MD5 Collision Attack Lab

I have setup my Lab in a docker container using the ubuntu 20.04 image from docker hub.

I have downloaded the `md5collgen` tool and set it up. Since I couldn't get a way to install the bless editor in the container, I have decided to use `xxd` to get the hex dump data from the binary file.

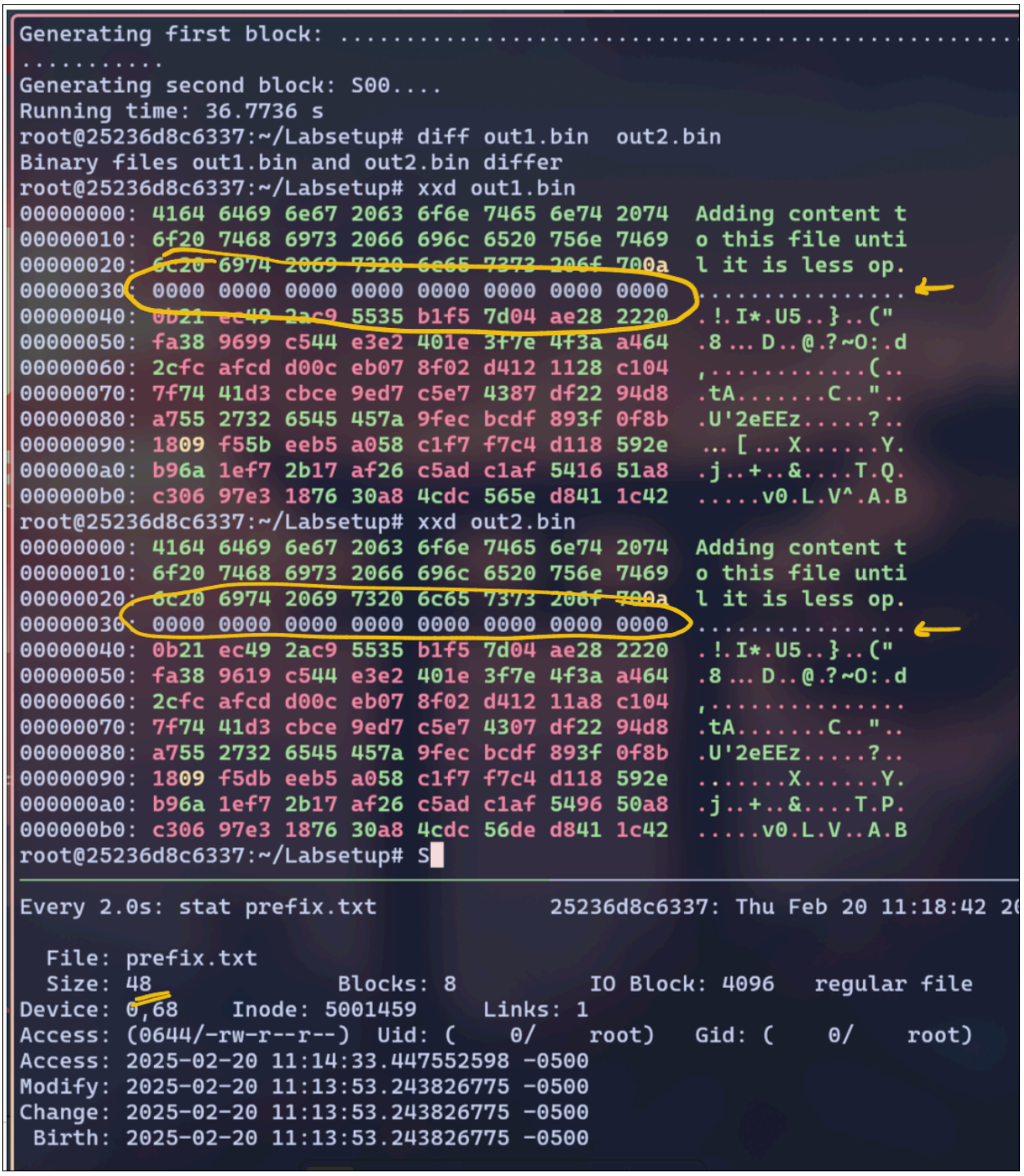
Lab Tasks

3.1 Generating Two Different Files with the Same MD5 Hash

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

Answer:

