

Lab 3 - Documentation

April 4, 2020

1 Lab 3 - MD5 Collision Attack Lab

1.1 ## By: Laura Perda

1.2 Task 1 - Generating Two Different Files with the Same MD5 Hash

In this task, I will generate two different files with the same MD5 hash values. The beginning parts of these files are required to be the same (share the same prefix), which can be accomplished using the md5collgen program. I created a text file named ‘prefix.txt’ that contains the word ‘welcome’. The picture below shows the following md5collgen command. When using the recommended ‘diff’ command on ‘out1.bin’ and ‘out2.bin’, it shows that the two files are different, but when comparing the hash value through the ‘md5sum’ command it shows the same.



The screenshot shows a desktop environment with a terminal window open. The terminal window has a title bar labeled "Terminal". Inside the terminal, the user runs the command "md5collgen -p prefix.txt -o out1.bin out2.bin". The output of the command is displayed, showing the generation of two MD5 collisions. The user then runs "diff out1.bin out2.bin" to verify that the files differ only in their random padding. Finally, the user runs "xxd out1.bin" to show the hex dump of the file, which includes the prefix "welcome...." followed by random data.

```
[04/04/20]seed@VM:~/.../Task 1$ 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: ffa382d7e6c48a1935e32049c9f5c83b

Generating first block: .....
Generating second block: S00.....
Running time: 6.53332 s
[04/04/20]seed@VM:~/.../Task 1$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[04/04/20]seed@VM:~/.../Task 1$ md5sum out?.bin
23c19dddea589407a008c93c6c3ea7ce  out1.bin
23c19dddea589407a008c93c6c3ea7ce  out2.bin
[04/04/20]seed@VM:~/.../Task 1$ xxd out1.bin
00000000: 7765 6c63 6f6d 650a 0000 0000 0000 0000 welcome.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 0173 c928 1a58 26d7 49e4 f0af 50fd 41b2 .s.(.X&.I...P.A.
00000050: 8297 a86d 0917 41c9 4a5e 8292 477c ab22 ...m..A.J^..G|."
00000060: 6adb c8ea f2c5 f980 0b68 d39c 0234 d405 j.....h...4..
00000070: f9ad 270d 56d3 7c7a dad0 7181 5013 f0dd ..'.V.|z..q.P...
00000080: 36f6 1501 e824 8fe0 a50d 6107 a9fd 8c50 6....$....a....P
00000090: ce59 210f 22bf 4055 c9fb 14c5 d561 5db2 .Y!.."@U.....a].
000000a0: b341 1fdf a5bb 7154 9947 cccf cbb9 4fac .A..V.qT.G....O.
000000b0: 951b dd19 ac7b 831c 2f6d 004a edbb 956b .....{/..m.J...k
```

Q1: If the length of your prefix file is not a multiple of 64, what is going to happen?

A: In the picture above, I also decided to check the hex dump of 'out1.bin' which show it was padded with zeroes to bring it up to 64 bytes and then followed by random data being appended to that. The hex dump for 'out2.bin' has the same 64 initial bytes, but differs slightly when it comes to the random data appended at the end of the file.

Q2: Create a prefix file with exactly 64 bytes & run the collision tool again to see what happens.

A: For this question, I truncated the prefix.txt file to be the size of 64 bytes thru the 'truncate' command. The results shown in the pictures below show that the outcomes were not padded this time. The only change is the type of random data appended to the prefix.

```

Terminal
[04/04/20]seed@VM:~/.../Task 1$ truncate -s 64 prefix.txt
[04/04/20]seed@VM:~/.../Task 1$ xxd prefix.txt
00000000: 7765 6c63 6f6d 650a 0000 0000 0000 0000 welcome.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
[04/04/20]seed@VM:~/.../Task 1$ xxd prefix_before.txt
00000000: 7765 6c63 6f6d 650a welcome.

Terminal
[04/04/20]seed@VM:~/.../Task 1$ 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: ffa382d7e6c48a1935e32049c9f5c83b

Generating first block: ...
Generating second block: W.
Running time: 3.25378 s
[04/04/20]seed@VM:~/.../Task 1$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[04/04/20]seed@VM:~/.../Task 1$ md5sum out?.bin
8bac26db86cf777724f24ce5f3f5675c out1.bin
8bac26db86cf777724f24ce5f3f5675c out2.bin
[04/04/20]seed@VM:~/.../Task 1$ xxd out1.bin
00000000: 7765 6c63 6f6d 650a 0000 0000 0000 0000 welcome.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 93bb 7723 347a a053 6b7c 9fee c101 82ba ..w#4z.Sk|....
00000050: 868c 83f6 aae2 80a7 3c1e 7a29 0944 b92b .....<.z).D.+
00000060: 5ecd 08df 3c6a 13df cbc3 645c c5c9 9dc1 ^...<j....d\....
00000070: 9ea5 7a08 e2ce 3ee9 d75e 083e d633 fcf9 ..z...>.^>.3..
00000080: e744 8372 7667 5ecb ffb6 e485 54e4 aa72 .D.rvg^.....T..r
00000090: 7c54 85a7 ee54 b7eb 0e65 a9d6 cc14 1ebe |T...T...e.....
000000a0: 85f7 bebb cff4 1429 164b 0d72 8374 01ff .....).K.r.t..
000000b0: c94c e824 72d9 c37f f56e 40f3 c988 5e59 .L.$r....n@....^Y

```

Q3: Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.

A: The picture below shows that there are a total of 6 different bytes from the random appended data that are different in the hex dumps.

```

Terminal
[04/04/20]seed@VM:~/.../Task 1$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[04/04/20]seed@VM:~/.../Task 1$ md5sum out?.bin
8bac26db86cf777724f24ce5f3f5675c  out1.bin
8bac26db86cf777724f24ce5f3f5675c  out2.bin
[04/04/20]seed@VM:~/.../Task 1$ xxd out1.bin
00000000: 7765 6c63 6f6d 650a 0000 0000 0000 0000 welcome.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 93bb 7723 347a a053 6b7c 9fee c101 82ba .w#4z.Sk|.....
00000050: 868c 83f6 aae2 80a7 3c1e 7a29 0944 b92b .v.....<.z).D.+
00000060: 5ecd 08df 3c6a 13df cbc3 645c c5c9 9dc1 ^...<j....d\.\n...
00000070: 9ea5 7a08 e2ce 3ee9 d75e 083e d633 fcf9 ..z...>..^..>..3..
00000080: e744 8372 7667 5ecb ffb6 e485 54e4 aa72 .D.rvg^....T..r
00000090: 7c54 85a7 ee54 b7eb 0e65 a9d6 cc14 1ebe |T.O.T...e.....
000000a0: 85f7 bebbcff4 1429 164b 0d72 8374 01ff .....).K.r.t...
000000b0: c94c e824 72d9 c37f f56e 40f3 c988 5e59 .L.$r....n@O..^Y
[04/04/20]seed@VM:~/.../Task 1$ xxd out2.bin
00000000: 7765 6c63 6f6d 650a 0000 0000 0000 0000 welcome.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 93bb 7723 347a a053 6b7c 9fee c101 82ba .w#4z.Sk|.....
00000050: 868c 8376 aae2 80a7 3c1e 7a29 0944 b92b .v.....<.z).D.+
00000060: 5ecd 08df 3c6a 13df cbc3 645c c549 9ec1 ^...<j....d\.\I...
00000070: 9ea5 7a08 e2ce 3ee9 d75e 08be d633 fcf9 ..z...>..^..>..3..
00000080: e744 8372 7667 5ecb ffb6 e485 54e4 aa72 .D.rvg^....T..r
00000090: 7c54 8527 ee54 b7eb 0e65 a9d6 cc14 1ebe |T.O.T...e.....
000000a0: 85f7 bebbcff4 1429 164b 0d72 83f4 00ff .....).K.r...
000000b0: c94c e824 72d9 c37f f56e 4073 c988 5e59 .L.$r....n@s..^Y
[04/04/20]seed@VM:~/.../Task 1$ █

```

1.3 Task 2 - Understanding MD5's Property

In this task, we will try to understand the MD5 algorithm and understand the importance of these properties. From what we have seen, MD5 divides the input data into blocks of 64 bytes and then computes the hash over these blocks. The core of the MD5 alg. is the compression function, which takes 2 inputs, a 64 byte block and the outcome of the previous iteration. It then produces a 128 bit IHV, which stands for “Intermediate Hash Value”, which is fed to the next iteration.

