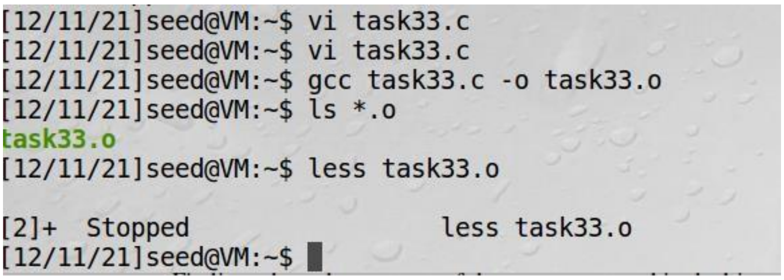
Then write command vi task33.c

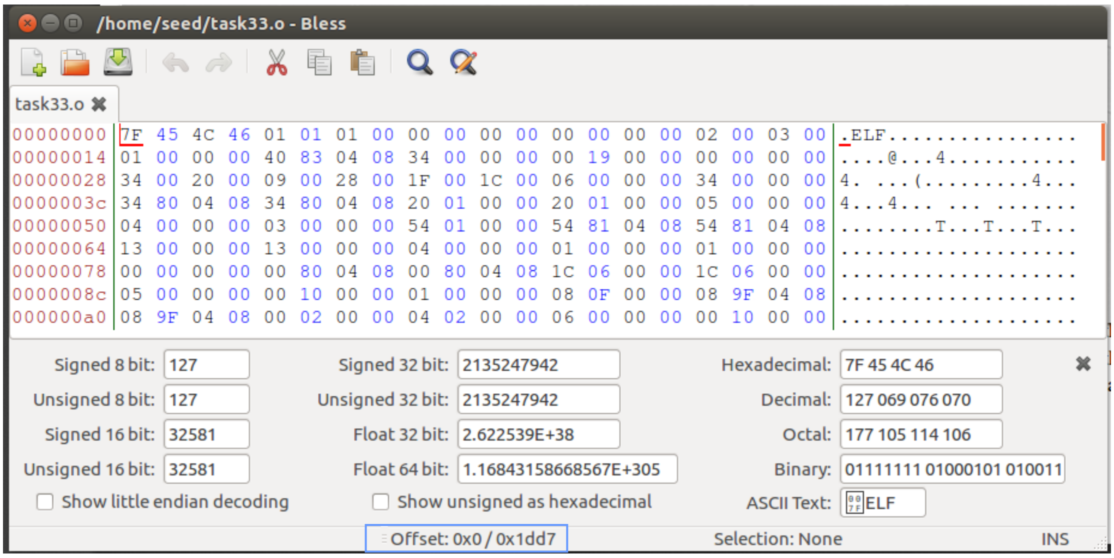
```
include <stdio.h>
unsigned char xyz[200] = {
0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
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
};
int main()
{
   inti;
   for (i=0; i<200; i++){
          printf("%x", xyz[i]);
}
       printf("\n");
-- INSERT --                                                  7,9          Top
```

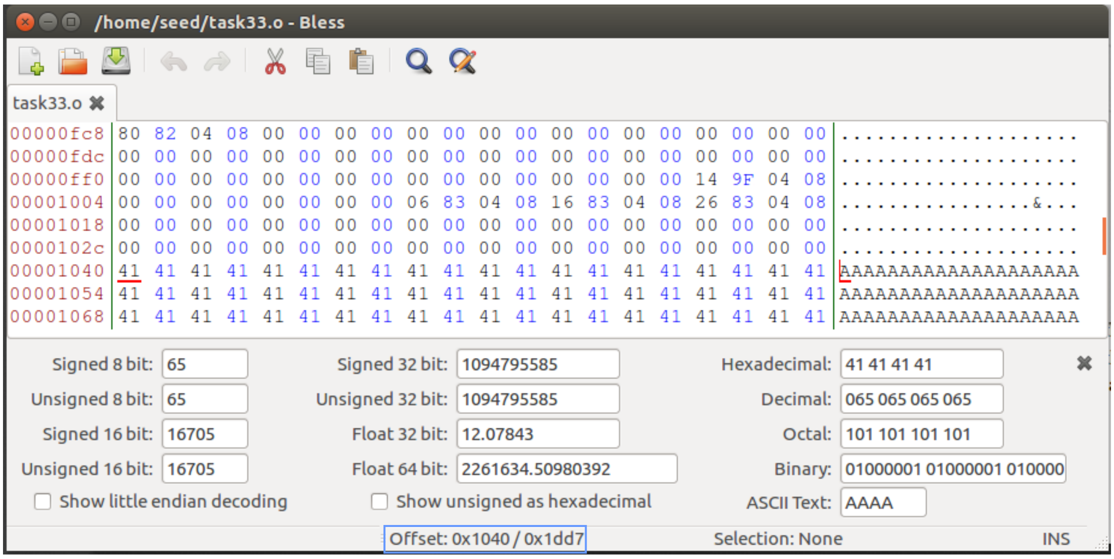
```
[12/11/21]seed@VM:~$ man gcc
```

```
GCC(1)                             GNU                              GCC(1)

NAME
       gcc - GNU project C and C++ compiler

SYNOPSIS
       gcc [-c|-S|-E] [-std=standard]
           [-g] [-pg] [-Olevel]
           [-Wwarn...] [-Wpedantic]
           [-Idir...] [-Ldir...]
           [-Dmacro[=defn]...] [-Umacro]
           [-foption...] [-mmachine-option...]
           [-o outfile] [@file] infile...

       Only the most useful options are listed here; see below for the
       remainder.  g++ accepts mostly the same options as gcc.

DESCRIPTION
       When you invoke GCC, it normally does preprocessing, compilation,
       assembly and linking.  The "overall options" allow you to stop this
       process at an intermediate stage.  For example, the -c option says not
       to run the linker.  Then the output consists of object files output by
       the assembler.
 Manual page gcc(1) line 1 (press h for help or q to quit)
```

