Assignment 3: MD5 Collision Attack Lab

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

Generate two files (outbin1 and 2) with the same prefix:
$ md5collgen -p prefix.txt -o o1.bin o2.bin
Check if they are the same (if they differ in some way):
$ diff o1.bin o2.bin
Check the MD5 hash of the files
$ md5sum o1.bin
$ md5sum o2.bin

```
[02/20/22]seed@VM:~$ md5collgen -p prefix.txt -o o1.bin o2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'o1.bin' and 'o2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 6327970a0919c5b058b22f6b5e7b64a4

Generating first block: ...................
Generating second block: S10.........................
Running time: 20.8566 s
[02/20/22]seed@VM:~$ diff o1.bin o2.bin
Binary files o1.bin and o2.bin differ
[02/20/22]seed@VM:~$ md5sum o1.bin
e2270fd48d04c8da17de8436f24a2ccb  o1.bin
[02/20/22]seed@VM:~$ md5sum o2.bin
e2270fd48d04c8da17de8436f24a2ccb  o2.bin
[02/20/22]seed@VM:~$
```

View the output files with bless and describe your observations*:*

o1.bin | o2.bin

```
00000000 68 69 20 6B 61 72 6C 61 0A 00 00 00 00 00 00 00 00 00 |hi karla..........|
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..................|
00000024 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..................|
00000036 00 00 00 00 00 00 00 00 00 00 81 69 BA 84 13 07 94 C4 |...........i......|
00000048 61 8E B7 C3 77 CA D0 25 08 FE 22 DB 83 D1 16 A2 2F FA |a...w..%.."...../.|
0000005a 80 AE B8 39 9C E9 43 15 E8 E5 3F EA 0B C1 EB 2B C5 39 |...9..C...?....+.9|
0000006c 86 37 9E 37 DB 13 4E 6B C4 22 AD 0B D6 EC 0F 36 7E 99 |.7.7..Nk."......6~.|
0000007e 80 F8 30 12 26 0A 93 8F B4 FB F5 07 6C A0 83 1F 82 5C |..0.&.......l....\|
00000090 49 84 82 E3 38 5D 93 28 AB C5 00 58 AA 5F 88 D8 CF 93 |I...8].(...X._....|
000000a2 2C 04 A7 45 1C 36 74 43 02 3B 6C 28 F5 DE 3E E1 55 23 |,..E.6tC.;l(..>.U#|
```

Recall the diagram that we were given at the beginning of the lab (Figure1) – also attached here.

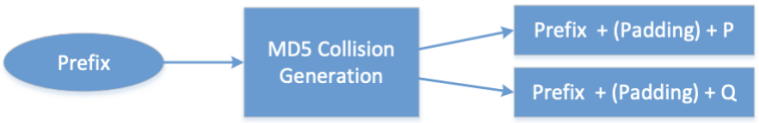


Figure 1: MD5 collision generation from a prefix

Compare the diagram to the outputs we have gotten. Note that we have the prefix which is followed by the padding and then additional data (P and Q) which follows. This is exactly what we see when we open the o1 and o2 files. We see that there is zero padding occuring after (or as part of) the prefix before the seemingly randomly appended information after. We note as foreshadowing to the first question, that this padding occurs when the prefix.txt does not meet a certain size requirement.

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

When we give m5collgen a prefix.txt file and its size is not a multiple of 64 we see that it is first padded so that its length will be a multiple of 64 bytes. When we open it with bless, we see specifically that the file is padded with zeros in all the x bytes such that $length = x \; mod \; 64$ and $x > 0$.
Note that out2.bin has the same prefix (as that is the command we used above), meaning that the first 64 bits (in this case - because the size of the prefix was <64) are identical to out1.bin

o1.bin | o2.bin

```
00000000 68 69 20 6B 61 72 6C 61 0A 00 00 00 00 00 00 00 00 |hi karla.........
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.................
00000024 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.................
00000036 00 00 00 00 00 00 00 00 00 00 00 81 69 BA 84 13 07 94 C4 |...........i.....
00000048 61 8E B7 C3 77 CA D0 25 08 FE 22 5B 83 D1 16 A2 2F FA |a...w..%.."[..../.
0000005a 80 AE B8 39 9C E9 43 15 E8 E5 3F EA 0B C1 EB 2B C5 39 |...9..C...?....+.9
0000006c 86 B7 9D 37 DB 13 4E 6B C4 22 AD 0B D6 EC 0F B6 7E 99 |...7..Nk."......~.
0000007e 80 F8 30 12 26 0A 93 8F B4 FB F5 07 6C A0 83 1F 82 5C |..0.&.......l....\
00000090 49 84 82 63 38 5D 93 28 AB C5 00 58 AA 5F 88 D8 CF 93 |I..c8].(...X._....
000000a2 2C 04 A7 45 1C 36 74 43 02 3B 6C A8 F5 DE 3E E1 55 23 |,..E.6tC.;l...>.U#
```

o1.bin | o2.bin

```
00000000 68 69 20 6B 61 72 6C 61 0A 00 00 00 00 00 00 00 00 |hi karla.........
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.................
00000024 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.................
00000036 00 00 00 00 00 00 00 00 00 00 00 81 69 BA 84 13 07 94 C4 |...........i.....
00000048 61 8E B7 C3 77 CA D0 25 08 FE 22 DB 83 D1 16 A2 2F FA |a...w..%."...../.
0000005a 80 AE B8 39 9C E9 43 15 E8 E5 3F EA 0B C1 EB 2B C5 39 |...9..C...?....+.9
0000006c 86 37 9E 37 DB 13 4E 6B C4 22 AD 0B D6 EC 0F 36 7E 99 |.7.7..Nk."......6~.
0000007e 80 F8 30 12 26 0A 93 8F B4 FB F5 07 6C A0 83 1F 82 5C |..0.&.......l....\
00000090 49 84 82 E3 38 5D 93 28 AB C5 00 58 AA 5F 88 D8 CF 93 |I...8].(...X._....
000000a2 2C 04 A7 45 1C 36 74 43 02 3B 6C 28 F5 DE 3E E1 55 23 |,..E.6tC.;l(..>.U#
```

I have highlighted the 64th byte in red within the above screenshots.

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

I have appended the bless hex view of the new prefix.txt we used to run the collision tool again. We notice that no zero padding occurs, because the prefix.txt file met the requirement $length = 0 \; mod \; 64 \; bytes$.

prefix.txt

```
00000000 68 69 20 6B 61 72 6C 61 20 74 68 69 73 20 63 6C 61 73 |hi karla this clas
00000012 73 20 69 73 20 61 20 6C 69 74 74 6C 65 20 68 61 72 64 |s is a little hard
00000024 62 75 74 20 69 74 73 20 73 74 69 6C 6C 20 66 75 6E 20 |but its still fun
00000036 69 20 67 75 65 73 73 21 2E 0A                         |i guess!..
```

o1.bin o2.bin

```
00000000 68 69 20 6B 61 72 6C 61 20 74 68 69 73 20 63 6C 61 73 |hi karla this clas|
00000012 73 20 69 73 20 61 20 6C 69 74 74 6C 65 20 68 61 72 64 |s is a little hard|
00000024 62 75 74 20 69 74 73 20 73 74 69 6C 6C 20 66 75 6E 20 |but its still fun |
00000036 69 20 67 75 65 73 73 21 2E 0A A1 81 53 86 55 1A 45 54 |i guess!....S.U.ET|
00000048 16 45 50 68 7A 54 29 E7 9D F6 21 FA B1 93 DC 7B 4F D2 |.EPhzT)...!....{O.|
0000005a 37 32 F4 20 41 58 AD B3 D0 08 3A 6A E2 30 4B 1A C0 01 |72. AX....:j.0K...|
0000006c 0E 3A E5 30 8E 18 BC 61 28 0E B6 EE 36 CA 0F 94 9F 8C |.:.0...a(...6.....|
0000007e FF F5 41 B9 D5 15 16 C0 7C 4E 29 E8 90 DC A2 95 87 D3 |..A.....|N).......|
00000090 39 B4 DC 83 50 18 A2 B7 AB B3 00 D7 3B 60 68 A6 D7 D3 |9...P.......;`h...|
000000a2 60 14 A7 C3 1B F6 76 08 C6 92 4C 1C F6 58 0F 15 3B 47 |`.....v...L..X..;G|
000000b4 91 F4 F9 EF 6C 81 90 B6 76 1D 07 A9                   |....l...v...      |
```

o1.bin o2.bin

```
00000000 68 69 20 6B 61 72 6C 61 20 74 68 69 73 20 63 6C 61 73 |hi karla this clas|
00000012 73 20 69 73 20 61 20 6C 69 74 74 6C 65 20 68 61 72 64 |s is a little hard|
00000024 62 75 74 20 69 74 73 20 73 74 69 6C 6C 20 66 75 6E 20 |but its still fun |
00000036 69 20 67 75 65 73 73 21 2E 0A A1 81 53 86 55 1A 45 54 |i guess!....S.U.ET|
00000048 16 45 50 68 7A 54 29 E7 9D F6 21 7A B1 93 DC 7B 4F D2 |.EPhzT)...!z...{O.|
0000005a 37 32 F4 20 41 58 AD B3 D0 08 3A 6A E2 30 4B 1A C0 01 |72. AX....:j.0K...|
0000006c 0E BA E5 30 8E 18 BC 61 28 0E B6 EE 36 CA 0F 14 9F 8C |...0...a(...6.....|
0000007e FF F5 41 B9 D5 15 16 C0 7C 4E 29 E8 90 DC A2 95 87 D3 |..A.....|N).......|
00000090 39 B4 DC 03 50 18 A2 B7 AB B3 00 D7 3B 60 68 A6 D7 D3 |9...P.......;`h...|
000000a2 60 14 A7 C3 1B F6 76 08 C6 92 4C 9C F5 58 0F 15 3B 47 |`.....v...L..X..;G|
000000b4 91 F4 F9 EF 6C 81 90 36 76 1D 07 A9                   |....l..6v...      |
```

To demonstrate fully, if we were to run the experiment again with a prefix file if size 99 bytes, we notice that md5collgen will pad the prefix with zeros until it is of size 128. I have highlighted the 128th byte in red in the screenshots below to show that the zero padding fills in the prefix and extends its length until it is a multiple of 64.

prefix.txt

```
00000000 68 69 20 6B 61 72 6C 61 20 74 68 69 73 20 63 6C 61 73 |hi karla this clas|
00000012 73 20 69 73 20 61 20 6C 69 74 74 6C 65 20 68 61 72 64 |s is a little hard|
00000024 62 75 74 20 69 74 73 20 73 74 69 6C 6C 20 66 75 6E 20 |but its still fun |
00000036 69 20 67 75 65 73 73 21 2E 61 66 20 6A 61 6C 64 66 61 |i guess!.af jaldfa|
00000048 61 64 66 61 65 73 64 64 0A 64 61 6B 6C 64 66 61 61 66 |adfaesdd.dakldfaaf|
0000005a 61 6B 64 66 65 65 65 64 0A                            |akdfeeed.         |
```

o1.bin o2.bin

```
00000000 68 69 20 6B 61 72 6C 61 20 74 68 69 73 20 63 6C 61 73 |hi karla this clas|
00000012 73 20 69 73 20 61 20 6C 69 74 74 6C 65 20 68 61 72 64 |s is a little hard|
00000024 62 75 74 20 69 74 73 20 73 74 69 6C 6C 20 66 75 6E 20 |but its still fun |
00000036 69 20 67 75 65 73 73 21 2E 61 66 20 6A 61 6C 64 66 61 |i guess!.af jaldfa|
00000048 61 64 66 61 65 73 64 64 0A 64 61 6B 6C 64 66 61 61 66 |adfaesdd.dakldfaaf|
0000005a 61 6B 64 66 65 65 65 64 0A 00 00 00 00 00 00 00 00 00 |akdfeeed..........|
0000006c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..................|
0000007e 00 00 20 02 E4 30 98 4F C4 37 BB B8 68 98 79 60 6A 6E |.. ..0.O.7..h.y`jn|
00000090 E7 D9 39 E5 4A EC 1C B6 0B 8E 7F 49 B2 B3 B2 0A 19 E3 |..9.J......I......|
000000a2 F7 C8 7E E4 69 25 CB 1D 63 32 1D 6C 4C 64 AC 5A F6 E5 |..~.i%..c2.lLd.Z..|
```

o1.bin | o2.bin

```
00000000 68 69 20 6B 61 72 6C 61 20 74 68 69 73 20 63 6C 61 73 |hi karla this clas|
00000012 73 20 69 73 20 61 20 6C 69 74 74 6C 65 20 68 61 72 64 |s is a little hard|
00000024 62 75 74 20 69 74 73 20 73 74 69 6C 6C 20 66 75 6E 20 |but its still fun |
00000036 69 20 67 75 65 73 73 21 2E 61 66 20 6A 61 6C 64 66 61 |i guess!.af jaldfa|
00000048 61 64 66 61 65 73 64 64 0A 64 61 6B 6C 64 66 61 61 66 |adfaesdd.dakldfaaf|
0000005a 61 6B 64 66 65 65 65 64 0A 00 00 00 00 00 00 00 00 00 |akdfeeed..........|
0000006c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..................|
0000007e 00 00 20 02 E4 30 98 4F C4 37 BB B8 68 98 79 60 6A 6E |.. ..0.O.7..h.y`jn|
00000090 E7 D9 39 65 4A EC 1C B6 0B 8E 7F 49 B2 B3 B2 0A 19 E3 |..9eJ......I......|
000000a2 F7 C8 7E E4 69 25 CB 1D 63 32 1D EC 4C 64 AC 5A F6 E5 |..~.i%..c2..Ld.Z..|
```

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.

The output files created are not identical, as there are differences that we can spot within the editor. Though the difference wasn't huge, there were certainly areas that has different bytes. This is confirmed with the Figure 1 in the lab. The data we are referring to is P and Q and thus we know they are unequal.

o1.bin | o2.bin

```
0000005a 61 6B 64 66 65 65 65 64 0A 00 00 00 00 00 00 00 00 00 |akdfeeed..........|
0000006c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..................|
0000007e 00 00 20 02 E4 30 98 4F C4 37 BB B8 68 98 79 60 6A 6E |.. ..0.O.7..h.y`jn|
00000090 E7 D9 39 E5 4A EC 1C B6 0B 8E 7F 49 B2 B3 B2 0A 19 E3 |..9.J......I......|
000000a2 F7 C8 7E E4 69 25 CB 1D 63 32 1D 6C 4C 64 AC 5A F6 E5 |..~.i%..c2.lLd.Z..|
000000b4 62 97 AE C6 07 7B 87 9F 4C 20 AC E9 E8 A0 D0 5C 33 5E |b....{..L .....\3^|
000000c6 ED CA 32 8B 3C 59 3E 27 D5 42 ED EF 88 47 19 41 3B 4C |..2.<Y>'.B...G.A;L|
000000d8 04 CD 18 F7 D0 0A A2 BE 81 DF 72 60 D9 72 9B 47 36 48 |..........r`.r.G6H|
000000ea 05 AA 93 E0 D2 9C D5 70 23 41 76 D6 13 F5 75 1B 21 66 |.......p#Av...u.!f|
000000cf E6 5F A1 02                                           |._..|
```

o1.bin | o2.bin

```
0000005a 61 6B 64 66 65 65 65 64 0A 00 00 00 00 00 00 00 00 00 |akdfeeed..........|
0000006c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |..................|
0000007e 00 00 20 02 E4 30 98 4F C4 37 BB B8 68 98 79 60 6A 6E |.. ..0.O.7..h.y`jn|
00000090 E7 D9 39 65 4A EC 1C B6 0B 8E 7F 49 B2 B3 B2 0A 19 E3 |..9eJ......I......|
000000a2 F7 C8 7E E4 69 25 CB 1D 63 32 1D EC 4C 64 AC 5A F6 E5 |..~.i%..c2..Ld.Z..|
000000b4 62 97 AE C6 07 7B 87 1F 4C 20 AC E9 E8 A0 D0 5C 33 5E |b....{..L .....\3^|
000000c6 ED CA 32 8B 3C 59 3E 27 D5 42 ED EF 88 C7 19 41 3B 4C |..2.<Y>'.B.....A;L|
000000d8 04 CD 18 F7 D0 0A A2 BE 81 DF 72 60 D9 72 9B 47 36 48 |..........r`.r.G6H|
000000ea 05 AA 93 60 D2 9C D5 70 23 41 76 D6 13 F5 75 1B 21 E6 |...`...p#Av...u.!.|
000000cf E6 5F A1 02                                           |._..|
```

I have included small red circles to show where P and Q differ.

Task 2: Understanding MD5's Property

Show that if $MD5(M) = MD5(N)$ (ie. The hashes of two distinct messages are equal), then $MD5(M||T) = MD5(N||T)$, for any input T

Design an experiment to demonstrates this property:

To do this we need to do a few things:

Step 1: Obtain N and M

We need to obtain two inputs/messages which produce the same hash. For this experiment, let us use the following string as the content in our prefix.txt: This class is great and the TA is awesome.

Generate N and M by running the following command:

$md5collgen -p prefix.txt -o file1 file2

```
[02/20/22]seed@VM:~$ md5collgen -p prefix.txt -o file1.bin file2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'file1.bin' and 'file2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 6b0b56adf8418836560c0ec84a6914a1

Generating first block: ...............
Generating second block: S10...............
Running time: 11.6467 s
```

We can use the `md5sum` command to check the MD5 hash of each output file. Verify that the hashes of file1 (N) and file2 (M) are the same through the following commands:

$mdsum file1

$mdsum file2

```
[02/20/22]seed@VM:~$ md5sum file1.bin
e115ee5f456a6f6a2bb5075f0f918228  file1.bin
[02/20/22]seed@VM:~$ md5sum file2.bin
e115ee5f456a6f6a2bb5075f0f918228  file2.bin
```

I have attached the output to show that they are equal.

Step 2: Concatenate: Obtain (N||T) and (M||T)

Once we have the different files which have the same hash. We will need to append the same suffix (T) to these and observe the modified hash of both of the files.

Let us create the suffix.txt: hello world.

$ cat file1 suffix > newfile1

$ cat file2 suffix > newfile2

```
[02/20/22]seed@VM:~$ md5collgen -p prefix.txt -o file1 file2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'file1' and 'file2'
Using prefixfile: 'prefix.txt'
Using initial value: 6b0b56adf8418836560c0ec84a6914a1

Generating first block: .....
Generating second block: S00..
Running time: 9.8849 s
[02/20/22]seed@VM:~$ echo "hello world" > suffix
[02/20/22]seed@VM:~$ cat suffix
hello world
[02/20/22]seed@VM:~$ cat file1 suffix > newfile1
[02/20/22]seed@VM:~$ cat file2 suffix > newfile2
```

Step 3: Verify the property: $MD5(M||T) = MD5(N||T)$

If the property is true, then the hash values for newfile1 and newfile2 should be identical.

$mdsum newfile1
$mdsum newfile2

```
[02/20/22]seed@VM:~$ md5sum newfile1
0d0f5f5f9e98701f8377e9153d7bee69  newfile1
[02/20/22]seed@VM:~$ md5sum newfile2
0d0f5f5f9e98701f8377e9153d7bee69  newfile2
```

We observe that the MD5 hashes remain identical.

Step 4: Conclusion
Because we were able to verify the property, we can conclude that this property holds true for MD5.

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

I included the original code below for reference:

```c
//original code given
unsigned char xyz[200] = { /* The actual contents of this array are up to you */ };
int main() {
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

The goal of this task is to create two distinct exe files such that their MD5 hash values are equal. Formally we are seeking to show $MD5(file1) = MD5(file2)$. In order to achieve this, we will follow the steps delineated in the pdf handout.

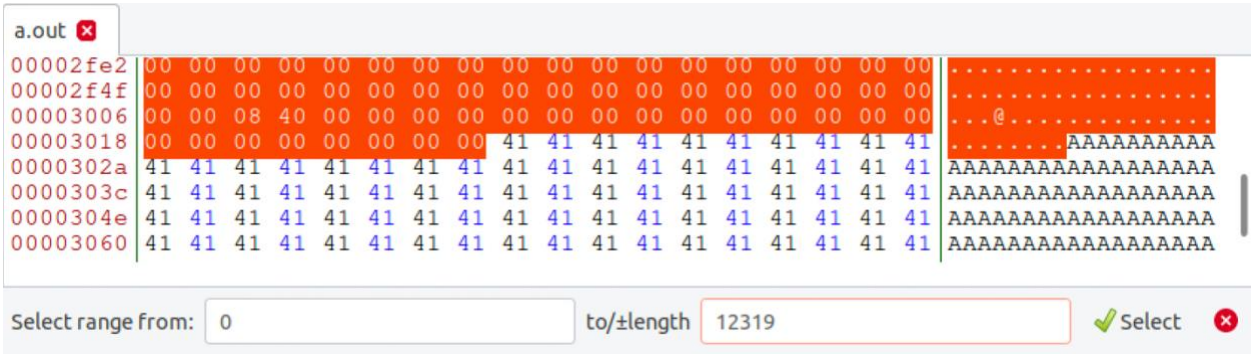
Step 1: Put some arbitrary values in the xyz array
Let us fill the array with something that is easily identifiable (fixed value) within the output of the bless program. The lab gives us the example of filling it with ASCII symbol for "A" - which in turn corresponds to the value 0x41.
Let us modify the code to reflect these actions:

```
unsigned char xyz[200] = {
        [0]: 0x41, [1]: 0x41, [2]: 0x41, [3]: 0x41, [4]: 0x41, [5]: 0x41, [6]: 0x41, [7]: 0x41, [8]: 0x41, [9]: 0x41,
        [10]: 0x41, [11]: 0x41, [12]: 0x41, [13]: 0x41, [14]: 0x41, [15]: 0x41, [16]: 0x41, [17]: 0x41, [18]: 0x41, [19]: 0x41,
        [20]: 0x41, [21]: 0x41, [22]: 0x41, [23]: 0x41, [24]: 0x41, [25]: 0x41, [26]: 0x41, [27]: 0x41, [28]: 0x41, [29]: 0x41,
        [30]: 0x41, [31]: 0x41, [32]: 0x41, [33]: 0x41, [34]: 0x41, [35]: 0x41, [36]: 0x41, [37]: 0x41, [38]: 0x41, [39]: 0x41,
        [40]: 0x41, [41]: 0x41, [42]: 0x41, [43]: 0x41, [44]: 0x41, [45]: 0x41, [46]: 0x41, [47]: 0x41, [48]: 0x41, [49]: 0x41,
        [50]: 0x41, [51]: 0x41, [52]: 0x41, [53]: 0x41, [54]: 0x41, [55]: 0x41, [56]: 0x41, [57]: 0x41, [58]: 0x41, [59]: 0x41,
        [60]: 0x41, [61]: 0x41, [62]: 0x41, [63]: 0x41, [64]: 0x41, [65]: 0x41, [66]: 0x41, [67]: 0x41, [68]: 0x41, [69]: 0x41,
        [70]: 0x41, [71]: 0x41, [72]: 0x41, [73]: 0x41, [74]: 0x41, [75]: 0x41, [76]: 0x41, [77]: 0x41, [78]: 0x41, [79]: 0x41,
        [80]: 0x41, [81]: 0x41, [82]: 0x41, [83]: 0x41, [84]: 0x41, [85]: 0x41, [86]: 0x41, [87]: 0x41, [88]: 0x41, [89]: 0x41,
        [90]: 0x41, [91]: 0x41, [92]: 0x41, [93]: 0x41, [94]: 0x41, [95]: 0x41, [96]: 0x41, [97]: 0x41, [98]: 0x41, [99]: 0x41,
        [100]: 0x41, [101]: 0x41, [102]: 0x41, [103]: 0x41, [104]: 0x41, [105]: 0x41, [106]: 0x41, [107]: 0x41, [108]: 0x41, [109]: 0x41,
        [110]: 0x41, [111]: 0x41, [112]: 0x41, [113]: 0x41, [114]: 0x41, [115]: 0x41, [116]: 0x41, [117]: 0x41, [118]: 0x41, [119]: 0x41,
        [120]: 0x41, [121]: 0x41, [122]: 0x41, [123]: 0x41, [124]: 0x41, [125]: 0x41, [126]: 0x41, [127]: 0x41, [128]: 0x41, [129]: 0x41,
        [130]: 0x41, [131]: 0x41, [132]: 0x41, [133]: 0x41, [134]: 0x41, [135]: 0x41, [136]: 0x41, [137]: 0x41, [138]: 0x41, [139]: 0x41,
        [140]: 0x41, [141]: 0x41, [142]: 0x41, [143]: 0x41, [144]: 0x41, [145]: 0x41, [146]: 0x41, [147]: 0x41, [148]: 0x41, [149]: 0x41,
        [150]: 0x41, [151]: 0x41, [152]: 0x41, [153]: 0x41, [154]: 0x41, [155]: 0x41, [156]: 0x41, [157]: 0x41, [158]: 0x41, [159]: 0x41,
        [160]: 0x41, [161]: 0x41, [162]: 0x41, [163]: 0x41, [164]: 0x41, [165]: 0x41, [166]: 0x41, [167]: 0x41, [168]: 0x41, [169]: 0x41,
        [170]: 0x41, [171]: 0x41, [172]: 0x41, [173]: 0x41, [174]: 0x41, [175]: 0x41, [176]: 0x41, [177]: 0x41, [178]: 0x41, [179]: 0x41,
        [180]: 0x41, [181]: 0x41, [182]: 0x41, [183]: 0x41, [184]: 0x41, [185]: 0x41, [186]: 0x41, [187]: 0x41, [188]: 0x41, [189]: 0x41,
        [190]: 0x41, [191]: 0x41, [192]: 0x41, [193]: 0x41, [194]: 0x41, [195]: 0x41, [196]: 0x41, [197]: 0x41, [198]: 0x41, [199]: 0x41}; //20 rows of 10


int task3() {
    int i;
    for (i=0; i<200; i++){
        printf("%x \n", xyz[i]);
    }
    printf("\n");
}
```

Step 2: compile the binary

From inside the array, we can find two locations, from where we can divide the executable file into three parts: a prefix, a 128-byte region, and a suffix. The length of the prefix needs to be multiple of 64 bytes. We open the hex editor 'bless' to locate the physical location of the xyz array. We notice that it is at 12319



Step 3: Then you can use a hex editor tool to modify the content of the xyz array directly in the binary file

First, we will run md5collgen on the prefix to generate two outputs that have the same MD5 hash value such that we obtain the following $MD5(prefix \,||\, P) = MD5(prefix \,||\, Q)$ where P and Q represent the next 128 bytes after the prefix. Note that the suffix is not shown in the above formula. Based on the property of MD5, we know that if we append the same suffix to the above two outputs, the resultant data will also have the same hash value. Basically, the following is true for any suffix: $MD5(prefix \,||P \,||\, suffix) \; = MD5(prefix \,||Q \,||\, suffix\,)$

As noted previously, the byte offset belonging to the prefix is 12352, we confirm this due to the fact that it is a multiple of 64 and it does not contain any of the array (which we know is contained in P). Using the command given to us in the assignment, we obtain:

$head -c 12352 a.out > prefix

Using the prefix we obtain, we now run a similar process to that in Question 2. Where we create two distinct files that produce the same hash $MD5(file1) = MD5(file2)$

$md5collgen -p prefix -o file1 file2

Now, using the property which we proved in Task 2: if $MD5(M) = MD5(N)$ (ie. The hashes of two distinct messages are equal), then $MD5(M||T) = MD5(N||T)$, for any input T.
We will extract the suffix from the original files and then append it to the new files (file1 and file2)

$ tail -c +12480 a.out > suffix NOTE THAT ITS 12352 + 128 TO FOLLOW FIGURE 4 STRUCTURE

$ cat file1 suffix > file1ps

$ cat file2 suffix > file2ps

$ chmod +x file1ps

$ chmod +x file2ps

$diff file1ps file2ps

Let us make these files executables once again and note that they are different. However, the md5sum of both the codes are the same.

$mdsum file1ps

$mdsum file2ps

```
[02/20/22]seed@VM:~$ gcc task3.c
[02/20/22]seed@VM:~$ head -c 12352 a.out > prefix
[02/20/22]seed@VM:~$ md5collgen -p prefix -o file1 file2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'file1' and 'file2'
Using prefixfile: 'prefix'
Using initial value: aefe77439646539fe6db56554d46ae83

Generating first block: ..
Generating second block: S11...
Running time: 1.18549 s
[02/20/22]seed@VM:~$ md5sum file1 file2
4613c2386a2cc2efa674cb40bbeb8fa4  file1
4613c2386a2cc2efa674cb40bbeb8fa4  file2
[02/20/22]seed@VM:~$ bless a.out
Gtk-Message: 23:29:22.217: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/seed/.config/bless/plugins'.
Could not find a part of the path '/home/seed/.config/bless/plugins'.
Could not find a part of the path '/home/seed/.config/bless/plugins'.
Could not find file "/home/seed/.config/bless/export_patterns"
[02/20/22]seed@VM:~$ tail -c +12480 a.out > suffix
[02/20/22]seed@VM:~$ cat file1 suffix > file1ps
[02/20/22]seed@VM:~$ cat file2 suffix > file2ps
[02/20/22]seed@VM:~$ chmod +x file1ps
[02/20/22]seed@VM:~$ chmod +x file2ps
[02/20/22]seed@VM:~$ md5sum file1 file2
4613c2386a2cc2efa674cb40bbeb8fa4  file1
4613c2386a2cc2efa674cb40bbeb8fa4  file2
```

sorry I realize I mistyped md5sum file1ps file2ps --- but the result is the same. We just wanted to demonstrate that the hash is the same

We have thus shown that we can create two different binaries from a single binary, both producing different output but having the same md5 hash value.

2.4 Task 4: Making the Two Programs Behave Differently

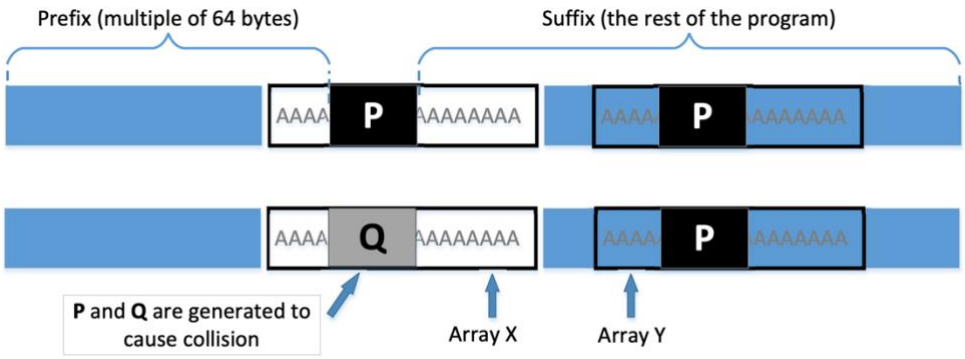


Figure 4: An approach to generate two hash-colliding programs with different behaviors.

Let us 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. In your work, what benign/malicious instructions are executed is not important; it is sufficient to demonstrate that the instructions executed by these two programs are different

Step 1 : The two hash-colliding programs produced by md5collgen need to share the same prefix

Step 2: we need to add some meaningful suffix to the outputs produced by md5collgen, the suffix added to both programs also needs to be the same

Program:
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

Version 1: Benign Code Execution

the code attached had to be in 2 screenshots

```c
#include <stdio.h>
unsigned char X[100] = {
        [0]: 0x41, [1]: 0x41, [2]: 0x41, [3]: 0x41, [4]: 0x41, [5]: 0x41, [6]: 0x41, [7]: 0x41, [8]: 0x41, [9]: 0x41,
        [10]: 0x41, [11]: 0x41, [12]: 0x41, [13]: 0x41, [14]: 0x41, [15]: 0x41, [16]: 0x41, [17]: 0x41, [18]: 0x41, [19]: 0x41,
        [20]: 0x41, [21]: 0x41, [22]: 0x41, [23]: 0x41, [24]: 0x41, [25]: 0x41, [26]: 0x41, [27]: 0x41, [28]: 0x41, [29]: 0x41,
        [30]: 0x41, [31]: 0x41, [32]: 0x41, [33]: 0x41, [34]: 0x41, [35]: 0x41, [36]: 0x41, [37]: 0x41, [38]: 0x41, [39]: 0x41,
        [40]: 0x41, [41]: 0x41, [42]: 0x41, [43]: 0x41, [44]: 0x41, [45]: 0x41, [46]: 0x41, [47]: 0x41, [48]: 0x41, [49]: 0x41,
        [50]: 0x41, [51]: 0x41, [52]: 0x41, [53]: 0x41, [54]: 0x41, [55]: 0x41, [56]: 0x41, [57]: 0x41, [58]: 0x41, [59]: 0x41,
        [60]: 0x41, [61]: 0x41, [62]: 0x41, [63]: 0x41, [64]: 0x41, [65]: 0x41, [66]: 0x41, [67]: 0x41, [68]: 0x41, [69]: 0x41,
        [70]: 0x41, [71]: 0x41, [72]: 0x41, [73]: 0x41, [74]: 0x41, [75]: 0x41, [76]: 0x41, [77]: 0x41, [78]: 0x41, [79]: 0x41,
        [80]: 0x41, [81]: 0x41, [82]: 0x41, [83]: 0x41, [84]: 0x41, [85]: 0x41, [86]: 0x41, [87]: 0x41, [88]: 0x41, [89]: 0x41,
        [90]: 0x41, [91]: 0x41, [92]: 0x41, [93]: 0x41, [94]: 0x41, [95]: 0x41, [96]: 0x41, [97]: 0x41, [98]: 0x41, [99]: 0x41}; //10

unsigned char Y[100] = {
        [0]: 0x41, [1]: 0x41, [2]: 0x41, [3]: 0x41, [4]: 0x41, [5]: 0x41, [6]: 0x41, [7]: 0x41, [8]: 0x41, [9]: 0x41,
        [10]: 0x41, [11]: 0x41, [12]: 0x41, [13]: 0x41, [14]: 0x41, [15]: 0x41, [16]: 0x41, [17]: 0x41, [18]: 0x41, [19]: 0x41,
        [20]: 0x41, [21]: 0x41, [22]: 0x41, [23]: 0x41, [24]: 0x41, [25]: 0x41, [26]: 0x41, [27]: 0x41, [28]: 0x41, [29]: 0x41,
        [30]: 0x41, [31]: 0x41, [32]: 0x41, [33]: 0x41, [34]: 0x41, [35]: 0x41, [36]: 0x41, [37]: 0x41, [38]: 0x41, [39]: 0x41,
        [40]: 0x41, [41]: 0x41, [42]: 0x41, [43]: 0x41, [44]: 0x41, [45]: 0x41, [46]: 0x41, [47]: 0x41, [48]: 0x41, [49]: 0x41,
        [50]: 0x41, [51]: 0x41, [52]: 0x41, [53]: 0x41, [54]: 0x41, [55]: 0x41, [56]: 0x41, [57]: 0x41, [58]: 0x41, [59]: 0x41,
        [60]: 0x41, [61]: 0x41, [62]: 0x41, [63]: 0x41, [64]: 0x41, [65]: 0x41, [66]: 0x41, [67]: 0x41, [68]: 0x41, [69]: 0x41,
        [70]: 0x41, [71]: 0x41, [72]: 0x41, [73]: 0x41, [74]: 0x41, [75]: 0x41, [76]: 0x41, [77]: 0x41, [78]: 0x41, [79]: 0x41,
        [80]: 0x41, [81]: 0x41, [82]: 0x41, [83]: 0x41, [84]: 0x41, [85]: 0x41, [86]: 0x41, [87]: 0x41, [88]: 0x41, [89]: 0x41,
        [90]: 0x41, [91]: 0x41, [92]: 0x41, [93]: 0x41, [94]: 0x41, [95]: 0x41, [96]: 0x41, [97]: 0x41, [98]: 0x41, [99]: 0x41}; //10

int main() {
    int i;
    int benign = 1;
    int count = 0;
    for (i = 0; i<100; i++){
        if(X[i] != Y[i]){
            benign = 0;
            printf("index %x: %x =/= %x \n", i, X[i], Y[i]);
            count++;
            break;
        }
        else{
            printf("index %x: %x = %x \n", i, X[i], Y[i]);
        }
    }
    if(benign){
        printf("good code \n");
    }
    else{
        printf("bad code \n");
    }
}
```

The steps we will take to produce the two executables illustrated in Figure 4 are very similar to the steps we took in Task 3. We will again take the prefix from this particular executable that we just created (denoted a.out as it is the executable of our version1.c file). To comply with figure 4 we must include some of the X array in the prefix and have the Y array be contained within the suffix.

```
$ head -c 30080 a.out > prefix
```

Then we will run the md5collgen command in order to create the two distinct binary files that share the same hash values.

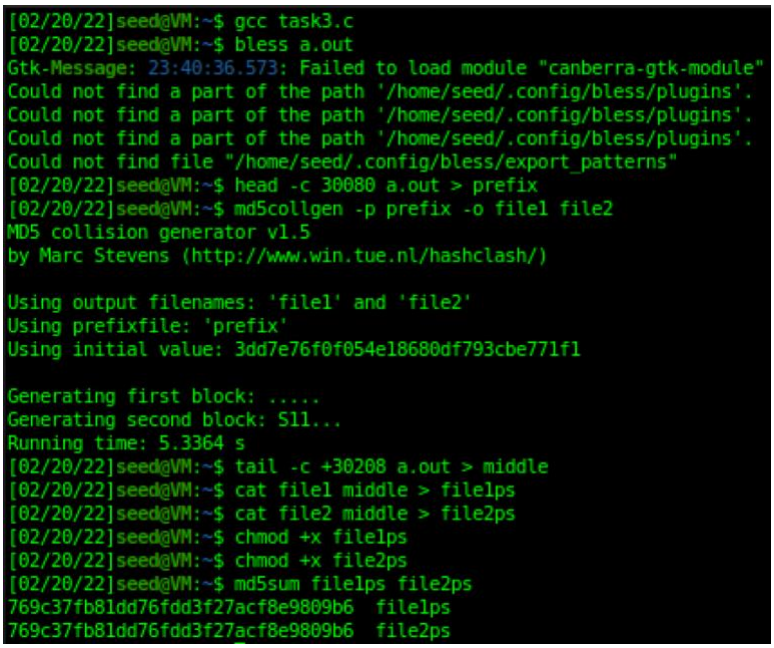
```
$ md5collgen -p prefix -o file1 file2
```

We will run the third command as a formality to check that they, in fact differ as we expect (sanity check). Additionally, verify that the hashes are identical.

```
$ differ file1 file2
$ md5sum file1 file2
```

To comply with the structure of Figure 4, let us include everything after the end of array X into the “middle” section.

```
$tail -c +30208 a.out > middle
```



In order to complete the array within file1 and file2 we need to obtain the excluded bits from Array Y and concatenate them to both files.