

MD5 COLLISION ATTACK LAB

Jasmine Joy (500924677)

Task 1: Generating two different files with the same MD5 Hash

A file 'prefix' was created. Below is the implementation of md5collgen

```
[03/07/20]seed@VM:~/.../Lab3$ cat prefix.txt
The quick brown fox jumps over the lazy dog.
[03/07/20]seed@VM:~/.../Lab3$ md5collgen -p prefix.txt
-o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

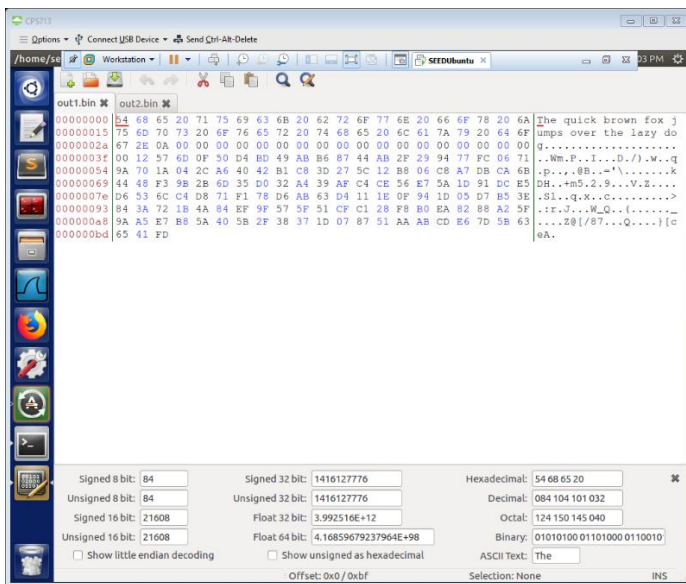
Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 9d40f4fb3ecb760166124e8c68388499

Generating first block: .....
Generating second block: S11...
Running time: 20.903 s
[03/07/20]seed@VM:~/.../Lab3$
```

Using checksum to determine if the output files out1.bin and out2.bin are the same.

```
[03/07/20]seed@VM:~/.../Lab3$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[03/07/20]seed@VM:~/.../Lab3$ md5sum out1.bin
dcd7205e747bb41fd21261b5d8911d34 out1.bin
[03/07/20]seed@VM:~/.../Lab3$ md5sum out2.bin
dcd7205e747bb41fd21261b5d8911d34 out2.bin
[03/07/20]seed@VM:~/.../Lab3$
```

Looking at the output files using hex editor:

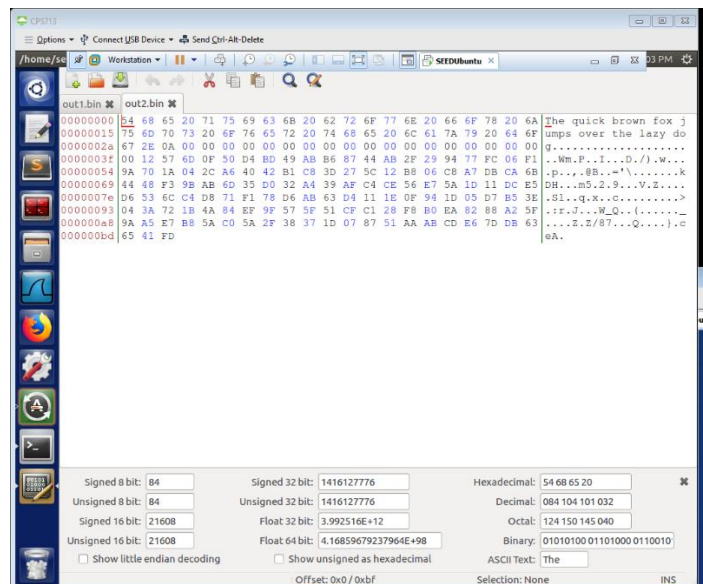


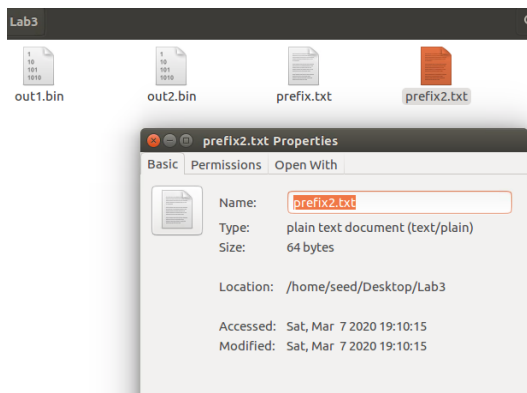
Answer 1

If the length is not a multiple of 64, then the text gets padded with 0s (as seen above in hex editor).

Answer 2

Creating a new textfile with arbitrary content (64 bytes).





Running the collision tool again gives the following:

```
[03/07/20]seed@VM:~/.../Lab3$ cat prefix2.txt
There once was a splendid velveteen rabbit who was fat n'bunchy
[03/07/20]seed@VM:~/.../Lab3$ md5collgen -p prefix2.txt -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: 'prefix2.txt'
Using initial value: 7b93c3bf7c78d233c90a0576f19640be

Generating first block: .....
Generating second block: W...
Running time: 68.6717 s
[03/07/20]seed@VM:~/.../Lab3$
```

Looking at the output files using hex editor:

As seen above, when the file is exactly 64 bytes, then no padding with zeros is applied.

Answer 3

Not all bytes are different for the two output files (as seen in output1.bin and output2.bun above).

The bytes differ at some positions (see highlighted region in red). The differences need not be always constant.

```
[03/07/20]seed@VM:~/.../Lab3$ md5sum out1.bin
b6168781a3e19548fc96b76cf5e66906 out1.bin
[03/07/20]seed@VM:~/.../Lab3$ md5sum out2.bin
b6168781a3e19548fc96b76cf5e66906 out2.bin
[03/07/20]seed@VM:~/.../Lab3$ diff out1.bin out2.bin
2c2
< 0060vG0Ks+0E/rK00Q-0yB0B90/000r#00
                                dJ<0c000hkf0000
                                0w00;050a00*le>00
x001\000000,00J00003E200AAb0000
                                0?+00000\00000I
\ No newline at end of file
---
> 0060vG0Ks+0E/0K00Q-0yB0B90/000r#00
                                dJ<00000hkf0000
                                0w00;050a00*le>00
x001\500000,00J00003E200AAb0000
                                0?+00000\0|000I
\ No newline at end of file
Trash )]seed@VM:~/.../Lab3$
```

Task 2: Understanding MD5s Property

The property explored in this task is based on the Target Collision Resistance property of Hash functions.

That is; for M and N where their hashes are the same $h(M) = h(N)$. If a segment X was added to M and N; then the hashes would be equal $h(M+x) = h(N+x)$

Using the file prefix above:

```
[03/09/20]seed@VM:~/.../Lab3$ cat prefix.txt
The quick brown fox jumps over the lazy dog.
[03/09/20]seed@VM:~/.../Lab3$ md5collgen -p prefix.txt -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.txt'
Using initial value: 9d40f4fb3ecb760166124e8c68388499

Generating first block: .....
Generating second block: S00.....
Running time: 18.3947 s
[03/09/20]seed@VM:~/.../Lab3$
```

The checksums of the original outputs of the file are, as expected, the same.

```
[03/09/20]seed@VM:~/.../Lab3$ md5sum out1 out2
ff7751768307fb0ee946a59222d03d2e out1
ff7751768307fb0ee946a59222d03d2e out2
[03/09/20]seed@VM:~/.../Lab3$
```

The word “random” was added into both outputs. And, as predicted, the hashes of the outputs stay the same, but a new hash was generated.

```
[03/09/20]seed@VM:~/.../Lab3$ echo random >> out1
[03/09/20]seed@VM:~/.../Lab3$ echo random >> out2
[03/09/20]seed@VM:~/.../Lab3$ cat out1
The quick brown fox jumps over the lazy dog.
x 0$50j0000S00010h;0w00'09100)0000q0K(kD9AS0Nh0010000C00xE000 BB}}^03B00500m001000-000{000p
}20000]0r 0L000G00000)00000000random
[03/09/20]seed@VM:~/.../Lab3$ cat out2
The quick brown fox jumps over the lazy dog.
x 0$50j0000S00010h;0w00'09100)0000q0K(kD9H0S0Nh0010000C00xE000 BB}}^03B00500m001000-000{000
p}20000]0r 0L000G00000)00000000random
[03/09/20]seed@VM:~/.../Lab3$ md5sum out1 out2
2910363135d25aedeb25a592db22250b out1
2910363135d25aedeb25a592db22250b out2
[03/09/20]seed@VM:~/.../Lab3$
```

Task 3: Generating two executable files with the same MD5 Hash

```
GNU nano 2.5.3 File: task3.c Modified
Search your computer

#include <stdio.h>

unsigned char a[200] = { 'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','$'
int main()
{
    int i;
    for(i = 0; i < 200; i++)
    {
        printf("%x", a[i]);
    }
    printf("\n");
}
```

```
[03/09/20]seed@VM:~/.../Lab3$ pico task3.c
[03/09/20]seed@VM:~/.../Lab3$ gcc task3.c -o task3.out
```

task3.out ✕

00001008	00 00 00 00 06 83 04 08 16 83 04 08 26 83 04 08 00 00&.....
0000101a	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000102c	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000103e	00 00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50	..ABCDEF GHI JKLMNOP
00001050	51 52 53 54 55 56 57 58 59 5A 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
00001062	38 39 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50	89ABCDEFGHI JKLMNOP
00001074	51 52 53 54 55 56 57 58 59 5A 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
00001086	38 39 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50	89ABCDEFGHI JKLMNOP
00001098	51 52 53 54 55 56 57 58 59 5A 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
000010aa	38 39 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50	89ABCDEFGHI JKLMNOP
000010bc	51 52 53 54 55 56 57 58 59 5A 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
000010ce	38 39 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50	89ABCDEFGHI JKLMNOP
000010e0	51 52 53 54 55 56 57 58 59 5A 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
000010f2	38 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	89.....
00001104	00 00 00 00 47 43 43 3A 20 28 55 62 75 6E 74 75 20 35GCC: (Ubuntu 5
00001116	2E 34 2E 30 2D 36 75 62 75 6E 74 75 31 7E 31 36 2E 30	.4.0-6ubuntu1~16.0
00001128	34 2E 34 29 20 35 2E 34 2E 30 20 32 30 31 36 30 36 30	4.4) 5.4.0 2016060
0000113a	39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	9.....
0000114c	00 00 00 00 54 81 04 08 00 00 00 00 03 00 01 00 00 00T.....
0000115e	00 00 68 81 04 08 00 00 00 00 03 00 02 00 00 00 00 00	..h.....
00001170	88 81 04 08 00 00 00 00 03 00 03 00 00 00 00 00 00 00AC 81
00001182	04 08 00 00 00 00 03 00 04 00 00 00 00 00 00 00 00 00
00001194	00 00 00 00 03 00 05 00 00 00 00 00 2C 82 04 08 00 00,
000011a6	00 00 03 00 06 00 00 00 00 00 80 82 04 08 00 00 00 00

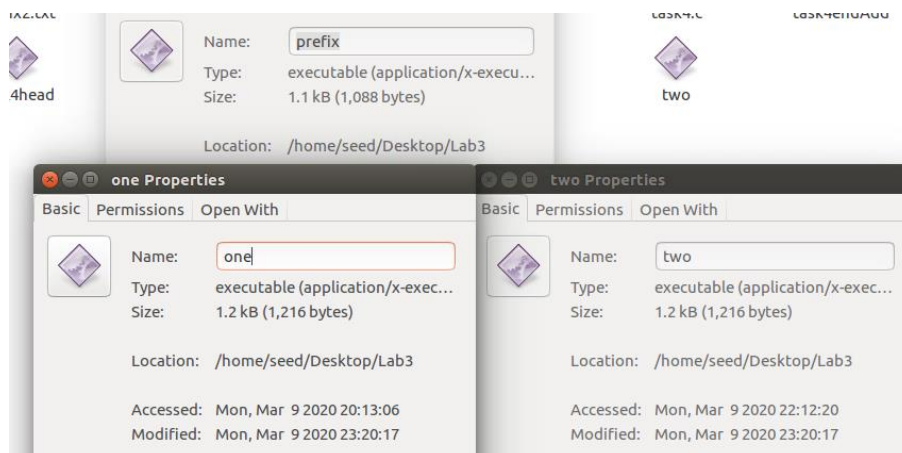
Signed 8 bit:	65	Signed 32 bit:	1094861636	Hexadecimal:	41 42 43 44	✕
Unsigned 8 bit:	65	Unsigned 32 bit:	1094861636	Decimal:	065 066 067 068	
Signed 16 bit:	16706	Float 32 bit:	12.14142	Octal:	101 102 103 104	
Unsigned 16 bit:	16706	Float 64 bit:	2393736.54120723	Binary:	01000001 01000010 010	
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	ABCD	
Offset: 0x1040 / 0x1dcf				Selection: None	INS	

As seen above, the test starts at the 1040th position.

$1040 \% 64 = 16$

The next integer that is fully divisible by 64 would be 1088.

Hence, the filesize needs to be 1088 bytes.



Dividing the binary file task3.out into two segments: prefix, and suffix.


```
[03/09/20]seed@VM:~/.../Lab3$ head -c 1088 task3.out > prefix
[03/09/20]seed@VM:~/.../Lab3$ md5collgen -p prefix -o one two
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'one' and 'two'
Using prefixfile: 'prefix'
Using initial value: f63809b1d4dc3365d4dc0f62dc1fc3ec

Generating first block: .....
Generating second block: S11.....
Running time: 39.7606 s
[03/09/20]seed@VM:~/.../Lab3$ tail -c 1217 task3.out > task3end
[03/09/20]seed@VM:~/.../Lab3$
```

Adding the tail-end segment to the output files of prefix

```
[03/09/20]seed@VM:~/.../Lab3$ cat task3end >> one
[03/09/20]seed@VM:~/.../Lab3$ cat task3end >> two
[03/09/20]seed@VM:~/.../Lab3$
```

Trying to make the outputs executable did not produce the desired result.

```
[03/09/20]seed@VM:~/.../Lab3$ chmod +x one
[03/09/20]seed@VM:~/.../Lab3$ chmod +x two
[03/09/20]seed@VM:~/.../Lab3$ ./one
Segmentation fault
[03/09/20]seed@VM:~/.../Lab3$ ./two
Segmentation fault
```

In theory, running the executable should provide. And trying to run them again should give me textfiles instead of binary. But since, I was unable to get the testfiles, did a checksum on the binaries (which provided the same hash), and a report on the difference between the binaries (they do have differences between them)

```
[03/09/20]seed@VM:~/.../Lab3$ md5sum one two
bf3703878db0ec0bdec7cfeb4e4161de one
bf3703878db0ec0bdec7cfeb4e4161de two
[03/09/20]seed@VM:~/.../Lab3$ diff one two
Binary files one and two differ
[03/09/20]seed@VM:~/.../Lab3$
```

Task 4: Making the Two Programs Behave Differently

Where arrays a and b are identical:

```
#include <stdio.h>

unsigned char a[200] = { 'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','$'
unsigned char b[200] = { 'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','$'
int main()
{
    int i, variable;
    for(i = 0; i < 200; i++)
    {
        if(a[i]!=b[i])
        {
            variable=0;
        }
        else {
            variable=1;
        }
    }

    if(variable=1){
        printf("Safe code");
    }
    else{
        printf("Malicious code");
    }
    printf("\n");
}
```

Hex editor view of file 'a.out' showing memory addresses and corresponding hexadecimal and ASCII values. The ASCII column contains a mix of printable characters and control characters, including a null byte at address 00001008.

Address	Hex	ASCII
00001008	00 00 00 00 06 83 04 08 16 83 04 08 26 83 04 08 00 00&.....
0000101a	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000102c	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000103e	00 00 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50	..ABCDEFGHIJKLMN
00001050	51 52 53 54 55 56 57 58 59 5a 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
00001062	38 39 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50	89ABCDEFGHIJKLMN
00001074	51 52 53 54 55 56 57 58 59 5a 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
00001086	38 39 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50	89ABCDEFGHIJKLMN
00001098	51 52 53 54 55 56 57 58 59 5a 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
000010aa	38 39 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50	89ABCDEFGHIJKLMN
000010bc	51 52 53 54 55 56 57 58 59 5a 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
000010ce	38 39 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50	89ABCDEFGHIJKLMN
000010e0	51 52 53 54 55 56 57 58 59 5a 30 31 32 33 34 35 36 37	QRSTUVWXYZ01234567
000010f2	38 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	89.....
00001104	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001116	00 00 00 00 00 00 00 00 00 00 41 42 43 44 45 46 47 48ABCDEFGH
00001128	49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a	IJKLMNOPQRSTUVWXYZ
0000113a	30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 47 48	0123456789ABCDEFGH
0000114c	49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a	IJKLMNOPQRSTUVWXYZ
0000115e	30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 47 48	0123456789ABCDEFGH
00001170	49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a	IJKLMNOPQRSTUVWXYZ
00001182	30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 47 48	0123456789ABCDEFGH
00001194	49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a	IJKLMNOPQRSTUVWXYZ

Conversion tool interface showing various data formats for the selected hex value 41 42 43 44:

- Signed 8 bit: 65
- Unsigned 8 bit: 65
- Signed 16 bit: 16706
- Unsigned 16 bit: 16706
- Show little endian decoding: ☐
- Signed 32 bit: 1094861636
- Unsigned 32 bit: 1094861636
- Float 32 bit: 12.14142
- Float 64 bit: 2393736.54120723
- Show unsigned as hexadecimal: ☐
- Hexadecimal: 41 42 43 44
- Decimal: 065 066 067 068
- Octal: 101 102 103 104
- Binary: 01000001 01000010 01000011
- ASCII Text: ABCD

Offset: 010100 / 017303 Selection: None INS

10100%64 = 52

However, 10176%64 = 0. Therefore:

Following the same process as in Task 3:

```
[03/09/20]seed@VM:~/.../Lab3$ head -c 10176 a.out > task4head
[03/09/20]seed@VM:~/.../Lab3$ md5collgen -p prefix -o task4onef task4twof
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'task4onef' and 'task4twof'
Using prefixfile: 'prefix'
Using initial value: f63809b1d4dc3365d4dc0f62dc1fc3ec

Generating first block: .....
Generating second block: W.....
Running time: 6.93946 s
[03/09/20]seed@VM:~/.../Lab3$ md5sum tas4onef task4twof
md5sum: tas4onef: No such file or directory
d73090199bcf397401a3047f48e04210 task4twof
[03/09/20]seed@VM:~/.../Lab3$
```

```
[03/09/20]seed@VM:~/.../Lab3$ head -c 4353 a.out > endtoAdd
[03/09/20]seed@VM:~/.../Lab3$ tail -c +129 endtoAdd > task4endAdd
[03/09/20]seed@VM:~/.../Lab3$
```

The hashes are the same, but the files are different.

```
[03/09/20]seed@VM:~/.../Lab3$ md5sum bCode mCode
d73090199bcf397401a3047f48e04210 bCode
d73090199bcf397401a3047f48e04210 mCode
[03/09/20]seed@VM:~/.../Lab3$ cp task4onef bCode
[03/09/20]seed@VM:~/.../Lab3$ cp task4twof mCode
[03/09/20]seed@VM:~/.../Lab3$ diff bCode mCode
Binary files bCode and mCode differ
```

Same issue as Task 3 encountered:

```
[03/09/20]seed@VM:~/.../Lab3$ chmod +x bCode mCode
[03/09/20]seed@VM:~/.../Lab3$ ./bCode
Segmentation fault
[03/09/20]seed@VM:~/.../Lab3$ ./mCode
Segmentation fault
[03/09/20]seed@VM:~/.../Lab3$
```

I wasn't able to figure out how to implement the code correctly to check if when benign code or malicious code is executed. But, in theory, `chmod +x` should've made the files executable.

To complete the task, I should find a way to make the two files distinct such that their hash is also different. This would make execute the 'malicious' portion of the code.