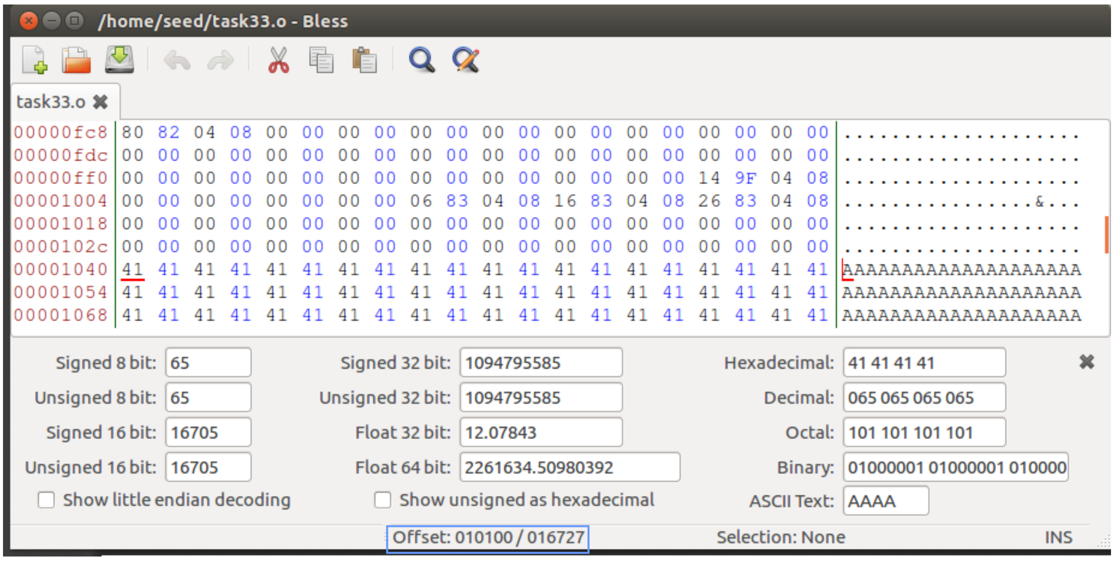
Compile the preceding code using gcc task33.c -o task33.o . It is simple to identify the position where the array values are written. In decimal terms, 0x1040 is 4160. This is divisible by 64, but for the time being, we'll go with a prefix length of 4224 bytes (0x1080). (This is optional, but I wanted the split to be in the center of the array.)



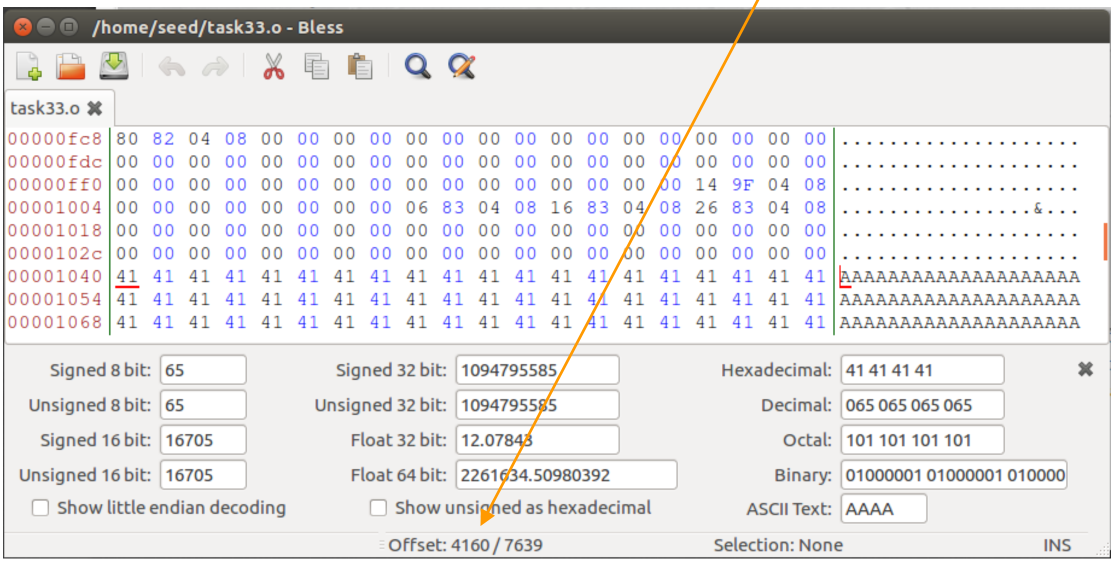




Click on here to get binary value

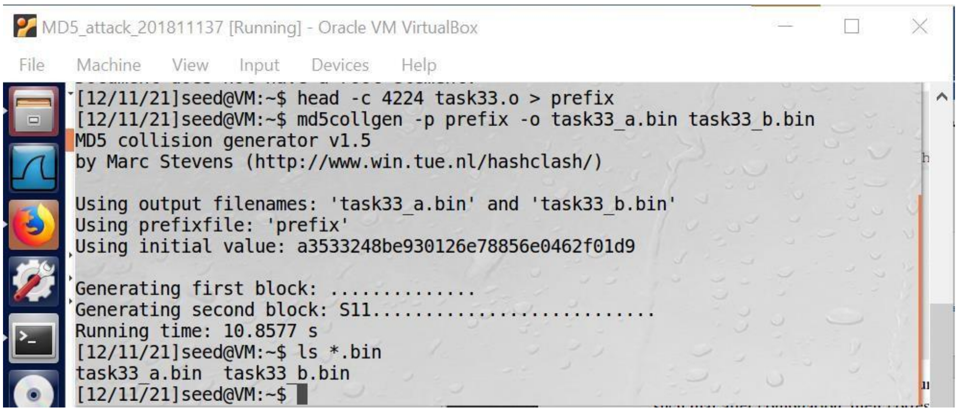


And if we click again, we will get Decimal

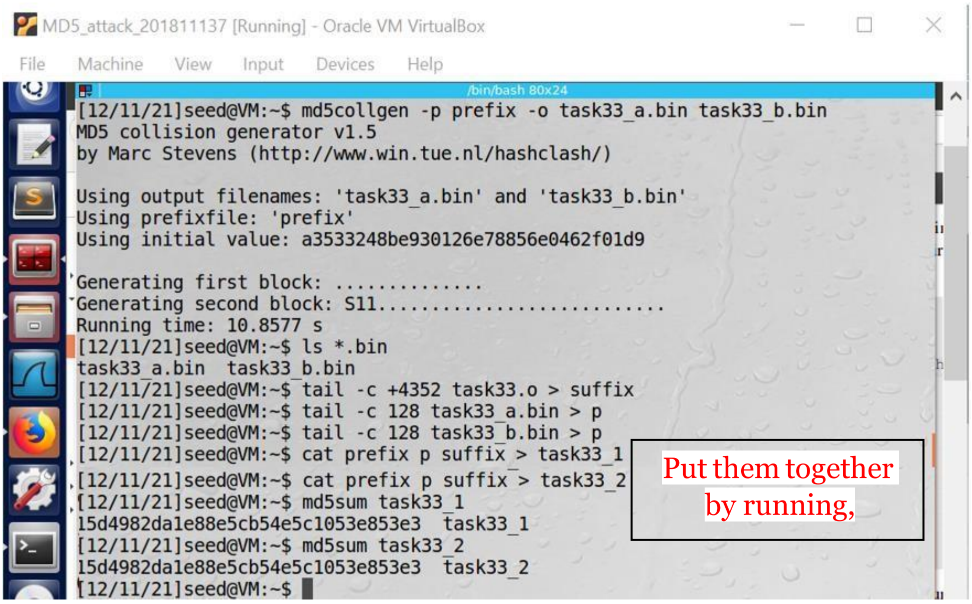


The length of the prefix needs to be multiple of 64 bytes.

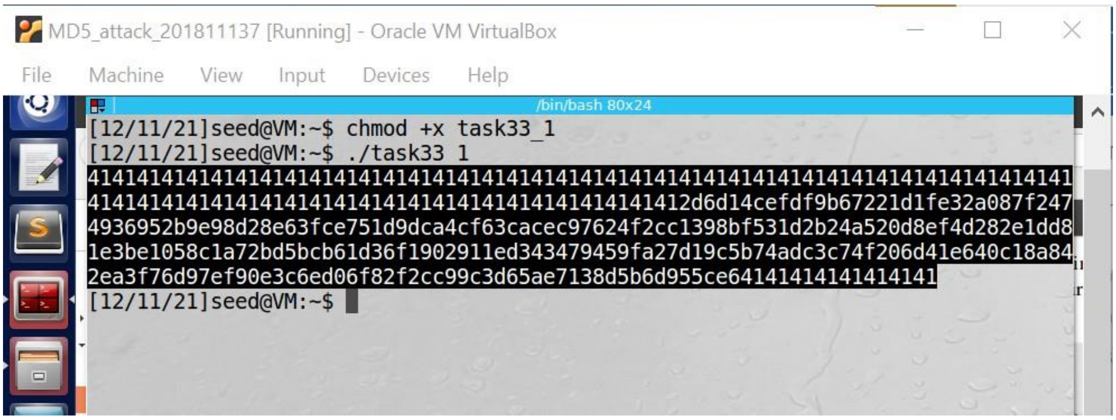
Running head -c 4224 task33.o > prefix and md5collgen -p prefix -o task33_a.bin task33_b.bin



Now we will get the common end to be appended using tail -c 4353 task33.o > suffix command.



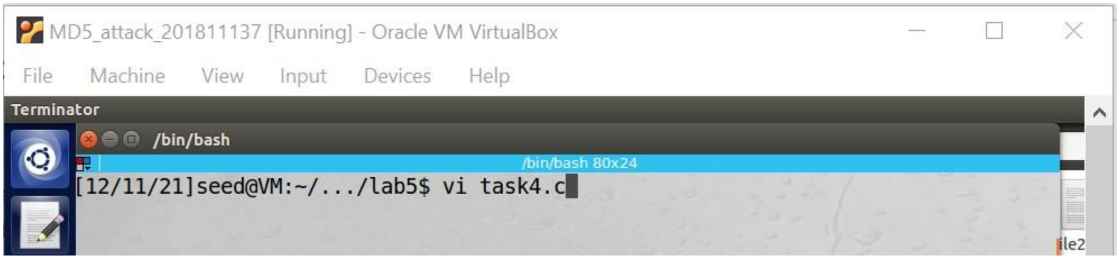
Add executable permission to both files and run them. Note that the output differs.



You can see the p value between the non-'A' values

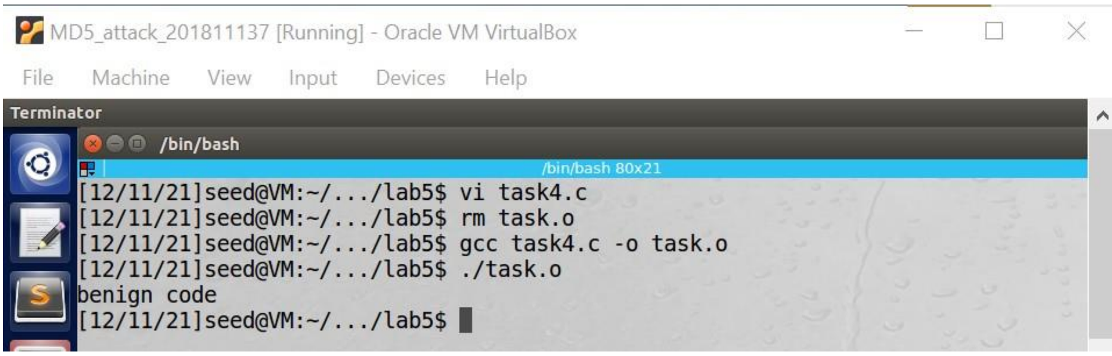
4.4 Task 4: Making the Two Programs Behave Differently.

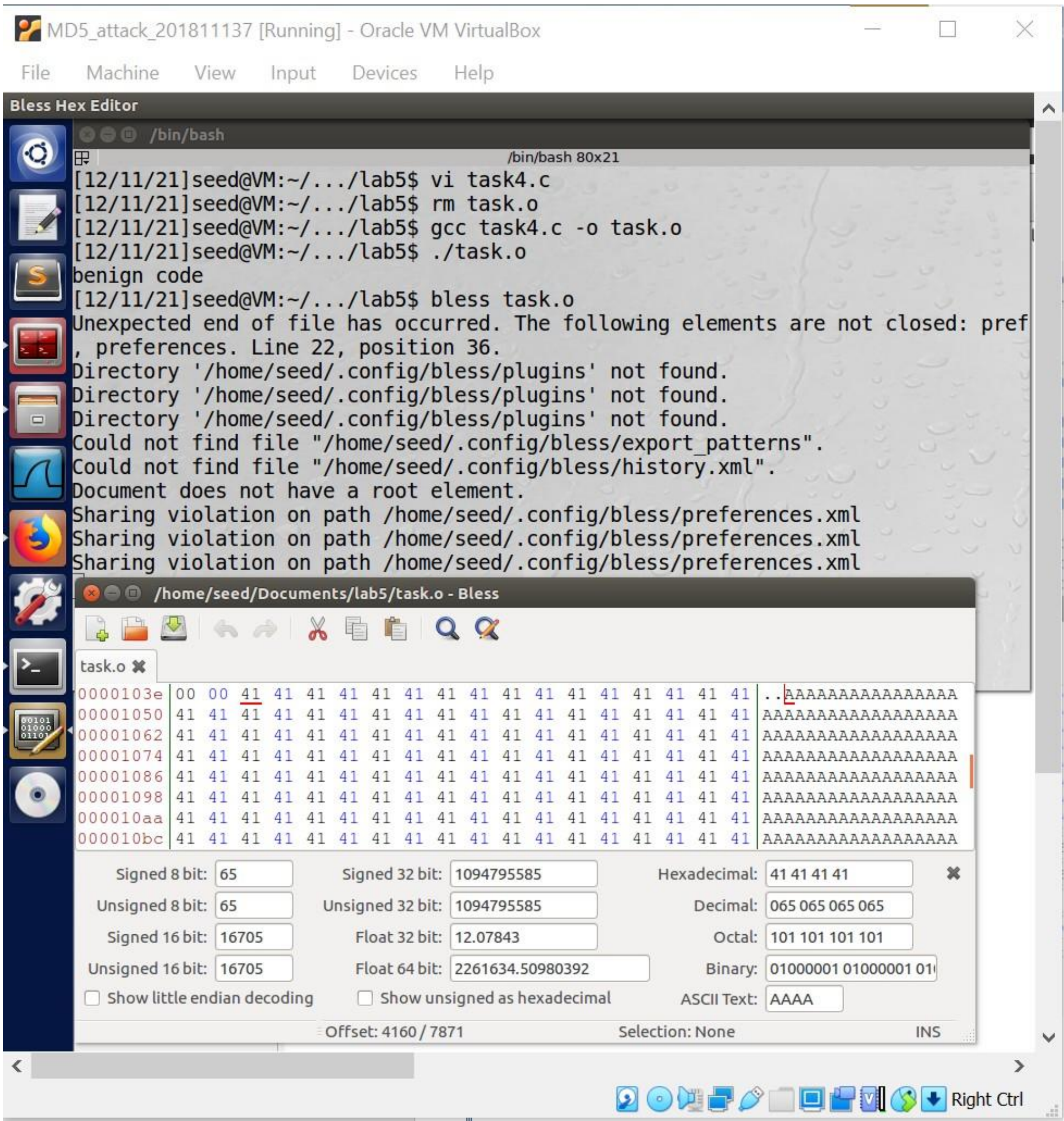
For this, I need to write two programs with the same MD5 sum but distinct behaviors. In this example, I will merely have the programs display different statements; but, in practice, the second program may run harmful code.



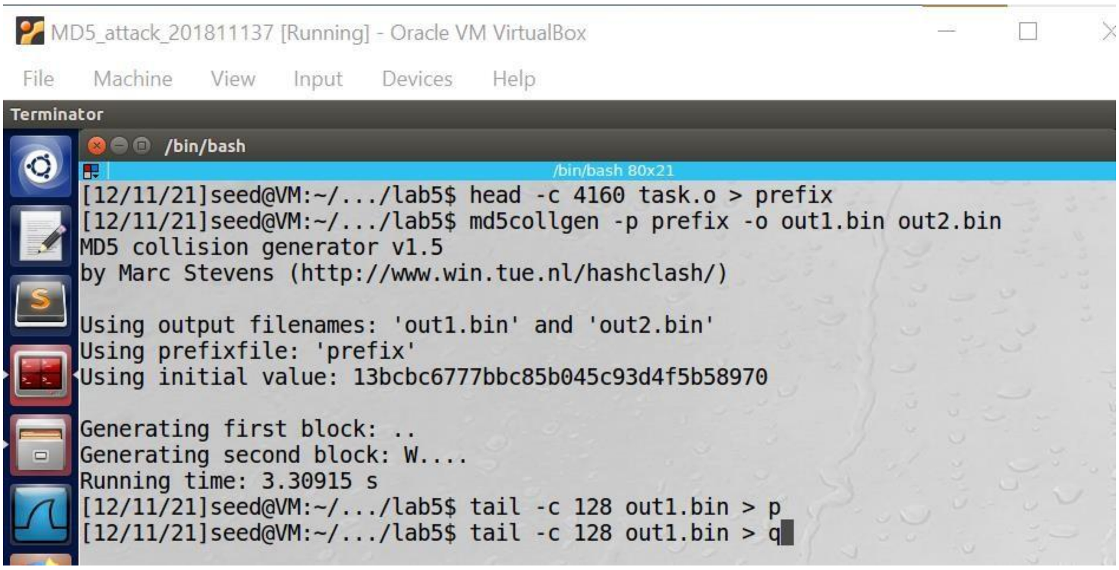
This is the program we'll be employing. Make a copy of the contents and save it as a task4.c.

