

Cryptography:
MD5 Collision Attack Lab
Batyi Amatov (amatobat@b-tu.de)

Table of Content

<u>Content</u>	<u>Page number</u>
Abstract	1
1. The Tasks Description	
Task 1: Generating Two Different Files with the Same MD5 Hash	1 - 3
Task 2: Understanding MD5's Property	4
Task 3: Generating Two Executable Files with the Same MD5 Hash	5- 7
Task 4: Making the Two Programs Behave Differently	8- 9
2. Conclusion	10
Annexure 1	11
Annexure 2	12
Reference	13

Abstract:

We know that it is very difficult to generate same hash function value from different content. But in this lab our main goal is to study and understand the impact of collision attacks. We have created two different files/programs which are not identical to each other. However from the different categories of files/program we are trying to generate collision attacks that share the same MD5 hash value but have completely different behaviors.

1. The Tasks Description:

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

Our task is to generate two different files with the same MD5 hash values. For this purpose we have used the md5collgen program to generate two files with the same hash value where the same prefix has been shared.

In figure 1, we have our initial input file with share prefix.



Figure 1: A prefix file with arbitrary content

Based on the given prefix (prefix.txt), we are generating two different files (out1.bin, out2.bin), as shown in Figure 3 and 4, which will give the same MD5 hash.

```
Terminal
[04/12/19]csl@vm:~/bin$ ls
again bin.tar.gz md5collgen task01 task02 task03 task04
[04/12/19]csl@vm:~/bin$ cd again
[04/12/19]csl@vm:~/bin/again$ cd task01/
[04/12/19]csl@vm:~/../task01$ ls
md5collgen
[04/12/19]csl@vm:~/../task01$ touch prefix.txt
[04/12/19]csl@vm:~/../task01$ vim prefix.txt
[04/12/19]csl@vm:~/../task01$ 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: 4d78ab4e9e87b837abdae646a10e844a

Generating first block: .....
Generating second block: S10.....
Running time: 110.787 s
[04/12/19]csl@vm:~/../task01$ ls
md5collgen out1.bin out2.bin prefix.txt
[04/12/19]csl@vm:~/../task01$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[04/12/19]csl@vm:~/../task01$ md5sum out1.bin
67fce0711222643edaba9b40a1a662fd out1.bin
[04/12/19]csl@vm:~/../task01$ md5sum out2.bin
67fce0711222643edaba9b40a1a662fd out2.bin
[04/12/19]csl@vm:~/../task01$
```

Figure 2: Command Line Instructions

[illegible]

out2.bin	
00000000	48 65 6C 6C 6F 20 69 74 20 69 73 20 6D 65 21 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 Hello it is me!.....
0000001e	00
0000003c	00 00 00 00 CD 2F 31 E2 4C 30 C9 C7 2E 7A F8 4D B6 AF 98 7C B6 CB 1F E2 4F F7 EB B9 60 DE /1.L0...z.M... ...O...`
0000005a	81 9D 57 2A 1D D7 7A AB BE E5 5A E8 8C 66 EB EB 51 19 CC 6D 70 1D B3 F8 54 F4 1C AD F5 78 ..W*..z...Z..f..Q..mp...T...x
00000078	24 98 76 F5 E5 89 A3 D6 8D 49 8B C3 80 D8 16 A1 29 56 5A 93 EC 88 10 1D BB A5 4F F8 70 5F \$.v.....I.....)VZ.....O.p
00000096	13 01 2B C5 10 67 8A 5F 68 C6 D7 95 30 44 97 C5 DC FB 3B FF 0F 3B 50 1C 0D CD 9D 7F EC F8 ..+.g._h...0D...;.;P.....
000000b4	83 A3 CA 4C 15 F9 86 2F A7 9E 04 76 ...L.../...v

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

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

[illegible]

	Untitled 1*	prefix64.txt	out1_64.bin	out2_64.bin	
00000000	41	41	41	41	AAAAA.....
0000001e	41	41	41	41	AAAAA.....
0000003c	41	41	0A FB DD 57 8B C2 D5 A3 E6 7D B5 8A C3 01 24 D4 58 DB BE 97 AF 30 79 A2 F0 22 3A		AAA...W....}....\$.X....Oy..":
0000005a	79	05 D0 A2 38 3C 07 9A B8 0F 48 08 88 6C 0B FC 67 FB 03 D2 62 2A 04 18 E3 59 E9 5A 7D 4A			y...<...H..l.g...b*.~Y.Z]J
00000078	94	1E 4C F8 CE FE 71 B1 5E 60 D5 8C 29 83 58 15 93 3C 5C BB 3A 83 F7 96 B7 C4 81 BC EA CB			..L...q.^...) .X.<\.:.....
00000096	06	2E 86 E9 8B CF 5F 51 A7 C1 1C AA AC 2E 86 0C 84 BE A9 8E 6F F1 AC F7 D1 22 E7 B3 5F 1C		Q.....o..."._.
000000b4	49	CC BB D0 58 E0 45 28 9B C9 90 DC			I...X.E(....

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

Untitled 1*	prefix64.txt	out_1_64.bin
00000000	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0000001e	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0000003c	41 41 41 0A FB DD 57 8B C2 D5 A3 E6 7D B5 8A C3 01 24 D4 58 DB BE 97 2F 30 79 A2 F0 22 3A	AAA...W....}...\$.X.../0y..":
0000005a	79 05 D0 A2 38 3C 07 9A B8 0F 48 08 88 6C 0B FC 67 FB 03 52 62 2A 04 18 E3 59 E9 5A 7D 4A	y...8<...H..l.g..Rb*...Y.Z}J
00000078	94 1E 4C 78 CE FE 71 B1 5E 60 D5 8C 29 83 58 15 93 3C 5C BB 3A 83 F7 96 B7 C4 81 3C EA CB	..Lx..q.^`.) .X.<\.:.....<..
00000096	06 2E 86 E9 8B CF 5F 51 A7 C1 1C AA AC 2E 86 0C 84 BE A9 8E 6F F1 AC 77 D2 22 E7 B3 5F 1C_Q.....o..w.."._.
000000b4	49 CC BB D0 58 E0 45 A8 9B C9 90 DC	I...X.E.....

Untitled 1*	prefix64.txt	out1_64.bin	out2_64.bin
00000000	41 41		AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0000001e	41 41		AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0000003c	41 41 41 0A FB DD 57 8B C2 D5 A3 E6 7D B5 8A C3 01 24 D4 58 DB BE 97 AF 30 79 A2 F0 22 3A		AAA...W....}...\$.X...0y..":
0000005a	79 05 D0 A2 38 3C 07 9A B8 0F 48 08 88 6C 0B FC 67 FB 03 D2 62 2A 04 18 E3 59 E9 5A 7D 4A		y...8<...H..l..g...b*...Y.Z}J
00000078	94 1E 4C F8 CE FE 71 B1 5E 60 D5 8C 29 83 58 15 93 3C 5C BB 3A 83 F7 96 B7 C4 81 BC EA CB		.L...q.^(..).X.<\.:.....
00000096	06 2E 86 E9 8B CF 5F 51 A7 C1 1C AA AC 2E 86 0C 84 BE A9 8E 6F F1 AC F7 D1 22 E7 B3 5F 1C	_Q.....o....."._.
000000b4	49 CC BB D0 58 E0 45 28 9B C9 90 DC		I...X.E(....

5

Task 2: Understanding MD5's Property

Our task to identify the properties of the MD5 algorithm.

In figure 9, we observed the property of the MD5 algorithm. If adding any input by concatenation to the existing two different files which share the same MD5 hash, we would be able to get a result in two outputs that have the same hash value.

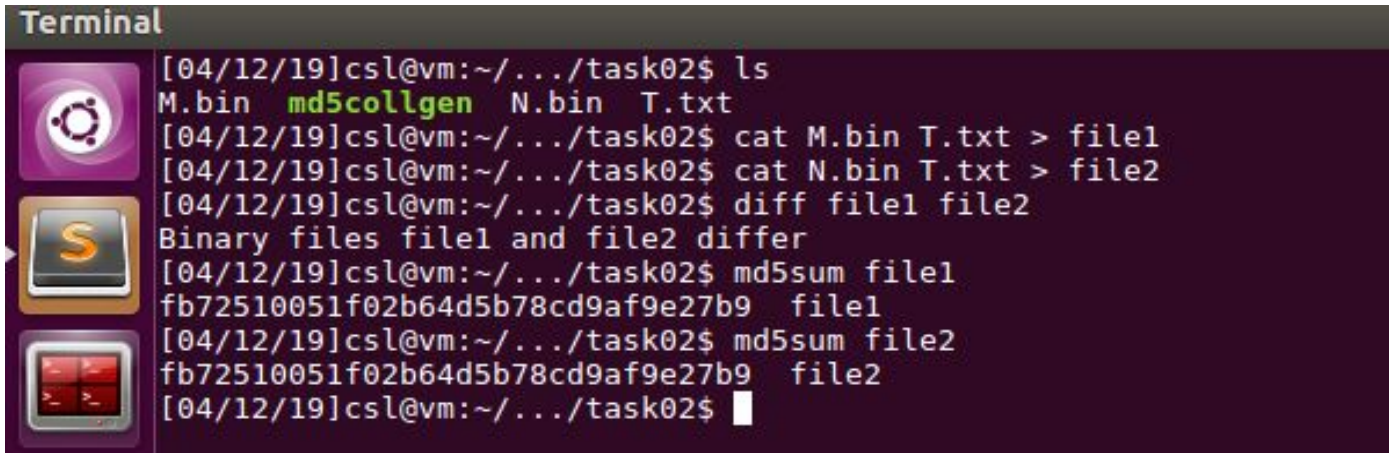
A terminal window titled "Terminal" with a dark background. On the left side, there are three application icons: a purple circle with a white gear, a yellow square with a white 'S', and a red square with a white 'X'. The terminal text shows a user 'csl' at a machine 'vm' in the directory '~/.../task02'. The user lists files 'M.bin', 'md5collgen', 'N.bin', and 'T.txt'. They then concatenate 'M.bin' and 'T.txt' into 'file1', and 'N.bin' and 'T.txt' into 'file2'. A 'diff' command shows the files differ. Finally, 'md5sum' is run on both files, resulting in the same MD5 hash: 'fb72510051f02b64d5b78cd9af9e27b9'.

Figure 9: To understand the properties of the MD5 algorithm

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

Our task is to generate two executable files with the same MD5 Hash based on the C code provided in “Annexure 1”. After compilation we got executable file which we need to divide into two parts: prefix which is multiple of 64 byte (which ends in the memory of array), then we left 128 byte untouched and get the rest as suffix which are shown in Figure 10, 11 and 12. For this purpose we used the following commands:

```
$ head -c 4160 codeExecutable > prefix
```

```
$ tail -c 3348 codeExecutable > suffix
```

After generating by mdcollgen on prefix we get P and Q files which share the same MD5 hash, but have differences in the last 128 bytes.

codeExecutable ✕

00000f1e	00 00 CC 82 04 08 0D 00 00 00 04 85 04 08 19 00 00 00 08 9F 04 08 1B 00 00 00 04 00 00 00
00000f3c	1A 00 00 00 0C 9F 04 08 1C 00 00 00 04 00 00 00 F5 FE FF 6F AC 81 04 08 05 00 00 00 2C 82O.....
00000f5a	04 08 06 00 00 00 CC 81 04 08 0A 00 00 00 54 00 00 00 0B 00 00 00 10 00 00 00 15 00 00 00T.....
00000f78	00 00 00 00 03 00 00 00 00 A0 04 08 02 00 00 00 18 00 00 00 14 00 00 00 11 00 00 00 17 00
00000f96	00 00 B4 82 04 08 11 00 00 00 AC 82 04 08 12 00 00 00 08 00 00 00 13 00 00 00 08 00 00 00
00000fb4	FE FF FF 6F 8C 82 04 08 FF FF FF 6F 01 00 00 00 F0 FF FF 6F 80 82 04 08 00 00 00 00 00 00	...O.....O.....O.....
00000fd2	00 00
00000ff0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 9F 04 08 00 00 00 00 00 00 00 06 83
0000100e	04 08 16 83 04 08 26 83 04 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00&.....
0000102c	00 00AAAAAAAA
0000104a	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00001068	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00001086	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010a4	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010c2	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010e0	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010fe	41 41	AAAAAAAAAGGCC: (Ubuntu 5.4.0-6
0000111c	75 62 75 6E 74 75 31 7E 31 36 2E 30 34 2E 34 29 20 35 2E 34 2E 30 20 32 30 31 36 30 36 30	ubuntu1~16.04.4) 5.4.0 2016060
0000113a	39 00	9.....T.....
00001158	03 00 01 00 00 00 00 00 68 81 04 08 00 00 00 00 03 00 02 00 00 00 00 00 88 81 04 08 00h.....
00001176	00 00 03 00 03 00 00 00 00 00 AC 81 04 08 00 00 00 00 03 00 04 00 00 00 00 00 CC 81 04 08
00001194	00 00 00 00 03 00 05 00 00 00 00 00 2C 82 04 08 00 00 00 00 03 00 06 00 00 00 00 80 82
000011b2	04 08 00 00 00 00 03 00 07 00 00 00 00 00 8C 82 04 08 00 00 00 00 03 00 08 00 00 00 00
000011d0	AC 82 04 08 00 00 00 00 03 00 09 00 00 00 00 00 B4 82 04 08 00 00 00 00 03 00 0A 00 00 00
000011ee	00 00 CC 82 04 08 00 00 00 00 03 00 0B 00 00 00 00 F0 82 04 08 00 00 00 00 03 00 0C 00
0000120c	00 00 00 00 30 83 04 08 00 00 00 00 03 00 0D 00 00 00 00 40 83 04 08 00 00 00 00 03 000.....@.....

Signed 8 bit: 127

Unsigned 8 bit: 127

Signed 16 bit: 32581

Unsigned 16 bit: 32581

☐ Show little endian decoding

Signed 32 bit: 2135247942

Unsigned 32 bit: 2135247942

Float 32 bit: 2.622539E+38

Float 64 bit: 1.16843158668567E+305

☐ Show unsigned as hexadecimal

Hexadecimal: 7F 45 4C 46 ✕

Decimal: 127 069 076 070

Octal: 177 105 114 106

Binary: 01111111 01000101 01001100 01000110

ASCII Text: 0912 ELF

Offset: 0 / 7635

Selection: 0 to 4159 (4160 bytes)

INS

Figure 10: Executable file with prefix

codeExecutable ✕		
00000f3c	1A 00 00 00 0C 9F 04 08 1C 00 00 00 04 00 00 00 F5 FE FF 6F AC 81 04 08 05 00 00 00 2C 82O.....
00000f5a	04 08 06 00 00 00 CC 81 04 08 0A 00 00 00 54 00 00 00 0B 00 00 00 10 00 00 00 15 00 00 00T.....
00000f78	00 00 00 00 03 00 00 00 00 A0 04 08 02 00 00 00 18 00 00 00 14 00 00 00 11 00 00 00 17 00
00000f96	00 00 B4 82 04 08 11 00 00 00 AC 82 04 08 12 00 00 00 08 00 00 00 13 00 00 00 08 00 00 00
00000fb4	FE FF FF 6F 8C 82 04 08 FF FF FF 6F 01 00 00 00 F0 FF FF 6F 80 82 04 08 00 00 00 00 00 00	...O...O...O.....
00000fd2	00 00
00000ff0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 9F 04 08 00 00 00 00 00 00 06 83
0000100e	04 08 16 83 04 08 26 83 04 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00&.....
0000102c	00 41 41 41 41 41 41 41 41 41AAAAAAAA
0000104a	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00001068	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00001086	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010a4	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABA
000010c2	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010e0	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010fe	41 41	AAAAAAAAAAAAGCC: (Ubuntu 5.4.0-6
0000111c	75 62 75 6E 74 75 31 7E 31 36 2E 30 34 2E 34 29 20 35 2E 34 2E 30 20 32 30 31 36 30 36 30	ubuntu1~16.04.4) 5.4.0 2016060
0000113a	39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 54 81 04 08 00 00 00 00	9.....T.....
00001158	03 00 01 00 00 00 00 00 68 81 04 08 00 00 00 00 03 00 02 00 00 00 00 88 81 04 08 00 00h.....
00001176	00 00 03 00 03 00 00 00 00 00 AC 81 04 08 00 00 00 00 03 00 04 00 00 00 00 CC 81 04 08
00001194	00 00 00 00 03 00 05 00 00 00 00 00 2C 82 04 08 00 00 00 00 03 00 06 00 00 00 00 80 82
000011b2	04 08 00 00 00 00 03 00 07 00 00 00 00 00 8C 82 04 08 00 00 00 00 03 00 08 00 00 00 00
000011d0	AC 82 04 08 00 00 00 00 03 00 09 00 00 00 00 00 B4 82 04 08 00 00 00 00 03 00 0A 00 00 00
000011ee	00 00 CC 82 04 08 00 00 00 00 03 00 0B 00 00 00 00 F0 82 04 08 00 00 00 00 03 00 0C 00
0000120c	00 00 00 00 30 83 04 08 00 00 00 00 03 00 0D 00 00 00 40 83 04 08 00 00 00 00 03 000.....@.....
0000122a	0E 00 00 00 00 00 04 85 04 08 00 00 00 00 03 00 0F 00 00 00 18 85 04 08 00 00 00 00
<div> <div>Signed 8 bit: 65</div> <div>Unsigned 8 bit: 65</div> <div>Signed 16 bit: 16705</div> <div>Unsigned 16 bit: 16705</div> <div><input type="checkbox"/> Show little endian decoding</div> </div> <div> <div>Signed 32 bit: 1094795585</div> <div>Unsigned 32 bit: 1094795585</div> <div>Float 32 bit: 12.07843</div> <div>Float 64 bit: 2261634.50980392</div> <div><input type="checkbox"/> Show unsigned as hexadecimal</div> </div> <div> <div>Hexadecimal: 41 41 41 41 ✕</div> <div>Decimal: 065 065 065 065</div> <div>Octal: 101 101 101 101</div> <div>Binary: 01000001 01000001 01000001 01000001</div> <div>ASCII Text: AAAA</div> </div>		
Offset: 4288 / 7635 Selection: 4160 to 4287 (128 bytes) INS		

Figure 11: Executable file with 128-byte region

codeExecutable ✕

00000f3c	1A 00 00 00 0C 9F 04 08 1C 00 00 00 04 00 00 00 F5 FE FF 6F AC 81 04 08 05 00 00 00 2C 82o.....,
00000f5a	04 08 06 00 00 00 CC 81 04 08 0A 00 00 00 54 00 00 00 0B 00 00 00 10 00 00 00 15 00 00 00T.....
00000f78	00 00 00 00 03 00 00 00 00 A0 04 08 02 00 00 00 18 00 00 00 14 00 00 00 11 00 00 00 17 00
00000f96	00 00 B4 82 04 08 11 00 00 00 AC 82 04 08 12 00 00 00 08 00 00 00 13 00 00 00 08 00 00 00
00000fb4	FE FF FF 6F 8C 82 04 08 FF FF FF 6F 01 00 00 00 F0 FF FF 6F 80 82 04 08 00 00 00 00 00 00	...o.....o.....o.....
00000fd2	00 00
00000ff0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 9F 04 08 00 00 00 00 00 00 00 06 83
0000100e	04 08 16 83 04 08 26 83 04 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00&.....
0000102c	00 41 41 41 41 41 41 41 41 41AAAAAAAA
0000104a	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00001068	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
00001086	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010a4	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010c2	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010e0	41 41	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
000010fe	41 41	AAAAAAAAAGCC: (Ubuntu 5.4.0-6
0000111c	75 62 75 6E 74 75 31 7E 31 36 2E 30 34 2E 34 29 20 35 2E 34 2E 30 20 32 30 31 36 30 36 30	ubuntul~16.04.4) 5.4.0 2016060
0000113a	39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 54 81 04 08 00 00 00 00 00	9.....T.....
00001158	03 00 01 00 00 00 00 00 00 68 81 04 08 00 00 00 00 03 00 02 00 00 00 00 00 88 81 04 08 00h.....
00001176	00 00 03 00 03 00 00 00 00 00 AC 81 04 08 00 00 00 00 03 00 04 00 00 00 00 00 CC 81 04 08
00001194	00 00 00 00 03 00 05 00 00 00 00 00 2C 82 04 08 00 00 00 00 03 00 06 00 00 00 00 00 80 82
000011b2	04 08 00 00 00 00 03 00 07 00 00 00 00 00 8C 82 04 08 00 00 00 00 03 00 08 00 00 00 00 00
000011d0	AC 82 04 08 00 00 00 00 03 00 09 00 00 00 00 00 B4 82 04 08 00 00 00 00 03 00 0A 00 00 00
000011ee	00 00 CC 82 04 08 00 00 00 00 03 00 0B 00 00 00 00 00 F0 82 04 08 00 00 00 00 03 00 0C 00
0000120c	00 00 00 00 30 83 04 08 00 00 00 03 00 0D 00 00 00 00 00 40 83 04 08 00 00 00 00 03 00	...0.....@.....
0000122a	0E 00 00 00 00 00 04 85 04 08 00 00 00 03 00 0F 00 00 00 00 18 85 04 08 00 00 00 00 00

Signed 8 bit:
Unsigned 8 bit:
Signed 16 bit:
Unsigned 16 bit:
☐ Show little endian decoding

Signed 32 bit:
Unsigned 32 bit:
Float 32 bit:
Float 64 bit:
☐ Show unsigned as hexadecimal

Hexadecimal: ✕
Decimal:
Octal:
Binary:
ASCII Text:

Offset: 7636 / 7635 Selection: 4288 to 7635 (3348 bytes) INS

Figure 12: Executable file with suffix

After it, we are adding suffix to both files in order to gather the program again and getting files: version1 and version2, as per using command in Figure 13. After checking by md5sum they share the same MD5 hash. So, we generate two executable files with the same MD5 Hash.

```
[04/12/19]csl@vm:~/.../task03$ ls
code.c codeExecutable md5collgen P.bin prefix Q.bin suffix version1
[04/12/19]csl@vm:~/.../task03$ cat P.bin suffix > version1
[04/12/19]csl@vm:~/.../task03$ cat Q.bin suffix > version2
[04/12/19]csl@vm:~/.../task03$ md5sum version1
aa45a6859179f28ba41aac36f4be02dc version1
[04/12/19]csl@vm:~/.../task03$ md5sum version2
aa45a6859179f28ba41aac36f4be02dc version2
[04/12/19]csl@vm:~/.../task03$
```

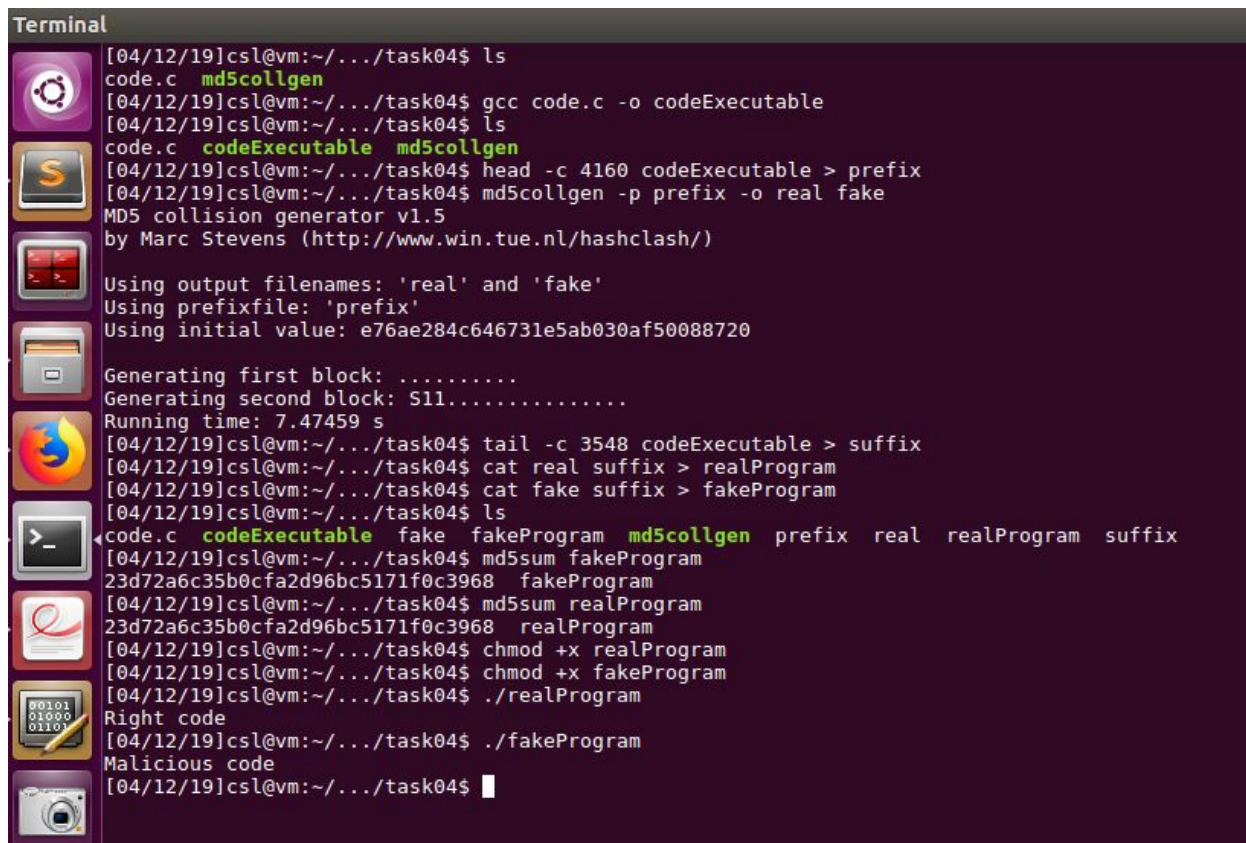
Figure 13: To generate two executable files with the same MD5 Hash

Task 4: Making the Two Programs Behave Differently

Our task is to prepare two different programs. One program will work properly and will go for certification, while the other program will be malicious, but will have the same MD5 hash as the first normal program. The source C code is provided in “Annexure 2”. After compilation we got executable file from which we need to get the prefix, which is multiple of 64 byte (which ends in the memory of array), then we left 128 byte untouched and got the rest as suffix. . For this purpose we used the following commands:

```
$ head -c 4160 codeExecutable > prefix
$ tail -c 3548 codeExecutable > suffix
```

After compilation the prefix by mdcollgen, we got two files: “real” and “fake”, which share the same MD5 hash, but had differences in the last 128 bytes, as shown in Figure 14.



```
Terminal
[04/12/19]csl@vm:~/.../task04$ ls
code.c  md5collgen
[04/12/19]csl@vm:~/.../task04$ gcc code.c -o codeExecutable
[04/12/19]csl@vm:~/.../task04$ ls
code.c  codeExecutable  md5collgen
[04/12/19]csl@vm:~/.../task04$ head -c 4160 codeExecutable > prefix
[04/12/19]csl@vm:~/.../task04$ md5collgen -p prefix -o real fake
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'real' and 'fake'
Using prefixfile: 'prefix'
Using initial value: e76ae284c646731e5ab030af50088720

Generating first block: .....
Generating second block: S11.....
Running time: 7.47459 s
[04/12/19]csl@vm:~/.../task04$ tail -c 3548 codeExecutable > suffix
[04/12/19]csl@vm:~/.../task04$ cat real suffix > realProgram
[04/12/19]csl@vm:~/.../task04$ cat fake suffix > fakeProgram
[04/12/19]csl@vm:~/.../task04$ ls
code.c  codeExecutable  fake  fakeProgram  md5collgen  prefix  real  realProgram  suffix
[04/12/19]csl@vm:~/.../task04$ md5sum fakeProgram
23d72a6c35b0cfa2d96bc5171f0c3968  fakeProgram
[04/12/19]csl@vm:~/.../task04$ md5sum realProgram
23d72a6c35b0cfa2d96bc5171f0c3968  realProgram
[04/12/19]csl@vm:~/.../task04$ chmod +x realProgram
[04/12/19]csl@vm:~/.../task04$ chmod +x fakeProgram
[04/12/19]csl@vm:~/.../task04$ ./realProgram
Right code
[04/12/19]csl@vm:~/.../task04$ ./fakeProgram
Malicious code
[04/12/19]csl@vm:~/.../task04$
```

Figure 14: Process of generating two different programs with the same MD5 Hash

In Figure 15 and 16 describe that, from the “real” program we took the generated by mdcollgen tool - 128 bytes at the end and copied it into the first 128 bytes of the second array in suffix file.

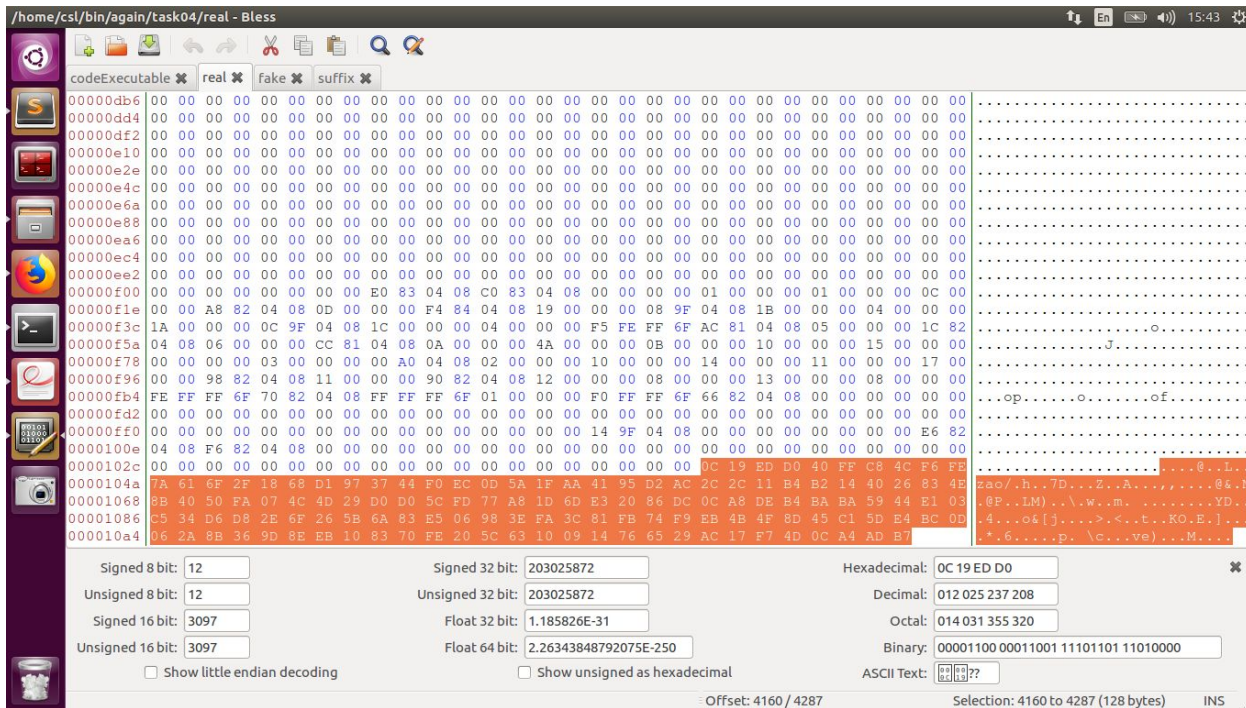


Figure 15: Copying the last 128 bytes from the real program



Figure 16: Putting the last 128 bytes from the real program into the first 128 bytes of the second array in suffix file.

After it, we are concatenating the updated suffix to both files (real, fake) in order to gather the program again and getting files: realProgram and fakeProgram. After checking by md5sum we have found that they shared the same MD5

hash, as shown in Figure 14. So, we generated two different behaviors of programs, based on the same source code, which share the same MD5 Hash.

2. Conclusion

Now a days the MD5 Hash function is a vulnerable hash function. The Lab is provide us the real life scenario based explanation, how the MD5 collision was happened, generating the same hash values for collision with different content and examined the properties of the hash function. Last but not the least, in the attacker point of view, it can also possible for us to generate a fake certificate with the help of real certificate whereas the content of the program were not the same that is one program is behaved and gave us correct output but the attacker program also can verify with the same certificate value due to generate same hash value (i.e collision occurs) though it will behave maliciously. So our observation is not to use the MD5 Hash function in any real world transactions.

Annexure 1

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

```
#include <stdio.h>
```

```
unsigned char xyz[200] =
```

[illegible]

```
int main() {
```

```
int i;
```

```
for(i = 0; i < 200; i++) {
```

```
printf("%x", xyz[i]); // Print the Array contents
```

}

```
printf("\n");
```

}

Annexure 2

Task 4: Making the Two Programs Behave Differently

```
#include <stdio.h>
```

```
unsigned char x[200] =
```

[illegible]

```
unsigned char y[200] =
```

[illegible]

```
int main() {
```

```
int f = 1;
```

```
for(int i = 0; i < 200; i++) {
```

```
if(x[i] != y[i]) // checking arrays for the equivalence
```

$$f = 0;$$

}

```
if(f == 1) // Array x's contents and Array y's contents are the same
```

```
printf("Right code\n");
```

```
else // For execute the Attacker's code
```

```
printf("Malicious code\n");
```

}

Reference

- [1] J. Black, M. Cochran, and T. Highland. “A Study of the MD5 Attacks: Insights and Improvements”. In: Proceedings of the 13th International Conference on Fast Software Encryption. FSE’06. Graz, Austria: Springer-Verlag, 2006, pp. 262–277. ISBN: 3-540-36597-4, 978-3540-36597-6. DOI: 10.1007/11799313_17. URL: http://dx.doi.org/10.1007/11799313_17 (cit. on p. 1).
- [2] M. Stevens. “On collisions for md5”. MA thesis. Eindhoven University of Technology, 2007 (cit. on p. 6).
- [3] M. Stevens, E. Bursztein, et al. The first collision for full SHA-1. Cryptology ePrint Archive, Report 2017/190. <https://eprint.iacr.org/2017/190>. 2017.