Prefix Size = 48 bytes



Here, my prefix is less than 64 bytes(not multiple), I see the out1.bin and out2.bin files are mismatching. The `m5collgen` tool adds padding to the prefix to make it 64 bytes to ensure the input is properly formatted.

- 16 bytes of padding were added. The padding bytes are just Zeroes
- The padding could add unintended differences in the output files, which might affect the collision.
- This is why the generated binary files were not perfectly identical in structure.

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

Answer:

```
root@25236d8c6337:~/Labsetup# ./md5collgen -p prefix.txt -o out1.bin out2.bin
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 10855ae3ebed22fda7d21391e874e68b

Generating first block: ......
Generating second block: S01.....................
Running time: 5.35488 s
root@25236d8c6337:~/Labsetup# xxd out1.bin
00000000: 4164 6469 6e67 2063 6f6e 7465 6e74 2074  Adding content t
00000010: 6f20 7468 6973 2066 696c 6520 756e 7469  o this file unti
00000020: 6c20 6974 2069 7320 3634 2062 7974 6573  l it is 64 bytes
00000030: 2069 6e20 746f 7461 6c20 7369 7a65 2e0a   in total size..
00000040: c303 ac83 3d5c 8ca0 d6d6 6d34 bb8b 0147  ....=\....m4...G
00000050: d52a 6e30 82ef 35f3 55f6 429e ad15 46cc  .*n0..5.U.B...F.
00000060: d013 fbcf 7c02 ca99 4bfa c23b 03ca f034  ....|...K..;...4
00000070: 05ba 6fc3 1c43 a5df a6ab bc93 1d1a b300  ..o..C..........
00000080: 270f febf 9863 b98e 2b51 ef64 c978 b972  '....c..+Q.d.x.r
00000090: 0efc 1f65 ffa5 2ee9 d35a 06b9 ecc6 01e6  ...e.....Z......
000000a0: fe65 a6d6 f4cf 034d 6884 5037 a8e8 c268  .e.....Mh.P7...h
000000b0: 3223 0a6e 0b1c cbc1 7c07 ce34 57ec a0da  2#.n....|..4W...
root@25236d8c6337:~/Labsetup# xxd out2.bin
00000000: 4164 6469 6e67 2063 6f6e 7465 6e74 2074  Adding content t
00000010: 6f20 7468 6973 2066 696c 6520 756e 7469  o this file unti
00000020: 6c20 6974 2069 7320 3634 2062 7974 6573  l it is 64 bytes
00000030: 2069 6e20 746f 7461 6c20 7369 7a65 2e0a   in total size..
00000040: c303 ac83 3d5c 8ca0 d6d6 6d34 bb8b 0147  ....=\....m4...G
00000050: d52a 6eb0 82ef 35f3 55f6 429e ad15 46cc  .*n...5.U.B...F.
00000060: d013 fbcf 7c02 ca99 4bfa c23b 034a f134  ....|...K..;.J.4
00000070: 05ba 6fc3 1c43 a5df a6ab bc13 1d1a b300  ..o..C..........
00000080: 270f febf 9863 b98e 2b51 ef64 c978 b972  '....c..+Q.d.x.r
00000090: 0efc 1fe5 ffa5 2ee9 d35a 06b9 ecc6 01e6  .........Z......
000000a0: fe65 a6d6 f4cf 034d 6884 5037 a868 c268  .e.....Mh.P7.h.h
000000b0: 3223 0a6e 0b1c cbc1 7c07 ceb4 57ec a0da  2#.n....|...W...
root@25236d8c6337:~/Labsetup#
```

```
Every 2.0s: stat prefix.txt

  File: prefix.txt
  Size: 64              Blocks: 8          IO Block: 4096   regular file
Device: 0,68    Inode: 5001459     Links: 1
```

Since the prefix file is now 64 bytes, the tool didn't need to add any padding. All it did is, it added `P` to `out1.bin` and `Q` to `out2.bin`.