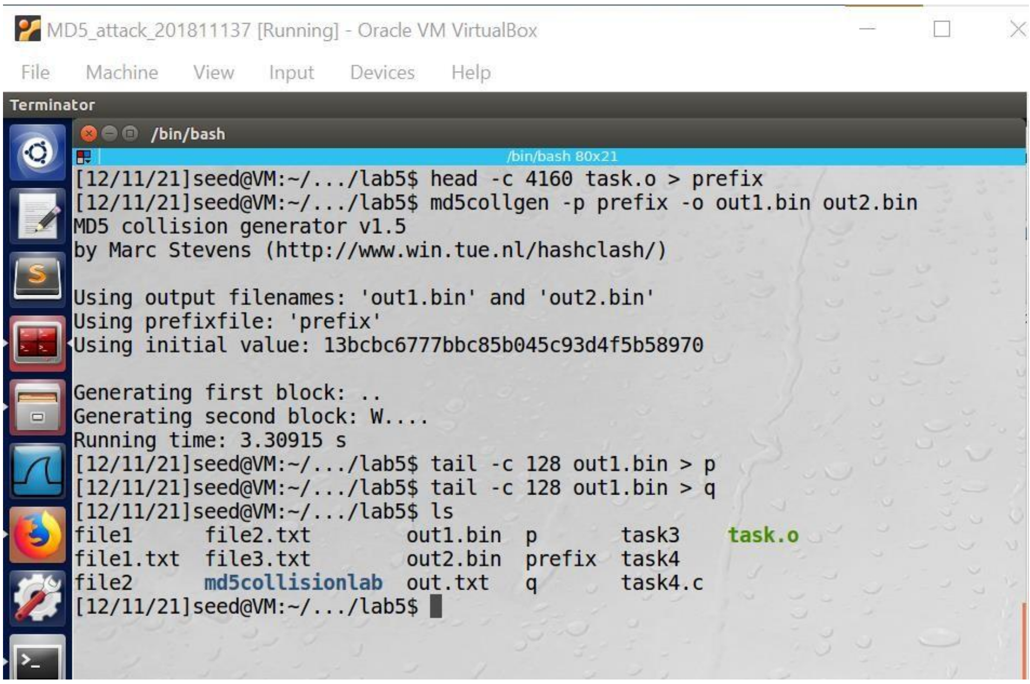
Now we compile the program with the gcc task4.c -o task.o. By default, the output binary is named task.o. Using bless task.o to examine the contents of task.o.

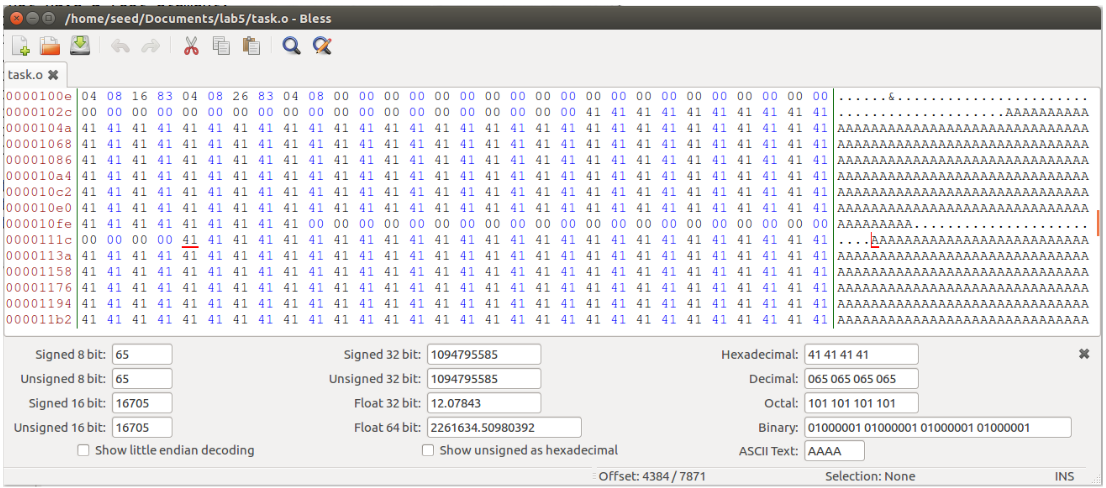


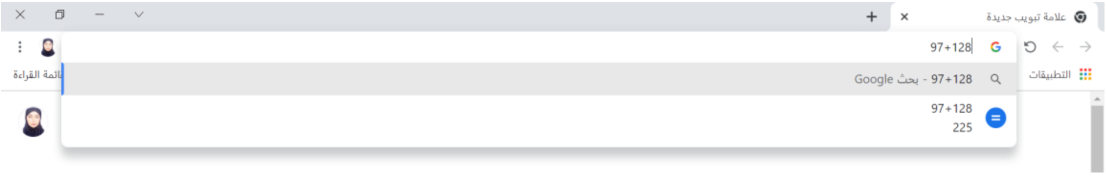


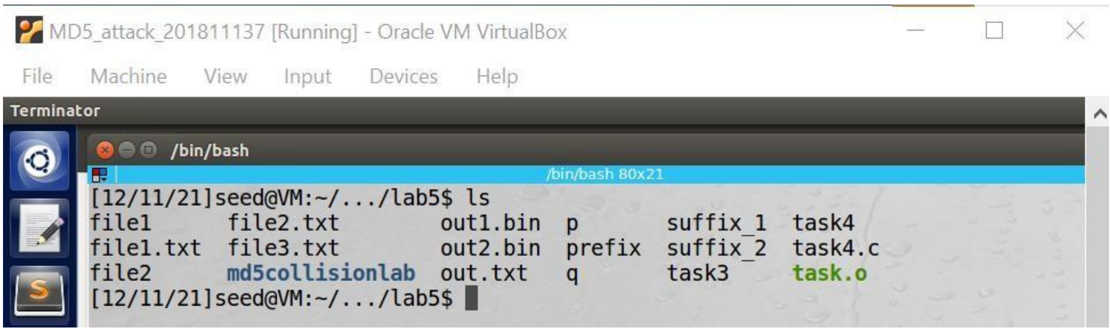
We can see that the first array begins at 0x1040 = 4160. As previously, the prefix can stretch till there, but I'll stick with 0x1080 to ensure that the created data begins in the center of the array.

