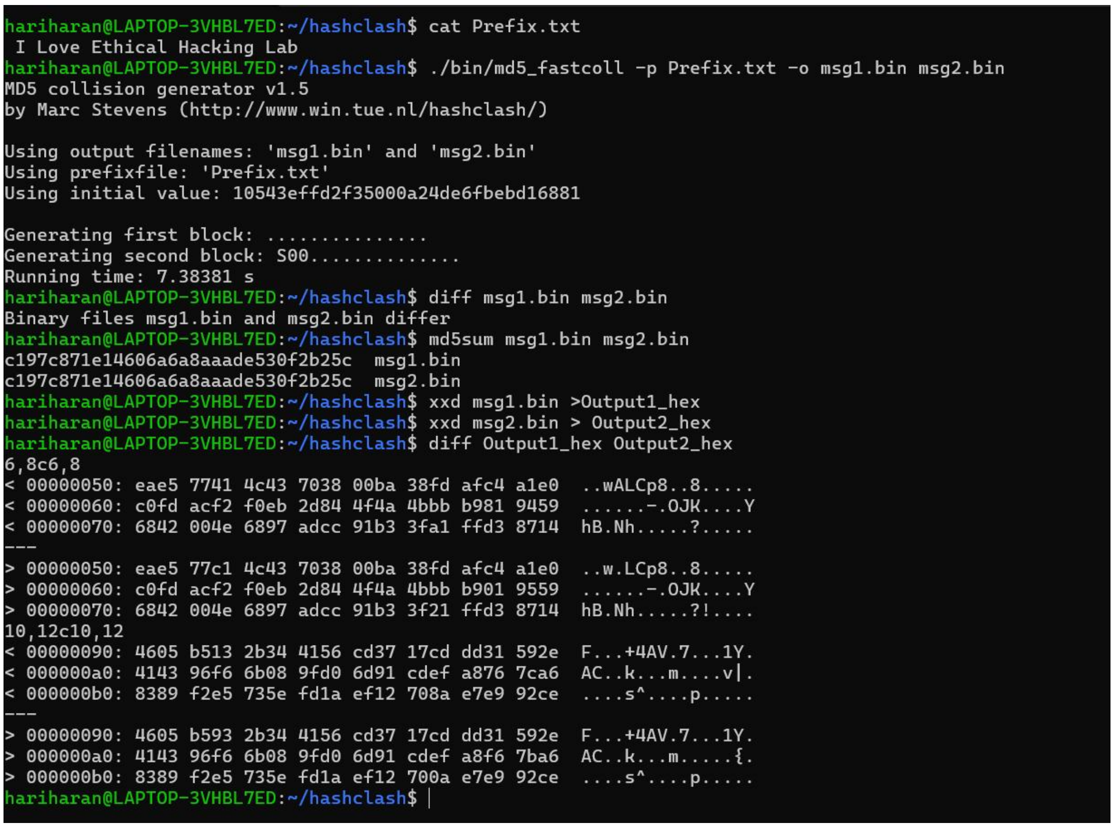
Cyber Security Lab (Ethical Hacking) MD5Collision Attacks

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



```
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ cat Prefix.txt
 I Love Ethical Hacking Lab
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ ./bin/md5_fastcoll -p Prefix.txt -o msg1.bin msg2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'Prefix.txt'
Using initial value: 10543effd2f35000a24de6fbebd16881

Generating first block: ...............
Generating second block: S00.............
Running time: 7.38381 s
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ diff msg1.bin msg2.bin
Binary files msg1.bin and msg2.bin differ
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ md5sum msg1.bin msg2.bin
c197c871e14606a6a8aaade530f2b25c  msg1.bin
c197c871e14606a6a8aaade530f2b25c  msg2.bin
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg1.bin >Output1_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg2.bin > Output2_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ diff Output1_hex Output2_hex
6,8c6,8
< 00000050: eae5 7741 4c43 7038 00ba 38fd afc4 a1e0  ..wALCp8..8.....
< 00000060: c0fd acf2 f0eb 2d84 4f4a 4bbb b981 9459  ......-.OJK....Y
< 00000070: 6842 004e 6897 adcc 91b3 3fa1 ffd3 8714  hB.Nh.....?.....
---
> 00000050: eae5 77c1 4c43 7038 00ba 38fd afc4 a1e0  ..w.LCp8..8.....
> 00000060: c0fd acf2 f0eb 2d84 4f4a 4bbb b901 9559  ......-.OJK....Y
> 00000070: 6842 004e 6897 adcc 91b3 3f21 ffd3 8714  hB.Nh.....?!....
10,12c10,12
< 00000090: 4605 b513 2b34 4156 cd37 17cd dd31 592e  F...+4AV.7...1Y.
< 000000a0: 4143 96f6 6b08 9fd0 6d91 cdef a876 7ca6  AC..k...m....v|.
< 000000b0: 8389 f2e5 735e fd1a ef12 708a e7e9 92ce  ....s^....p.....
---
> 00000090: 4605 b593 2b34 4156 cd37 17cd dd31 592e  F...+4AV.7...1Y.
> 000000a0: 4143 96f6 6b08 9fd0 6d91 cdef a8f6 7ba6  AC..k...m.....{.
> 000000b0: 8389 f2e5 735e fd1a ef12 700a e7e9 92ce  ....s^....p.....
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ |
```

This attack creates two files with the same data but with different padding. The two files share the same prefix i.e., the remaining part of the two files are not the same. They are generated by the collision tool “fastcoll”. This tool generates two different files with the same prefix and same md5 checksums.

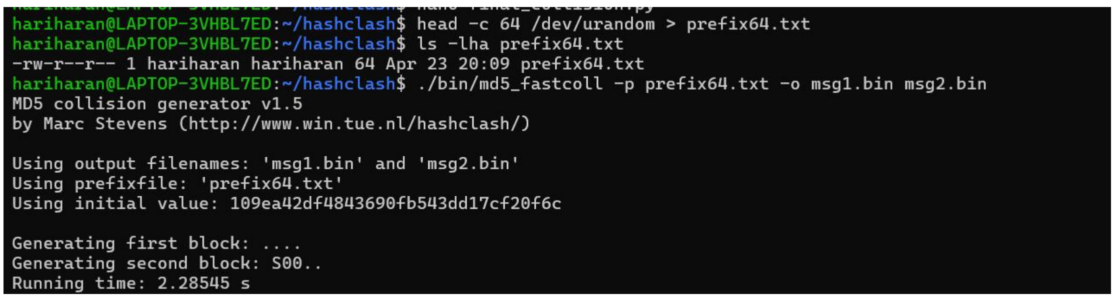
a) What would happen if the length of your prefix file is not multiple of 64? Zeros will be appended/padded to the rest of the text.

When the byte length of the prefix file is not a multiple of 64 bytes (MD5 processes data in 64-byte blocks), the tool will **automatically add padding** to align the data. This ensures that the total length of the message (prefix + collision block) is a multiple of 64 bytes, which is required for MD5 to process it correctly.

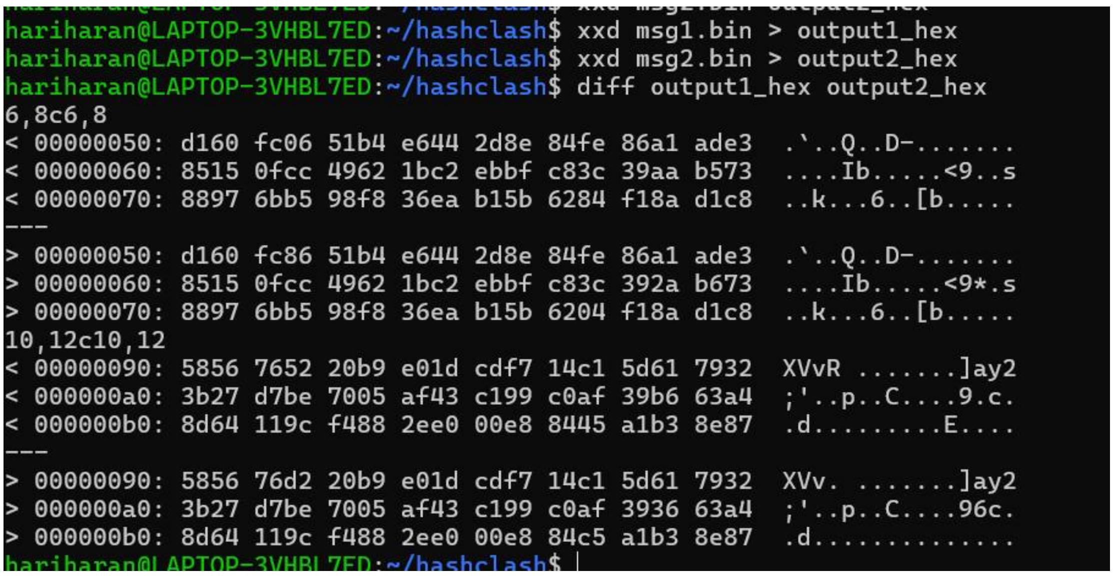
For example, if the prefix is 50 bytes, the tool will pad it with extra bytes so that the total length is a multiple of 64 before the MD5 collision generation begins. The padding is invisible in the final collision but ensures correct processing.

This is because the majority of cryptographic functions, such as MD5, process data in fixed-sized blocks. In MD5, a block size of 64 bytes is used.MD5 processes 512-bit (64-byte) blocks, and if the file is not of this block size, padding is performed.

b) Create a prefix file that has exactly a length of 64 bytes and run again the collision generation described above. Describe what happens.

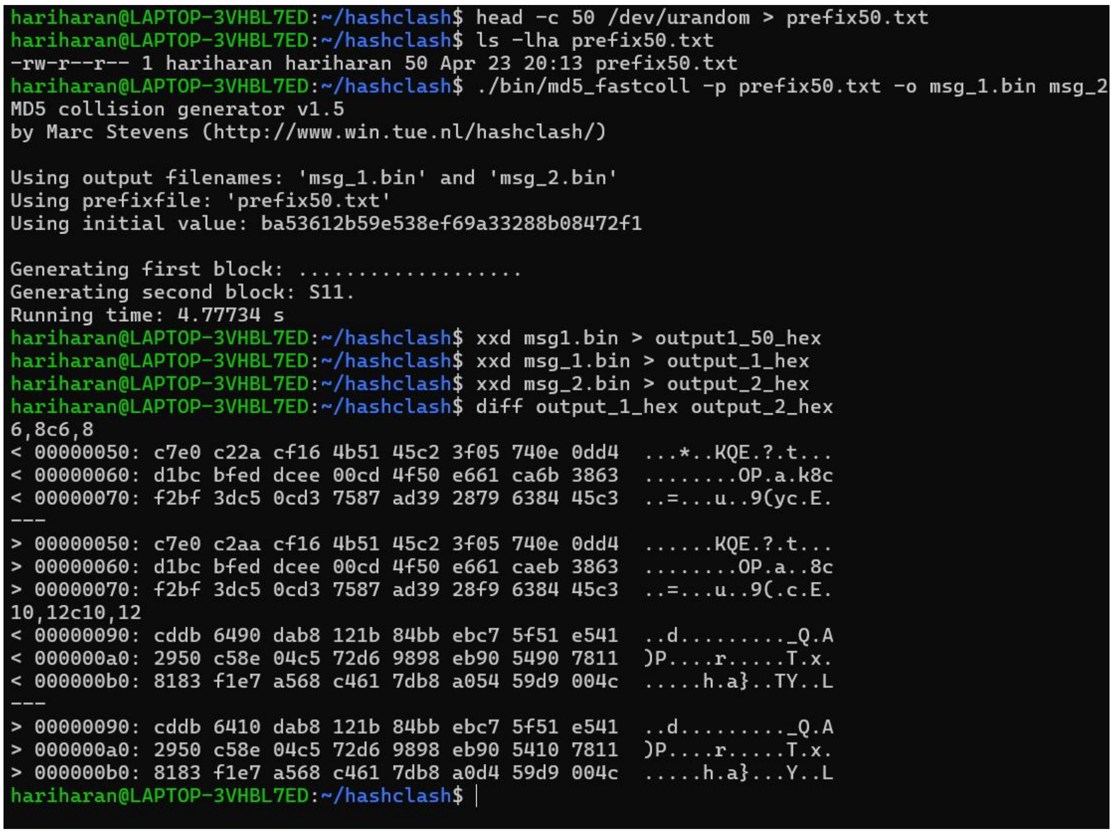


```
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ head -c 64 /dev/urandom > prefix64.txt
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ ls -lha prefix64.txt
-rw-r--r-- 1 hariharan hariharan 64 Apr 23 20:09 prefix64.txt
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ ./bin/md5_fastcoll -p prefix64.txt -o msg1.bin msg2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'prefix64.txt'
Using initial value: 109ea42df4843690fb543dd17cf20f6c

Generating first block: ....
Generating second block: S00..
Running time: 2.28545 s
```



