CPS633 Section 07 Fall2021

Lab 04 Report

MD5 Collision attack Lab

Name: Tusaif Azmat (group leader)
Student#: 500660278.
And
Name: Ankit Sodhi
Student#: 500958004

Group 04.

CPS633 Fall2021 Section 07 Group 04 Name: Tusaif Azmat - 500660278
Name: Ankit Sodhi - 500958004

CPS 633 - Lab 4 Report

MD5 Collision Attack Lab

2 Lab Tasks:

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

In this task, we will generate two different files with the same MD5 hash values. The beginning parts of these two files need to be the same, i.e., they share the same prefix. We can achieve this using the md5collgen program, which allows us to provide a prefix file with any arbitrary content.
The following command generates two output files, out1.bin and out2.bin, for a given a prefix file prefix.txt:

```
$ md5collgen -p prefix.txt -o out1.bin out2.bin
```

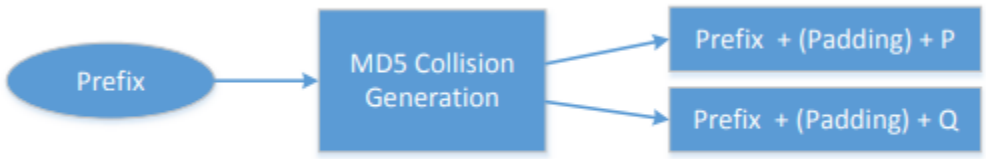
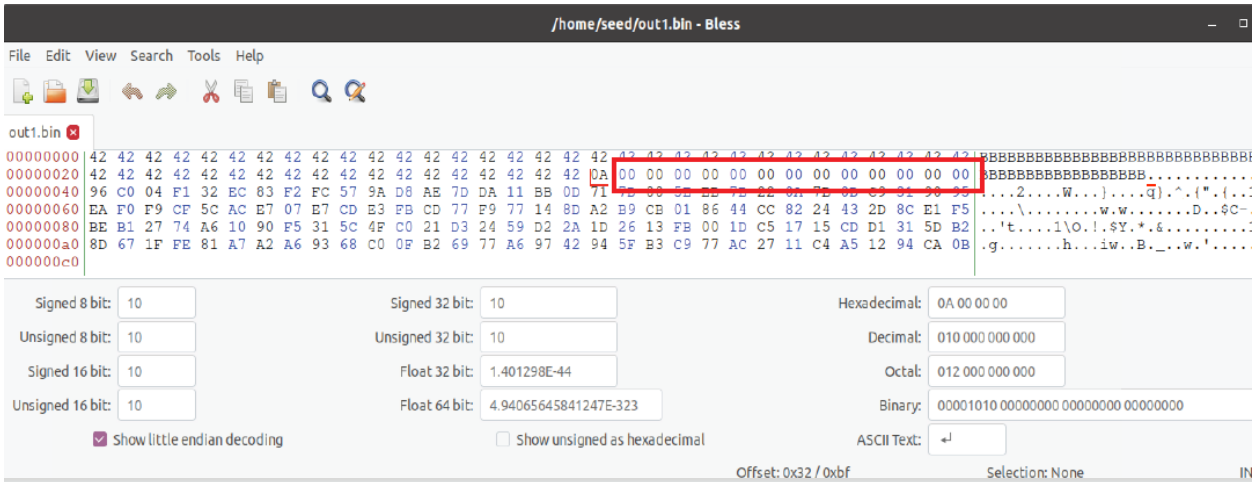


Figure 1: MD5 collision generation from a prefix

We can check whether the output files are distinct or not using the diff command. We can also use the md5sum command to check the MD5 hash of each output file. See the following commands.

```
$ diff out1.bin out2.bin
$ md5sum out1.bin
$ md5sum out2.bin
```

– Question 1. If the length of your prefix file is not multiple of 64, what is going to happen?
Answer: let's say we have 50 bytes and use for prefix.txt with that number which is less than 64. After running the commands we will see our output file is padded with extra zeros to make it 64 bytes.
As you could see in the image below:




```
seed@VM:~$ echo $(python3 -c 'print("\x42"*50)') > prefix.txt
seed@VM:~$ 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: 9f52f1281b667f0e1d05729e2269afd8

Generating first block: ............
Generating second block: S00.....
Running time: 16.6777 s
seed@VM:~$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
seed@VM:~$ md5sum out1.bin
1681bbddffb1d3a7890e264132ad20de  out1.bin
seed@VM:~$ md5sum out2.bin
1681bbddffb1d3a7890e264132ad20de  out2.bin
```

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

Answer: If we create a prefix file with exactly 64 bytes and run the collision tool again. We will see that it has not added any extra padding to it.

See screen shot below:

```
seed@VM:~$ echo $(python3 -c 'print("\x42"*63)') > prefix_64
seed@VM:~$ md5collgen  -p prefix 64 -o out1 64.bin out2 64.bin
```

```
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1_64.bin' and 'out2_64.bin'
Using prefixfile: 'prefix_64'
Using initial value: 1a411e7066d64ae6b60256017d130eda

Generating first block: ..
Generating second block: W...............................
seed@VM:~$ xxd out1_64.bin
4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
4242 4242 4242 4242 4242 4242 4242 4242  BBBBBBBBBBBBBBBB
4242 4242 4242 4242 4242 4242 4242 420a  BBBBBBBBBBBBBBB.
8f32 dac8 f8aa f756 ad51 7e73 ec88 5a3c  .2.....V.Q~s..Z<
f3c6 061d 4cf9 269b 30be 7951 92cb 3802  ....L.&.0.yQ..8.
79f5 01c1 d0a5 9f89 cbfb 4032 174e be6a  y.........@2.N.j
dabf d82b ba51 7d4a 4108 6668 8f29 c694  ...+.Q}JA.fh.)..
f280 18bc b917 6323 03bf cb34 97ce 61a4  ......c#...4..a.
4f9d 5aa8 480f 81ec 182d 9b76 ca2a fece  O.Z.H....-.v.*..
fd8f b7d5 da6f 3688 f656 093a 981c 961f  .....o6..V.:....
db74 c87f 7ac4 b33b cf5d 40dd 4df9 6f05  .t..z..;.]@.M.o.
seed@VM:~$
```

– 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: we ran the following commands and find that there are few differences. We suspect this may cause a collision error, but we shall see.

As you could see below:

```
seed@VM:~$ echo $(python3 -c 'print("\x43"*127)') > prefix_128
seed@VM:~$ cat prefix_128
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
seed@VM:~$ md5collgen -p prefix_128 -o out1_128.bin out2_128.bin
```

```
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1_128.bin' and 'out2_128.bin'
Using prefixfile: 'prefix_128'
Using initial value: 0dca8edb6993206e503fcbb81e110dac

Generating first block: ...
Generating second block: S10.....
Running time: 2.73359 s
seed@VM:~$ xxd out1_128.bin > o.txt
seed@VM:~$ xxd out2_128.bin > p.txt
seed@VM:~$ diff o.txt p.txt
10,12c10,12
< 00000090: 2859 2b60 db01 fb00 35da 800d 0925 3cdd  (Y+`....5....%<.
< 000000a0: 66d1 b3ea 5ec5 1b92 eb1b d4bc 1c7c 016c  f...^........|.l
< 000000b0: 818a a7df eb44 b7a2 be6c 4e68 8929 a2a0  .....D...lNh.)..
---
> 00000090: 2859 2be0 db01 fb00 35da 800d 0925 3cdd  (Y+.....5....%<.
> 000000a0: 66d1 b3ea 5ec5 1b92 eb1b d4bc 1cfc 016c  f...^..........l
> 000000b0: 818a a7df eb44 b7a2 be6c 4ee8 8929 a2a0  .....D...lN..)..
14,16c14,16
< 000000d0: 2fa8 c677 601b 9321 a9b3 1050 1b60 48d8  /..w`..!...P.`H.
< 000000e0: d755 5858 9f87 81fc 3b44 cc5a 5938 0e61  .UXX....;D.ZY8.a
< 000000f0: ac06 8f24 c93d 4124 f061 947d 98ca 217c  ...$.=A$.a.}..!|
---
> 000000d0: 2fa8 c6f7 601b 9321 a9b3 1050 1b60 48d8  /...`..!...P.`H.
> 000000e0: d755 5858 9f87 81fc 3b44 cc5a 59b8 0d61  .UXX....;D.ZY..a
> 000000f0: ac06 8f24 c93d 4124 f061 94fd 98ca 217c  ...$.=A$.a....!|
```

2.2 Task 2: Understanding MD5's Property

To achieve this task of understanding MD5's property, we create a prefix_t2 with "`cpssixthreet`" and suffix_t2 "`63306330C`" respectively. By doing md5collgen on prefix_t2 to create t2_1 and t2_2 to test that the two different files have same hash values. We tested the two strings created above as pairs. We tested string t2_1with suffix_t2 together and t2_2 with suffix_t2 and created two more files t2_1_done and t2_2_done. We tested if they were same or different but should have the same hash values. It proved to the property of MD5's.
As we can observe our test below:

```
seed@VM:~$ echo "cpssixthreet" > prefix_t2
seed@VM:~$ md5collgen -p prefix_t2 -o t2_1 t2_2
```

```
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 't2_1' and 't2_2'
Using prefixfile: 'prefix_t2'
Using initial value: 350afad547aef02777454b8a43b91311

Generating first block: ..
Generating second block: W..
Running time: 4.94848 s
seed@VM:~$ md5sum t2_1
8895134948b0f2488724af291649a1ae  t2_1
seed@VM:~$ md5sum t2_2
8895134948b0f2488724af291649a1ae  t2_2
seed@VM:~$ diff t2_1 t2_2
Binary files t2_1 and t2_2 differ
seed@VM:~$ echo "63306330C" > suffix_t2
seed@VM:~$ cat t2_1 suffix_t2 > t2_1_done
seed@VM:~$ cat t2_2 suffix_t2 > t2_2_done
seed@VM:~$ md5sum t2_1_done
107a30cda23b731846d83715b9503fcf  t2_1_done
seed@VM:~$ md5sum t2_2_done
107a30cda23b731846d83715b9503fcf  t2_2_done
seed@VM:~$ diff t2_1_done t2_2_done
Binary files t2_1_done and t2_2_done differ
```

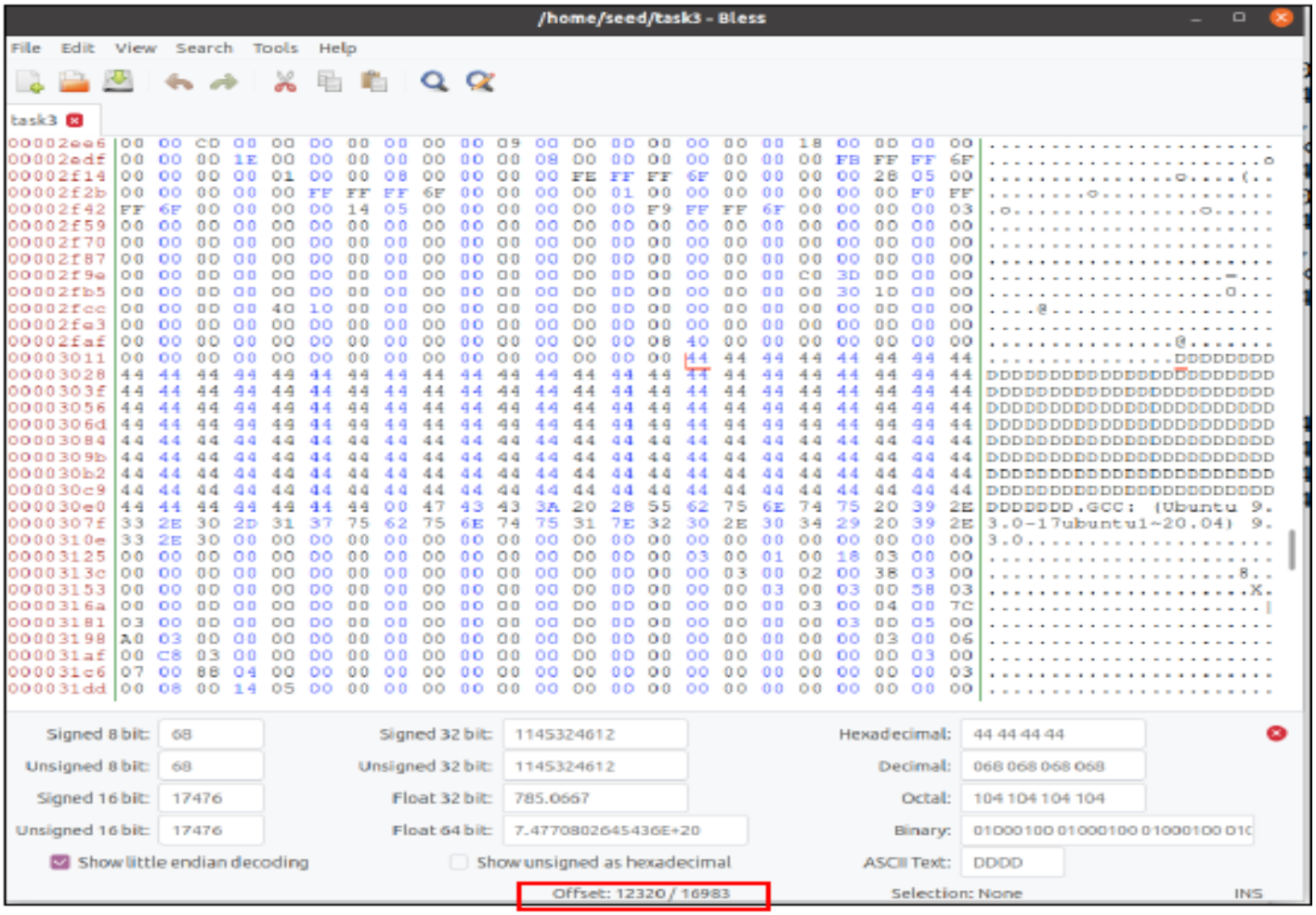
2.3 Task 3: Generating Two Executable Files with the Same MD5 Hash

For this task we created the two executable files with same MD5 hash to perform our findings.
To achieve our task, first we created a file name task3.c with array of char 200 and save the output in D.txt file.

Open task3.c Save

```c
#include <stdio.h>
unsigned char xyz[200] = {
0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x4
/* The actual contents of this array are up to you */
};
int main()
{
int i;
for (i=0; i<200; i++){
printf("%x", xyz[i]);
}
printf("\n");
}
```

```
seed@VM:~$ echo $(python3 -c 'print("0x44,"*199)') > D.txt
seed@VM:~$ cat D.txt
0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x
44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44
,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0
x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x4
4,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,
0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x
44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44
,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0
x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x4
4,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,
0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,0x44,
seed@VM:~$ gedit task3.c
seed@VM:~$ gcc -o task3 task3.c
seed@VM:~$ ./task3
444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444440
seed@VM:~$ bless task3
```



From the above file task.3 bless you see that the offset for the D is 12320. According to the tile the prefix length must be an integer multiple of 64, we get after calculating 12320 mod 64 = 32, and add it to the offset 12320+32=12352 and makes the prefix_t3 of 128 D that followed by suffix and we get suffix's offset that is 12352+128=12480. We get our p and q values here.

When we finish prefix_t3 and suffix_t3, we use prefix_t3 to generate task3_1.bin and task3_2.bin. After generation of the two files we take separately 128 bytes from the end of each file to get as p and q values.

We only need to put prefix_t3 p|q suffix_t3 string together to get the two file. Then we verify the MD5 has value and execute it with chmod +x <file> that will help us to confirm the differences among them.

Check below for the screen shots of the process:

```
seed@VM:~$ head -c 12352 task3 > prefix_t3
seed@VM:~$ tail -c +12480 task3 > suffix_t3
seed@VM:~$ md5collgen -p prefix_t3 -o task3_1.bin task3_2.bin
```

```
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'task3_1.bin' and 'task3_2.bin'
Using prefixfile: 'prefix_t3'
Using initial value: 1b9ce8f65b339765c5158f11cc5e8be2

Generating first block: ...............
Generating second block: W...
seed@VM:~$ tail -c 128 task3_1.bin > p
seed@VM:~$ tail -c 128 task3_2.bin > q
seed@VM:~$ cat prefix_t3 p suffix_t3 > t3_1_done
seed@VM:~$ cat prefix_t3 q suffix_t3 > t3_2_done
seed@VM:~$ md5sum t3_1_done
ca789cd0ce72113e79f59e08fd170a2f  t3_1_done
seed@VM:~$ md5sum t3_2_done
ca789cd0ce72113e79f59e08fd170a2f  t3_2_done
seed@VM:~$ chmod +x t3_1_done
seed@VM:~$ ./t3_1_done
4444444444444444444444444444444444444444444444444444444444444444444467514373123 97f3045f636
f18b9d9f48c8522b69f7d979b21214a9214115f7c4ee233260fe0d22121729cd4f767261b4f796c604be5d
5375e2b92de96661278fc8d36a214cdd0e93f35946c5e7d4d14c926d7ca983d8ec5473abed718d274289d9
6a3e16755f19db5835f7f58ab62ef2d36ed4444444444444444444444444444444444444444444444444
44444444444444444444
seed@VM:~$ chmod +x t3_2_done
seed@VM:~$ ./t3_2_done
4444444444444444444444444444444444444444444444444444444444444444444467514373123 97f3045f636
f18b9d9f4c8522b69f7d979b21214a9214115f7c4ee233260fe0d22121f29cd4f767261b4f796c604be5de
375e2b92de96661278fc8d36a214cdd0e93f359c6c5e7d4d14c926d7ca983d8ec5473abed718d274289d97
93e16755f19db5835f7f58abe2ef2d36ed4444444444444444444444444444444444444444444444444
44444444444444444444
```

2.4 Task 4: Making the Two Programs Behave Differently

In our approach, we create two arrays X and Y. We compare the contents of these two arrays; if they are the same, the benign code is executed; otherwise, the malicious code is executed.
We create code.c program and make our judgement from there. If an array X and Y are same then could will begin execution otherwise it's a malicious code.

CPS633 Fall2021 Section 07 Group 04 Name: Tusaif Azmat - 500660278

Name: Ankit Sodhi - 500958004

```c
#include <stdio.h>
unsigned char x[200] =
{0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x
unsigned char y[200] =
{0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x
int main() {
  int i;
  for (i=0; i<200; i++){
    if(x[i] != y[i]){ break; }
  }

  if(i == 200){ printf("%s", "benign code"); } /* x = y */
  else{ printf("%s", "WARNING: malicious code"); } /* x != y */

  printf("\n");
```

/home/seed/code - Bless

File Edit View Search Tools Help

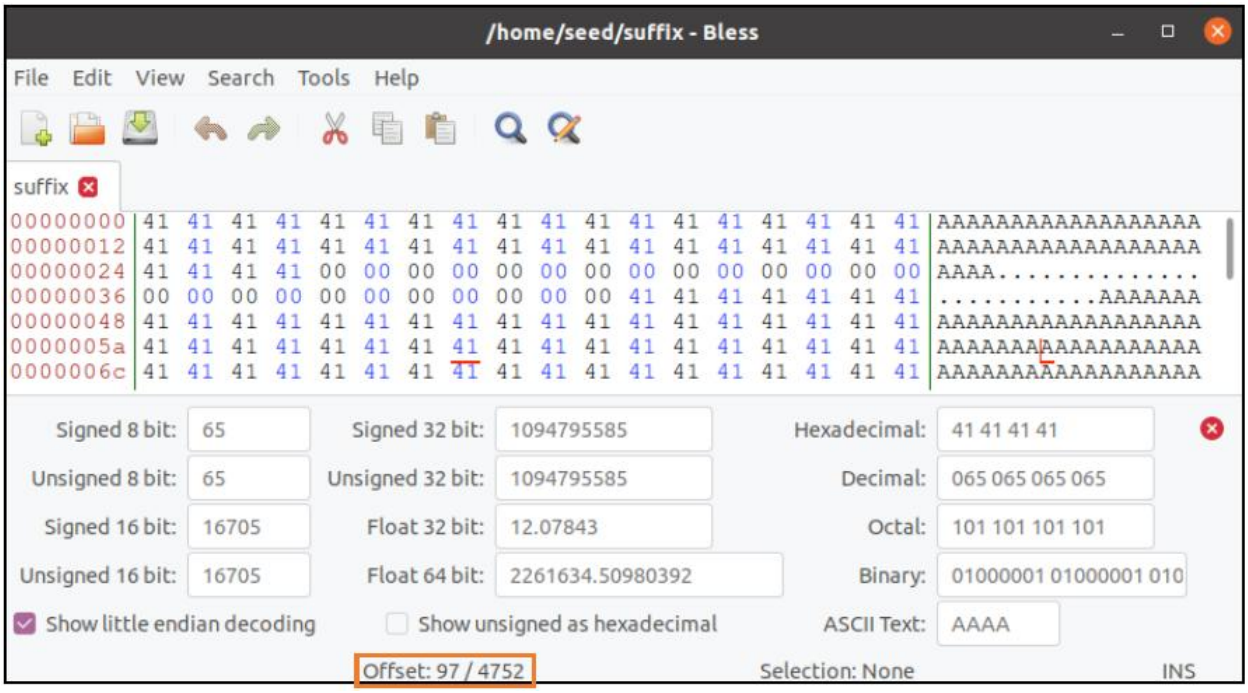
suffix code

```
00002f80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....................
00002f94 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....................
00002fa8 00 00 00 00 00 00 00 00 C0 3D 00 00 00 00 00 00 00 00 00 00 .........=..........
00002fbc 00 00 00 00 00 00 00 00 00 00 00 00 30 10 00 00 00 00 00 00 ............0.......
00002fd0 40 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @...................
00002fe4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....................
00002f8f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08 40 00 00 ..................@..
0000300c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....................
00003020 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAA
00003034 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAA
00003048 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAA
0000305c 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAA
00003070 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAA
```

Signed 8 bit:	65	Signed 32 bit:	1094795585	Hexadecimal:	41 41 41 41
Unsigned 8 bit:	65	Unsigned 32 bit:	1094795585	Decimal:	065 065 065 065
Signed 16 bit:	16705	Float 32 bit:	12.07843	Octal:	101 101 101 101
Unsigned 16 bit:	16705	Float 64 bit:	2261634.50980392	Binary:	01000001 01000001 010000
				ASCII Text:	AAAA

Show little endian decoding Show unsigned as hexadecimal

Offset: 12320 / 17231 Selection: None INS

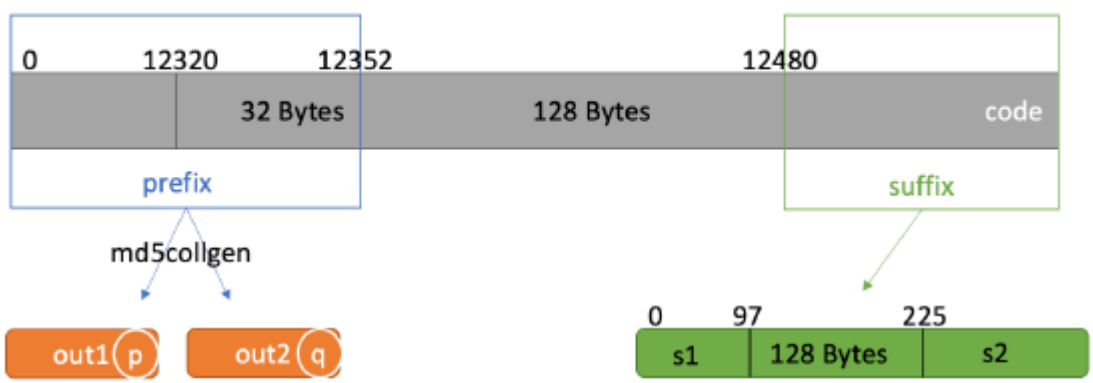


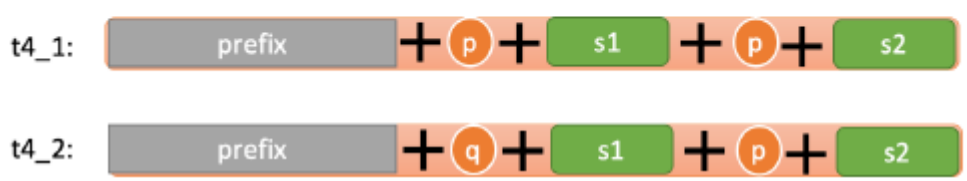
Our Goal is to do as follows:

| 12352 | p | 64 | p | 4512 | > task4_1

| 12352 | q | 64 | p | 4512 | > task4_2

As can been seen below:



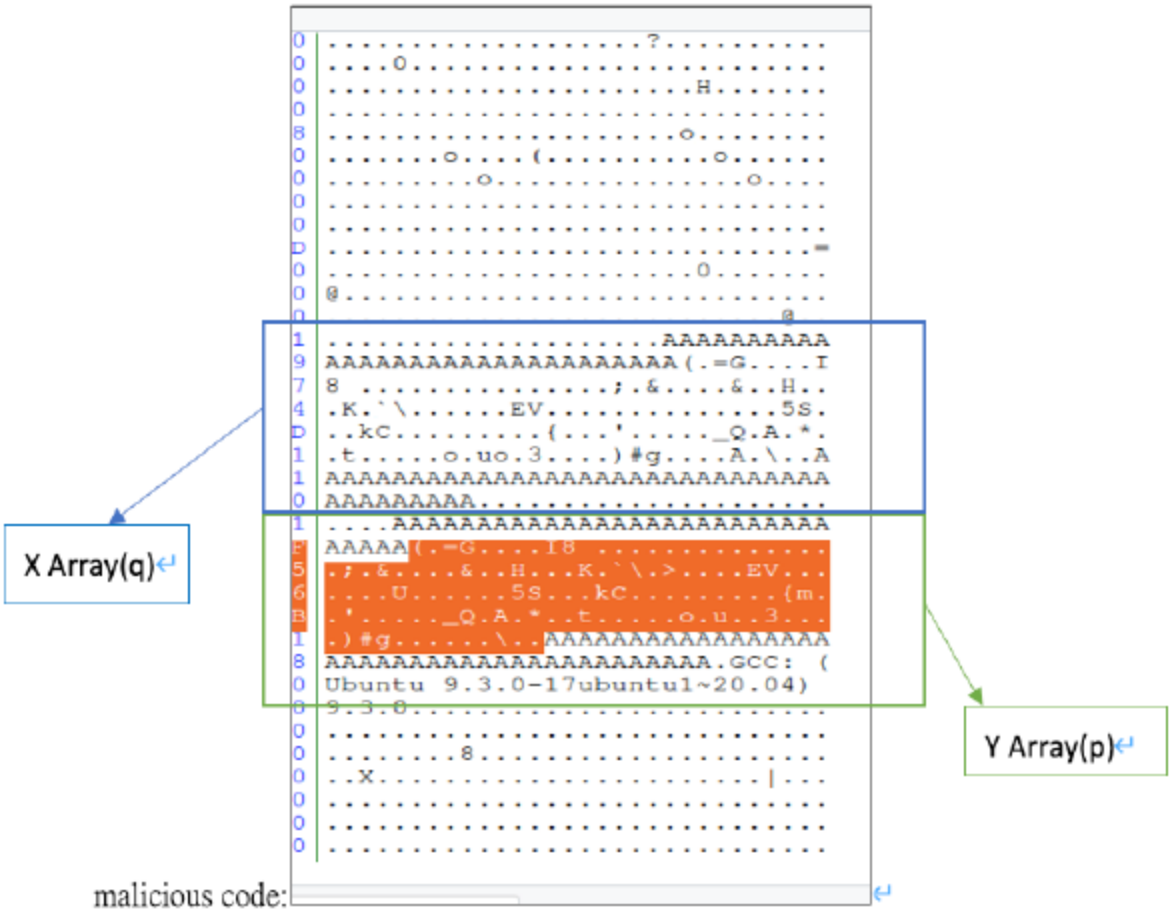


X Array(p)

```
.................................
.........................o.......
........o....(..........o......
.........o................o.....
.................................
.................................
...........................=....
....................0.......@....
.................................
.......................@.........
..............AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA(.=G....I8 .......
........;.&....&..H...K.`\.>...
.EV.......U......5S...kC.......
..(m..'.....__Q.A.*..t.....o.u..
3....)#g......\..AAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAA.....
.....................AAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA(.=G....I8
................;.&....&..H...K.
`\.>....EV.......U......5S...kC
.........(m..'.....__Q.A.*..t...
..o.u..3....)#g......\..AAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA.GCC: (Ubuntu 9.3.0-17ubuntu1
~20.04) 9.3.0..................
.................................
...............8................
.......X........................
|................................
.................................
.................................
..............................(...
```

Y Array(p)

benign code:



This is how we conclude the task 4:

```
seed@VM:~$ head -c 12351 code > prefix
seed@VM:~$ tail -c +12480 code > suffix
seed@VM:~$ md5collgen -p prefix -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'
Using initial value: 4dcfaeb9b0a401ded87aaf3261645348

Generating first block: .
Generating second block: S11.................
Running time: 2.45297 s
```

```
seed@VM:~$ tail -c 128 out1 > p
seed@VM:~$ tail -c 128 out2 > q
seed@VM:~$ head -c 97 suffix > s1
seed@VM:~$ tail -c +225 suffix > s2
seed@VM:~$ cat prefix p s1 p s2 > t4_1
seed@VM:~$ cat prefix q s1 p s2 > t4_2
seed@VM:~$ chmod +x t4_1
seed@VM:~$ chmod +x t4_2
seed@VM:~$ ./t4_1
benign code
seed@VM:~$ ./t4_2
WARNING: malicious code
```

Hence, the “WARNING: malicious code” proves that the two programs behaved differently