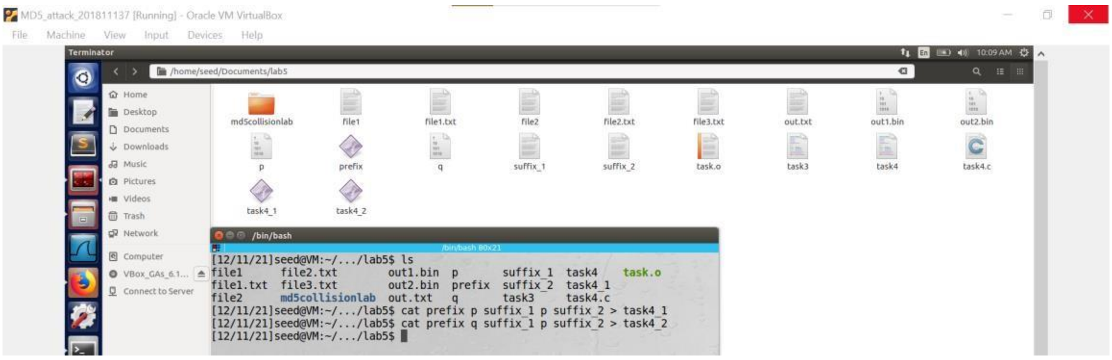

MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
/bin/bash
/bin/bash 80x21
[12/11/21]seed@VM:~/.../lab5$ head -c 4160 task.o > prefix
[12/11/21]seed@VM:~/.../lab5$ md5collgen -p prefix -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix'
Using initial value: 13bcbc6777bbc85b045c93d4f5b58970
Generating first block: ..
Generating second block: W....
Running time: 3.30915 s
[12/11/21]seed@VM:~/.../lab5$ tail -c 128 out1.bin > p
[12/11/21]seed@VM:~/.../lab5$ tail -c 128 out1.bin > q

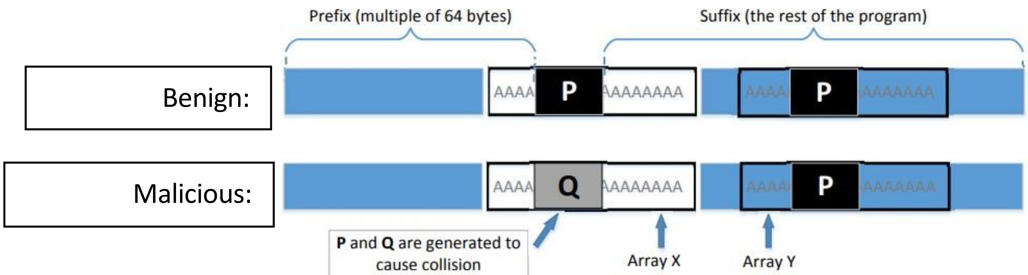

MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
/bin/bash
/bin/bash 80x21
[12/11/21]seed@VM:~/.../lab5$ head -c 4160 task.o > prefix
[12/11/21]seed@VM:~/.../lab5$ md5collgen -p prefix -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix'
Using initial value: 13bcbc6777bbc85b045c93d4f5b58970
Generating first block: ..
Generating second block: W....
Running time: 3.30915 s
[12/11/21]seed@VM:~/.../lab5$ tail -c 128 out1.bin > p
[12/11/21]seed@VM:~/.../lab5$ tail -c 128 out1.bin > q
[12/11/21]seed@VM:~/.../lab5$ ls
file1 file2.txt out1.bin p task3 task.o
file1.txt file3.txt out2.bin prefix task4
file2 md5collisionlab out.txt q task4.c
[12/11/21]seed@VM:~/.../lab5$

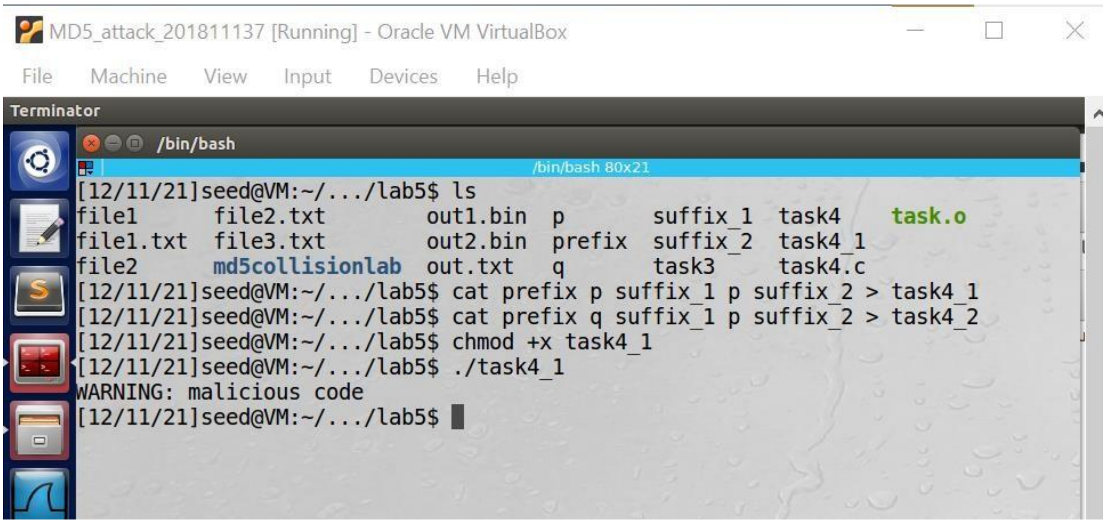

/home/seed/Documents/lab5/task.o - Bless
task.o
Signed 8 bit: 65
Unsigned 8 bit: 65
Signed 16 bit: 16705
Unsigned 16 bit: 16705
Show little endian decoding
Signed 32 bit: 1094795585
Unsigned 32 bit: 1094795585
Float 32 bit: 12.07843
Float 64 bit: 2261634.50980392
Show unsigned as hexadecimal
Hexadecimal: 41 41 41 41
Decimal: 065 065 065 065
Octal: 101 101 101 101
Binary: 01000001 01000001 01000001 01000001
ASCII Text: AAAA
Offset: 4384 / 7871
Selection: None
INS


علامة تبويب جديدة
97+128
Google بحث - 97+128
التطبيقات
قائمة القراءة
97+128
225


MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
/bin/bash
/bin/bash 80x21
[12/11/21]seed@VM:~/.../lab5$ ls
file1 file2.txt out1.bin p suffix_1 task4
file1.txt file3.txt out2.bin prefix suffix_2 task4.c
file2 md5collisionlab out.txt q task3 task.o
[12/11/21]seed@VM:~/.../lab5$

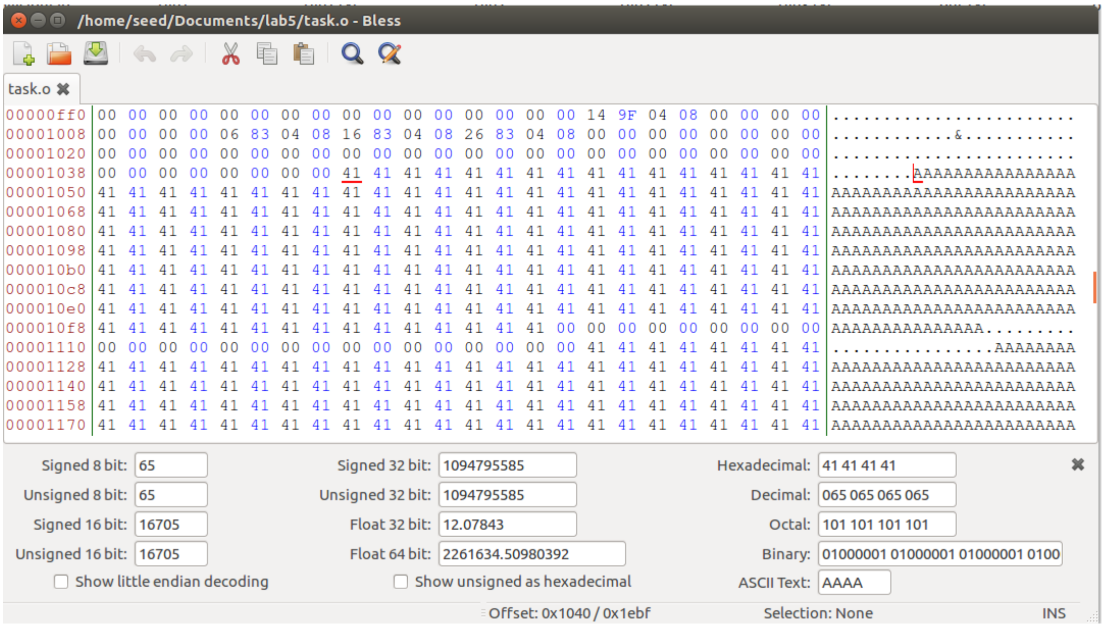

MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
/home/seed/Documents/lab5
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Network
Computer
Connect to Server
md5collisionlab file1 file1.txt file2 file2.txt file3.txt out.txt out1.bin out2.bin
p prefix q suffix_1 suffix_2 task.o task3 task4 task4.c
task4_1 task4_2
/bin/bash
[12/11/21]seed@VM:~/.../lab5$ ls
file1 file2.txt out1.bin p suffix_1 task4 task.o
file1.txt file3.txt out2.bin prefix suffix_2 task4_1
file2 md5collisionlab out.txt q task3 task4.c
[12/11/21]seed@VM:~/.../lab5$ cat prefix p suffix_1 p suffix_2 > task4_1
[12/11/21]seed@VM:~/.../lab5$ cat prefix q suffix_1 p suffix_2 > task4_2
[12/11/21]seed@VM:~/.../lab5$


Prefix (multiple of 64 bytes)
Suffix (the rest of the program)
Benign:
AAAA
P
AAAAAAAA
AAAA
P
AAAAAAA
Malicious:
AAAA
Q
AAAAAAAA
AAAA
P
AAAAAAA
P and Q are generated to cause collision
Array X
Array Y