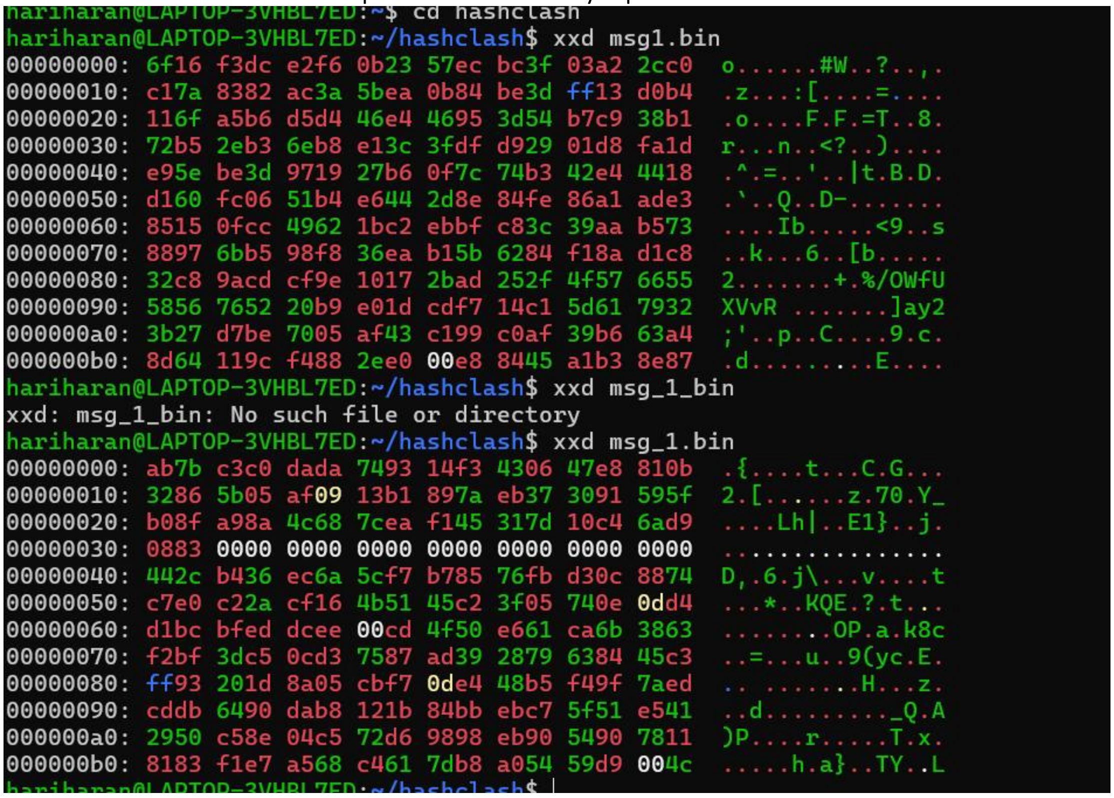
```
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg1.bin > output1_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg2.bin > output2_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ diff output1_hex output2_hex
6,8c6,8
< 00000050: d160 fc06 51b4 e644 2d8e 84fe 86a1 ade3  .`..Q..D-.......
< 00000060: 8515 0fcc 4962 1bc2 ebbf c83c 39aa b573  ....Ib.....<9..s
< 00000070: 8897 6bb5 98f8 36ea b15b 6284 f18a d1c8  ..k...6..[b.....
---
> 00000050: d160 fc86 51b4 e644 2d8e 84fe 86a1 ade3  .`..Q..D-.......
> 00000060: 8515 0fcc 4962 1bc2 ebbf c83c 392a b673  ....Ib.....<9*.s
> 00000070: 8897 6bb5 98f8 36ea b15b 6204 f18a d1c8  ..k...6..[b.....
10,12c10,12
< 00000090: 5856 7652 20b9 e01d cdf7 14c1 5d61 7932  XVvR .......]ay2
< 000000a0: 3b27 d7be 7005 af43 c199 c0af 39b6 63a4  ;'..p..C....9.c.
< 000000b0: 8d64 119c f488 2ee0 00e8 8445 a1b3 8e87  .d.........E....
---
> 00000090: 5856 76d2 20b9 e01d cdf7 14c1 5d61 7932  XVv. .......]ay2
> 000000a0: 3b27 d7be 7005 af43 c199 c0af 3936 63a4  ;'..p..C....96c.
> 000000b0: 8d64 119c f488 2ee0 00e8 84c5 a1b3 8e87  .d..............
hariharan@LAPTOP-3VHBL7ED:~/hashclash$
```

When I create a prefix file of the total size of 64 bytes and then run the MD 5 collision generation the file now exactly fits into a 64-byte block. No padding needed: Since the file is already 64 bytes, it exactly fits into one block.MD5 collision generation , will generate two different inputs that have the same hash. The prefix file here becomes part of this process.



```
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ head -c 50 /dev/urandom > prefix50.txt
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ ls -lha prefix50.txt
-rw-r--r-- 1 hariharan hariharan 50 Apr 23 20:13 prefix50.txt
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ ./bin/md5_fastcoll -p prefix50.txt -o msg_1.bin msg_2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg_1.bin' and 'msg_2.bin'
Using prefixfile: 'prefix50.txt'
Using initial value: ba53612b59e538ef69a33288b08472f1

Generating first block: ..................
Generating second block: S11.
Running time: 4.77734 s
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg1.bin > output1_50_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg_1.bin > output_1_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg_2.bin > output_2_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ diff output_1_hex output_2_hex
6,8c6,8
< 00000050: c7e0 c22a cf16 4b51 45c2 3f05 740e 0dd4  ...*..KQE.?.t...
< 00000060: d1bc bfed dcee 00cd 4f50 e661 ca6b 3863  ........OP.a.k8c
< 00000070: f2bf 3dc5 0cd3 7587 ad39 2879 6384 45c3  ..=...u..9(yc.E.
---
> 00000050: c7e0 c2aa cf16 4b51 45c2 3f05 740e 0dd4  ......KQE.?.t...
> 00000060: d1bc bfed dcee 00cd 4f50 e661 caeb 3863  ........OP.a..8c
> 00000070: f2bf 3dc5 0cd3 7587 ad39 28f9 6384 45c3  ..=...u..9(.c.E.
10,12c10,12
< 00000090: cddb 6490 dab8 121b 84bb ebc7 5f51 e541  ..d........._Q.A
< 000000a0: 2950 c58e 04c5 72d6 9898 eb90 5490 7811  )P....r.....T.x.
< 000000b0: 8183 f1e7 a568 c461 7db8 a054 59d9 004c  .....h.a}..TY..L
---
> 00000090: cddb 6410 dab8 121b 84bb ebc7 5f51 e541  ..d........._Q.A
> 000000a0: 2950 c58e 04c5 72d6 9898 eb90 5410 7811  )P....r.....T.x.
> 000000b0: 8183 f1e7 a568 c461 7db8 a0d4 59d9 004c  .....h.a}...Y..L
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ |
```

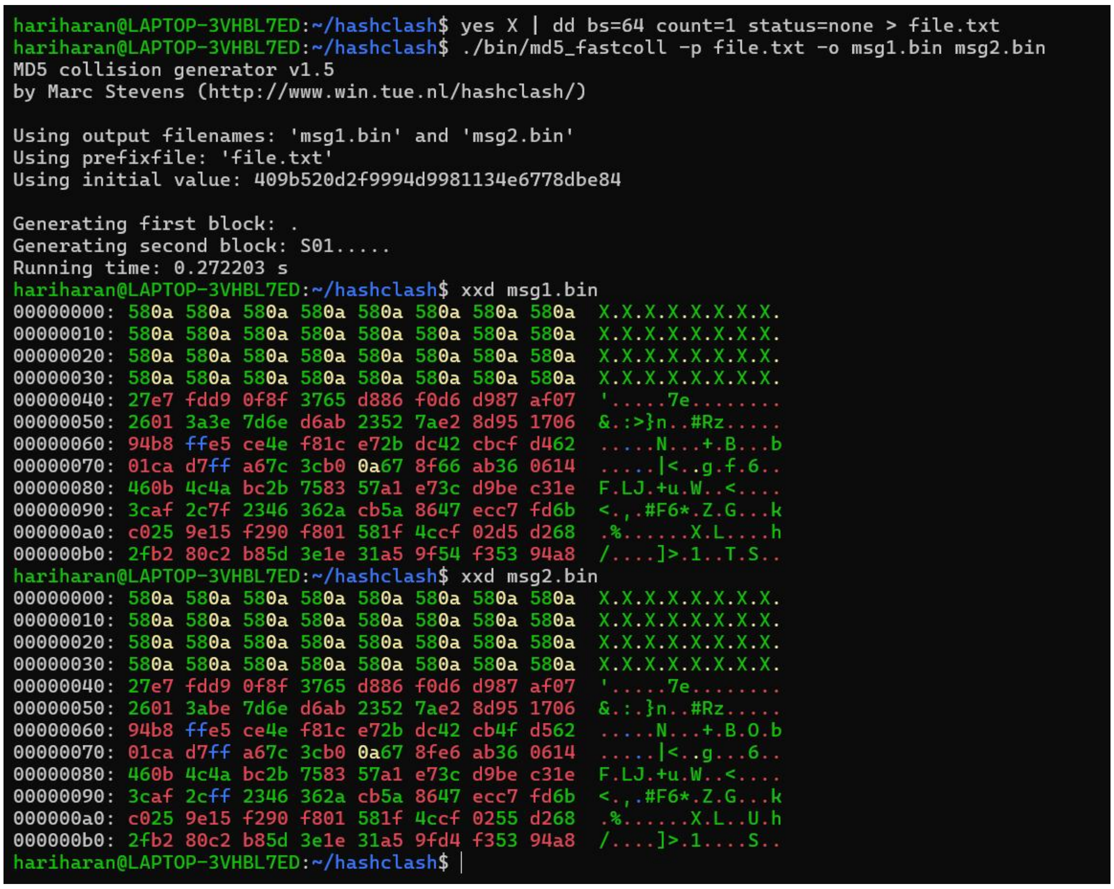
Difference between direct 64 bit prefix file and a 50-byte prefix file



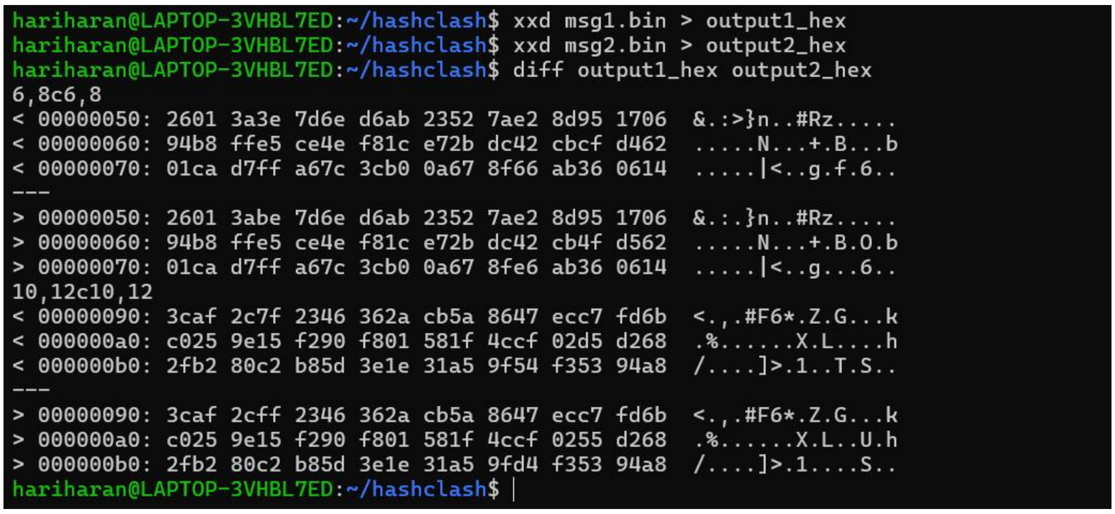
```
hariharan@LAPTOP-3VHBL7ED:~$ cd hashclash
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg1.bin
00000000: 6f16 f3dc e2f6 0b23 57ec bc3f 03a2 2cc0  o......#W..?..,.
00000010: c17a 8382 ac3a 5bea 0b84 be3d ff13 d0b4  .z...:[....=....
00000020: 116f a5b6 d5d4 46e4 4695 3d54 b7c9 38b1  .o....F.F.=T..8.
00000030: 72b5 2eb3 6eb8 e13c 3fdf d929 01d8 fa1d  r...n..<?..)....
00000040: e95e be3d 9719 27b6 0f7c 74b3 42e4 4418  .^.=..'..|t.B.D.
00000050: d160 fc06 51b4 e644 2d8e 84fe 86a1 ade3  .`..Q..D-.......
00000060: 8515 0fcc 4962 1bc2 ebbf c83c 39aa b573  ....Ib.....<9..s
00000070: 8897 6bb5 98f8 36ea b15b 6284 f18a d1c8  ..k...6..[b.....
00000080: 32c8 9acd cf9e 1017 2bad 252f 4f57 6655  2.......+.%/OWfU
00000090: 5856 7652 20b9 e01d cdf7 14c1 5d61 7932  XVvR ........]ay2
000000a0: 3b27 d7be 7005 af43 c199 c0af 39b6 63a4  ;'..p..C....9.c.
000000b0: 8d64 119c f488 2ee0 00e8 8445 a1b3 8e87  .d.........E....
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg_1_bin
xxd: msg_1_bin: No such file or directory
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg_1.bin
00000000: ab7b c3c0 dada 7493 14f3 4306 47e8 810b  .{....t...C.G...
00000010: 3286 5b05 af09 13b1 897a eb37 3091 595f  2.[......z.70.Y_
00000020: b08f a98a 4c68 7cea f145 317d 10c4 6ad9  ....Lh|..E1}..j.
00000030: 0883 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 442c b436 ec6a 5cf7 b785 76fb d30c 8874  D,.6.j\...v....t
00000050: c7e0 c22a cf16 4b51 45c2 3f05 740e 0dd4  ...*..KQE.?.t...
00000060: d1bc bfed dcee 00cd 4f50 e661 ca6b 3863  ........OP.a.k8c
00000070: f2bf 3dc5 0cd3 7587 ad39 2879 6384 45c3  ..=...u..9(yc.E.
00000080: ff93 201d 8a05 cbf7 0de4 48b5 f49f 7aed  .. ......H...z.
00000090: cddb 6490 dab8 121b 84bb ebc7 5f51 e541  ..d........._Q.A
000000a0: 2950 c58e 04c5 72d6 9898 eb90 5490 7811  )P....r.....T.x.
000000b0: 8183 f1e7 a568 c461 7db8 a054 59d9 004c  .....h.a}..TY..L
```

In the 50 bit prefix file the zeros are padded

c) Is the data (128 bytes) generated by md5 fastcoll completely different for both output files? Please identify all the bytes that are different.



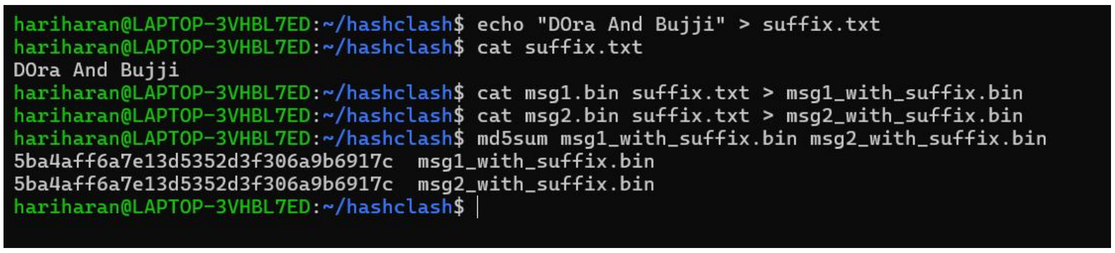
```
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ yes X | dd bs=64 count=1 status=none > file.txt
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ ./bin/md5_fastcoll -p file.txt -o msg1.bin msg2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'msg1.bin' and 'msg2.bin'
Using prefixfile: 'file.txt'
Using initial value: 409b520d2f9994d9981134e6778dbe84

Generating first block: .
Generating second block: S01.....
Running time: 0.272203 s
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg1.bin
00000000: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000010: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000020: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000030: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000040: 27e7 fdd9 0f8f 3765 d886 f0d6 d987 af07  '.....7e........
00000050: 2601 3a3e 7d6e d6ab 2352 7ae2 8d95 1706  &.:>}n..#Rz.....
00000060: 94b8 ffe5 ce4e f81c e72b dc42 cbcf d462  .....N...+.B...b
00000070: 01ca d7ff a67c 3cb0 0a67 8f66 ab36 0614  .....|<..g.f.6..
00000080: 460b 4c4a bc2b 7583 57a1 e73c d9be c31e  F.LJ.+u.W..<....
00000090: 3caf 2c7f 2346 362a cb5a 8647 ecc7 fd6b  <.,.#F6*.Z.G...k
000000a0: c025 9e15 f290 f801 581f 4ccf 02d5 d268  .%......X.L....h
000000b0: 2fb2 80c2 b85d 3e1e 31a5 9f54 f353 94a8  /....]>.1..T.S..
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg2.bin
00000000: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000010: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000020: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000030: 580a 580a 580a 580a 580a 580a 580a 580a  X.X.X.X.X.X.X.X.
00000040: 27e7 fdd9 0f8f 3765 d886 f0d6 d987 af07  '.....7e........
00000050: 2601 3abe 7d6e d6ab 2352 7ae2 8d95 1706  &.:.}n..#Rz.....
00000060: 94b8 ffe5 ce4e f81c e72b dc42 cb4f d562  .....N...+.B.O.b
00000070: 01ca d7ff a67c 3cb0 0a67 8fe6 ab36 0614  .....|<..g...6..
00000080: 460b 4c4a bc2b 7583 57a1 e73c d9be c31e  F.LJ.+u.W..<....
00000090: 3caf 2cff 2346 362a cb5a 8647 ecc7 fd6b  <.,.#F6*.Z.G...k
000000a0: c025 9e15 f290 f801 581f 4ccf 0255 d268  .%......X.L..U.h
000000b0: 2fb2 80c2 b85d 3e1e 31a5 9fd4 f353 94a8  /....]>.1....S..
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ |
```



```
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg1.bin > output1_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ xxd msg2.bin > output2_hex
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ diff output1_hex output2_hex
6,8c6,8
< 00000050: 2601 3a3e 7d6e d6ab 2352 7ae2 8d95 1706  &.:>}n..#Rz.....
< 00000060: 94b8 ffe5 ce4e f81c e72b dc42 cbcf d462  .....N...+.B...b
< 00000070: 01ca d7ff a67c 3cb0 0a67 8f66 ab36 0614  .....|<..g.f.6..
---
> 00000050: 2601 3abe 7d6e d6ab 2352 7ae2 8d95 1706  &.:.}n..#Rz.....
> 00000060: 94b8 ffe5 ce4e f81c e72b dc42 cb4f d562  .....N...+.B.O.b
> 00000070: 01ca d7ff a67c 3cb0 0a67 8fe6 ab36 0614  .....|<..g...6..
10,12c10,12
< 00000090: 3caf 2c7f 2346 362a cb5a 8647 ecc7 fd6b  <.,.#F6*.Z.G...k
< 000000a0: c025 9e15 f290 f801 581f 4ccf 02d5 d268  .%......X.L....h
< 000000b0: 2fb2 80c2 b85d 3e1e 31a5 9f54 f353 94a8  /....]>.1..T.S..
---
> 00000090: 3caf 2cff 2346 362a cb5a 8647 ecc7 fd6b  <.,.#F6*.Z.G...k
> 000000a0: c025 9e15 f290 f801 581f 4ccf 0255 d268  .%......X.L..U.h
> 000000b0: 2fb2 80c2 b85d 3e1e 31a5 9fd4 f353 94a8  /....]>.1....S..
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ |
```

When generating two different files with the same MD5 hash by using md5_fastcoll, the generated data for both files will be distinct.Both the files will consist of the same prefix (64 bytes), but the suffix generated by the collision generation tool will be distinct.Having distinct bytes in the suffix ensures that the resulting files are distinct while having the same MD5 hash.

Task 2: Understanding MD5's Property Since the files generated for the above task out1_64.bin and out2_64.bin share the same prefix, we can just append a suffix to the two files and then check the md5sum of the two files to see if they are still the same.



```
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ echo "DOra And Bujji" > suffix.txt
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ cat suffix.txt
DOra And Bujji
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ cat msg1.bin suffix.txt > msg1_with_suffix.bin
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ cat msg2.bin suffix.txt > msg2_with_suffix.bin
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ md5sum msg1_with_suffix.bin msg2_with_suffix.bin
5ba4aff6a7e13d5352d3f306a9b6917c  msg1_with_suffix.bin
5ba4aff6a7e13d5352d3f306a9b6917c  msg2_with_suffix.bin
hariharan@LAPTOP-3VHBL7ED:~/hashclash$ |
```

Task 3: Generating Two Files with the Same MD5 Hash

a) **Which property of a secure hash function should be broken in order to be able to execute the attack? Elaborate on this question for each of the scenarios described above**.

Scenario 1:

The property which needs to be broken in case of scenario 1 is the second preimage resistance,which is to create a different file that hashes to the same of the hash of a fixed,unchanged file.

X ! = Y such that H(X) = H(Y)

Scenario 2:

The property that needs to be broken is the "collision resistance" property. It makes sure that it is computationally infeasible to find 2 different inputs that hash to generate the same output (collision). In scenario 2 described, the attacker modifies both files to generate a collision. The file modifications should not end up changing the visible content of the file.

b)Is this property broken in case of the MD5 algorithm? Elaborate on this question for each of the scenarios described above.

Scenario 1:

As far as my research,there is no such attack which breaks the MD5 algorithm.But could be possible in the future with more computational power and time.

Scenario2:

The property of collision resistance is broken with the help of Hashclash tools.

C)Elaborate on limitations and possibilities of executing such an attack for each of the scenarios.

Scenario 1: The limitations are that the attacker cannot change the visible content of the first PNG file, which means that the attack is limited to changing the hidden data in the file. Also, the attacker can only modify one PNG file, which limits the attack further. Another limitation is that the attacker must find a collision in the MD5 function which produces the same hash for both PNGs.

Scenario 2: The limitations of this attack are similar to Scenario 1. The attacker cannot change the visible content of the PNG files, which means that the attack is limited to changing the hidden data in the files. Additionally, the attacker must find a collision in the MD5 hash function that produces the same hash output for both PNG files. I chose to go with scenario 2. Here is some information about the PNG file format. File header: The PNG file format starts with an 8-byte header that identifies the file as a PNG file. The header consists of the following bytes: 137 80 78 71 13 10 26 10. The first byte is a magic number that identifies the file as a PNG file. The remaining bytes are control characters that ensure that the file is properly formatted.

Image data: The image data in a PNG file is stored as a series of pixels. Each pixel contains information about the color and transparency of the image.

Chunks: Chunks are optional sections of the PNG file that provide additional image information.

So the format of a chunk will be like:

4 byte: The data length

4 byte: Name of the chunk

Variable length: Data segment

4 byte : CRC

There are several types of chunks, including:

● IHDR(ImageHeader) chunk: This chunk provides image information, such as width, height, and color depth.

● PLTE(Palette) chunk: This chunk contains a color palette that can be used to represent the image.

● IDAT(Image Data) chunk: This chunk contains the compressed image data.

IEND marks the image end; the data field of the IEND chunk has 0 bytes/is empty.

● tEXt(Textual Data) chunk: This chunk contains text information about the image, such as title or author.

● tIME(Time) chunk: This chunk contains the time that the image was last modified.

In Scenario 2, the attacker can modify certain parts of the PNG file format.

In my case for the attack,the basic understanding of how these collisions work is depicted in this picture.

Rules to achieve this attack:

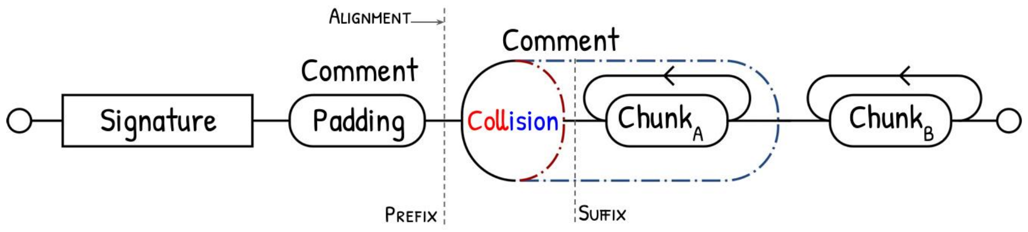
1) The prefix should be of 64 bytes.

2) The 10th byte after the prefix will have a difference of +/- 1.

3) The last prefix block which is passed to the Unicoll should be a multiple of 4.I chose 16 bytes.If its more than 20 bytes,it requires a lot of time to compute the

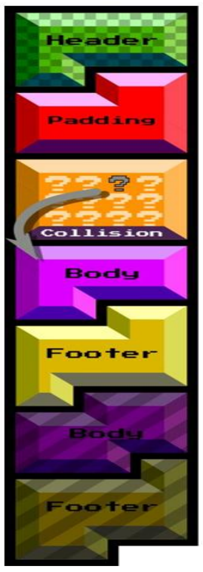
collision.
Basically we start with the prefix which is the png header and padding with fake chunk to be used as appending.
And then crafted a fake chunk dORA where its length is exactly the 10th byte which differs by +/-1 .Then the chunk name and only the first character for the collision block to make the length exactly a multiple of 4.I made the fake chunk length to be 0x71 (113) where the 112 bytes is filled as the collision block by the Unicoll script.
Then the last 4 bytes of the CRC is appended.
Then another chunk jUMP is created which has the length such it could contain the data of 2nd png file.A fake CRC is appended to the end of the jUMP chunk and then the png1 file data.

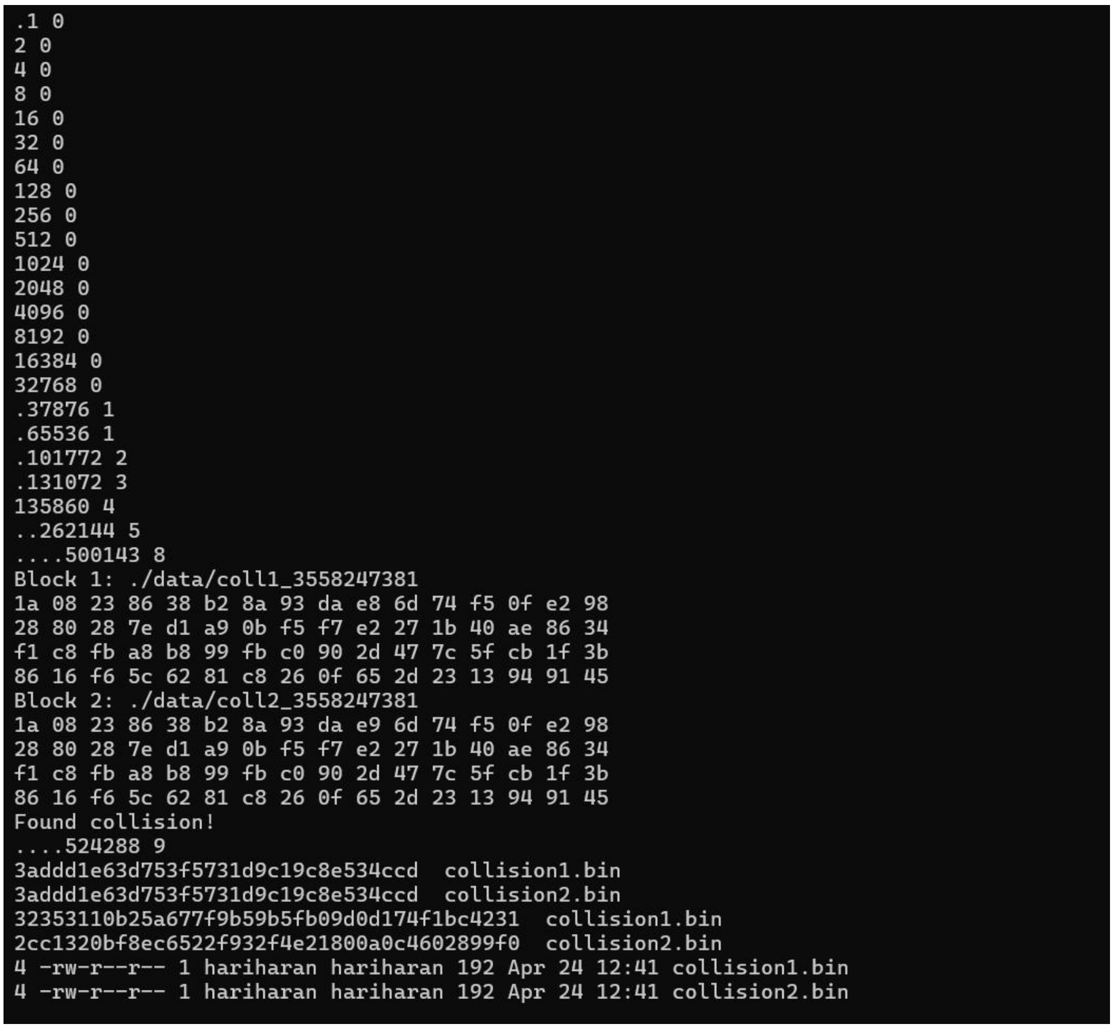
The concept how the two png files work having the same MD5 hash and the same content is explained in this picture.



For the first png file:
The fake chunk dORA will have a length of 0x71,so it correctly ends and the another fake chunk jUMP is read.Since its not meaningful it is ignored as comment and the png file 1 data will be read without getting corrupted .

In case of the second png file:
Since the 10th byte after the prefix is +1,the length of the chunk dORA will be 0x171 which is 0x100(256 bytes extra). The jUMP chunk is part of the fake chunk dORA itself and the second png file Content is read properly without getting corrupted.
Once after the IEND the png data will just be ignored by the png viewer.





```
.1 0
2 0
4 0
8 0
16 0
32 0
64 0
128 0
256 0
512 0
1024 0
2048 0
4096 0
8192 0
16384 0
32768 0
.37876 1
.65536 1
.101772 2
.131072 3
135860 4
..262144 5
....500143 8
Block 1: ./data/coll1_3558247381
1a 08 23 86 38 b2 8a 93 da e8 6d 74 f5 0f e2 98
28 80 28 7e d1 a9 0b f5 f7 e2 27 1b 40 ae 86 34
f1 c8 fb a8 b8 99 fb c0 90 2d 47 7c 5f cb 1f 3b
86 16 f6 5c 62 81 c8 26 0f 65 2d 23 13 94 91 45
Block 2: ./data/coll2_3558247381
1a 08 23 86 38 b2 8a 93 da e9 6d 74 f5 0f e2 98
28 80 28 7e d1 a9 0b f5 f7 e2 27 1b 40 ae 86 34
f1 c8 fb a8 b8 99 fb c0 90 2d 47 7c 5f cb 1f 3b
86 16 f6 5c 62 81 c8 26 0f 65 2d 23 13 94 91 45
Found collision!
....524288 9
3addd1e63d753f5731d9c19c8e534ccd  collision1.bin
3addd1e63d753f5731d9c19c8e534ccd  collision2.bin
32353110b25a677f9b59b5fb09d0d174f1bc4231  collision1.bin
2cc1320bf8ec6522f932f4e21800a0c4602899f0  collision2.bin
4 -rw-r--r-- 1 hariharan hariharan 192 Apr 24 12:41 collision1.bin
4 -rw-r--r-- 1 hariharan hariharan 192 Apr 24 12:41 collision2.bin
```