MD5 Collision Attack Lab — A Cryptographic Security SEEDLab

Shruti Vinodh Follow
Jun 21, 2018 · 6 min read

Introduction

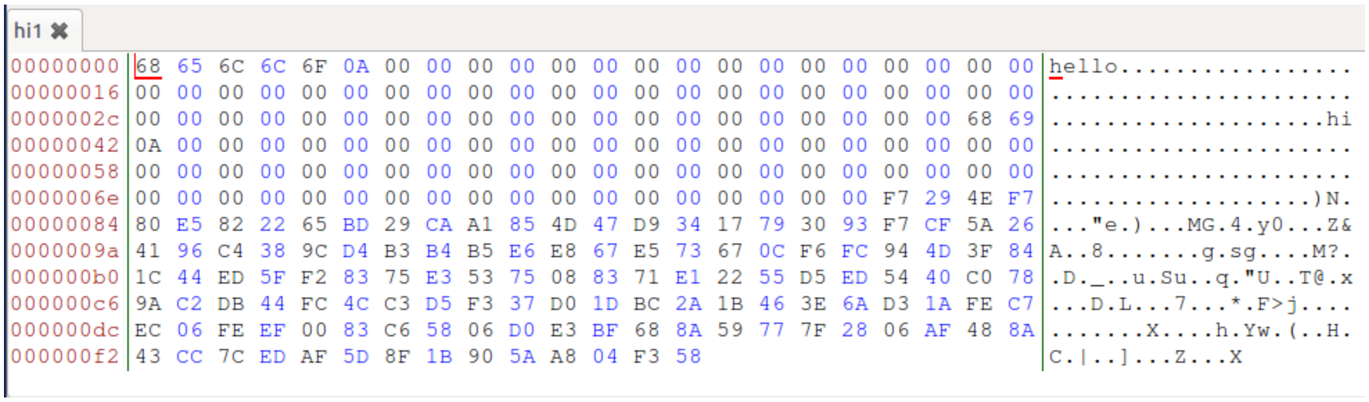
This is a guide for the SEEDLab MD5 Collision Attack Lab. This lab delves into the MD5 collision attack which makes use of its length extension property.

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

We do this by using `md5collgen -p PREFIX_FILE -o OUTPUTFILE1 OUTPUTFILE2` .

> *Question 1: If the length of your prefix file is not a multiple of 64, what is going to happen?*
>
> *Answer: It will be padded with zeros.*

To test this out, I created a file **hi.txt** and truncated it using `truncate -s YOUR_DESIRED_SIZE hi.txt` . After running `md5collgen -p hi.txt -o hi1 hi2` looking at the results using `bless hi1` , we can see that it has been padded with zeros. This is because MD5 processes blocks of size 64 bytes.

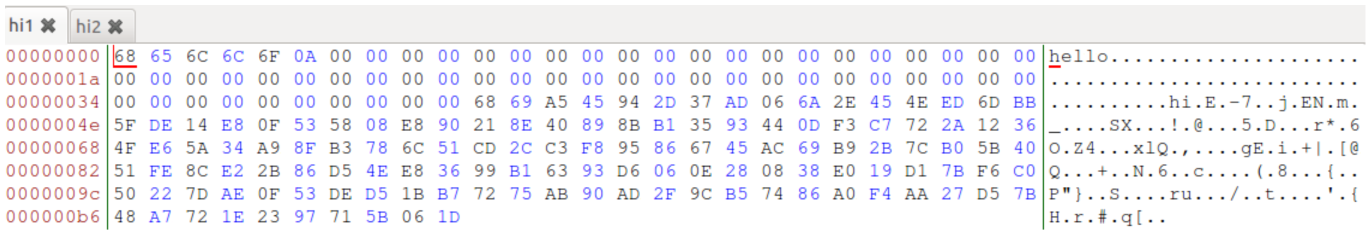


```
00000000 68 65 6C 6C 6F 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 hello.................
00000016 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
0000002c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 68 69 ....................hi
00000042 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
00000058 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......................
0000006e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 F7 29 4E F7 ....................)N.
00000084 80 E5 82 22 65 BD 29 CA A1 85 4D 47 D9 34 17 79 30 93 F7 CF 5A 26 ..."e.)...MG.4.y0...Z&
0000009a 41 96 C4 38 9C D4 B3 B4 B5 E6 E8 67 E5 73 67 0C F6 FC 94 4D 3F 84 A..8.......g.sg....M?.
000000b0 1C 44 ED 5F F2 83 75 E3 53 75 08 83 71 E1 22 55 D5 ED 54 40 C0 78 .D._..u.Su..q."U..T@.x
000000c6 9A C2 DB 44 FC 4C C3 D5 F3 37 D0 1D BC 2A 1B 46 3E 6A D3 1A FE C7 ...D.L...7...*.F>j....
000000dc EC 06 FE EF 00 83 C6 58 06 D0 E3 BF 68 8A 59 77 7F 28 06 AF 48 8A .......X....h.Yw.(..H.
000000f2 43 CC 7C ED AF 5D 8F 1B 90 5A A8 04 F3 58                         C.|..]...Z...X
```

Note the zero padding. (The word "hi" was concatenated to hi.txt to mark the end of the file. hi.txt took up 67 bytes.)

> *Question 2: Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.*
>
> *Answer: No zero padding is observed.*

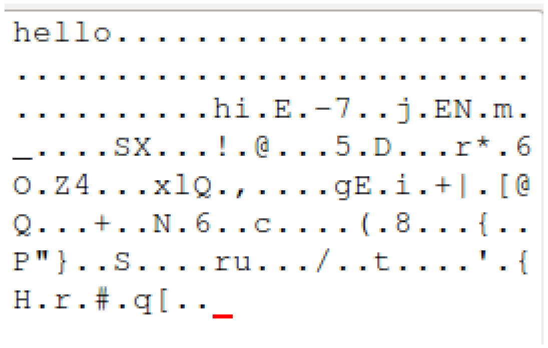
To see this in action, create a file and use `truncate -s 64 YOUR_FILE` . Run `md5collgen -p hi -o hi1 hi2` and view the output (`bless hi1 hi2``).



```
00000000 68 65 6C 6C 6F 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 hello.....................
0000001a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..........................
00000034 00 00 00 00 00 00 00 00 00 00 68 69 A5 45 94 2D 37 AD 06 6A 2E 45 4E ED 6D BB ..........hi.E.-7..j.EN.m.
0000004e 5F DE 14 E8 0F 53 58 08 E8 90 21 8E 40 89 8B B1 35 93 44 0D F3 C7 72 2A 12 36 _.....SX...!.@...5.D...r*.6
00000068 4F E6 5A 34 A9 8F B3 78 6C 51 CD 2C C3 F8 95 86 67 45 AC 69 B9 2B 7C B0 5B 40 O.Z4...xlQ.,....gE.i.+|.[@
00000082 51 FE 8C E2 2B 86 D5 4E E8 36 99 B1 63 93 D6 06 0E 28 08 38 E0 19 D1 7B F6 C0 Q...+..N.6..c....(.8...{..
0000009c 50 22 7D AE 0F 53 DE D5 1B B7 72 75 AB 90 AD 2F 9C B5 74 86 A0 F4 AA 27 D5 7B P"}..S....ru.../..t....'.{
000000b6 48 A7 72 1E 23 97 71 5B 06 1D                                                 H.r.#.q[..
```

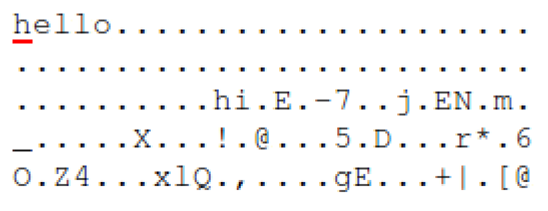
No zero padding observed.

> *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: No, not all bytes are different. In previous case, the bytes only differ at the positions 20, 60, 84 and 110. After multiple trials, it is noted that these differences are not constant.*

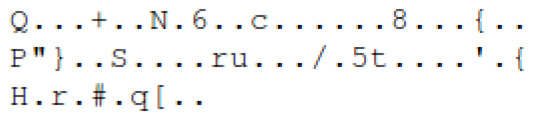


```
hello.....................
..........................
..........hi.E.-7..j.EN.m.
_.....SX...!.@...5.D...r*.6
O.Z4...xlQ.,....gE.i.+|.[@
Q...+..N.6..c....(.8...{..
P"}..S....ru.../..t....'.{
H.r.#.q[..
```

hi1



```
hello.....................
..........................
..........hi.E.-7..j.EN.m.
_......X...!.@...5.D...r*.6
O.Z4...xlQ.,....gE...+|.[@
```



```
Q...+..N.6..c......8...{..
P"}..S....ru.../.5t....'.{
H.r.#.q[..
```

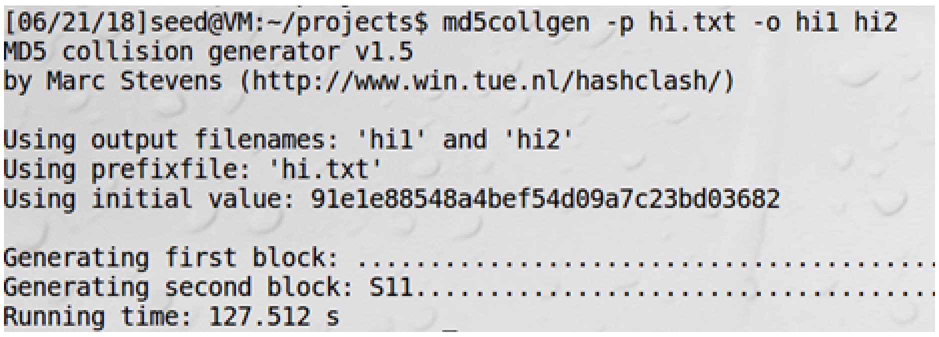
hi2

2.2 Task 2: Understanding MD5's Property

One interesting point is that you can append the same file to the outputs generated by md5collgen and while it changes the MD5 hash, both the new hashes will be identical. This is because many hash functions like SHA-1, SHA-2 and MD5 are subject to length extension. The hash function has an internal state and processes the message in fixed blocks by applying a compression function to the current state and the current block.

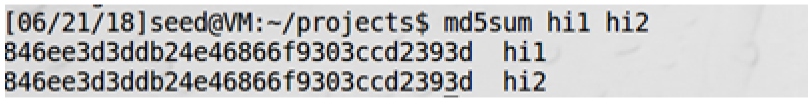
Since the MD5 hashes of both the files were the same, it is safe to assume that the internal state after the algorithm has been run was the same. Let us call this internal state *s*. So, after appending a file there will be a stage which involves running the compression algorithm on *s* and the current block, which would be identical for both modified files. This would result in the same MD5 hash for both files.

To test this create a file **hi.txt** and run `md5collgen -p hi.txt -o hi1 hi2`.



```
[06/21/18]seed@VM:~/projects$ md5collgen -p hi.txt -o hi1 hi2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'hi1' and 'hi2'
Using prefixfile: 'hi.txt'
Using initial value: 91e1e88548a4bef54d09a7c23bd03682

Generating first block: .....................................
Generating second block: S11.................................
Running time: 127.512 s
```

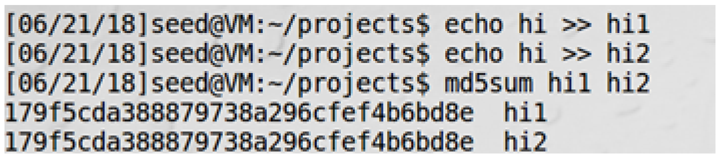
Verify that the MD5 hashes are the same using `md5sum hi1 hi2`.



```
[06/21/18]seed@VM:~/projects$ md5sum hi1 hi2
846ee3d3ddb24e46866f9303ccd2393d  hi1
846ee3d3ddb24e46866f9303ccd2393d  hi2
```

Now let's append a random string to the end of both files and check the MD5 hashes of both files again.

```
echo hi >> hi1
echo hi >> hi2
md5sum hi1 hi2
```



```
[06/21/18]seed@VM:~/projects$ echo hi >> hi1
[06/21/18]seed@VM:~/projects$ echo hi >> hi2
[06/21/18]seed@VM:~/projects$ md5sum hi1 hi2
179f5cda388879738a296cfef4b6bd8e  hi1
179f5cda388879738a296cfef4b6bd8e  hi2
```

We can see that the MD5 hashes remain identical.

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

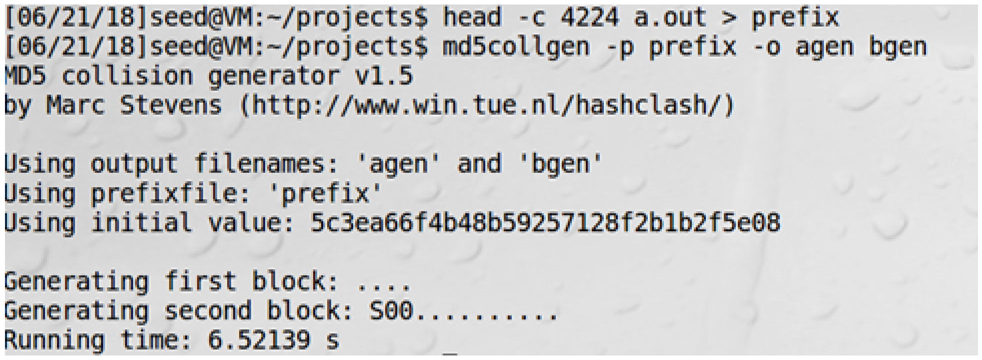
Save the following code in a file **program.c.**

```c
#include <stdio.h>

unsigned char a[200] = { 'H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A'
int main()
{
        int i;
        a[199]='D';
        a[198]='N';
        a[197]='E';
        for(i = 0; i < 200; i++)
        {
                printf("%x", a[i]);
        }
        printf("\n");
}
```

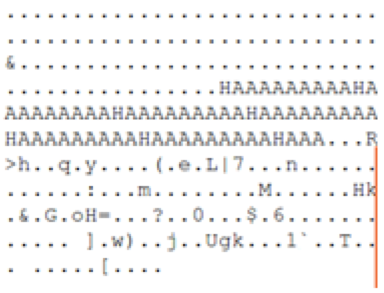
program.c hosted with ♡ by GitHub view raw

Compile the above with `gcc program.c -o program.out` . The location where the array values are written can easily be identified. **0x1040** works out to be **4160** in decimal. This is divisible by 64 but for now we'll select a prefix length of 4224 bytes (**0x1080**). *(This is not necessary, but I wanted to have the split in the middle of the array.)*

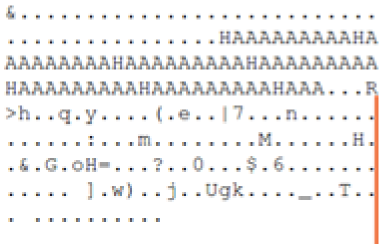
Running `head -c 4224 a.out > prefix` and `md5collgen -p prefix -o agen bgen` ,


```
[06/21/18]seed@VM:~/projects$ head -c 4224 a.out > prefix
[06/21/18]seed@VM:~/projects$ md5collgen -p prefix -o agen bgen
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'agen' and 'bgen'
Using prefixfile: 'prefix'
Using initial value: 5c3ea66f4b48b59257128f2b1b2f5e08

Generating first block: ....
Generating second block: S00.........
Running time: 6.52139 s
```

Now we have two files with the same MD5 hash but different suffixes. Looking at both **agen** and **bgen** in bless (`bless agen bgen`).


```
............................
............................
&...........................
................HAAAAAAAAAAHA
AAAAAAAAHAAAAAAAAAAHAAAAAAAAAA
HAAAAAAAAAAHAAAAAAAAAAAHAAA...R
>h..q.y....(.e.L|7...n......
......:...m.........M......Hk
.&.G.oH=...?..0...$.6.......
.....  ].w)..j..Ugk...1`..T..
.  .....[....
```

A part of agen


```
&...........................
................HAAAAAAAAAAHA
AAAAAAAAHAAAAAAAAAAHAAAAAAAAAA
HAAAAAAAAAAHAAAAAAAAAAAHAAA...R
>h..q.y....(.e..|7...n......
......:...m.........M......H.
.&.G.oH=...?..0...$.6.......
.....  ].w)..j..Ugk...._..T..
.  ..........
```

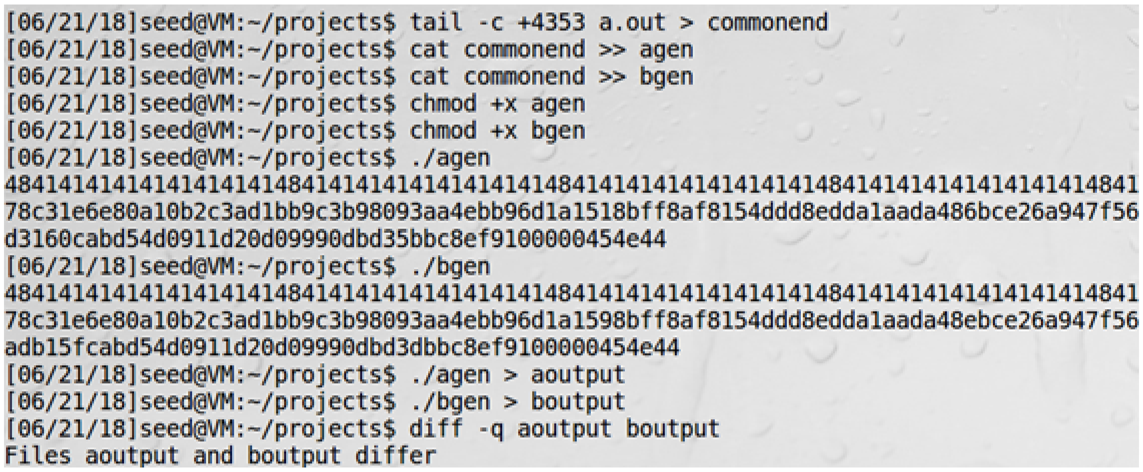
A portion of bgen

Now we will get the common end to be appended using `tail -c 4353 a.out > commonend`.

Put them together by running,

```
cat commonend >> agen
cat commonend >> bgen
```

Add executable permission to both files and run them. Note that the output differs.



```
[06/21/18]seed@VM:~/projects$ tail -c +4353 a.out > commonend
[06/21/18]seed@VM:~/projects$ cat commonend >> agen
[06/21/18]seed@VM:~/projects$ cat commonend >> bgen
[06/21/18]seed@VM:~/projects$ chmod +x agen
[06/21/18]seed@VM:~/projects$ chmod +x bgen
[06/21/18]seed@VM:~/projects$ ./agen
484141414141414141414148414141414141414141414841414141414141414141414841414141414141414141414841
78c31e6e80a10b2c3ad1bb9c3b98093aa4ebb96d1a1518bff8af8154ddd8edda1aada486bce26a947f56
d3160cabd54d0911d20d09990dbd35bbc8ef9100000454e44
[06/21/18]seed@VM:~/projects$ ./bgen
484141414141414141414148414141414141414141414841414141414141414141414841414141414141414141414841
78c31e6e80a10b2c3ad1bb9c3b98093aa4ebb96d1a1598bff8af8154ddd8edda1aada48ebce26a947f56
adb15fcabd54d0911d20d09990dbd3dbbc8ef9100000454e44
[06/21/18]seed@VM:~/projects$ ./agen > aoutput
[06/21/18]seed@VM:~/projects$ ./bgen > boutput
[06/21/18]seed@VM:~/projects$ diff -q aoutput boutput
Files aoutput and boutput differ
```

2.4 Task 4: Making the Two Programs Behave Differently

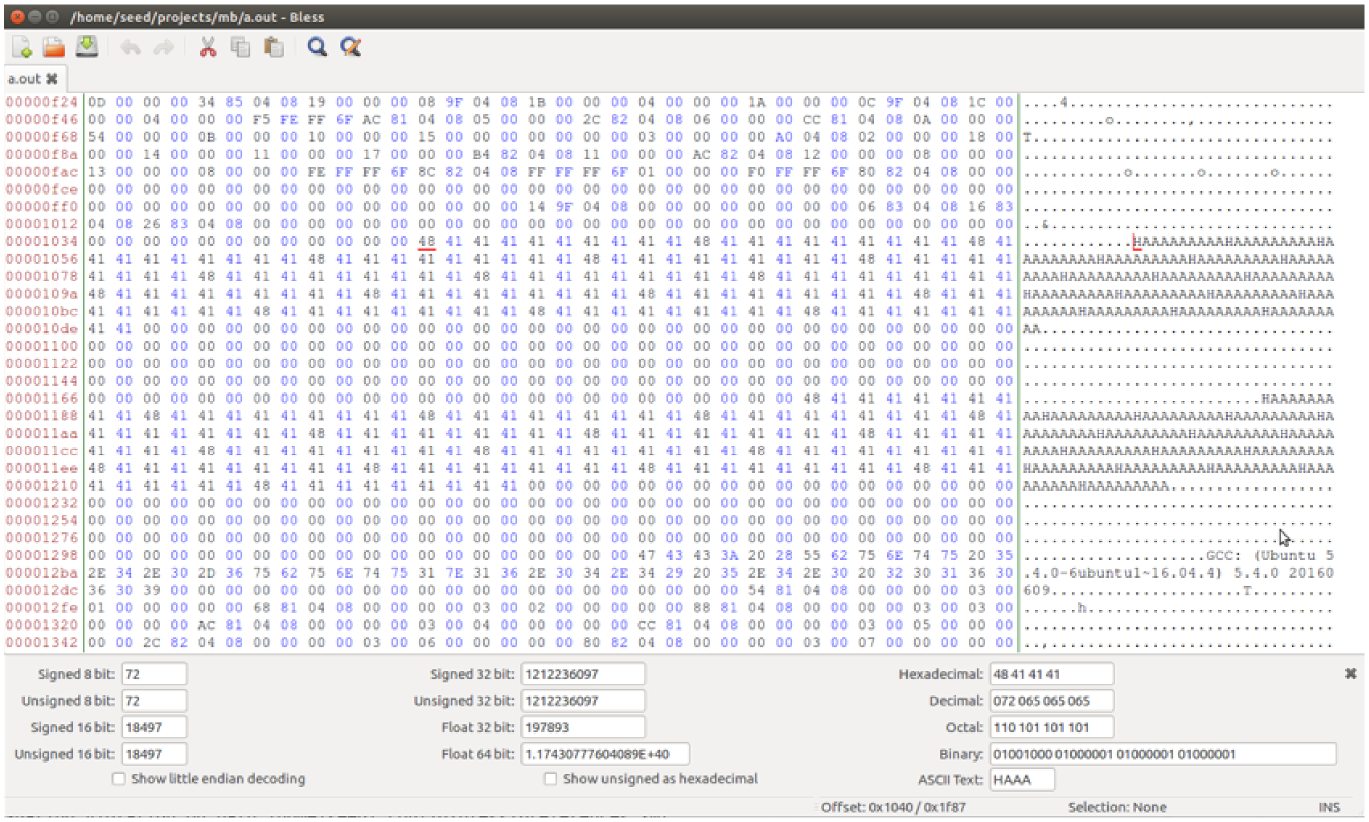
For this, we have to create two programs which have the same MD5 sum, but behave differently. In this example, we will make the programs simply print different statements, however, practically speaking, the second program can execute some malicious code.

This is the program we will be using. Copy the contents and save it as **program.c.**

```c
#include <stdio.h>
unsigned char a[300] = { 'H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A'
unsigned char b[300] = { 'H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A'
int main()
{
	int i;
	int isSame=1;
	for(i = 0; i < 200; i++)
	{
		if(a[i]!=b[i])
			isSame=0;
	}
	if(isSame)
		printf("This would run the safe code and display the intended behaviour");
	else
		printf("This is where malicious code would be run");
	printf("\n");
}
```

program.c hosted with ♡ by GitHub view raw

Now we compile the program using `gcc program.c`. The output binary is stored by default as **a.out**. Looking at the contents of a.out using `bless a.out`.



```
/home/seed/projects/mb/a.out - Bless
a.out
00000f24 0D 00 00 00 34 85 04 08 19 00 00 00 08 9F 04 08 1B 00 00 00 04 00 00 00 1A 00 00 00 0C 9F 04 08 1C 00 ....4.............................
00000f46 00 00 04 00 00 00 F5 FE FF 6F AC 81 04 08 05 00 00 00 2C 82 04 08 06 00 00 00 CC 81 04 08 0A 00 00 00 .........o.......,................
00000f68 54 00 00 00 0B 00 00 00 10 00 00 00 15 00 00 00 00 00 00 00 03 00 00 00 00 A0 04 08 02 00 00 00 18 00 T.................................
00000f8a 00 00 14 00 00 00 11 00 00 00 17 00 00 00 B4 82 04 08 11 00 00 00 AC 82 04 08 12 00 00 00 08 00 00 00 ..................................
00000fac 13 00 00 00 08 00 00 00 FE FF FF 6F 8C 82 04 08 FF FF FF 6F 01 00 00 00 F0 FF FF 6F 80 82 04 08 00 00 ...........o.......o.......o......
00000fce 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................................
00000ff0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 9F 04 08 00 00 00 00 00 00 00 00 06 83 04 08 16 83 ..................................
00001012 04 08 26 83 04 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..&...............................
00001034 00 00 00 00 00 00 00 00 00 00 00 00 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 ............HAAAAAAAAAHAAAAAAAAAHA
00001056 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 AAAAAAAAHAAAAAAAAAHAAAAAAAAAHAAAAA
00001078 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 AAAAHAAAAAAAAAHAAAAAAAAAHAAAAAAAAA
0000109a 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 HAAAAAAAAAHAAAAAAAAAHAAAAAAAAAHAAA
000010bc 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 AAAAAAHAAAAAAAAAHAAAAAAAAAHAAAAAAA
000010de 41 41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AA................................
00001100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................................
00001122 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................................
00001144 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................................
00001166 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 48 41 41 41 41 41 41 41 ..........................HAAAAAAA
00001188 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 AAHAAAAAAAAAHAAAAAAAAAHAAAAAAAAAHA
000011aa 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 AAAAAAAAHAAAAAAAAAHAAAAAAAAAHAAAAA
000011cc 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 AAAAHAAAAAAAAAHAAAAAAAAAHAAAAAAAAA
000011ee 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 48 41 41 41 HAAAAAAAAAHAAAAAAAAAHAAAAAAAAAHAAA
00001210 41 41 41 41 41 41 48 41 41 41 41 41 41 41 41 41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AAAAAAHAAAAAAAAA..................
00001232 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................................
00001254 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................................
00001276 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................................
00001298 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 47 43 43 3A 20 28 55 62 75 6E 74 75 20 35 ....................GCC: (Ubuntu 5
000012ba 2E 34 2E 30 2D 36 75 62 75 6E 74 75 31 7E 31 36 2E 30 34 2E 34 29 20 35 2E 34 2E 30 20 32 30 31 36 30 .4.0-6ubuntu1~16.04.4) 5.4.0 20160
000012dc 36 30 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 54 81 04 08 00 00 00 00 03 00 609.....................T.........
000012fe 01 00 00 00 00 00 68 81 04 08 00 00 00 00 03 00 02 00 00 00 00 00 88 81 04 08 00 00 00 00 03 00 03 00 ......h...........................
00001320 00 00 00 00 AC 81 04 08 00 00 00 00 03 00 04 00 00 00 00 00 CC 81 04 08 00 00 00 00 03 00 05 00 00 00 ..................................
00001342 00 00 2C 82 04 08 00 00 00 00 03 00 06 00 00 00 00 00 80 82 04 08 00 00 00 00 03 00 07 00 00 00 00 00 ..,...............................

  Signed 8 bit: 72            Signed 32 bit: 1212236097             Hexadecimal: 48 41 41 41
Unsigned 8 bit: 72          Unsigned 32 bit: 1212236097                 Decimal: 072 065 065 065
 Signed 16 bit: 18497          Float 32 bit: 197893                       Octal: 110 101 101 101
Unsigned 16 bit: 18497         Float 64 bit: 1.17430777604089E+40        Binary: 01001000 01000001 01000001 01000001
  ☐ Show little endian decoding   ☐ Show unsigned as hexadecimal      ASCII Text: HAAA
                                                   Offset: 0x1040 / 0x1f87      Selection: None      INS
```

bless a.out

We note that the first array starts at position **0x1040 = 4160.** As before, he prefix can extend until there, but I will continue to use the position **0x1080** so that the generated data will start in the middle of the array.

Since **0x1080** will refer to the first 4224 bytes of the data, we will run `head -c 4224 a.out > prefix` to store the first 4224 bytes in **prefix**. Then we will run `md5collgen -p prefix -o pgen qgen` which will generate two binaries with the same MD5 sum but they will differ.

Opening both in bless using `bless pgen qgen`.

We can see the 128 characters generated by md5collgen. The MD5 sums of **pgen** and **qgen** will be identical *(Verify this:* `md5sum pgen qgen` *)*

We can then select and create two files **p** and **q** having the generated values found in **pgen** and **qgen** respectively. To get the middle section note where the generated data begins in the first array. In the second instance of *"HAAAAAAAAAAHAAAA…"* (where the second area is located) we place a marker at where the data should be replaced and take it out. *(You can do this by creating a new file `head -c +4353 a.out > end` and selecting until the marker.)* The resulting **middle** should look like this.

To create the common **end** file, we would need the characters from the **end** file created earlier after accounting for a 128+192 *(size of **middle**)* offset. Do this by running `tail -c +321 end > commonend` .

Now to put all the files together. Run

```
cp pgen benigncode
cp qgen maliciouscode
cat middle >> benigncode
cat middle >> maliciouscode
cat p >> benigncode
cat p >> maliciouscode
cat commonend >> benigncode
cat commonend >> maliciouscode
```

Verify using `md5sum benigncode maliciouscode` and check if the hashes are the same. Also, make sure the size of **a.out**, **benigncode** and **maliciouscode** are the same.

Finally, run

```
chmod +x benigncode maliciouscode
./benigncode
./maliciouscode
```

. . .

Hash Length Extension:
https://www.eecs.umich.edu/courses/eecs398.f10/homework/eecs398-hw3-f10.pdf

Security Seedlab Md5 Cryptography Hashing