Crypto MD5 Collision Lab

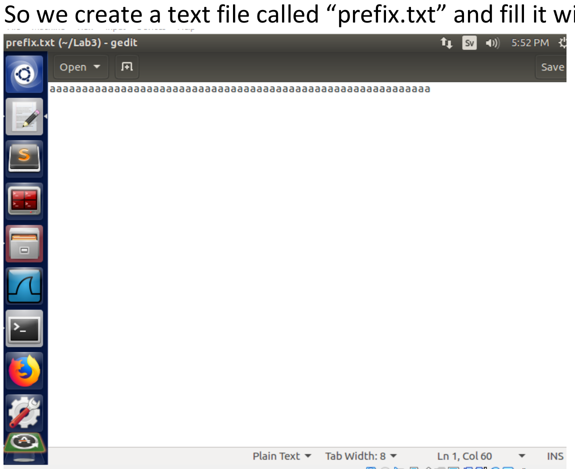
In this lab we will investigate the MD5 collision attack, a secure one way hash needs a one-way property and the collision resistance property. It ensures that with an hash value of h, it's infeasible to find the input M provided by the user. The learning objective of this lab is for us students to understand the impact of collision attacks and why the collision resistance property is important, we will see first hand what damages it can cause if the collision resistance property is broken. To achieve this we need to launch collision attacks against the has function, be able to create two different programs that share the same MD5 but have completely different behaviors. We will cover One-way hash functions, the collision resistance property, launch collision attacks and learn about MD5. The lab will be done in the pre-built ubuntu virtual machine provided by seed labs. We'll use a tool called "Fast MD5 collision generation"

Task 1

For task 1 we first need to create a prefix.txt file and fill it with contents, if we don't create this file the md5collgen will throw an error

```
[04/14/21]seed@VM:~/Lab3$ md5collgen -p prefix.txt -o o
ut1.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'
Error: cannot open inputfile: 'prefix.txt'
[04/14/21]seed@VM:~/Lab3$
```

So we create a text file called "prefix.txt" and fill it with contents as such and fill it with 60 'a's



Now if I re-launch the provided syntax we get the following result:

```
[04/14/21]seed@VM:~/Lab3$ md5collgen -p prefix.txt -o o
ut1.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: 4224e33d87b0f1396b4ffa6120177833

Generating first block: .....
Generating second block: S01.........
Running time: 7.00675 s
[04/14/21]seed@VM:~/Lab3$
```

Marcus Roos Maro1904
maro1904@student.miun.se

To check if the output files are the same we'll use a difftool to see if there's any difference.

```
[04/14/21]seed@VM:~/Lab3$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
```

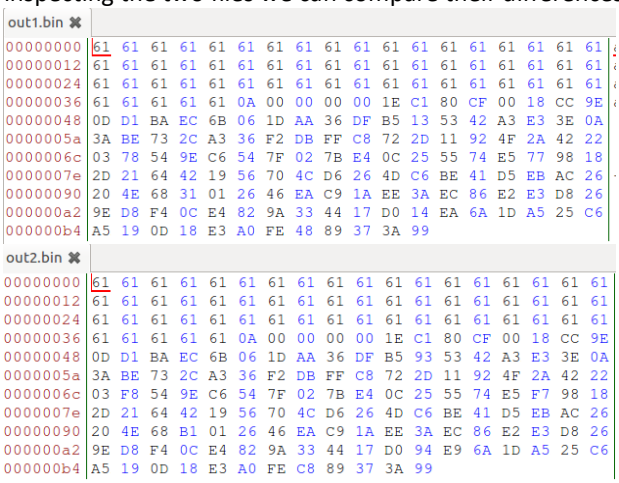
As we can see the two files are different even when the contents of prefix.txt were the same. We then check if their md5sum is the same or not. We can see that the contents of the files are different, but the md5 sums are exactly the same.

```
[04/14/21]seed@VM:~/Lab3$ md5sum out1.bin
3a1f2316d8308a848cfac1efa0777950  out1.bin
[04/14/21]seed@VM:~/Lab3$ md5sum out2.bin
3a1f2316d8308a848cfac1efa0777950  out2.bin
[04/14/21]seed@VM:~/Lab3$
```

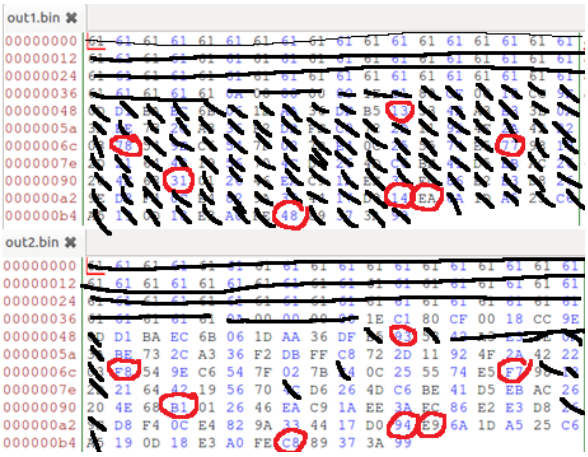
We need to open those two files so we can see what makes them different, for this we are going to use the software called "bless" , we then need to answer 3 questions:

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

As the prefix file we used had a length of 60 bytes we can use it to determine what happens. By running the bless software and inspecting the two files we can compare their differences.



The 61s that we see are the hex values of the 'a's we put in, as we can see its the same in both files. I did use Paint for this (I know it's terrible, but it got the job done. And I could find 7 hex values which differed, the ones circled in red are the same for both tables. For bigger files I'd definitely recommending using a proper difftool (such as kdiff or vimdiff).



Marcus Roos
maro1904@student.miun.se
Maro1904

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

For this I added 4 more ‘a’s to my prefix.txt file to extend it to 64 bytes and ran the collision tool again.
Running the diff tool now instead gives us a vastly different result but we can still see differences in the both prints meaning the files are still different from one another.

```
[04/14/21]seed@VM:~/Lab3$ diff out1.bin out2.bin
2c2
:::-@y1@9@@@@@@tK\z@@@J@@@:@@@@{@@Z@@@@9@@;@@@\@lsN
       @@]@@@Yqy@MI^@kRv@@@/@"@@s@@@|@@@W='b@
\ No newline at end of file
---
:::-@y1@9@@@@@@tK\z@9@J@@@:@@@@{@@Z@@@@9@@;@@@\@lsN@
@@       @@]@@@Yqy@MI^@kRv@@@/@@@s@@@|@@@@=@'b@
\ No newline at end of file
[04/14/21]seed@VM:~/Lab3$
```

In the bless tool we can see they are still different

out1.bin

```
00000000 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aa
00000012 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aa
00000024 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aa
00000036 61 61 61 61 61 61 61 61 0A 52 22 2B 3D 16 ED DB 87 aa
00000048 47 9A 06 D4 CF 45 3E 0C 8F FA F8 BA 8E 07 08 79 4D 9A G.
0000005a 82 69 E8 AB 13 CA 45 1E FC E6 58 2A 7E 1A E7 45 7B 04 .i
0000006c 0F 86 84 8E 99 DF F3 F8 DB DA 1D 89 B3 D7 CB CA D9 CD ..
0000007e 51 6C 2A 85 D3 F3 33 88 F4 0D 5D F2 21 CA 35 27 F0 00 Ql
00000090 C0 AB 01 12 FE FD 20 56 C9 1B 16 C1 51 59 79 B2 37 89 ..
000000a2 D7 BA 56 02 7E 31 5B 99 D0 AF 5D 6E 76 C4 E0 BC B1 1B ..
000000b4 5E 99 9E 66 69 22 B1 74 48 23 22 0C ^.
```

out2.bin

```
00000000 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 a
00000012 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 a
00000024 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 a
00000036 61 61 61 61 61 61 61 61 0A 52 22 2B 3D 16 ED DB 87 a
00000048 47 9A 06 D4 CF 45 3E 0C 8F FA F8 3A 8E 07 08 79 4D 9A G
0000005a 82 69 E8 AB 13 CA 45 1E FC E6 58 2A 7E 1A E7 45 7B 04 .
0000006c 0F 06 85 8E 99 DF F3 F8 DB DA 1D 89 B3 D7 CB 4A D9 CD .
0000007e 51 6C 2A 85 D3 F3 33 88 F4 0D 5D F2 21 CA 35 27 F0 00 Q
00000090 C0 AB 01 92 FE FD 20 56 C9 1B 16 C1 51 59 79 B2 37 89 .
000000a2 D7 BA 56 02 7E 31 5B 99 D0 AF 5D EE 75 C4 E0 BC B1 1B .
000000b4 5E 99 9E 66 69 22 B1 F4 48 23 22 0C ^
```

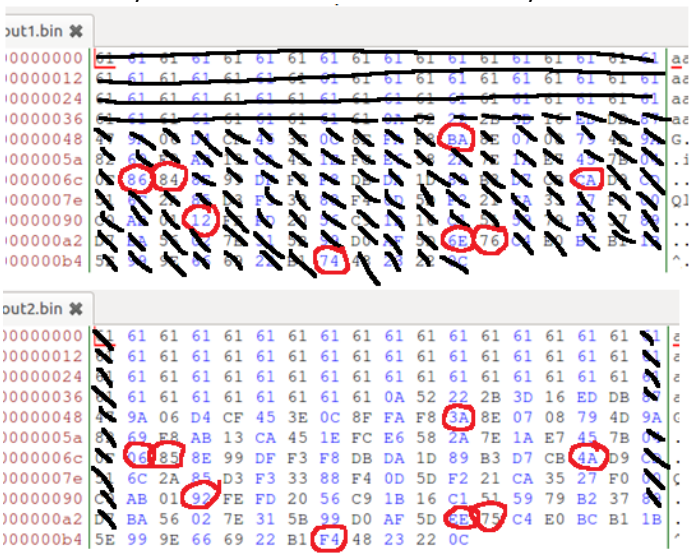
Using the md5sum however we can see that the values are still the same, just as when we had 60 bytes.

```
[04/14/21]seed@VM:~/Lab3$ md5sum out2.bin
f08a969e5f0c853239071fdf89f39786  out2.bin
[04/14/21]seed@VM:~/Lab3$ md5sum out1.bin
f08a969e5f0c853239071fdf89f39786  out1.bin
[04/14/21]seed@VM:~/Lab3$
```

Marcus Roos
maro1904@student.miun.se
Maro1904

– 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.

As we are checking the 128 bytes (2*64) in the two different files generated from the prefix file of 64 bytes we'll compare them as we did in the first question and identify all the different bytes. The unedited values can be compared from question 2 if there's any doubt. As we can see there are 8 bytes that differ this time.



Marcus Roos
maro1904@student.miun.se
Maro1904

Task 2

Understanding the MD5's property, in this task we will learn about some of the properties of the MD5 algorithm. For this we can use our previous two files we generated (out1.bin and out2.bin), and run the 'cat' command such as "cat out1.bin prefix.txt > out3.bin" and "cat out2.bin prefix.txt > out4.bin".
This will concatenate the contents of out2.bin into contents of out1.bin and store the results in out3.bin, just as the other quoted command will concatenate the contents of out1.bin into contents of out2.bin and store the results in out4.bin.
We know from previous task that the md5sum from the both bin files are the same, and the byte values differ from one another. We then do the same for the concatenated files.

```
[04/15/21]seed@VM:~/Lab3$ md5sum out1.bin
f08a969e5f0c853239071fdf89f39786  out1.bin
[04/15/21]seed@VM:~/Lab3$ md5sum out2.bin
f08a969e5f0c853239071fdf89f39786  out2.bin
[04/15/21]seed@VM:~/Lab3$ cat out1.bin prefix.txt > out
3.bin
[04/15/21]seed@VM:~/Lab3$ cat out2.bin prefix.txt > out
4.bin
[04/15/21]seed@VM:~/Lab3$ md5sum out3.bin
58f7cfcfa00fbb3d6b7763644c4adf9b  out3.bin
[04/15/21]seed@VM:~/Lab3$ md5sum out4.bin
58f7cfcfa00fbb3d6b7763644c4adf9b  out4.bin
[04/15/21]seed@VM:~/Lab3$ cat prefix.txt
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaa
[04/15/21]seed@VM:~/Lab3$ diff out3.bin out4.bin
Binary files out3.bin and out4.bin differ
[04/15/21]seed@VM:~/Lab3$
```

As we can see the MD5 sum for out3 and out4.bin differ from that of out1 and out2.bin, but the md5 sum for out3 and out4.bin are the same to one another, we concatenated the contenst of prefix.txt to out1 and out2.bin and placed the result in out3 and out4.bin respectively. This proves that concatenating does change the md5 sum but the results are still the same if the starting sum is the same, however we need to keep in mind that the bytes themselves differ, as we can see if we run the diff command. From the lab we can confirm that the statement holds true:
"Given two inputs M and N, if MD5(M) = MD5(N), i.e., the MD5 hashes of M and N are the same, then for any input T, MD5(M k T) = MD5(N k T), where k represents concatenation"

Task 3

For this task we are generating two executable files with the same MD5 hash, we've been given a C program and our job is to create two different versions of the program, so that their xyz arrays are different but the hash values of the exec are the same, the following is the code we've been given:

```c
#include <stdio.h>
unsigned char xyz[200] = {
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61
};
int main()
{
        int i;
        for (i=0; i<200; i++){
                printf("%x", xyz[i]);
        }
        printf("\n");
}
```

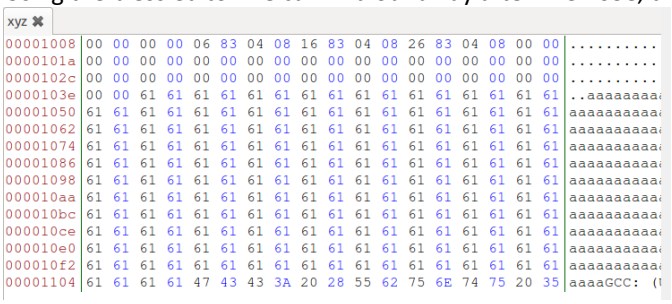
As I've been using lower case 'a' for the previous tasks I filled the array with lower case 'a's hex form, this makes it easier to find the array when we're using bless as all we need to do is to find the area where a lot of "61" are located, that's the place of our array.

Marcus Roos Maro1904
maro1904@student.miun.se

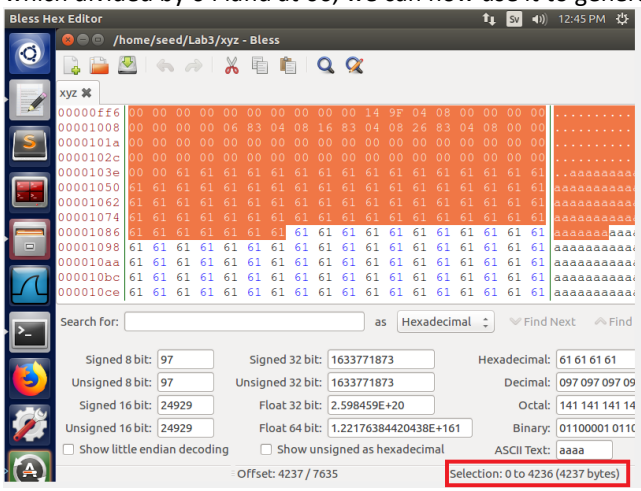
We compile the program as following

```
[04/15/21]seed@VM:~/Lab3$ gcc -o xyz xyz.c
```

Using the bless editor we can find our array after line 103e, there is a lot of padding before our array shows up.



To change the contents of the xyz we can take an offset from the bless editor, including a bit of our array and change the values slightly. To do this we select from the start of the file to a bit through our array and copy the offset. In the following screenshot we can see the offset from the start to the array being 4237 bytes, we'll use this to edit our array, but we can't divide 4237 by 64 (size of our prefix) which is what we want, so we increment until we can divide it by 64 and land at 4224, which divided by 64 land at 66, we can now use it to generate our files with an offset value of 4224.



We use the commands given to us by the lab description and adjust it for our specific case:

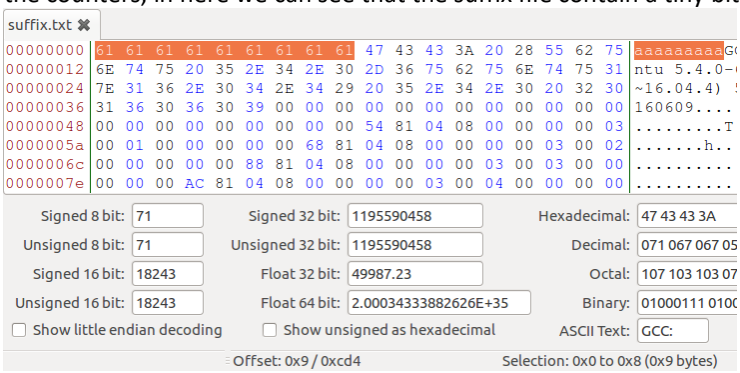
head –c 4224 xyz > prefix.txt

We also do the same for the suffix, but we append 128 bytes in this case, as we want to replace 128 bytes as per the description. We add a '+' to indicate we want to start saving contents from 4352 offset until end of the file, while the head indicates we want to save from the start of the file until the 4224 offset.

tail –c +4352 xyz > suffix.txt

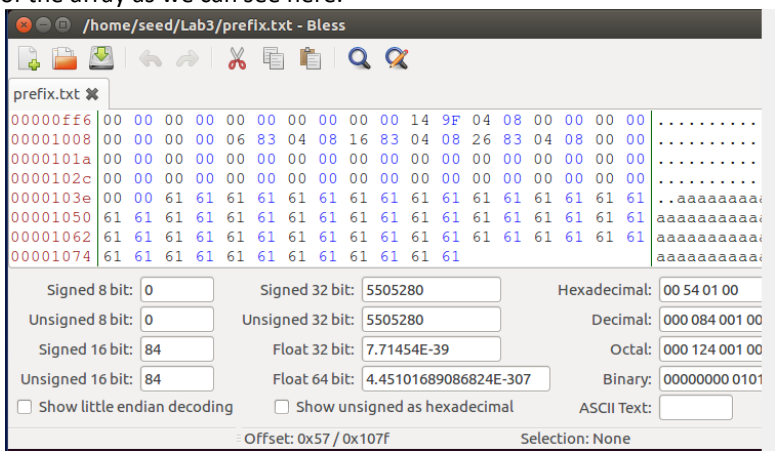
```
[04/15/21]seed@VM:~/Lab3$ head -c 4224 xyz > prefix.txt
[04/15/21]seed@VM:~/Lab3$ tail -c +4352 xyz > suffix.tx
t
```

The contents of those two files are intelligible, but it has copied the contents from the xyz file and stored it into the prefix.txt and suffix.txt files respectively. We could however use bless tool once again and open up the prefix/suffix.txt files to inspect the counters, in here we can see that the suffix file contain a tiny bit of the array.



Marcus Roos Maro1904
maro1904@student.miun.se

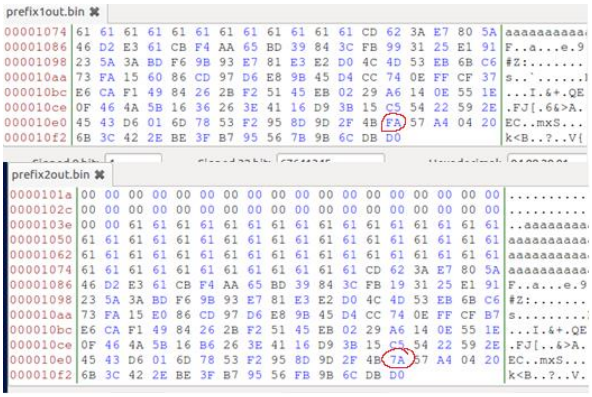
Meanwhile the prefix.txt file contain the same contents as xyz do up until the 4224 byte, meaning it ends towards the middle of the array as we can see here.



From the prefix.txt we then create two new files., prefix1out.bin and prefix2out.bin.

```
[04/15/21]seed@VM:~/Lab3$ md5collgen -p prefix.txt -o p
refix1out.bin prefix2out.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'prefix1out.bin' and 'prefix2ou
t.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 34265d72c4079fc17f975b24906eb301

Generating first block: .....
Generating second block: S00.
Running time: 6.36172 s
[04/15/21]seed@VM:~/Lab3$
```

We open up those two files in the bless editor and look at their last bytes.



To make sure they are different I ran the diff tool, and then located at least one byte that differs. As per the instructions I save the 128 bytes from prefix1out.bin and prefix2out.bin and store them in P and Q respectively

```
[04/16/21]seed@VM:~/Lab3$ tail -c 128 prefix1out.bin >
P
[04/16/21]seed@VM:~/Lab3$ tail -c 128 prefix2out.bin >
Q
```

Marcus Roos Maro1904
maro1904@student.miun.se

As with the previos task I concatenate the P and Q to their respective programs.

```
[04/16/21]seed@VM:~/Lab3$ cat prefix.txt P suffix.txt >
 suffix1.out
[04/16/21]seed@VM:~/Lab3$ cat prefix.txt Q suffix.txt >
 suffix2.out
[04/16/21]seed@VM:~/Lab3$ chmod 4755 suffix1.out suffix
2.out
```

Using the same method as before I'll check whether they are different or not, and I check whether their MD5sum is the same or not.

```
[04/16/21]seed@VM:~/Lab3$ diff suffix1.out suffix2.out
Binary files suffix1.out and suffix2.out differ
[04/16/21]seed@VM:~/Lab3$ md5sum suffix1.out
ec5bb515a8c1313a0d9b9e231c1718fe  suffix1.out
[04/16/21]seed@VM:~/Lab3$ md5sum suffix2.out
ec5bb515a8c1313a0d9b9e231c1718fe  suffix2.out
```

If we run both programs they will print their array, we can see some differences in this code as well as we would expect because the diff tool told us so, nevertheless we manually check it to prove it holds true.

```
[04/16/21]seed@VM:~/Lab3$ suffix1.out
6161616161616161616161616161616161616161616161616161616161616161
1616161616161616161616161616161616161616161616161616161616161616
6161616161616161cd623ae7805a46d2e361cbf4aa65bd39843cf
b993125e191235a3abdf69b93e781e3e2d04c4d53eb6bc673fa1560
86cd97d6e89b45d4cc74effcf37e6caf14984262bf25145eb229a61
4e551ef464a5b1636263e4116d93b15c55422592e4543d616d7853f
2958d9d2f4bfa57a44206b3c422ebe3fb795567b9b6cdbd06161616
161616161
[04/16/21]seed@VM:~/Lab3$ suffix2.out
6161616161616161616161616161616161616161616161616161616161616161
1616161616161616161616161616161616161616161616161616161616161616
6161616161616161cd623ae7805a46d2e361cbf4aa65bd39843cf
b193125e191235a3abdf69b93e781e3e2d04c4d53eb6bc673fa15e0
86cd97d6e89b45d4cc74effcfb7e6caf14984262bf25145eb229a61
4e551ef464a5b16b6263e4116d93b15c55422592e4543d616d7853f
2958d9d2f4b7a57a44206b3c422ebe3fb79556fb9b6cdbd06161616
161616161
```

The mission of this task was to create two different executable files with the same md5 hash, as we can see from the above screenshots that's what we have accomplished. The files differ, but their md5sum is the same, the contents of their array is different, but the md5sum is still the same.

Marcus Roos Maro1904
maro1904@student.miun.se

Task 4

For this task we are going to make the two programs we previously created behave differently. Right now they work likewise as both print out the contents of their array and they still execute the same instructions, for this task we will change their behaviors from one another. As copied from the lab instructions we can easily see what we want to achieve, and why it would be beneficial for someone with malign intent to achieve this.

"Assume that you have created a software which does good things. You send the software to a trusted authority to get certified. The authority conducts a comprehensive testing of your software, and concludes that your software is indeed doing good things. The authority will present you with a certificate, stating that your program is good. To prevent you from changing your program after getting the certificate, the MD5 hash value of your program is also included in the certificate; the certificate is signed by the authority, so you cannot change anything on the certificate or your program without rendering the signature invalid. You would like to get your malicious software certified by the authority, but you have zero chance to achieve that goal if you simply send your malicious software to the authority. However, you have noticed that the authority uses MD5 to generate the hash value. You got an idea. You plan to prepare two different programs. One program will always execute benign instructions and do good things, while the other program will execute malicious instructions and cause damages. You find a way to get these two programs to share the same MD5 hash value."

We will use a similar approach as to when created the previous two files but we will follow the pseudo-code provided to us by the lab instructions:

```
Array X;
Array Y;

main()
{
  if(X's contents and Y's contents are the same)
      run benign code;
  else
      run malicious code;
  return;
}
```

My plan here is to fill the contents of those array with the same values so they are easier to find in the bless editor, we then iterate through both the arrays to see if they are different from each other, if they aren't different at all we will execute the benign code, if they are different, the malicious code will run instead. I wrote the following code, and once compiled and ran it will execute either malicious code or benign code depending if the array contents are the same. This is for demonstration purposes, the last byte (0x52) found in arrayTwo is changed so I could test if the bool statement worked, it's in fact change to an 'a' in the actual compiler (0x61).

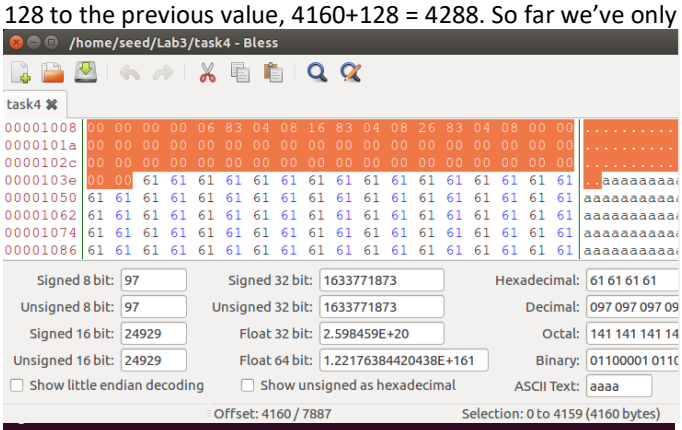
```
#include <stdio.h>
#include <stdbool.h>
unsigned char arrayOne[200] = {
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61
};

unsigned char arrayTwo[200] = {
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61, 0x61,
0x61, 0x61, 0x61, 0x61, 0x52
};
int main()
{
bool loop = true;
// The for loop will break if a divergence is found in the contents and the loop
will switch to false to indicate a divergence
        for (int i=0; i<200; i++){
                if(arrayOne[i] != arrayTwo[i]){
                                loop = false;
                                break;
                                }

        }
        if (loop) {
                printf("%s\n","Executing benign code");
        }
        else {
                printf("%s\n","Executing malicious code");
        }
        printf("\n");
}
```

I first ran this program with the contents being identical, then I changed the last byte to verify whether it works or not.

```
[04/16/21]seed@VM:~/Lab3$ gcc task4.c -o task4
[04/16/21]seed@VM:~/Lab3$ task4
Executing benign code

[04/16/21]seed@VM:~/Lab3$ gcc task4.c -o task4
[04/16/21]seed@VM:~/Lab3$ task4
Executing malicious code

[04/16/21]seed@VM:~/Lab3$ gcc task4.c -o task4
[04/16/21]seed@VM:~/Lab3$ task4
Executing benign code
```

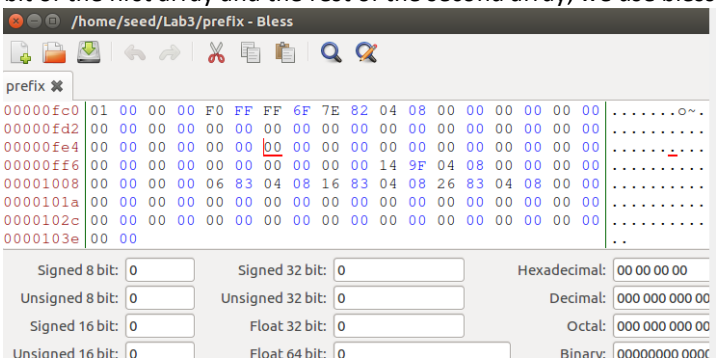
We now have a premise and a two programs which behave the same, and have the same contents in their array, we now want to change their behavior, we will treat this task the same as we did with task 4, in that we will check their contents in the bless editor, find where our array starts and grab the byte offset and push it into a prefix file, and pull the offset+128 bytes into the suffix file.

Using bless editor I find the start of the first array at 4160 bytes. We also need a value for our suffix, for this we'll simply add 128 to the previous value, 4160+128 = 4288. So far we've only done exactly the same as we did in the previous task.

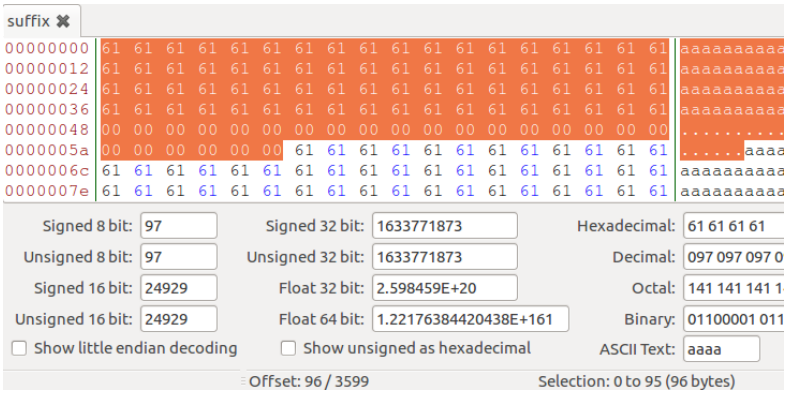



```
[04/16/21]seed@VM:~/Lab3$ head -c 4160 task4 > prefix
[04/16/21]seed@VM:~/Lab3$ tail -c +4288 task4 > suffix
[04/16/21]seed@VM:~/Lab3$ md5collgen -p prefix -o task4
 1.bin task4_2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'task4_1.bin' and 'task4_2.bin'
Using prefixfile: 'prefix'
Using initial value: f76bda5d92edc188765736efb7a3ce83

Generating first block: ...
Generating second block: S01.....................
.................
Running time: 8.59004 s
[04/16/21]seed@VM:~/Lab3$ tail -c 128 task4_1.bin > P
[04/16/21]seed@VM:~/Lab3$ tail -c 128 task4_2.bin > Q
[04/16/21]seed@VM:~/Lab3$
```

The prefix file should contain from start of file until well through our first array, while the suffix file should contain the very last bit of the first array and the rest of the second array, we use bless to check if this holds true.



Marcus Roos Maro1904
maro1904@student.miun.se



Our second array is entirely in the suffix file, we want to split this suffix file up even further so we can edit the contents of the second array, the second array begins at an offset of 96 bytes, we know from before that we want to add 128 bytes to this for the second part. Meaning first suffix contain contents from start of the file until byte 97, and the second one will be split at 128 bytes until end of file. 128+96 = 224 byte offset.

Marcus Roos
maro1904@student.miun.se
Maro1904

We now need to create our two programs, the plan is for them to have the same md5sum but behave differently once launched, one of them should print “executing benign code” while the other should print “executing malicious code”.
We create our two programs as in task 3 by using “cat” and including our prefix, P/Q values, suffix1 and suffix2, this took a bit of trial and error to get working correctly but I found this to be working as it should.

```
[04/16/21]seed@VM:~/Lab3$ head -c 96 suffix > suffix1
[04/16/21]seed@VM:~/Lab3$ tail -c +224 suffix > suffix2
[04/16/21]seed@VM:~/Lab3$ cat prefix P suffix1 P suffix
2 > task4benign.out
[04/16/21]seed@VM:~/Lab3$ cat prefix Q suffix1 P suffix
2 > task4malicious.out
```

As with task 4, we need to give them the right permissions, chmod 4755 should do the trick.

```
[04/16/21]seed@VM:~/Lab3$ chmod 4755 task4benign.out ta
sk4malicious.out
[04/16/21]seed@VM:~/Lab3$ task4benign.out
Executing benign code

[04/16/21]seed@VM:~/Lab3$ task4malicious.out
Executing malicious code

[04/16/21]seed@VM:~/Lab3$ md5sum task4benign.out
dbe3869a62a96fceffaa034d0e6d196f  task4benign.out
[04/16/21]seed@VM:~/Lab3$ md5sum task4malicious.out
dbe3869a62a96fceffaa034d0e6d196f  task4malicious.out
[04/16/21]seed@VM:~/Lab3$
```

We can see that in the end we managed to execute benign code with our first program, and malicious code with our second program, while their md5sum still remains the same.

Marcus Roos Maro1904
maro1904@student.miun.se