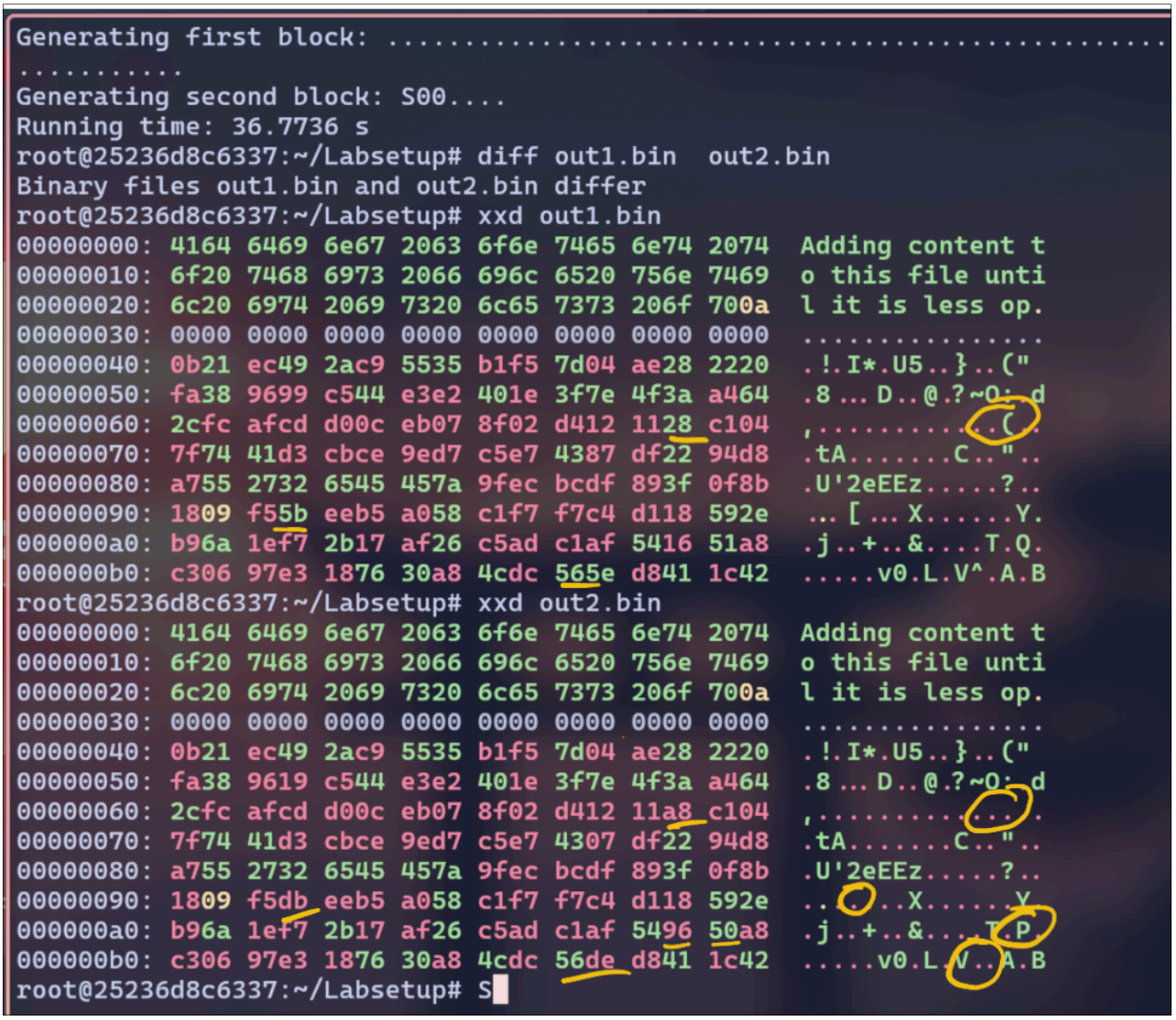
Question 3. Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.

