SHIQI LIU
LAB REPORT
CS458

Introduction:

This lab delves into the MD5 collision attack which makes use of its length extension property. In this lab, I compare two different files with the same MD5 Hash, and understand MD5 property deeply, then I created two different versions of the program, and make them behave differently. I found an interesting observations is a hash function is said to be secure if it is a one-way hash function and is collision-resistant. The one-way property ensures that given a hash value h, it is computationally infeasible to find an input m such that hash(m) = h.

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

In this task, we need to find out two different files with same MD5 Hash. First, I create a test file. Command is –

```
1. echo TEST > prefix.txt
```

Then, I generated two files with the same md5 hash

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

```
[09/28/20]seed@VM:~$ echo TEST > prefix.txt
[09/28/20]seed@VM:~$ md5collgen -p prefix.txt -o out1.b
in 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.txt'
Using initial value: c07521db2bf9baff14a5b3e496f5fdf8

Generating first block: .......
Generating second block: S11.........................
Running time: 3.5838 s
[09/28/20]seed@VM:~$
```

Then, I checked that the md5sum of these files are the same:

```
[09/28/20]seed@VM:~$ md5sum ./out1.bin
8db6996a1a142ebbd70e6a591cbf85e4  ./out1.bin
[09/28/20]seed@VM:~$ md5sum ./out2.bin
8db6996a1a142ebbd70e6a591cbf85e4  ./out2.bin
[09/28/20]seed@VM:~$
```

Difference of these files are shown blow:

```
[09/28/20]seed@VM:~$ diff -y <(xxd ./out1.bin) <(xxd ./
out2.bin)
00000000: 5445 5354 0a00 0000 0000 0000 0000 0000  TEST
......  00000000: 5445 5354 0a00 0000 0000 0000 0000 00
00  TEST......
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ....
......  00000010: 0000 0000 0000 0000 0000 0000 0000 00
00  ..........
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ....
......  00000020: 0000 0000 0000 0000 0000 0000 0000 00
00  ..........
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ....
......  00000030: 0000 0000 0000 0000 0000 0000 0000 00
00  ..........
00000040: a44f cdb5 059a 3fe0 bc18 4af6 4443 89a5  .O..
..?...  00000040: a44f cdb5 059a 3fe0 bc18 4af6 4443 89
a5  .O....?...
00000050: 9c72 09f6 5c87 3538 07be 78e6 9109 169e  .r..
\.58.. |        00000050: 9c72 0976 5c87 3538 07be 78e6
 9109 169e  .r.v\.58..
00000060: 02d9 e6e5 ea25 5694 e7e3 5df1 0cbe 1b4c  ....
.%V... |        00000060: 02d9 e6e5 ea25 5694 e7e3 5df1
```

Question 1: If the length of your prefix file is not a multiple of 64, what is going to happen?
Answer: It will be padded with zeros.
Question 2: Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.
Answer: It's almost the same as previous experiments, no zero padding is observed.
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: No, most bytes are the same, but there still are few bytes are different.

Task 2: Understanding MD5's Property

At high level, MD5 divides its data into blocks of 64 bytes and then computes the hash iteratively on these blocks. The core of MD5 is a compression function which produces a 128 bit IHV or intermediate hash value. The input for the first iteration i.e., IHV_0 is fixed. Based on the working of the MD5 algorithm, we can derive a property which is - Given two inputs `M` and `N`, if `MD5(M) = MD5(N)`, then for any input `T`, `MD5(M || T) = MD5(N || T)`. Therefore, adding a particular suffix to any two distinct messages having the same MD5 hash, gives two new longer messages for by concatenation of the original and the suffix messages, both of which also have the same MD5 hash.

Therefore, in this task we reuse the outputs from task 1. We have generated two sets of files, there are two files in each set with the same md5sum. We randomly choose file A from the first set and append same content to it to genetate file C. If the properity of MD5 holds, then the MD5 of C remains the same.

```
[09/27/20]seed@VM:~/Desktop$ echo "ABCDEFGHIJKLMNOPQRSTUVWXYZ" > suffix.txt
[09/27/20]seed@VM:~/Desktop$ cat out1.bin suffix.txt > outlong1.bin
[09/27/20]seed@VM:~/Desktop$ cat out2.bin suffix.txt > outlong2.bin
[09/27/20]seed@VM:~/Desktop$ diff outlong1.bin outlong2.bin
Binary files outlong1.bin and outlong2.bin differ
```

using md5sum command:

```
[09/27/20]seed@VM:~/Desktop$ md5sum outlong1.bin
0c03d401f908cbd9cfb74aa505858201  outlong1.bin
[09/27/20]seed@VM:~/Desktop$ md5sum outlong2.bin
0c03d401f908cbd9cfb74aa505858201  outlong2.bin
```

using “xxd” command:

```
[09/27/20]seed@VM:~/Desktop$ xxd outlong1.bin
00000000: dfcd c5df 0edf cf5c af81 9e8c 77ff 1ac2  .......\....w...
00000010: 8968 dca2 cced 5f4c 270a 81f5 974a bc04  .h...._L'....J..
00000020: 4ea3 b6dd 704b 4dbe eb03 de41 a475 bb60  N...pKM....A.u.`
00000030: 965c 7772 4643 6d3b a5e4 763c a0f1 b644  .\wrFCm;..v<...D
00000040: bf23 f002 5acc e7a3 2b62 fd29 acae ad9f  .#..Z...+b.)....
00000050: ade9 9a9a 849f 1200 abb5 10cf 1760 48e7  ............`H.
00000060: d795 2258 9fc5 7e1c 3940 c69b 6a44 e687  .."X..~.9@..jD..
00000070: 9584 c260 a3bb 9e01 71b4 53e9 8072 4be1  ...`....q.S..rK.
00000080: 4142 4344 4546 4748 494a 4b4c 4d4e 4f50  ABCDEFGHIJKLMNOP
00000090: 5152 5354 5556 5758 595a 0a               QRSTUVWXYZ.
[09/27/20]seed@VM:~/Desktop$ xxd outlong2.bin
00000000: dfcd c5df 0edf cf5c af81 9e8c 77ff 1ac2  .......\....w...
00000010: 8968 dc22 cced 5f4c 270a 81f5 974a bc04  .h."._L'....J..
00000020: 4ea3 b6dd 704b 4dbe eb03 de41 a4f5 bb60  N...pKM....A...`
00000030: 965c 7772 4643 6d3b a5e4 76bc a0f1 b644  .\wrFCm;..v....D
00000040: bf23 f002 5acc e7a3 2b62 fd29 acae ad9f  .#..Z...+b.)....
00000050: ade9 9a1a 849f 1200 abb5 10cf 1760 48e7  ............`H.
00000060: d795 2258 9fc5 7e1c 3940 c69b 6ac4 e587  .."X..~.9@..j...
00000070: 9584 c260 a3bb 9e01 71b4 5369 8072 4be1  ...`....q.Si.rK.
00000080: 4142 4344 4546 4748 494a 4b4c 4d4e 4f50  ABCDEFGHIJKLMNOP
00000090: 5152 5354 5556 5758 595a 0a               QRSTUVWXYZ.
```

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

Given a code in C, create two different versions of this code such that the difference in them lies in the array contents, but the hash values of their executables are the same. The code is -

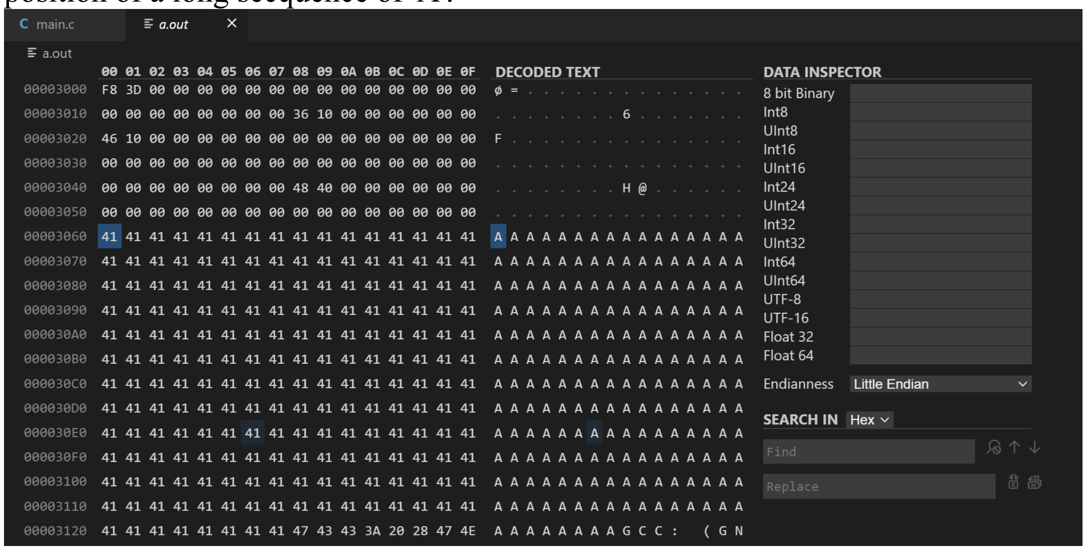
```c
#include <stdio.h>
unsigned char xyz[200] = {
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
```

```
};
int main() {
int i;
for (i = 0; i < 200; i++) {
if(i%25 == 0 && i > 0){ printf("\n");
}
printf("%x ", xyz[i]); }
printf("\n"); }
```

the output of this program is:

```
[09/27/20]seed@VM:~/Desktop$ gcc zheng.c && ./a.out
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
```

To find the position of the array, I used the hex plugin with VS code, it is easy to identify the position of a long secquence of 'A'.



Then, creating prefix and suffix files:

```
[09/27/20]seed@VM:~/Desktop$ head -c 16705 a.out > prefix
[09/27/20]seed@VM:~/Desktop$ tail -c +18243 a.out > suffix
```

Output files:

```
[09/27/20]seed@VM:~/Desktop$ md5collgen -p prefix -o out1 out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1' and 'out2'
Using prefixfile: 'prefix'
Using initial value: d0c64a70a4cce6aa44268721d8019037

Generating first block: ...
Generating second block: S10.......
Running time: 1.94065 s
```

Then, assemble these two new programs:

```
[09/27/20]seed@VM:~/Desktop$ cp out1 p1
[09/27/20]seed@VM:~/Desktop$ cp out2 p2
[09/27/20]seed@VM:~/Desktop$ cat suffix >> p1
[09/27/20]seed@VM:~/Desktop$ cat suffix >> p2
```

Checking the md5sum od these two programs:

```
[09/27/20]seed@VM:~/Desktop$ md5sum p1
ff4e7611579254db633de7051bb44112  p1
[09/27/20]seed@VM:~/Desktop$ md5sum p2
ff4e7611579254db633de7051bb44112  p2
```

Finally checking the outputs are the same.

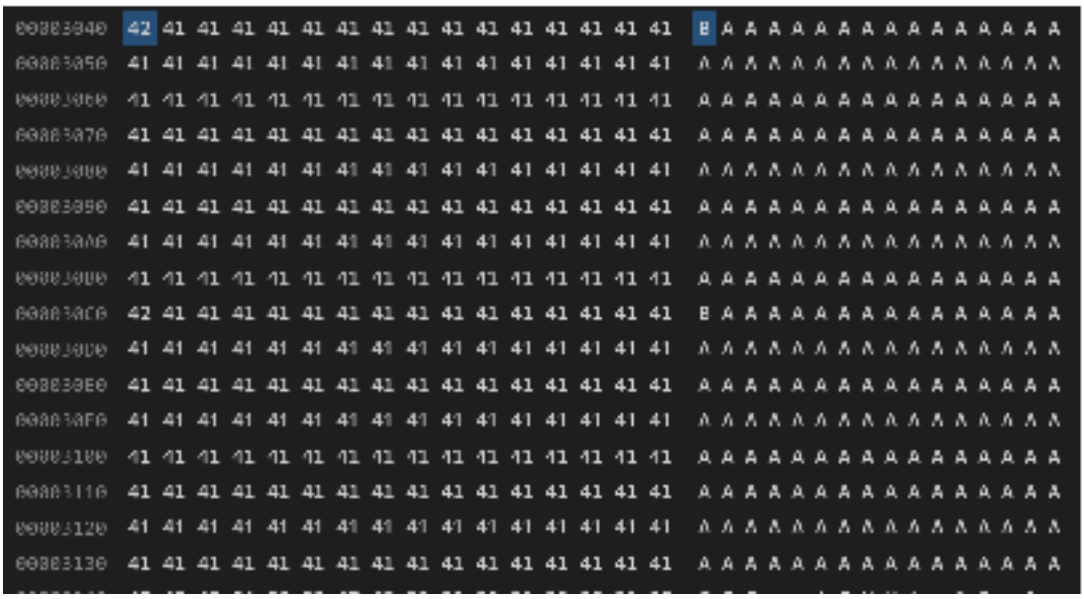
Task 4 : Making two programs behave differently

In this task, we need to design a program to make two programs that share the same MD5 hash behave differently.
Therefore, write a program as follows.

```
1. #include <stdio.h>
2. unsigned char A[128] = {
3. "BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAA
   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"};
4. unsigned char B[128] = { "BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   AA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAA
   AAAAAAAAAAAA"};
5. int main() {
6. for (int i = 0; i < 128; i += 1) {
```

```
7. if (A[i] != B[i]) {
8. printf("Malicious Code Executed!\n");
9. return 0; }
10. }
11. printf("Every Thing Seems Good!\n"); return 0; )
```

If contents in array A and B are the same, the program will print
Every Thing Seems Good!, otherwise the program will print Malicious Code Executed. Similiar to previous task, I used the VS-code to find address of A and B



then we generate two prefix with the same MD5

```
[09/27/20]seed@VM:~/Desktop$ md5collgen -p prefix -o out1 out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1' and 'out2'
Using prefixfile: 'prefix'
Using initial value: 38cfbd078390b95c5f3b1e4e2d66c972
```

then we copy the A array from output1 to B array and generate the suffix

```
[09/27/20]seed@VM:~/Desktop$ tail out1 -c 128 > suffix
[09/27/20]seed@VM:~/Desktop$ tail a.out -c +12608 > temp
[09/27/20]seed@VM:~/Desktop$ cat ./temp >> ./suffix
[09/27/20]seed@VM:~/Desktop$ cp out1 p1
[09/27/20]seed@VM:~/Desktop$
[09/27/20]seed@VM:~/Desktop$ cp ou2 p2
cp: cannot stat 'ou2': No such file or directory
[09/27/20]seed@VM:~/Desktop$ cp out2 p2
[09/27/20]seed@VM:~/Desktop$ cat suffix >> p1
[09/27/20]seed@VM:~/Desktop$ cat suffix >> p2
```

Then we check the MD5 value and the output of these two programs:

```
[09/27/20]seed@VM:~/Desktop$ md5sum out1
ff4e7611579254db633de7051bb44112  out1
[09/27/20]seed@VM:~/Desktop$ md5sum out2
ff4e7611579254db633de7051bb44112  out2
```

Therefore, the same MD5 but behaves differently.

The full script that carry out this experiment are listed below:

```
1.  gcc task4.c
2.  ./a.out
3.  head -c 12352 a.out > prefix
4.  ./md5collgen -p prefix -o out1 out2
5.  tail out1 -c 128 > suffix
6.  tail a.out -c +12608 > temp
7.  cat ./temp >> ./suffix
8.  cp out1 p1
9.  cp out2 p2
10. cat suffix >> p1
11. cat suffix >> p2
12. md5sum p1
13. md5sum p2
14. ./p1
15. ./p2
```