This is not good

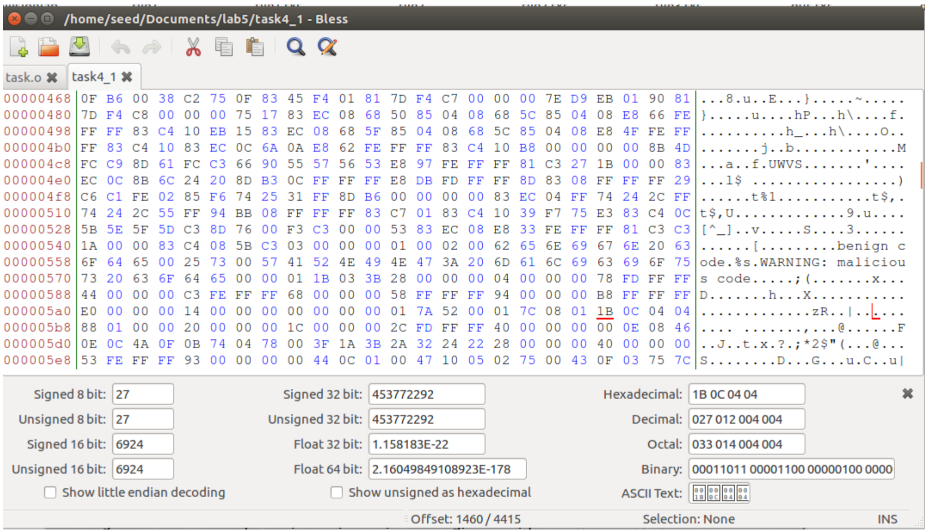
Time for some debugging



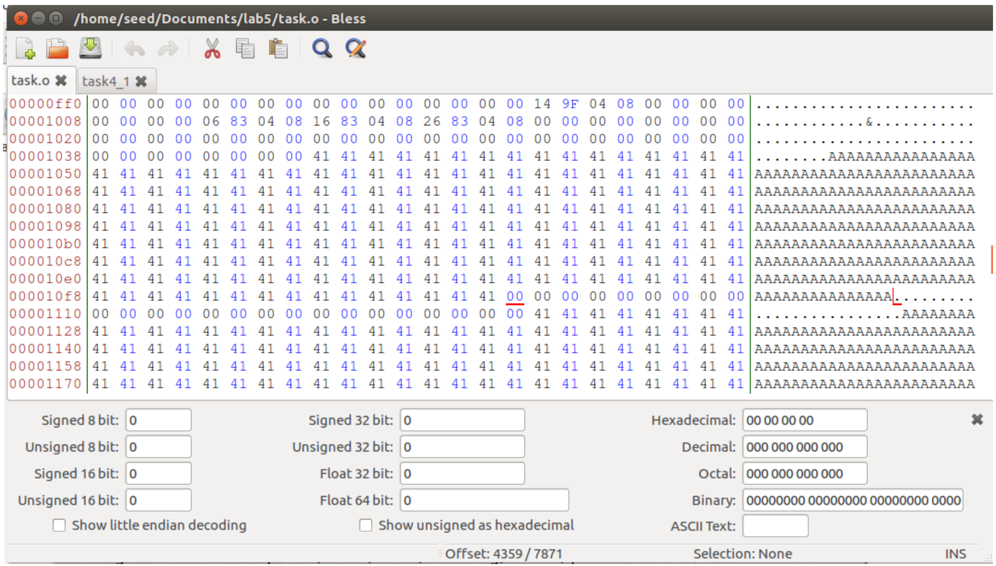
Attempting to align exec files..

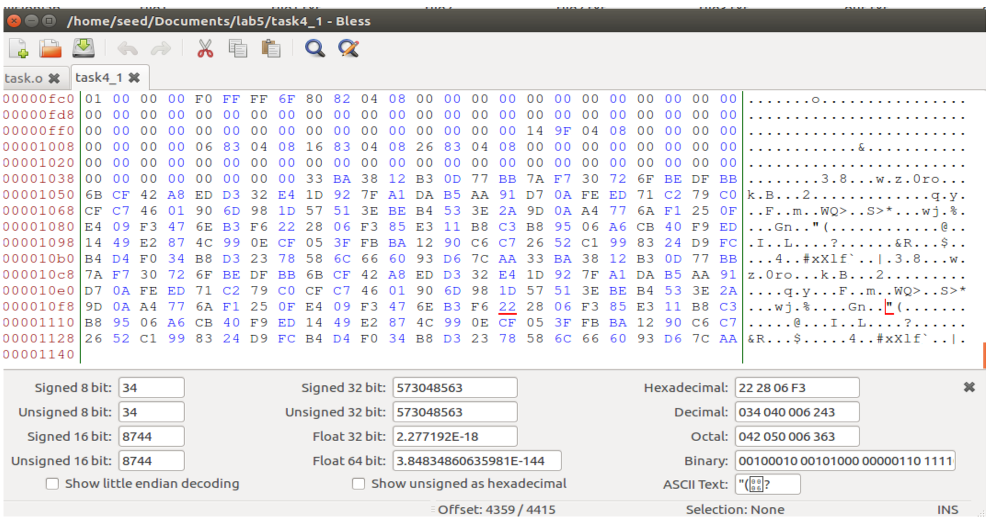
Array x start @ 0x1040

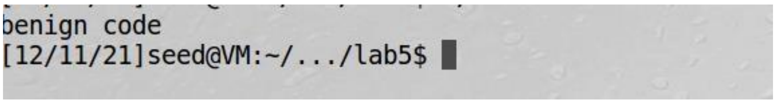
(i.e. offset of 4160-bytes)



Array x ends 4359-bytes into file







So, in last I conclude that Task4 develops two programs with distinct behaviour. Task 3 merely prints the material and follows the same set of instructions. It is feasible to sign malware using hash collision for certificates that first sign the file md5 and then the hash.

5 Summary:

Task 1:

In task 1, I tried with the "md5collgen" tool to explore how to generate various files with the same MD5 hash. I specifically prepared a file called prefix.txt that only includes the string "HAFSA saif" and then executed the command:

md5collgen -p prefix.txt -o output1.bin output2.bin

Task 2:

I devised several experiments to investigate the structure of the MD5 hash method as well as the application of iterated compression algorithms. To complete this work, I needed to produce particular binary files, therefore I learnt about the "bless" hex editor and used it to build the required files.

Task 3:

In task 3, we need to construct two alternative copies of this program, with different xyz array contents but the identical hash values for their executables.

Tasks 4:

In this task, we need to write two programs with the same MD5 sum but distinct behaviors. In this example, we will merely have the programs display different statements; but, in practice, the second program may run harmful code.

6 Conclusion:

In conclusion, from my notes I can discretion This SEED lab investigated the MD5 hash code and strategies for producing collisions with it. Collisions could be constructed fast for files with a highly organized format, and as a final assignment, we created two executable programs that performed quite different duties (one good and one evil) but had the same hash.

This lab demonstrated that the MD5 hash function cannot offer the requisite security for usage in an anti-virus system that detects file changes using the hash function. A malicious software creator, for example, may create an innocuous but appealing application (such as a free game) and then replace it with a harmful program that has the same MD5 hash value as the game. Because it has the same hash value, anti-virus software cannot identify that the file has changed!

While this is a specialized attack against MD5, I think that the MD5 algorithm has major flaws and should not be utilized.

7 References:

- Du, W. (2019). Computer & Internet Security: A Hands-on Approach. Wenliang Du. Injection, and the Slowloris Attack , by Avi Kak / https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture27.pdf
- A Meaningful MD5 Hash Collision Attack/ Narayana D. Kashyap / https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1020&context=etd_projects
- Collision attack/ https://academic.microsoft.com/topic/87538441/publication/search?q=Collision%20attack&qe=And(Composite(F.FId%253D87538441)%252CTy%253D%270%27)&f=&orderBy=0