Answer:

The 128 bytes generated by `md5collgen` are completely different for both the cases. Whether the prefix is padded to get 64 bytes or not, the P and Q are different but they do have some similarities.

here are the differences b/w out1.bin and out2.bin

With padding file:



Without Padding:

```
root@25236d8c6337:~/Labsetup# xxd out1.bin
00000000: 4164 6469 6e67 2063 6f6e 7465 6e74 2074  Adding content t
00000010: 6f20 7468 6973 2066 696c 6520 756e 7469  o this file unti
00000020: 6c20 6974 2069 7320 3634 2062 7974 6573  l it is 64 bytes
00000030: 2069 6e20 746f 7461 6c20 7369 7a65 2e0a   in total size..
00000040: c303 ac83 3d5c 8ca0 d6d6 6d34 bb8b 0147  ....=\....m4...G
00000050: d52a 6e30 82ef 35f3 55f6 429e ad15 46cc  .*n0..5.U.B...F.
00000060: d013 fbcf 7c02 ca99 4bfa c23b 03ca f034  ....|...K..;...4
00000070: 05ba 6fc3 1c43 a5df a6ab bc93 1d1a b300  ..o..C..........
00000080: 270f febf 9863 b98e 2b51 ef64 c978 b972  '....c..+Q.d.x.r
00000090: 0efc 1f65 ffa5 2ee9 d35a 06b9 ecc6 01e6  ...e.....Z......
000000a0: fe65 a6d6 f4cf 034d 6884 5037 a8e8 c268  .e.....Mh.P7...h
000000b0: 3223 0a6e 0b1c cbc1 7c07 ce34 57ec a0da  2#.n....|..4W...
root@25236d8c6337:~/Labsetup# xxd out2.bin
00000000: 4164 6469 6e67 2063 6f6e 7465 6e74 2074  Adding content t
00000010: 6f20 7468 6973 2066 696c 6520 756e 7469  o this file unti
00000020: 6c20 6974 2069 7320 3634 2062 7974 6573  l it is 64 bytes
00000030: 2069 6e20 746f 7461 6c20 7369 7a65 2e0a   in total size..
00000040: c303 ac83 3d5c 8ca0 d6d6 6d34 bb8b 0147  ....=\....m4...G
00000050: d52a 6eb0 82ef 35f3 55f6 429e ad15 46cc  .*n...5.U.B...F.
00000060: d013 fbcf 7c02 ca99 4bfa c23b 034a f134  ....|...K..;.J.4
00000070: 05ba 6fc3 1c43 a5df a6ab bc13 1d1a b300  ..o..C..........
00000080: 270f febf 9863 b98e 2b51 ef64 c978 b972  '....c..+Q.d.x.r
00000090: 0efc 1fe5 ffa5 2ee9 d35a 06b9 ecc6 01e6  .........Z......
000000a0: fe65 a6d6 f4cf 034d 6884 5037 a868 c268  .e.....Mh.P7.h.h
000000b0: 3223 0a6e 0b1c cbc1 7c07 ceb4 57ec a0da  2#.n....|...W...
root@25236d8c6337:~/Labsetup#
```

I can see a pattern here. The P and Q even though they are random strings, they are almost same.
Only bytes 47, 84, 110, 124 differ from out1.bin and out2.bin.

3.2 Task 2: Understanding MD5's Property

For out1.bin and out2.bin, I ran the `md5sum` command and the below is the output:

```
root@25236d8c6337:~/Labsetup# md5sum out1.bin out2.bin
f9b824b94fc51c819a69cc91df6f7acf  out1.bin
f9b824b94fc51c819a69cc91df6f7acf  out2.bin
```

Created a new suffix file called `suffix.bin`:

```
root@25236d8c6337:~/Labsetup# echo "new suffix data to binary" > suffix.bin
root@25236d8c6337:~/Labsetup# cat suffix.bin
new suffix data to binary
root@25236d8c6337:~/Labsetup# xxd suffix.bin
00000000: 6e65 7720 7375 6666 6978 2064 6174 6120  new suffix data
00000010: 746f 2062 696e 6172 790a                 to binary.
```

I have concatenated the out1.bin with suffix.bin file and the same for out2.bin file.

```
root@25236d8c6337:~/Labsetup# cat out1.bin suffix.bin > out1_suffix.bin
root@25236d8c6337:~/Labsetup# cat out2.bin suffix.bin > out2_suffix.bin
root@25236d8c6337:~/Labsetup# md5sum out1_suffix.bin out2_suffix.bin
720b3e90a395d5a4c90d073c69de9878  out1_suffix.bin
720b3e90a395d5a4c90d073c69de9878  out2_suffix.bin
root@25236d8c6337:~/Labsetup# 
```

MD5(out1.bin) = MD5(out2.bin)
and MD5(out1.bin I suffix.bin) = MD5(out2.bin I suffix.bin)

where I represents concatenation.

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

MyCode:

```c
#include <stdio.h>

unsigned char xyz[200] = {
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
};

int main() {
    int i;
    for (i=0; i<200; i++){
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

Compile it:

```
gcc code.c
```

Check the hex output using `xxd`

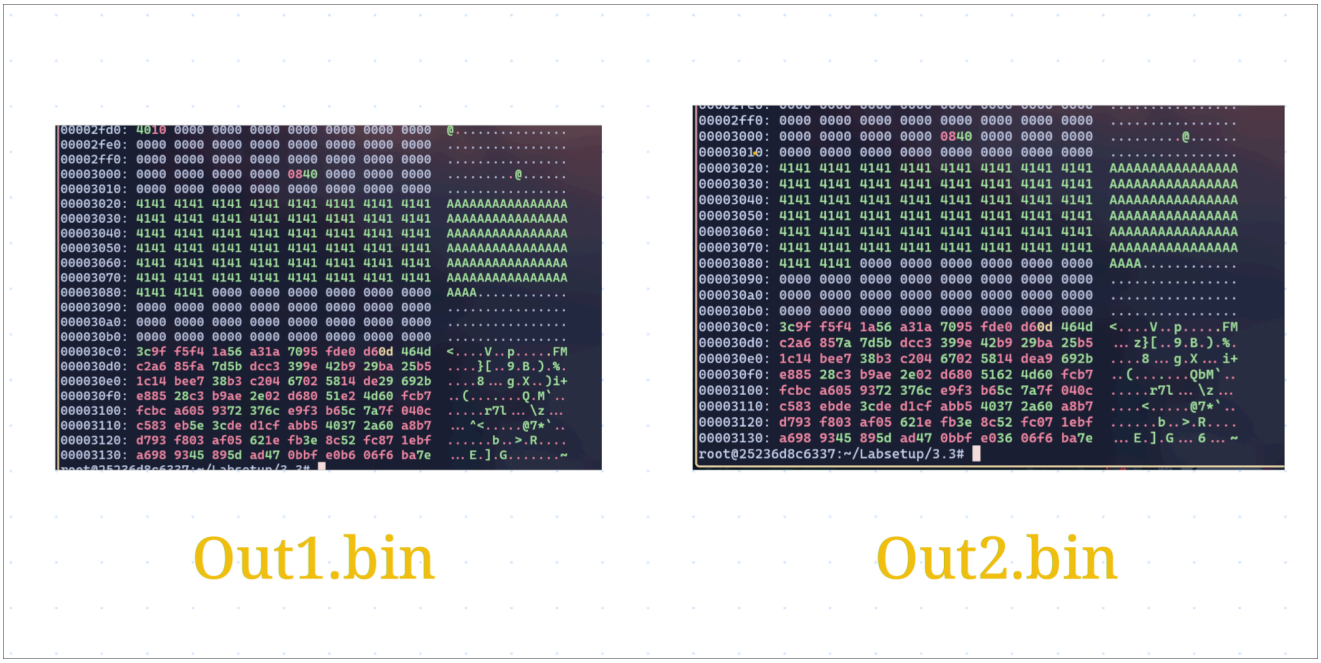
```
xxd ./a.out
```

Now, the letters A start at 3020, but I wanted to split it at half of the char array(at 3084 or 12420 in decimal) so I get all A's in the prefix.

```
head -c 12420 a.out > prefix
```

Generating the out1.bin and out2.bin

```
./md5collgen -p prefix -o out1.bin out2.bin
```

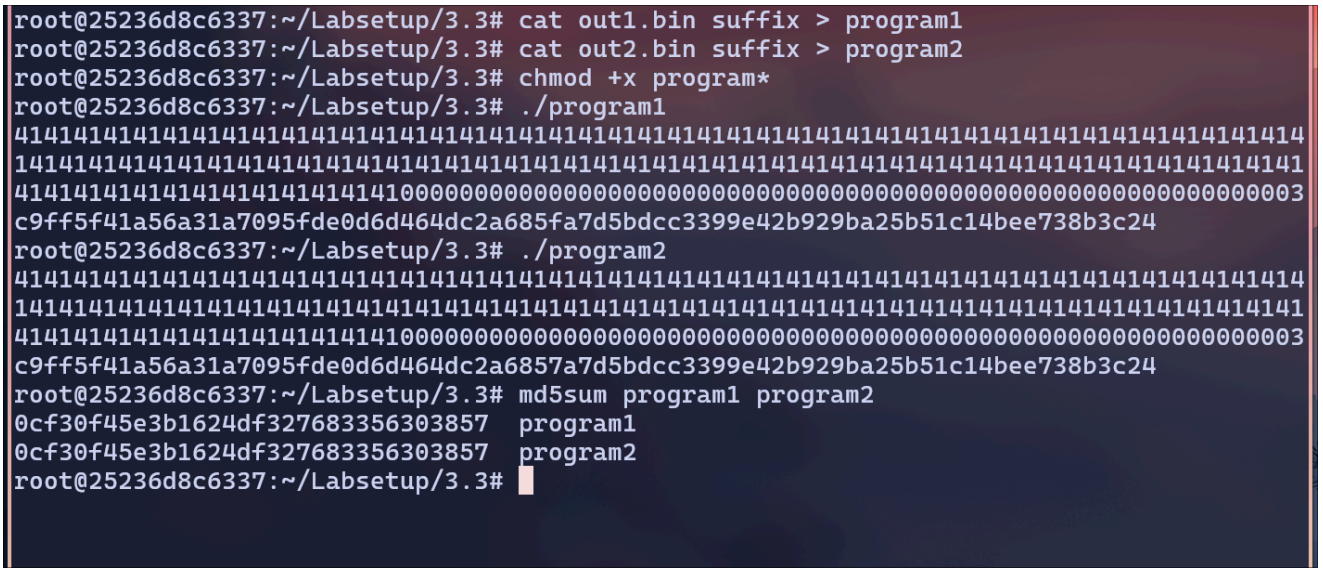


Adding the remaining text to suffix (from hex 3085 or 12421)

```
tail -c +12421 a.out > suffix
```

```
root@25236d8c6337:~/Labsetup/3.3# tail -c +12421 a.out > suffix
root@25236d8c6337:~/Labsetup/3.3# xxd suffix | head
00000000: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000010: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000020: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000030: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000040: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000050: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000060: 4242 4242 4743 433a 2028 5562 756e 7475  BBBBGCC: (Ubuntu
00000070: 2031 332e 332e 302d 3675 6275 6e74 7532   13.3.0-6ubuntu2
00000080: 7e32 342e 3034 2920 3133 2e33 2e30 0000  ~24.04) 13.3.0..
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
root@25236d8c6337:~/Labsetup/3.3#
```

Add the suffix to out1 and out2 to create to files using `cat` command. Also provide the executable permissions and check the output of the files and their md5sum.



The hashes match and we get the same output for both `program1` and `program2` files.

3.4 Task 4: Making the Two Programs Behave Differently

The below code is used for creating two programs that behave differently.

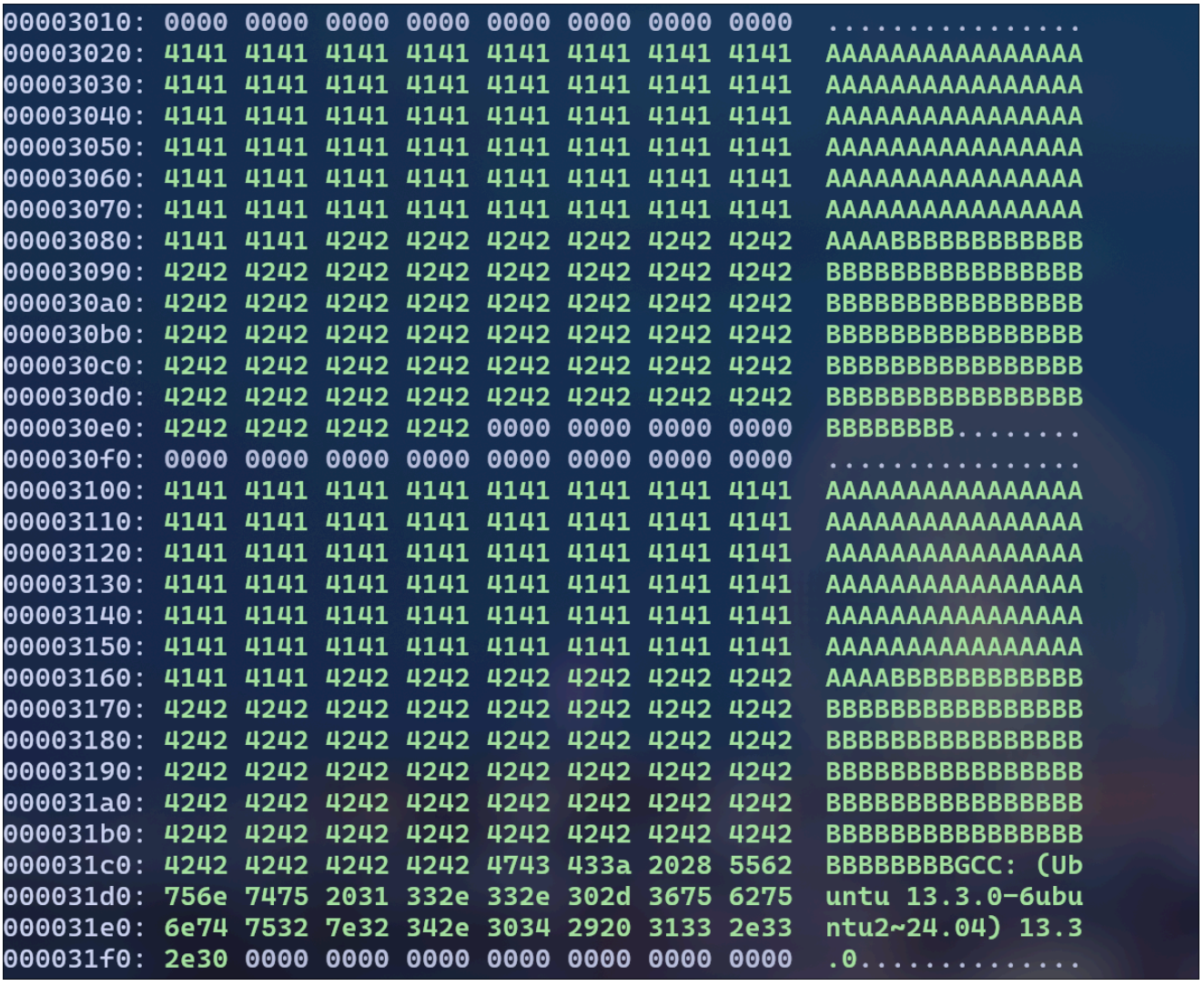
```
#include <stdio.h>
unsigned char x[400] = { 'A','A','A','A','A','A','A','A','A','A','A','A','
unsigned char y[400] = { 'A','A','A','A','A','A','A','A','A','A','A','A','

int main()
{
   int i;
   int foundMatch=1;
   for(i = 0; i < 400; i++)
   {
      if(x[i]!=y[i])
        foundMatch=0;
}

if(foundMatch)
  printf("This is Benign code\n");
else
  printf("This is a Malicious code \n");
}
```

compile the code:

```
gcc code.c
```



We can observe the hex values for first array starting at 3020 and so I will extract the data before this to prefix.

```
head -c 12420 a.out > prefix
```

Using `md5collgen` command to create 2 binary files with the same hash

```
./md5collgen -p prefix -o P_prefix Q_prefix
```

```
root@25236d8c6337:~/Labsetup/3.4# md5sum P_prefix Q_prefix
da89403e832f806d701db7f8d62382f3  P_prefix
da89403e832f806d701db7f8d62382f3  Q_prefix
root@25236d8c6337:~/Labsetup/3.4#
```

Now, we need to connect the prefix files to correct suffix files.

Upon on observation of the prefix files, there is 188 bytes of difference b/w P_prefix and prefix file and the same for Q_prefix.

```
root@25236d8c6337:~/Labsetup/3.4# ls -l *pref*
-rw-r--r-- 1 root root 12608 Feb 21 22:11 P_prefix
-rw-r--r-- 1 root root 12608 Feb 21 22:11 Q_prefix
-rw-r--r-- 1 root root 12420 Feb 21 22:08 prefix
root@25236d8c6337:~/Labsetup/3.4#
```

So, the suffix file is everything after the first 12421 bytes from the `a.out` file.

Create suffix:

```
tail -c +12609 a.out > suffix
```

To get P file, we know 188 bytes are generated extra by `md5collgen`

```
tail -c 188 P_prefix > P
tail -c 188 Q_prefix > Q
```

checking suffix:

```
root@25236d8c6337:~/Labsetup/3.4# xxd suffix
00000000: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000010: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000020: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000030: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000040: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000050: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000060: 4242 4242 0000 0000 0000 0000 0000 0000  BBBB............
00000070: 0000 0000 0000 0000 0000 0000 4141 4141  ............AAAA
00000080: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000090: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
000000a0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
000000b0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
000000c0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
000000d0: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
000000e0: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
000000f0: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000100: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
00000110: 4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
```

Now, we observe second array starts at 7c(124). I'm adding extra 100 bytes to get middle of second array at e0.

```
head -c 150 suffix > middle
```

So, everything after `middle` file will be our `commonsuffix` file. Since `P` is 188 bytes and `middle` is about 174 bytes, the `commonsuffix` file would be everything in the `suffix` file after 412 *(188+154)* bytes.

```
tail -c 339 suffix > commonsuffix
```

At final, we will concatenate everything together:

```
cat P_prefix middle P commonsuffix > benignCode
cat Q_prefix middle Q commonsuffix > maliciousCode
```

```
root@25236d8c6337:~/Labsetup/3.4# ls -l a.out *Code
-rwxr-xr-x 1 root root 16456 Feb 21 23:53 a.out
-rw-r--r-- 1 root root 13285 Feb 21 23:55 benignCode
-rw-r--r-- 1 root root 13285 Feb 21 23:55 maliciousCode
root@25236d8c6337:~/Labsetup/3.4#
```

Verifying MD5 Hash:

```
md5sum program1 program2
```

> We have same hash.

Running the programs and checking their behavior.

Program1: