

# MD5 Collision Attack Lab

## Part 2.1 – Generating two different files with the same MD5 hash

Created a text file named **prefix.txt**, having the text “sameprefix”.

Using the following command in the terminal, **md5collgen -p prefix.txt -o out1.bin out2.bin**, it generates two output files that are different but sharing the same prefixes.

```
[09/22/20]seed@VM:~/.../MD5$ 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: 27240a9c0d5207f36628394d351415cb

Generating first block: ..
Generating second block: W.....
Running time: 1.55001 s
[09/22/20]seed@VM:~/.../MD5$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[09/22/20]seed@VM:~/.../MD5$ md5sum out1.bin
47dba89a446f552092c5e9420ca3e188 out1.bin
[09/22/20]seed@VM:~/.../MD5$ md5sum out2.bin
47dba89a446f552092c5e9420ca3e188 out2.bin
```

Using **diff** and **md5sum**, we can see that the two files differ but they have the same md5 hash values.

Opening the two files with a text editor, we see the following:

out1.bin ✖		
00000000	73 61 6D 65 70 72 65 66 69 78 0A 00 00 00 00 00 00	sameprefix.....
00000012	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	.....
00000036	00 00 00 00 00 00 00 00 00 00 27 7A 70 EA 49 57 69 7C	.....'zp.IWi
00000048	39 48 2C 9A 5A 73 C3 43 7C 9B 30 8E 02 E7 C2 C4 06 42	9H,.Zs.C .0.....B
0000005a	7C 46 2D 88 09 53 A0 AD FA E6 3A 6E 0C 6B 07 70 71 E3	F-..S....:n.k.pq.
0000006c	A4 87 1F 7F 3A 2D EF 33 7B 70 FD 38 D9 40 40 70 F9 33	....-.3{p.8.@p.3
0000007e	31 EC 6C 5F 7E C4 5D B8 B6 43 BC 18 BC E3 A1 24 D2 CA	1.l_~.]..C.....\$..
00000090	3F 5B DB 46 76 1A 7F 4B 10 C9 E6 65 CB FB 91 FE B5 CF	?[.Fv..K...e.....
000000a2	FA 2B 4C 50 86 05 C2 3A CD 39 8A 68 79 3F B1 04 51 27	..+LP....:9.hy?...Q'
000000b4	CF D1 83 E4 FA 2B 98 5D 55 8D F5 79	.....+.]U..y

out2.bin ✖		
00000000	73 61 6D 65 70 72 65 66 69 78 0A 00 00 00 00 00 00	sameprefix.....
00000012	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	.....
00000036	00 00 00 00 00 00 00 00 00 00 27 7A 70 EA 49 57 69 7C	.....'zp.IWi
00000048	39 48 2C 9A 5A 73 C3 43 7C 9B 30 8E 02 E7 C2 C4 06 42	9H,.Zs.C .0.....B
0000005a	7C 46 2D 88 09 53 A0 AD FA E6 3A 6E 0C 6B 07 70 71 E3	F-..S....:n.k.pq.
0000006c	A4 07 20 7F 3A 2D EF 33 7B 70 FD 38 D9 40 40 F0 F9 33	.. :-.3{p.8.@p.3
0000007e	31 EC 6C 5F 7E C4 5D B8 B6 43 BC 18 BC E3 A1 24 D2 CA	1.l_~.]..C.....\$..
00000090	3F 5B DB C6 76 1A 7F 4B 10 C9 E6 65 CB FB 91 FE B5 CF	?[.v..K...e.....
000000a2	FA 2B 4C 50 86 05 C2 3A CD 39 8A E8 78 3F B1 04 51 27	..+LP....:9.x?...Q'
000000b4	CF D1 83 E4 FA 2B 98 DD 55 8D F5 79	.....+.U..y

Looking at both files, there are some minor differences. The beginning is similar due to the same prefixes, and the adding of zeroes afterwards is similar, but after that there are some differences.

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

I made another file that is greater than 64 but not multiple of 64 (77 bytes), and the result is this:

77p1.bin ✖	
00000000	6F 7A 4E 4B 55 31 4F 46 4A 4E 41 51 6F 75 6E 4A 78 41 ozNKU1OFJNAQounJxA
00000012	76 65 4B 47 53 5A 38 68 39 59 42 77 67 4B 6C 70 49 59 veKGSZ8h9YBwgKlpIY
00000024	72 33 74 51 4B 75 50 56 63 56 49 33 4F 6B 45 56 77 4F r3tQKuPvCVI3OkEVwO
00000036	48 6E 66 56 51 51 4F 36 64 53 6F 7A 4E 4B 55 31 4F 46 HnfVQQO6dS0zNKU1OF
00000048	4A 4E 41 51 6F 00 00 00 00 00 00 00 00 00 00 00 00 JNAQo.....
0000005a	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000006c	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000007e	00 00 93 B0 24 70 75 CF 93 A7 7D 9D D0 C0 27 B6 65 A1 ....\$pu...}...'e.
00000090	16 A2 36 AC 69 65 8D 7A 34 B2 3C 2E 92 C0 23 31 CB E3 ..6.ie.z4.<...#1..
000000a2	BC DC DC 33 48 22 47 4A 3D 02 C4 B5 75 1C 57 DF DC A1 ...3H"GJ=...u.W...
000000b4	A7 BF FC 74 9A 03 9F 8D A7 5D 95 D9 33 6A 66 11 A2 C9 ...t.....]..3jf...

From looking at the previous prefix and this one, there will be a bunch of added zeroes if it is not a multiple of 64.

Thus, if the length of the prefix file is not a multiple of 64, there will be zero padding.

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

Made a new prefix text file, making sure it is 64 bytes by using **truncate**. Looking at the two output files, there are no zero paddings. Thus, if the prefix file is exactly 64 bytes, there is no zero padding.

64p1.bin ✖	
00000000	6F 7A 4E 4B 55 31 4F 46 4A 4E 41 51 6F 75 6E 4A 78 41 76 65 ozNKU1OFJNAQounJxAve
00000014	4B 47 53 5A 38 68 39 59 42 77 67 4B 6C 70 49 59 72 33 74 51 KGSZ8h9YBwgKlpIYr3tQ
00000028	4B 75 50 56 63 56 49 33 4F 6B 45 56 77 4F 48 6E 66 56 51 51 KuPvCVI3OkEVwOHnfVQQ
0000003c	4F 36 64 53 32 62 24 DB 2A AF AD 37 82 58 4A 69 0B 7E 59 4B O6dS2b\$.*...7.XJi.~YK
00000050	73 54 8E 58 2B 63 94 77 31 06 7B 1D 15 1E 98 BE 1A 0B B4 ED sT.X+c.wl.{.....
00000064	C2 48 DF 32 0B 64 74 19 07 58 69 11 89 99 49 0E 37 F3 54 AD .H.2.dt..Xi...I.7.T.
00000078	A4 CB 00 4A 22 D3 94 BC 2F B2 A7 C7 90 CA CF 08 5C 94 EB AC ...J"../.....\...
0000008c	45 11 93 70 0A 3E EA 77 F7 A5 3E EA CB 5A 76 37 EC 47 E2 E3 E..p.>.w...>..Zv7.G..
000000a0	DE 84 BE 94 02 11 FF 10 CA 33 74 97 CE 28 75 AC 35 E5 E9 7D .....3t..(u.5..}
000000b4	C3 22 2D 1E 49 5A F7 2E 7E 3E A8 4E ."-..IZ...~>..N

64p2.bin ✖	
00000000	6F 7A 4E 4B 55 31 4F 46 4A 4E 41 51 6F 75 6E 4A 78 41 ozNKU1OFJNAQounJxA
00000012	76 65 4B 47 53 5A 38 68 39 59 42 77 67 4B 6C 70 49 59 veKGSZ8h9YBwgKlpIY
00000024	72 33 74 51 4B 75 50 56 63 56 49 33 4F 6B 45 56 77 4F r3tQKuPvCVI3OkEVwO
00000036	48 6E 66 56 51 51 4F 36 64 53 32 62 24 DB 2A AF AD 37 HnfVQQO6dS2b\$.*...7
00000048	82 58 4A 69 0B 7E 59 4B 73 54 8E D8 2B 63 94 77 31 06 .XJi.~YKsT...+c.wl.
0000005a	7B 1D 15 1E 98 BE 1A 0B B4 ED C2 48 DF 32 0B 64 74 19 {.....H.2.dt.
0000006c	07 D8 69 11 89 99 49 0E 37 F3 54 AD A4 CB 00 CA 22 D3 ..i...I.7.T....."
0000007e	94 BC 2F B2 A7 C7 90 CA CF 08 5C 94 EB AC 45 11 93 70 ../.....\...E..p
00000090	0A 3E EA F7 F7 A5 3E EA CB 5A 76 37 EC 47 E2 E3 DE 84 .>....>..Zv7.G....
000000a2	BE 94 02 11 FF 10 CA 33 74 97 CE A8 74 AC 35 E5 E9 7D .....3t...t.5..}
000000b4	C3 22 2D 1E 49 5A F7 AE 7E 3E A8 4E ."-..IZ...~>..N

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

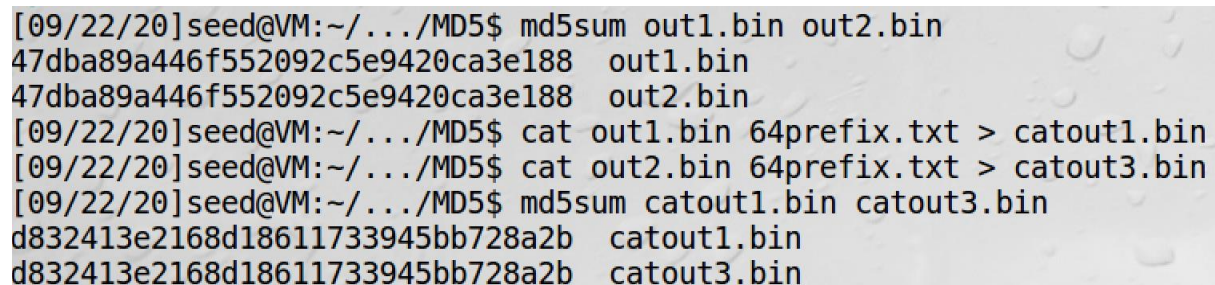
The data generated by md5collgen are not completely different for the two output files, most are the same but some bytes are different. Looking at the previous prefix file I made, the bolded are the differences:

```
6F 7A 4E 4B 55 31 4F 46 4A 4E 41 51 6F 75 6E 4A 78 41 76 65 4B 47 53 5A 38 68 39 59 42 77 67 4B 6C 70
49 59 72 33 74 51 4B 75 50 56 63 56 49 33 4F 6B 45 56 77 4F 48 6E 66 56 51 51 4F 36 64 53 32 62 24 DB
2A AF AD 37 82 58 4A 69 0B 7E 59 4B 73 54 8E 58 2B 63 94 77 31 06 7B 1D 15 1E 98 BE 1A 0B B4 ED C2
48 DF 32 0B 64 74 19 07 58 69 11 89 99 49 0E 37 F3 54 AD A4 CB 00 4A 22 D3 94 BC 2F B2 A7 C7 90 CA
CF 08 5C 94 EB AC 45 11 93 70 0A 3E EA 77 F7 A5 3E EA CB 5A 76 37 EC 47 E2 E3 DE 84 BE 94 02 11 FF 10
CA 33 74 97 CE 28 75 AC 35 E5 E9 7D C3 22 2D 1E 49 5A F7 2E 7E 3E A8 4E
```

---

```
6F 7A 4E 4B 55 31 4F 46 4A 4E 41 51 6F 75 6E 4A 78 41 76 65 4B 47 53 5A 38 68 39 59 42 77 67 4B 6C 70
49 59 72 33 74 51 4B 75 50 56 63 56 49 33 4F 6B 45 56 77 4F 48 6E 66 56 51 51 4F 36 64 53 32 62 24 DB
2A AF AD 37 82 58 4A 69 0B 7E 59 4B 73 54 8E D8 2B 63 94 77 31 06 7B 1D 15 1E 98 BE 1A 0B B4 ED C2
48 DF 32 0B 64 74 19 07 D8 69 11 89 99 49 0E 37 F3 54 AD A4 CB 00 CA 22 D3 94 BC 2F B2 A7 C7 90 CA
CF 08 5C 94 EB AC 45 11 93 70 0A 3E EA F7 F7 A5 3E EA CB 5A 76 37 EC 47 E2 E3 DE 84 BE 94 02 11 FF 10
CA 33 74 97 CE A8 74 AC 35 E5 E9 7D C3 22 2D 1E 49 5A F7 AE 7E 3E A8 4E
```

## Part 2.2 – Understanding MD5's Property



```
[09/22/20]seed@VM:~/.../MD5$ md5sum out1.bin out2.bin
47dba89a446f552092c5e9420ca3e188 out1.bin
47dba89a446f552092c5e9420ca3e188 out2.bin
[09/22/20]seed@VM:~/.../MD5$ cat out1.bin 64prefix.txt > catout1.bin
[09/22/20]seed@VM:~/.../MD5$ cat out2.bin 64prefix.txt > catout3.bin
[09/22/20]seed@VM:~/.../MD5$ md5sum catout1.bin catout3.bin
d832413e2168d18611733945bb728a2b catout1.bin
d832413e2168d18611733945bb728a2b catout3.bin
```

The screenshot above shows that, with two files having the same MD5 hashes, and adding the same suffix (concatenating both files with the same text file) to both of them will result in two outputs that have the same MD5 hash value. This value can be found using **md5sum**, and before and after the concatenation, the two files have same hash values. Thus, given two inputs M and N, if  $MD5(N)=MD5(N)$ , then for any input T,  $MD5(M || T) = MD5(N || T)$ , where  $||$  represents concatenation.

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

Compiling the above code and using **bleess** to view the binary executable file, we get this:

We see the “A”s from our code, and before that we can set as the prefix, and after all the “A”s, is the suffix. Due to how MD5’s property, as long as the same suffix is appended to both files, the MD5 value will still be the same for both files, even if the data in the middle are different.

I will then divide the executable file into 3 parts, prefix, a 128-byte region, and a suffix. The prefix is the data before 4160, which is fine because it is a multiple of 64. Then 128 bytes after, and after that is the suffix.

```
[09/23/20]seed@VM:~/.../MD5$ head -c 4160 task3 > task3prefix
[09/23/20]seed@VM:~/.../MD5$ tail -c 4289 task3 > task3suffix
```



Generating two files that have the same MD5 hash value and 128 bytes after the prefix, we get that  $\text{MD5}(\text{prefix} || p) = \text{MD5}(\text{prefix} || q)$ :

```
[09/23/20]seed@VM:~/.../MD5$ md5collgen -p task3prefix -o p1 q1
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'p1' and 'q1'
Using prefixfile: 'task3prefix'
Using initial value: 348b2aec40d504b882c85286c7ba71c8

Generating first block: .....
Generating second block: S01.....
Running time: 11.1261 s
[09/23/20]seed@VM:~/.../MD5$ md5sum p1 q1
9864e59a4c930f7f3cc2f5cec4e60fcf p1
9864e59a4c930f7f3cc2f5cec4e60fcf q1
[09/23/20]seed@VM:~/.../MD5$ tail -c 128 p1 > p
[09/23/20]seed@VM:~/.../MD5$ tail -c 128 q1 > q
```

Now, we can concatenate p and q with the prefix and suffix, and get two executable files that have the same MD5 hash value but behave differently: MD5(prefix || p || suffix) = MD5(prefix || q || suffix)

[illegible]

Thus, we now know and have two executable files with the same MD5 hash but what they print out are different.

Made the following c program, that has two arrays, X and Y, that are the same. If they are the same, then it will print out "Good code", if not, then it will print out "Bad code".

Doing a similar process of the previous task, I get the prefix of the executable, right before the data of the first array and also the suffix afterwards. But, we also want to change the array content of the second array so we want the data between them and another suffix afterwards.

Now the following could be done:

Generating two files again that have the same MD5 hash value and getting the 128 bytes after the prefix:

```
[09/23/20]seed@VM:~/.../MD5$ md5collgen -p task4prefix -o 4good 4bad
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: '4good' and '4bad'
Using prefixfile: 'task4prefix'
Using initial value: 6d3578a2e26c59f9395f61b0cf759daf

Generating first block: .....
Generating second block: S10.....
Running time: 6.27343 s
[09/23/20]seed@VM:~/.../MD5$ tail -c 128 4good > good
[09/23/20]seed@VM:~/.../MD5$ tail -c 128 4bad > bad
```

“good” and “bad” will be used to change the content of the arrays. Inserting “good” in both arrays, results in good code, while inserting “bad” and then “good” will result in malicious code because it sees that the arrays are not the same. Thus, concatenating them with the prefix and suffixes, we get two executable programs with the same MD5 hash but one executes malicious code and other is good code. The process can be written as an equation similar to previous tasks:

$$\text{MD5}(\text{prefix} || \text{good} || \text{suffix1} || \text{good} || \text{suffix2}) = \text{MD5}(\text{prefix} || \text{bad} || \text{suffix1} || \text{good} || \text{suffix2})$$

Because of the insertions, the program sees that the arrays are no longer the same, so it executes the malicious code. However, because of MD5’s property, the MD5 hash stays the same for both programs even though the array content of the other program is different.

```
[09/23/20]seed@VM:~/.../MD5$ cat task4prefix good task4suffix1 good task4suffix2 > task4good
[09/23/20]seed@VM:~/.../MD5$ cat task4prefix bad task4suffix1 good task4suffix2 > task4bad
[09/23/20]seed@VM:~/.../MD5$ md5sum task4good task4bad
545ffb970ba6392be8761b5df8d163ac task4good
545ffb970ba6392be8761b5df8d163ac task4bad
[09/23/20]seed@VM:~/.../MD5$ chmod +x task4good task4bad
[09/23/20]seed@VM:~/.../MD5$ task4good
Good code
[09/23/20]seed@VM:~/.../MD5$ task4bad
Bad Code
```

Thus, both programs have same MD5 hash value. The first version, “task4good”, have the contents of the X and Y array the same, so it runs good code. However, the second version, “task4bad”, the contents of the array are different which makes the program run malicious code.

## Final Thoughts

This was a very interesting lab, as I have never known about MD5 before this class, and also how easy it is to essentially to “mask” malicious code as good due to having the same MD5 hash value of a certified benign program. Also, this was my first time really using Ubuntu and VirtualBox, so it was a nice introduction to those as well.