We have to design an experiment on the simple fact that if inputs M and N have the same hash, adding the suffix T to them will result in two outputs that have the same hash value. This was already mostly accomplished in Task 1. Thus, the steps in Task 1 will be repeated and then show an extra step in which we append some suffix and redo the hash.

Below shows that the M & N have the same hash value.

```
[04/04/20]seed@VM:~/.../Task 2$ 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: ffa382d7e6c48a1935e32049c9f5c83b

Generating first block: ...
Generating second block: S11..... .
Running time: 5.34753 s
[04/04/20]seed@VM:~/.../Task 2$ md5sum out?.bin
19f650bff0a8242ed1687f7c689e7fd3  out1.bin
19f650bff0a8242ed1687f7c689e7fd3  out2.bin
[04/04/20]seed@VM:~/.../Task 2$ █
```

The picture below shows that after appending the prefix.txt file to the end of both .bin files, the ‘md5sum’ command shows that the hash value remains the same.

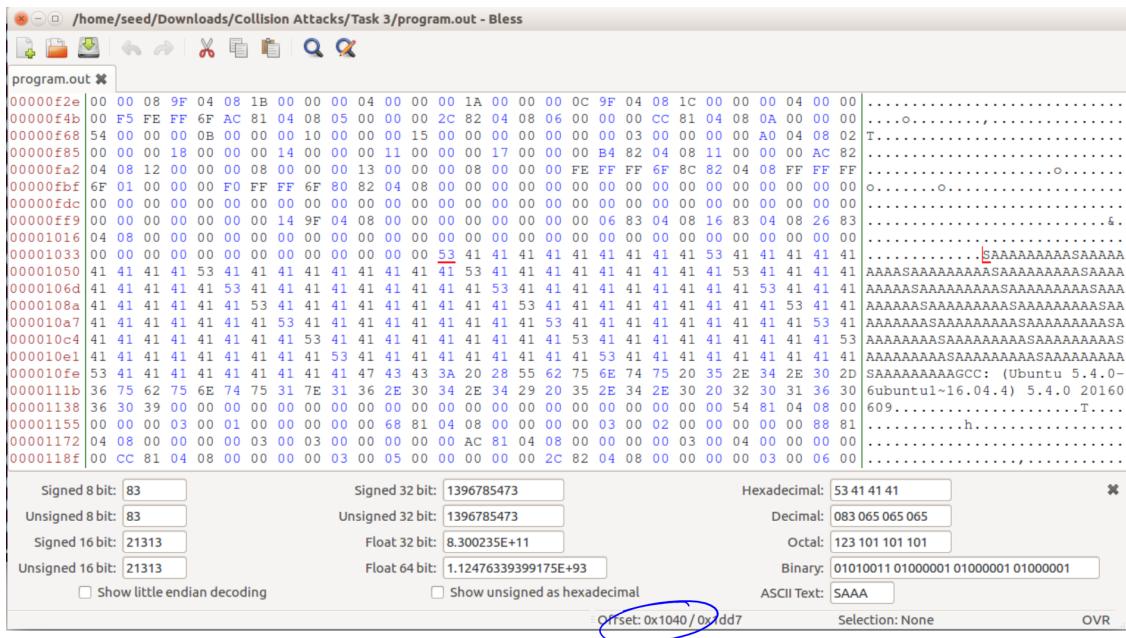
```
[04/04/20]seed@VM:~/.../Task 2$ echo prefix.txt >> out1.bin
[04/04/20]seed@VM:~/.../Task 2$ echo prefix.txt >> out2.bin
[04/04/20]seed@VM:~/.../Task 2$ md5sum out?.bin
244897207a10a7d6d31af80c8f873c67  out1.bin
244897207a10a7d6d31af80c8f873c67  out2.bin
[04/04/20]seed@VM:~/.../Task 2$ █
```

As a result, we can say the MD5 property holds.

1.4 Task 3 - Generating Two Executable Files with the same MD5 Hash

In this task, we are given a program that we must modify with some array and so that we can divide the executable file into three parts: prefix, 128 bit region, and suffix. Length of prefix should be a multiple of 64. The point of this is to generate two different executables with the same MD5 hash value and check if their hash value remains the same after appending some suffix.

Compiling the provided program and checking the location of the beginning of the array of values shows that it is at an offset of 0x1040, which converts to 4160 in decimal (divisible by 64). Since I would like to see a bit of the array within the prefix, I chose to add an extra 64 byte block, so the cut off would be 4224 bytes. The picture below demonstrates a bit of this.

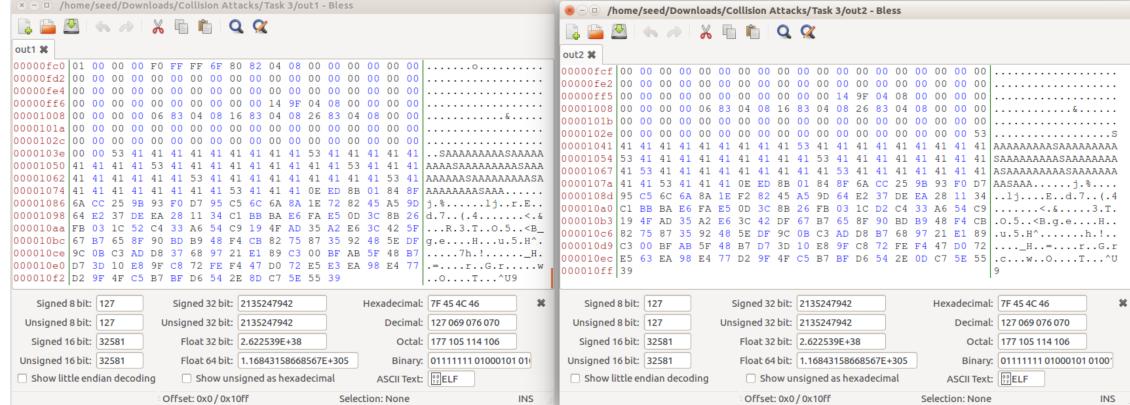


The picture below shows that after using the “md5collgen” to create the same hash but with different endings (P & Q).

```
[04/04/20]seed@VM:~/.../Task 3$ 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: acb20c63240e7d1398448abb2d4f91ff

Generating first block: .
Generating second block: $10.....
Running time: 4.62089 s
[04/04/20]seed@VM:~/.../Task 3$ md5sum out1 out2
17f0f6e47b6ala7242c9d26f6698eed out1
17f0f6e47b6ala7242c9d26f6698eed out2
[04/04/20]seed@VM:~/.../Task 3$
```



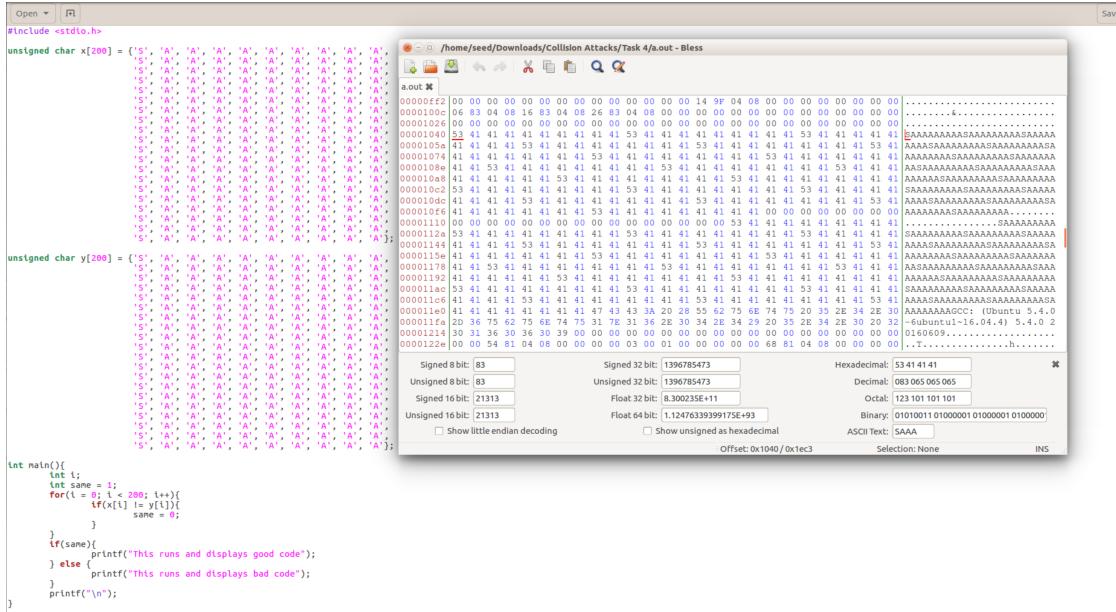
Now, we can append the tail of the prefix to both files and check whether the same hash value is true when they receive the same suffixes. Creating the files into executable files does not change the outcome.

```
[04/04/20]seed@VM:~/.../Task 3$ tail -c +4353 prefix.txt > same
[04/04/20]seed@VM:~/.../Task 3$ cat same >> out1
[04/04/20]seed@VM:~/.../Task 3$ cat same >> out2
[04/04/20]seed@VM:~/.../Task 3$ md5sum out1 out2
17f0f6e47b6a1a7242c9d26f6698eead  out1
17f0f6e47b6a1a7242c9d26f6698eead  out2
[04/04/20]seed@VM:~/.../Task 3$
```

1.5 Task 4 - Making the 2 Programs Behave Differently

In the previous task, we manage to make two programs to have the same MD5 hash, but their behaviors were different. Their differences were in the print out, but they still execute the same sequence of instructions. In this task, we want to create two different programs to have the same MD5 hash value.

Same case as before but with a different source code. The two arrays are the same, so it should execute the first print statement. Again, I while the array starts at x1040, I will like to see a bit of the array, so x1080 is fine for the prefix.

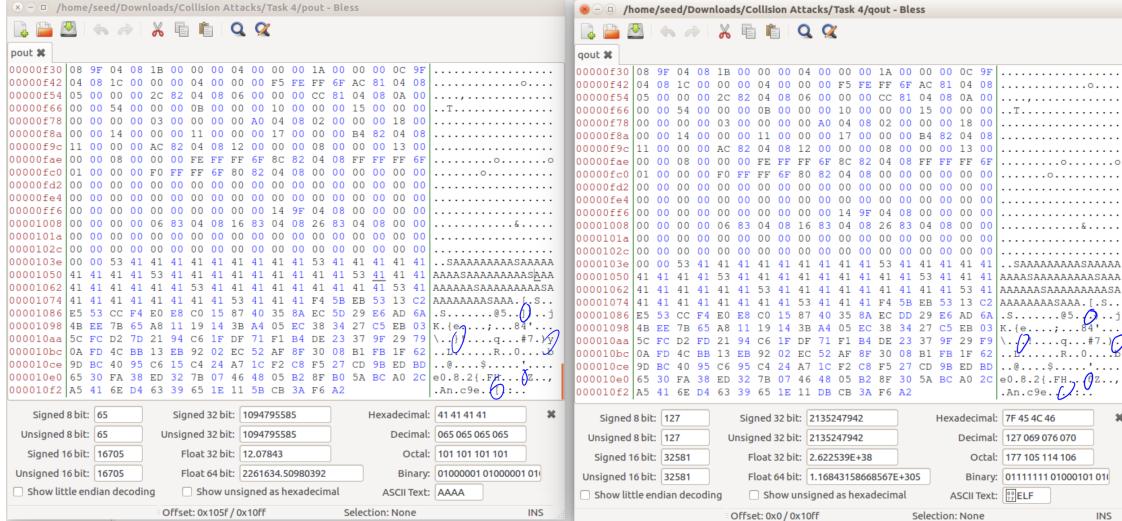


After using the “md5collgen” command to create the files pout & qout, I checked to see if the hash values were the same by using ‘md5sum’. Both result being the same despite being a slight difference in the middle section as shown through bless hex. Note that the last 8 elements of the first array are missing, so we need to grab those and append to the two created files.

```
[04/04/20]seed@VM:~/.../Task 4$ head -c 4224 a.out > prefix
[04/04/20]seed@VM:~/.../Task 4$ md5collgen -p prefix -o pout qout
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'pout' and 'qout'
Using prefixfile: 'prefix'
Using initial value: df03be71d83ec3d9806700fe4dae2ecf

Generating first block: .....
Generating second block: W.....
Running time: 13.8796 s
[04/04/20]seed@VM:~/.../Task 4$ md5sum pout qout
3a36bb9044c7b82524ed43b0a323647f pout
3a36bb9044c7b82524ed43b0a323647f qout
[04/04/20]seed@VM:~/.../Task 4$
```



After I had checked the hash values, I had placed the rest of a.out file into a file named ‘test’. This is done in order to grab the missing 8 bytes of the first array. Using this, I concatenated the ‘missing’ file to ‘pout’ & ‘qout’ to create a complete version to both.

```
[04/04/20]seed@VM:~/.../Task 4$ tail +4353 a.out > test
tail: cannot open '+4353' for reading: No such file or directory
[04/04/20]seed@VM:~/.../Task 4$ tail -c +4353 a.out > test
[04/04/20]seed@VM:~/.../Task 4$ head -c 8 test > missing
[04/04/20]seed@VM:~/.../Task 4$ cat pout missing > poutcomplete
[04/04/20]seed@VM:~/.../Task 4$ cat qout missing > qoutcomplete
[04/04/20]seed@VM:~/.../Task 4$ tail -c +9 test > suffix
```

Then I needed to add the bytes between the end of the 1st array & the beginning of the 2nd array, which I named ‘tillNext’. I stored the beginning of the second array in ‘suffix’ into ‘test’, so that I can add them to the complete versions of ‘pout’ & ‘qout’.

```
[04/04/20]seed@VM:~/.../Task 4$ tail -c +25 suffix > test
[04/04/20]seed@VM:~/.../Task 4$ head -c 24 suffix > tillNext
[04/04/20]seed@VM:~/.../Task 4$ cat poutcomplete tillNext > f1tillNext
[04/04/20]seed@VM:~/.../Task 4$ cat qoutcomplete tillNext > f2tillNext
```

At this point, the 2 programs are different executables but only have the content up to the 2nd array. To make these two files do different outcomes (good vs bad), the content of the 2nd array should be equal to the generated array. So, I had to place the bytes after 2nd array from ‘test’ to ‘suffix’. Then I copied the 1st array from ‘poutcomplete’ to ‘parray’, which was appended to the previously made files in the picture above. From this, I was able to create the 2 good & bad executables.

```
[04/04/20]seed@VM:~/.../Task 4$ tail -c +201 test > suffix
[04/04/20]seed@VM:~/.../Task 4$ tail -c +4161 poutcomplete > parray
[04/04/20]seed@VM:~/.../Task 4$ cat f1tillNext parray suffix > firstEx
[04/04/20]seed@VM:~/.../Task 4$ cat f2tillNext parray suffix > secEx
```

Now that I have the 2 executables, it’s time to test whether they still produce the same hash value despite being different. I made them executable, checked their hash values, and executed both just to make sure the outcome was correct. It was interesting to see that so much work was necessary in order to exploit the MD5 algorithm.

```
[04/04/20]seed@VM:~/.../Task 4$ chmod +x firstEx
[04/04/20]seed@VM:~/.../Task 4$ chmod +x SecEx
chmod: cannot access 'SecEx': No such file or directory
[04/04/20]seed@VM:~/.../Task 4$ chmod +x secEx
[04/04/20]seed@VM:~/.../Task 4$ md5sum firstEx secEx
20c911aa6c0c9d876801c46251ab41ac  firstEx
20c911aa6c0c9d876801c46251ab41ac  secEx
[04/04/20]seed@VM:~/.../Task 4$ ./firstEx
This runs and displays good code
[04/04/20]seed@VM:~/.../Task 4$ ./secEx
This runs and displays bad code
[04/04/20]seed@VM:~/.../Task 4$ █
```

[]: