

MD5 Collision Attack Lab

Table of Contents:

Title page.....	1
Table of contents	2
Abstract	3
Introduction	3
Objective	3
Tasks	4
Task 1: Generating Two Different Files with the Same MD5 Hash.....	4
Task 2: Understanding MD5's Property.....	8
Task 3: Generating Two Executable Files with the Same MD5 Hash.....	9
Task 4: Making the Two Programs Behave Differently.....	14
Summary	20
Conclusion	20
References	21

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 to 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.

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

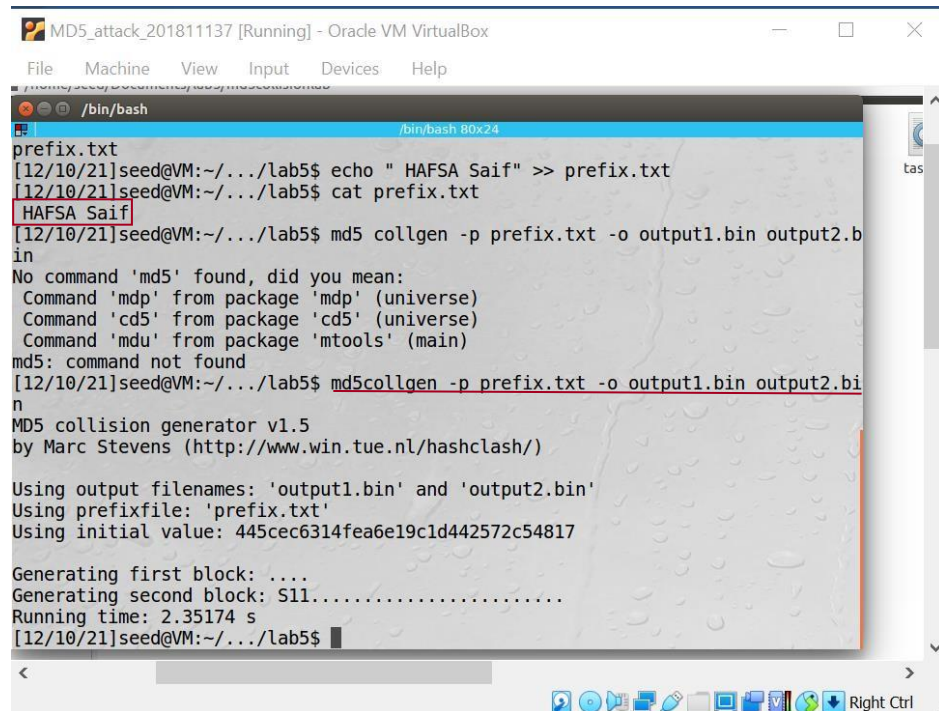
4 Tasks:

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
```

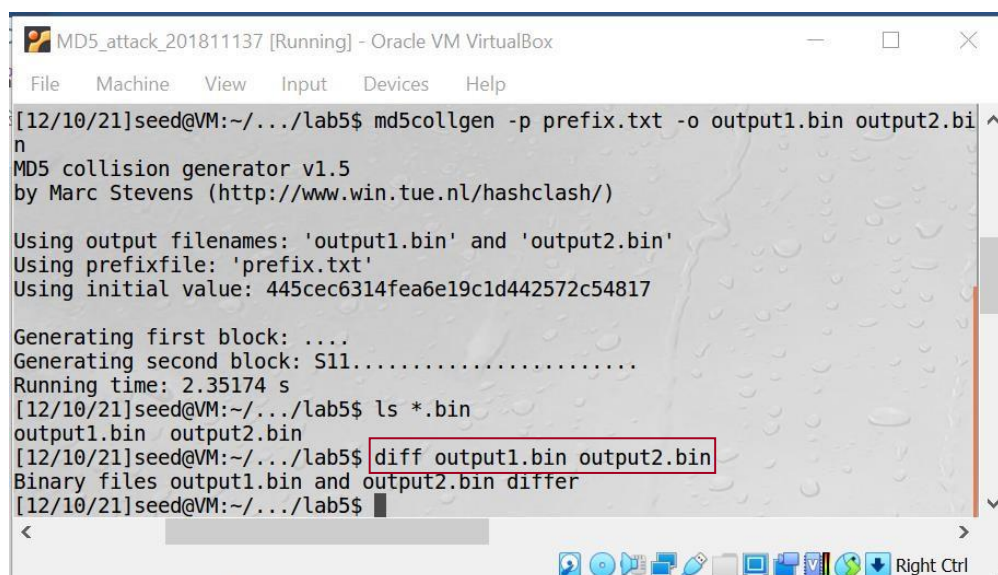


```
MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
prefix.txt
[12/10/21]seed@VM:~/.../lab5$ echo " HAFSA Saif" >> prefix.txt
[12/10/21]seed@VM:~/.../lab5$ cat prefix.txt
HAFSA Saif
[12/10/21]seed@VM:~/.../lab5$ md5 collgen -p prefix.txt -o output1.bin output2.b
in
No command 'md5' found, did you mean:
  Command 'mdp' from package 'mdp' (universe)
  Command 'cd5' from package 'cd5' (universe)
  Command 'mdu' from package 'mtools' (main)
md5: command not found
[12/10/21]seed@VM:~/.../lab5$ md5collgen -p prefix.txt -o output1.bin output2.bi
n
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'output1.bin' and 'output2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 445cec6314fea6e19c1d442572c54817

Generating first block: ....
Generating second block: S11.....
Running time: 2.35174 s
[12/10/21]seed@VM:~/.../lab5$
```

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:



```
MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
[12/10/21]seed@VM:~/.../lab5$ md5collgen -p prefix.txt -o output1.bin output2.bi
n
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'output1.bin' and 'output2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 445cec6314fea6e19c1d442572c54817

Generating first block: ....
Generating second block: S11.....
Running time: 2.35174 s
[12/10/21]seed@VM:~/.../lab5$ ls *.bin
output1.bin  output2.bin
[12/10/21]seed@VM:~/.../lab5$ diff output1.bin output2.bin
Binary files output1.bin and output2.bin differ
[12/10/21]seed@VM:~/.../lab5$
```

```

[12/10/21]seed@VM:~/.../lab5$ xxd output1.bin
00000000: 2048 4146 5341 2053 6169 660a 0000 0000  HAFSA Saif.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000040: 27c9 addb 1e61 2b45 5a22 c7f8 8d64 b43f  '....a+EZ"...d.?
00000050: 9de7 a1dc 3432 f302 307e 42b1 ada9 95ab  ....42..0-B....
00000060: 54fb f6c3 fda3 21b8 4706 eb73 1fd2 eb66  T.....!G..s..f
00000070: 38e1 e860 5505 2eb6 ff87 3b9c 0269 c4ba  8..`U.....;..i..
00000080: 77f8 805e f004 d3e0 5285 cbee eb95 a019  w..^....R.....
00000090: 3d97 01a5 ebcf 682e 86af dfdf 5f20 a341  =....h....._A
000000a0: 367a b489 744c abf7 ed9a eb38 c126 f208  6z..tL.....8.&..
000000b0: ae68 7cd5 0e81 c41f fa47 dba2 0a0f d3b5  .h|.....G.....
[12/10/21]seed@VM:~/.../lab5$ xxd output2.bin
00000000: 2048 4146 5341 2053 6169 660a 0000 0000  HAFSA Saif.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000040: 27c9 addb 1e61 2b45 5a22 c7f8 8d64 b43f  '....a+EZ"...d.?
00000050: 9de7 a15c 3432 f302 307e 42b1 ada9 95ab  ...42..0-B....
00000060: 54fb f6c3 fda3 21b8 4706 eb73 1f52 ec66  T.....!G..s.R.f
00000070: 38e1 e860 5505 2eb6 ff87 3b1c 0269 c4ba  8..`U.....;..i..
00000080: 77f8 805e f004 d3e0 5285 cbee eb95 a019  w..^....R.....
00000090: 3d97 0125 ebcf 682e 86af dfdf 5f20 a341  =..%.h....._A
000000a0: 367a b489 744c abf7 ed9a eb38 c1a6 f108  6z..tL.....8....
000000b0: ae68 7cd5 0e81 c41f fa47 db22 0a0f d3b5  .h|.....G."....
[12/10/21]seed@VM:~/.../lab5$

```

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.

```

[12/10/21]seed@VM:~/.../lab5$ diff output1.bin output2.bin
Binary files output1.bin and output2.bin differ
[12/10/21]seed@VM:~/.../lab5$ md5sum output1.bin
2a24eda854ae21dd54883c9f0bd60c21  output1.bin
[12/10/21]seed@VM:~/.../lab5$ md5sum output2.bin
2a24eda854ae21dd54883c9f0bd60c21  output2.bin
[12/10/21]seed@VM:~/.../lab5$

```

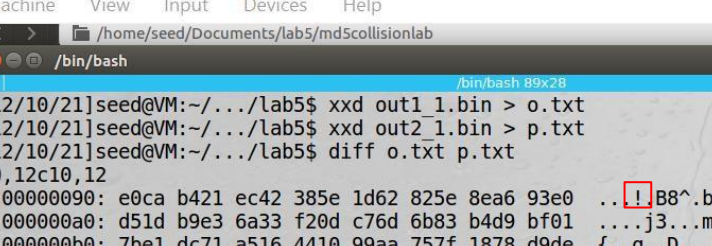
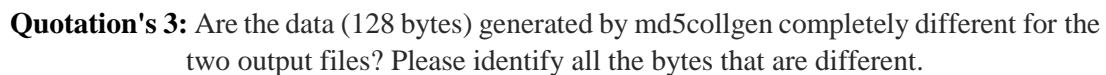
I produced a 92-byte prefix file and repeated the experiment to see what occurs with different prefix sizes. According to the hex dump, the padding bytes are utilized to make the prefix 128 bytes long. Then I carefully adjusted the prefix file to make it precisely 64 bytes long, and the md5collgen tool 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.

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.



MD5_attack_201811137 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

/home/seed/Documents/lab5/md5collisionlab

/bin/bash

/bin/bash 89x28

```
[12/10/21]seed@VM:~/.../lab5$ xxd out1_1.bin > o.txt
[12/10/21]seed@VM:~/.../lab5$ xxd out2_1.bin > p.txt
[12/10/21]seed@VM:~/.../lab5$ diff o.txt p.txt
10,12c10,12
< 00000090: e0ca b421 ec42 385e 1d62 825e 8ea6 93e0  ..!B8^b.^...
< 000000a0: d51d b9e3 6a33 f20d c76d 6b83 b4d9 bf01  ....j3...mk....
< 000000b0: 7be1 dc71 a516 4410 99aa 757f 1878 d9de  {...q..D...u..X..
...
> 00000090: e0ca b4a1 ec42 385e 1d62 825e 8ea6 93e0  ..B8^b.^...
> 000000a0: d51d b9e3 6a33 f20d c76d 6b83 b459 c001  ....j3...mk..Y..
> 000000b0: 7be1 dc71 a516 4410 99aa 75ff 1878 d9de  {...q..D...u..X..
14,16c14,16
< 000000d0: 2afe 9d46 2b66 16ee d31a 76cd ec47 e2f1  *..F+f.....v..G..
< 000000e0: 2864 c69b 06d5 e7bd ba8c 6df7 b9b4 97ec  (d.....m.....
< 000000f0: 01ce f9cd 00f7 04a5 045a a835 2687 7652  .......Z..5&..vR
...
> 000000d0: 2afe 9dc6 2b66 16ee d31a 76cd ec47 e2f1  *..+f.....v..G..
> 000000e0: 2864 c69b 06d5 e7bd ba8c 6df7 b934 97ec  (d.....m.....4..
> 000000f0: 01ce f9cd 00f7 04a5 045a a8b5 2687 7652  .......Z..&..vR
[12/10/21]seed@VM:~/.../lab5$
```

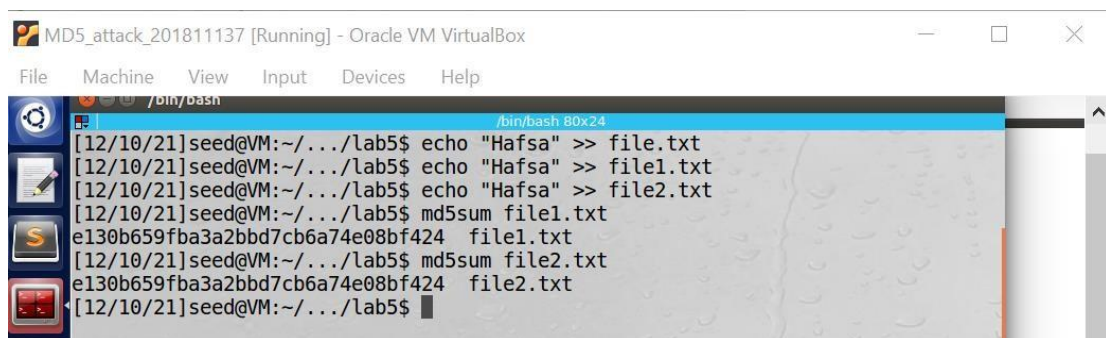
7

4.2 Task 2: Understanding MD5's Property.

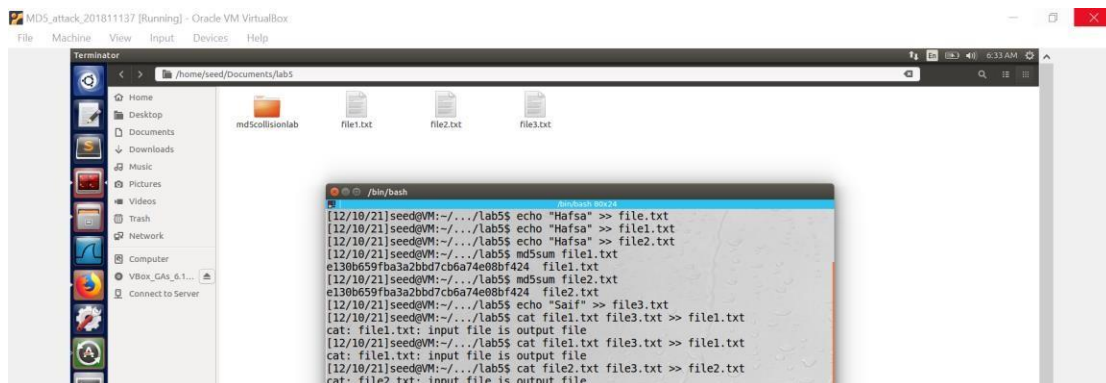
One interesting thing is that you may append the same file to the outputs of md5collgen and, while the MD5 hash changes, the resulting hashes will be same. This is due to the fact that many hash algorithms, such as SHA-1, SHA-2, and MD5, may be extended in length. The hash function keeps an internal 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.

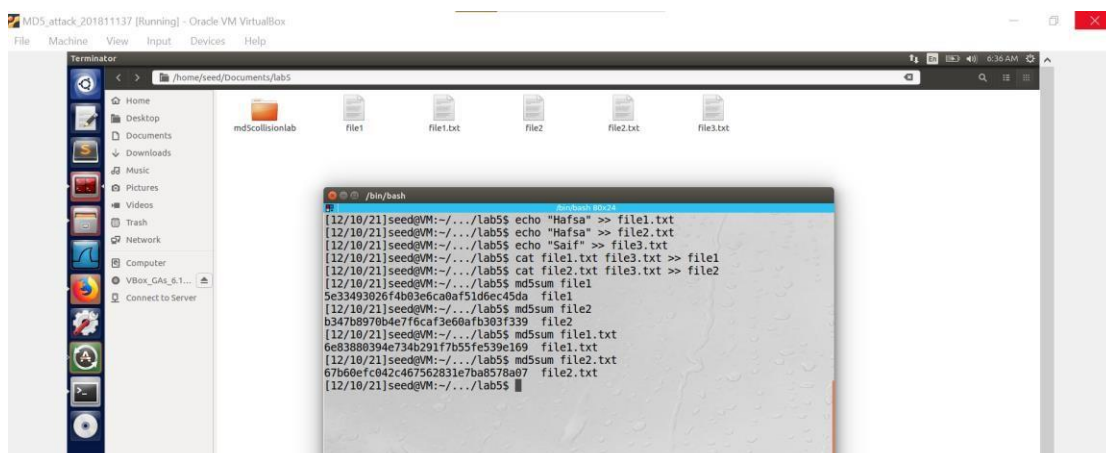
Here I was making 2 files



```
MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/bin/bash
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file.txt
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file1.txt
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file2.txt
[12/10/21]seed@VM:~/.../lab5$ md5sum file1.txt
e130b659fba3a2bbd7cb6a74e08bf424 file1.txt
[12/10/21]seed@VM:~/.../lab5$ md5sum file2.txt
e130b659fba3a2bbd7cb6a74e08bf424 file2.txt
[12/10/21]seed@VM:~/.../lab5$
```



```
MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
/home/seed/Documents/lab5
md5collisionlab file1.txt file2.txt file3.txt
/bin/bash
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file1.txt
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file2.txt
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file3.txt
[12/10/21]seed@VM:~/.../lab5$ md5sum file1.txt
e130b659fba3a2bbd7cb6a74e08bf424 file1.txt
[12/10/21]seed@VM:~/.../lab5$ md5sum file2.txt
e130b659fba3a2bbd7cb6a74e08bf424 file2.txt
[12/10/21]seed@VM:~/.../lab5$ cat file1.txt file3.txt >> file1.txt
cat: file1.txt: input file is output file
[12/10/21]seed@VM:~/.../lab5$ cat file1.txt file3.txt >> file1.txt
cat: file1.txt: input file is output file
[12/10/21]seed@VM:~/.../lab5$ cat file2.txt file3.txt >> file2.txt
cat: file2.txt: input file is output file
```



```
MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
/home/seed/Documents/lab5
md5collisionlab file1 file1.txt file2 file2.txt file3.txt
/bin/bash
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file1.txt
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file2.txt
[12/10/21]seed@VM:~/.../lab5$ echo "Hafsa" >> file3.txt
[12/10/21]seed@VM:~/.../lab5$ cat file1.txt file3.txt >> file1
[12/10/21]seed@VM:~/.../lab5$ cat file2.txt file3.txt >> file2
[12/10/21]seed@VM:~/.../lab5$ md5sum file1
5e33493026f4b03e6ca8af51d6ec45da file1
[12/10/21]seed@VM:~/.../lab5$ md5sum file2
b347b8970b4e7f6caf3e60afb303f339 file2
[12/10/21]seed@VM:~/.../lab5$ md5sum file1.txt
6e83880394e734b291f7b55fe539e169 file1.txt
[12/10/21]seed@VM:~/.../lab5$ md5sum file2.txt
67b6befe942c467562831e7ba8578a07 file2.txt
[12/10/21]seed@VM:~/.../lab5$
```


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

```

/bin/bash
/bin/bash 80x24
[12/11/21]seed@VM:~$ echo "$(python -c 'print("0x41,"*199)')" > out.txt
[12/11/21]seed@VM:~$ vi task33.c

```

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

```
/bin/bash
/bin/bash 80x24
#include <stdio.h>
unsigned char xyz[200] = {
};
int main()
{
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

[illegible]

Then write command `vi task33.c`

```
/bin/bash /bin/bash 80x24
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,
0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
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 --
```



A terminal window with a dark gray title bar containing window control icons and the text `/bin/bash`. The terminal has a light blue header bar with `/bin/bash 80x24` on the right. The main area is white with a black cursor. The prompt is `[12/11/21]seed@VM:~$` and the command `man gcc` has been entered.

```

/bin/bash
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.)

```
[12/11/21]seed@VM:~$ vi task33.c
[12/11/21]seed@VM:~$ vi task33.c
[12/11/21]seed@VM:~$ gcc task33.c -o task33.o
[12/11/21]seed@VM:~$ ls *.o
task33.o
[12/11/21]seed@VM:~$ less task33.o

[2]+  Stopped                  less task33.o
[12/11/21]seed@VM:~$
```

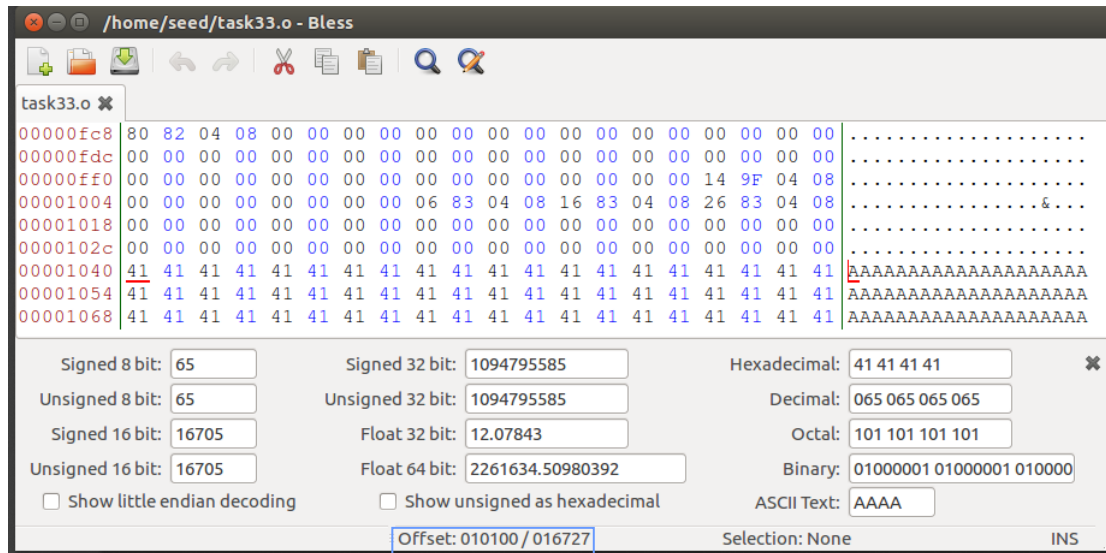
The screenshot shows the Bless debugger interface. The top panel displays the memory view of the task33.o file, showing the ELF header. The bottom panel shows the console view with the following data:

Signed 8 bit:	127	Signed 32 bit:	2135247942	Hexadecimal:	7F 45 4C 46
Unsigned 8 bit:	127	Unsigned 32 bit:	2135247942	Decimal:	127 069 076 070
Signed 16 bit:	32581	Float 32 bit:	2.622539E+38	Octal:	177 105 114 106
Unsigned 16 bit:	32581	Float 64 bit:	1.16843158668567E+305	Binary:	01111111 01000101 010011
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	ELF
Offset: 0x0 / 0x1dd7				Selection: None	INS

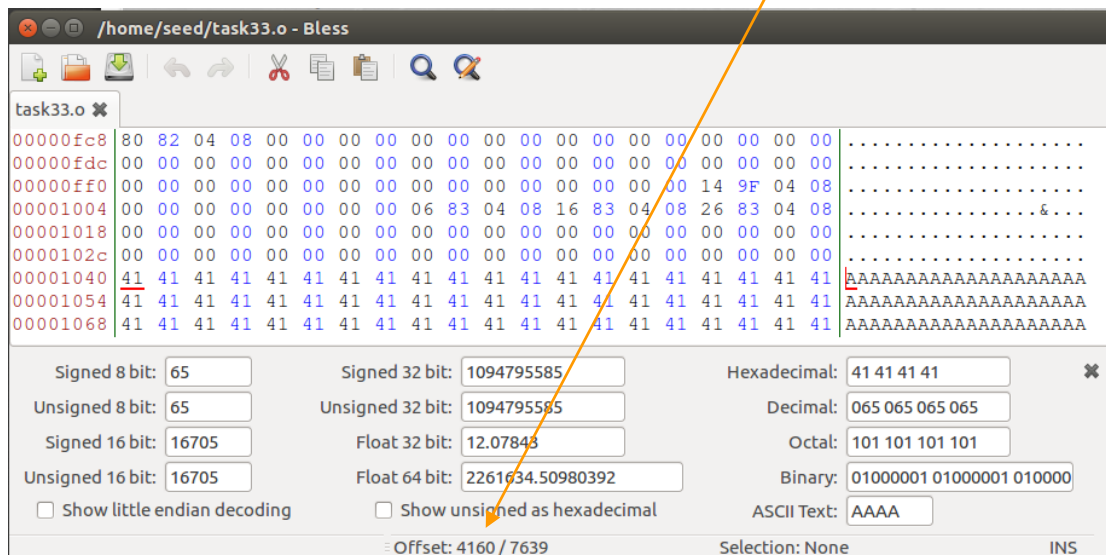
The screenshot shows the Bless debugger interface. The top panel displays the memory view of the task33.o file, showing the array data. The bottom panel shows the console view with the following data:

Signed 8 bit:	65	Signed 32 bit:	1094795585	Hexadecimal:	41 41 41 41
Unsigned 8 bit:	65	Unsigned 32 bit:	1094795585	Decimal:	065 065 065 065
Signed 16 bit:	16705	Float 32 bit:	12.07843	Octal:	101 101 101 101
Unsigned 16 bit:	16705	Float 64 bit:	2261634.50980392	Binary:	01000001 01000001 010000
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	AAAA
Offset: 0x1040 / 0x1dd7				Selection: None	INS

[Click on here to get binary value](#)



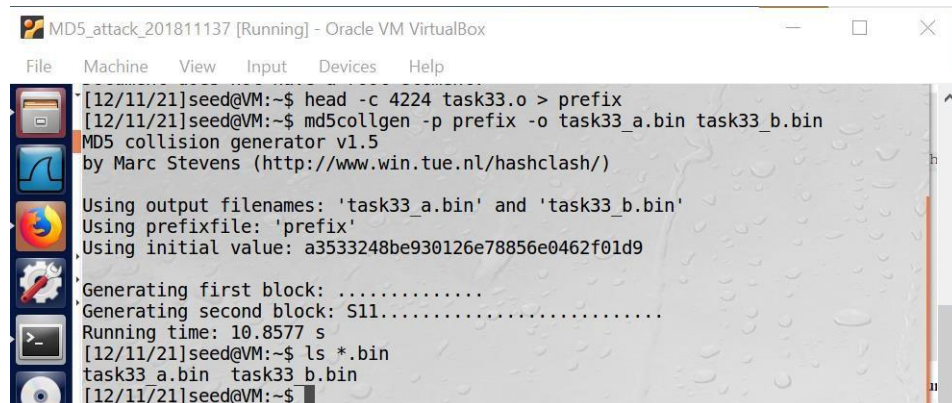
And if we click again, we will get Decimal



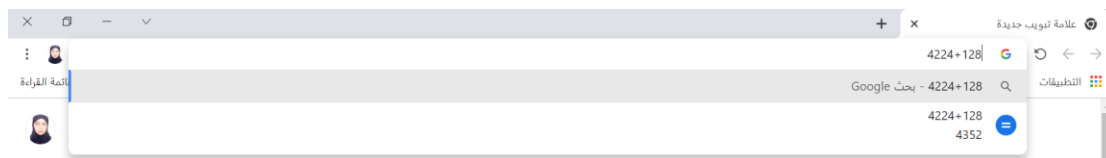
The length of the prefix needs to be multiple of 64 bytes.



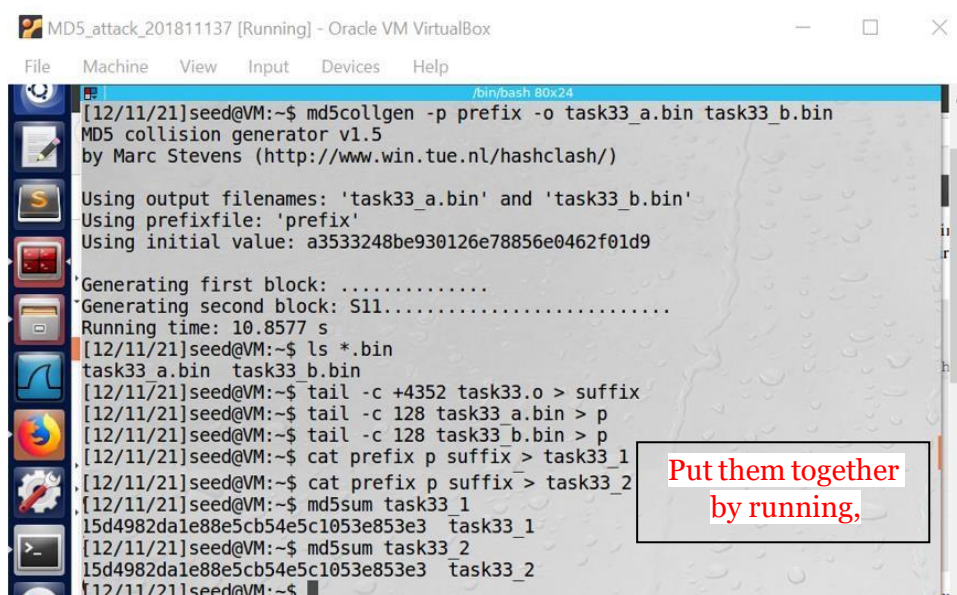
Running `head -c 4224 task33.o > prefix` and `md5collgen -p prefix -o task33_a.bin task33_b.bin`



```
MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
[12/11/21]seed@VM:~$ head -c 4224 task33.o > prefix
[12/11/21]seed@VM:~$ md5collgen -p prefix -o task33_a.bin task33_b.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
Using output filenames: 'task33_a.bin' and 'task33_b.bin'
Using prefixfile: 'prefix'
Using initial value: a3533248be930126e78856e0462f01d9
Generating first block: .....
Generating second block: S11.....
Running time: 10.8577 s
[12/11/21]seed@VM:~$ ls *.bin
task33_a.bin task33_b.bin
[12/11/21]seed@VM:~$
```



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




```
MD5_attack_201811137 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
[12/11/21]seed@VM:~$ md5collgen -p prefix -o task33_a.bin task33_b.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
Using output filenames: 'task33_a.bin' and 'task33_b.bin'
Using prefixfile: 'prefix'
Using initial value: a3533248be930126e78856e0462f01d9
Generating first block: .....
Generating second block: S11.....
Running time: 10.8577 s
[12/11/21]seed@VM:~$ ls *.bin
task33_a.bin task33_b.bin
[12/11/21]seed@VM:~$ tail -c +4352 task33.o > suffix
[12/11/21]seed@VM:~$ tail -c 128 task33_a.bin > p
[12/11/21]seed@VM:~$ tail -c 128 task33_b.bin > p
[12/11/21]seed@VM:~$ cat prefix p suffix > task33_1
[12/11/21]seed@VM:~$ cat prefix p suffix > task33_2
[12/11/21]seed@VM:~$ md5sum task33_1
15d4982dale88e5cb54e5c1053e853e3 task33_1
[12/11/21]seed@VM:~$ md5sum task33_2
15d4982dale88e5cb54e5c1053e853e3 task33_2
[12/11/21]seed@VM:~$
```

Put them together by running,

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

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`.



MD5_attack_201811137 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

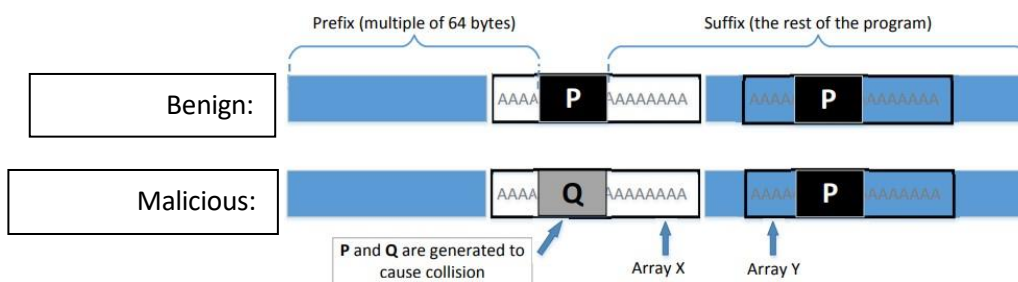
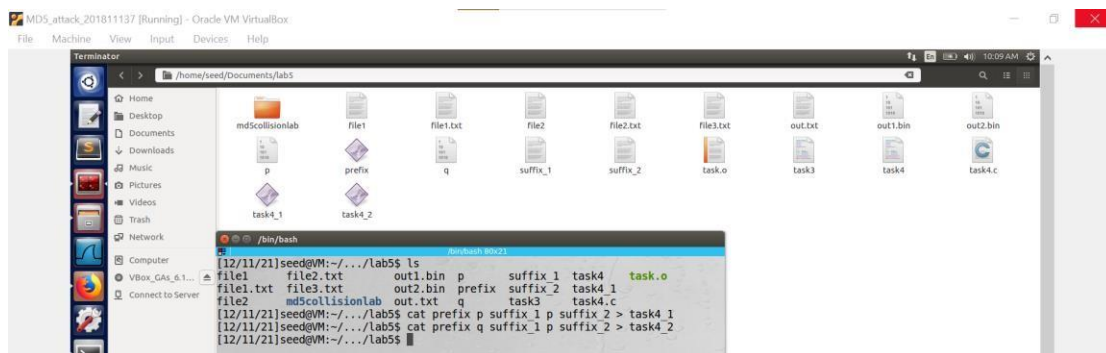
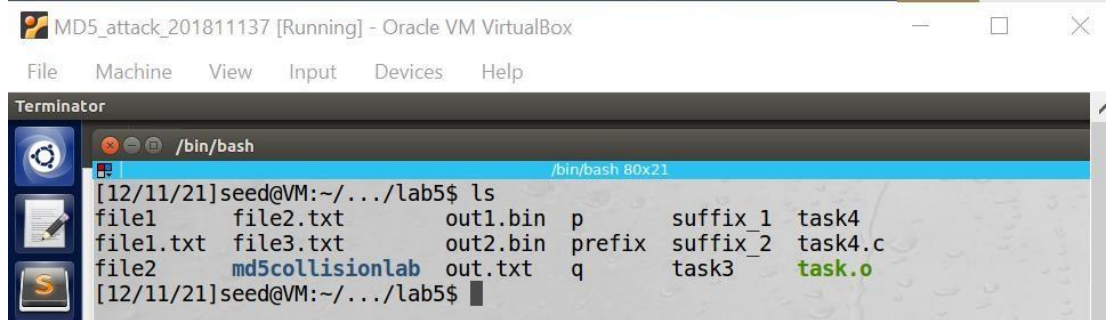
Terminator

/bin/bash

/bin/bash 80x21

```
[12/11/21]seed@VM:~/.../lab5$ vi task4.c
[12/11/21]seed@VM:~/.../lab5$ rm task.o
[12/11/21]seed@VM:~/.../lab5$ gcc task4.c -o task.o
[12/11/21]seed@VM:~/.../lab5$ ./task.o
benign code
[12/11/21]seed@VM:~/.../lab5$
```

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

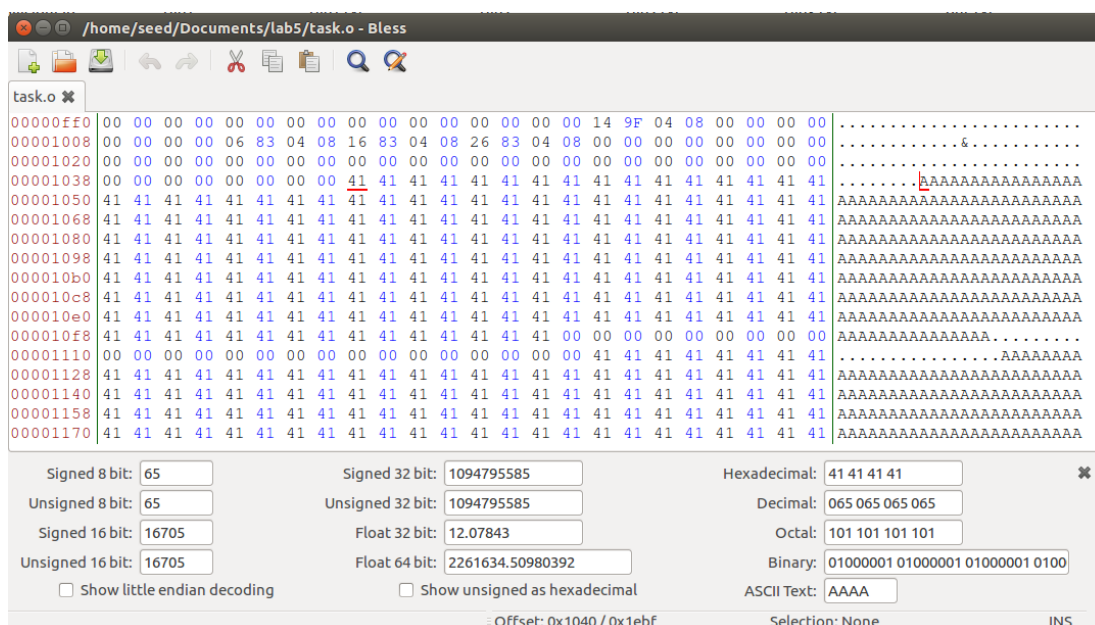
Terminator
/bin/bash

[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$ chmod +x task4_1
[12/11/21]seed@VM:~/.../lab5$ ./task4_1
WARNING: malicious code
[12/11/21]seed@VM:~/.../lab5$

```

This is not good

Time for some debugging



Attempting to align exec files..

Array x start @ 0x1040

(i.e. offset of 4160-bytes)

task.o task4_1

```

00000468 0F B6 00 38 C2 75 0F 83 45 F4 01 81 7D F4 C7 00 00 00 7E D9 EB 01 90 81 ...8.u..E..}.....~.....
00000480 7D F4 C8 00 00 00 75 17 83 EC 08 68 50 85 04 08 68 5C 85 04 08 E8 66 FE }.....u...hP...h\...f.
00000498 FF FF 83 C4 10 EB 15 83 EC 08 68 5F 85 04 08 68 5C 85 04 08 E8 4F FE FF .....h...h\...O..
000004b0 FF 83 C4 10 83 EC 0C 6A 0A E8 62 FE FF FF 83 C4 10 B8 00 00 00 00 8B 4D .....j..b.....M
000004c8 FC C9 8D 61 FC C3 66 90 55 57 56 53 E8 97 FE FF FF 81 C3 27 1B 00 00 83 ...a..f.UWVS.....'....
000004e0 EC 0C 8B 6C 24 20 8D B3 0C FF FF FF E8 DB FD FF FF 8D 83 08 FF FF FF 29 ...l$ .....(
000004f8 C6 C1 FE 02 85 F6 74 25 31 FF 8D B6 00 00 00 83 EC 04 FF 74 24 2C FF .....t%l.....t$,
00000510 74 24 2C 55 FF 94 BB 08 FF FF FF 83 C7 01 83 C4 10 39 F7 75 E3 83 C4 0C t$,U.....9.u....
00000528 5B 5E 5F 5D C3 8D 76 00 F3 C3 00 00 53 83 EC 08 E8 33 FE FF FF 81 C3 C3 [^_].v.....S....3....
00000540 1A 00 00 83 C4 08 5B C3 03 00 00 00 01 00 02 00 62 65 6E 69 67 6E 20 63 .....[.....benign c
00000558 6F 64 65 00 25 73 00 57 41 52 4E 49 4E 47 3A 20 6D 61 6C 69 63 69 6F 75 ode.%s.WARNING: maliciou
00000570 73 20 63 6F 64 65 00 00 01 1B 03 3B 28 00 00 00 04 00 00 00 78 FD FF FF s code.%s;(.....X...
00000588 44 00 00 00 C3 FE FF FF 68 00 00 58 FF FF FF 94 00 00 00 B8 FF FF FF D.....h...X.....
000005a0 E0 00 00 00 14 00 00 00 00 00 00 00 01 7A 52 00 01 7C 08 01 1B 0C 04 04 .....zR.....L....
000005b8 88 01 00 00 20 00 00 00 1C 00 00 2C FD FF FF 40 00 00 00 0E 08 46 .....F
000005d0 0E 0C 4A 0F 0B 74 04 78 00 3F 1A 3B 2A 32 24 22 28 00 00 00 40 00 00 ..J..t.x.?.*2$(...@...
000005e8 53 FE FF FF 93 00 00 00 00 44 0C 01 00 47 10 05 02 75 00 43 0F 03 75 7C S.....D...G...u.C..u|

```

Signed 8 bit: 27 Signed 32 bit: 453772292 Hexadecimal: 1B 0C 04 04

Unsigned 8 bit: 27 Unsigned 32 bit: 453772292 Decimal: 027 012 004 004

Signed 16 bit: 6924 Float 32 bit: 1.158183E-22 Octal: 033 014 004 004

Unsigned 16 bit: 6924 Float 64 bit: 2.16049849108923E-178 Binary: 00011011 00001100 00000100 0000

☐ Show little endian decoding ☐ Show unsigned as hexadecimal ASCII Text: [Hex] [Bin] [Dec] [Oct] [Bin]

Offset: 1460 / 4415 Selection: None INS

Array x ends 4359-bytes into file

task.o task4_1

```

00000ff0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 9F 04 08 00 00 00 00 .....&.....
00001008 00 00 00 00 06 83 04 08 16 83 04 08 26 83 04 08 00 00 00 00 00 00 .....&.....
00001020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....&.....
00001038 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....&.....
00001050 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
00001068 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
00001080 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
00001098 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
000010b0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
000010c8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
000010e0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
000010f8 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 00 00 00 00 .....AAAAAAAAAAAAAAAAAA
00001110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 41 41 41 41 .....AAAAAAAAAAAAAAAA
00001128 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
00001140 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
00001158 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA
00001170 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .....AAAAAAAAAAAAAAAAAAAA

```

Signed 8 bit: 0 Signed 32 bit: 0 Hexadecimal: 00 00 00 00

Unsigned 8 bit: 0 Unsigned 32 bit: 0 Decimal: 000 000 000 000

Signed 16 bit: 0 Float 32 bit: 0 Octal: 000 000 000 000

Unsigned 16 bit: 0 Float 64 bit: 0 Binary: 00000000 00000000 00000000 0000

☐ Show little endian decoding ☐ Show unsigned as hexadecimal ASCII Text: [Hex] [Bin] [Dec] [Oct] [Bin]

Offset: 4359 / 7871 Selection: None INS

task.o task4_1

```

00000fc0 01 00 00 00 F0 FF FF 6F 80 82 04 08 00 00 00 00 00 00 00 00 00 00 .....C.....
00000fd8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....&.....
00000ff0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 9F 04 08 00 00 00 00 .....&.....
00001008 00 00 00 00 06 83 04 08 16 83 04 08 26 83 04 08 00 00 00 00 00 00 .....&.....
00001020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....&.....
00001038 00 00 00 00 00 00 00 00 00 00 33 BA 38 12 B3 0D 77 BB 7A F7 30 72 6F BE DF BB .....3.8...w.z.0ro...
00001050 6B CF 42 A8 ED D3 32 E4 1D 92 7F A1 DA B5 AA 91 D7 0A FE ED 71 C2 79 C0 k.B...2.....q.y.
00001068 CF C7 46 01 90 6D 98 1D 57 51 3E BE B4 53 3E 2A 9D 0A A4 77 6A F1 25 0F ..F...m..WQ>...S>*..wj.%
00001080 E4 09 F3 47 6E B3 F6 22 28 06 F3 85 E3 11 B8 C3 B8 95 06 A6 CB 40 F9 ED ...Gn...*(.....@...
00001098 14 49 E2 87 4C 99 0E CF 05 3F FB BA 12 90 C6 C7 26 52 C1 99 83 24 D9 FC .I..L...?.....&R...$.
000010b0 B4 D4 F0 34 B8 D3 23 78 58 6C 66 60 93 D6 7C AA 33 BA 38 12 B3 0D 77 BB ...4..#x1f'...|.38...w.
000010c8 7A F7 30 72 6F BE DF BB 6B CF 42 A8 ED D3 32 E4 1D 92 7F A1 DA B5 AA 91 z.0ro...k.B...2.....
000010e0 D7 0A FE ED 71 C2 79 C0 CF C7 46 01 90 6D 98 1D 57 51 3E BE B4 53 3E 2A ...q.y...F...m..WQ>...S>*
000010f8 9D 0A A4 77 6A F1 25 0F E4 09 F3 47 6E B3 F6 22 28 06 F3 85 E3 11 B8 C3 ...wj.%...Gn...*(.....
00001110 B8 95 06 A6 CB 40 F9 ED 14 49 E2 87 4C 99 0E CF 05 3F FB BA 12 90 C6 C7 ...@...I..L...?.....
00001128 26 52 C1 99 83 24 D9 FC B4 D4 F0 34 B8 D3 23 78 58 6C 66 60 93 D6 7C AA &R...$.4..#x1f'...|.

```

Signed 8 bit: 34 Signed 32 bit: 573048563 Hexadecimal: 22 28 06 F3

Unsigned 8 bit: 34 Unsigned 32 bit: 573048563 Decimal: 034 040 006 243

Signed 16 bit: 8744 Float 32 bit: 2.277192E-18 Octal: 042 050 006 363

Unsigned 16 bit: 8744 Float 64 bit: 3.84834860635981E-144 Binary: 00100010 00101000 00000110 1111

☐ Show little endian decoding ☐ Show unsigned as hexadecimal ASCII Text: [Hex] [Bin] [Dec] [Oct] [Bin]

Offset: 4359 / 4415 Selection: None INS

```
benign code  
[12/11/21]seed@VM:~/.../lab5$ █
```

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](https://academic.microsoft.com/topic/87538441/publication/search?q=Collision%20attack&qe=And(Composite(F.FId%253D87538441)%252CTy%253D%270%27)&f=&orderBy=0)