CSCI 476: Lab 10

Nathan Stouffer

April 18, 2020

Task 1

In Task 1, the MD5 collision attack is investigated. This is done by using the md5collgen tool already installed on the Virtual Machine.

Task 1A

For Task 1A, any prefix may be used. Figure 1 shows the command line process to for creating the hash collision. Using the prefix "secret message:)" the md5collgen tool generates two binary files with the same hash. However, using the diff command, Figure 1 shows that the two binary files are different. So out1.bin and out2.bin are a hash collision (hash(m_1) = hash(m_2) but $m_1 \neq m_2$).

Figure 1: Generating hash collision

Task 1B

In Task 1B, it is wondered what happens when the prefix file is not a multiple of 64. The prefix used in Task 1A is one such prefix, so it is certain a hash collision can be generated for such files. Now take a closer look at the contents of out1.bin and out2.bin (the two inputs that collide in the hash space). Figure 2 shows the contents of out1.bin while Figure 3 shows out2.bin. Note that the tool generating the collisions pads the prefix up to 64 bytes before appending the binary that forces the collision. Precisely 64 bytes are required because that is the input size for the underlying algorithm.

Figure 2: Contents of out1.bin

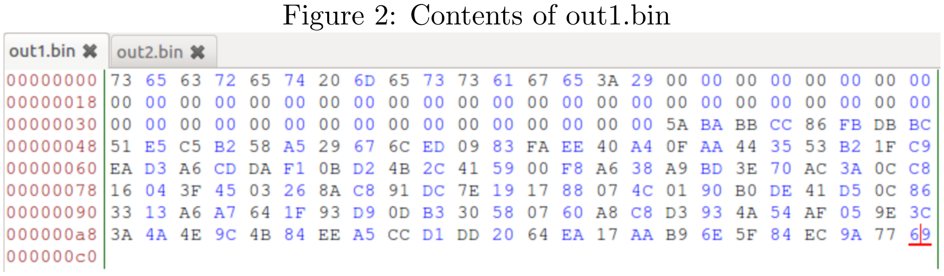


Figure 3: Contents of out2.bin

```
out1.bin  out2.bin
00000000 73 65 63 72 65 74 20 6D 65 73 73 61 67 65 3A 29 00 00 00 00 00 00 00 00
00000018 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5A BA BB CC 86 FB DB BC
00000048 51 E5 C5 B2 58 A5 29 67 6C ED 09 03 FA EE 40 A4 0F AA 44 35 53 B2 1F C9
00000060 EA D3 A6 CD DA F1 0B D2 4B 2C 41 59 00 78 A7 38 A9 BD 3E 70 AC 3A 0C C8
00000078 16 04 3F C5 03 26 8A C8 91 DC 7E 19 17 88 07 4C 01 90 B0 DE 41 D5 0C 86
00000090 33 13 A6 27 64 1F 93 D9 0D B3 30 58 07 60 A8 C8 D3 93 4A 54 AF 05 9E 3C
000000a8 3A 4A 4E 9C 4B 04 EE A5 CC D1 DD 20 64 EA 17 AA B9 6E 5F 04 EC 9A 77 69
000000c0
```

Task 1C

Task 1C asks that the experiment from Task 1B is repeated with a 64 byte block. The process of creating the hash collision and proving the two output files are different is shown in Figure 4. Then Figure 5 shows that their MD5 hashes collide.

Figure 4: Generating hash collision

```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/64-bytes
$ ls -l
total 4
-rw-rw-r-- 1 seed seed 64 Apr 15 09:18 prefix-64.txt

10.0.2.4 seed ~/Documents/comp-security/lab10/code/64-bytes
$ md5collgen -p prefix-64.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-64.txt'
Using initial value: b7820955b1304ef5d125be080b861780

Generating first block: ......
Generating second block: S01....
Running time: 8.3238 s

10.0.2.4 seed ~/Documents/comp-security/lab10/code/64-bytes
$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
```

Figure 5: Showing hash collision

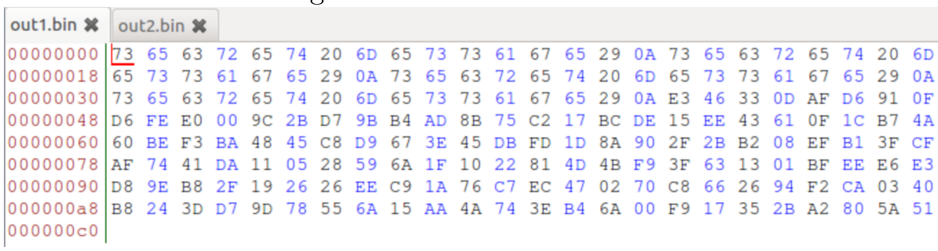
```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/64-bytes
$ md5sum out1.bin
18e7c1bd3a24afc26f9af39fa424d2c7  out1.bin

10.0.2.4 seed ~/Documents/comp-security/lab10/code/64-bytes
$ md5sum out2.bin
18e7c1bd3a24afc26f9af39fa424d2c7  out2.bin
```

It should be noted that there is no padding added in the case where the input is 64 bytes long (see Figures 6 and 7 for proof).

Task 1D

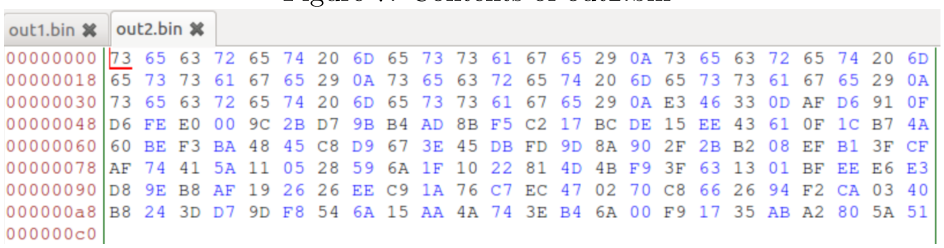
The respective 128 bytes appended to each file to create the collision are not entirely different. In fact, the bytes differ only by 7 bits. From the contents of each file displayed in Figures 6 and 7, the differences can be extracted.

Figure 6: Contents of out1.bin



```
out1.bin  out2.bin
00000000 73 65 63 72 65 74 20 6D 65 73 73 61 67 65 29 0A 73 65 63 72 65 74 20 6D
00000018 65 73 73 61 67 65 29 0A 73 65 63 72 65 74 20 6D 65 73 73 61 67 65 29 0A
00000030 73 65 63 72 65 74 20 6D 65 73 73 61 67 65 29 0A E3 46 33 0D AF D6 91 0F
00000048 D6 FE E0 00 9C 2B D7 9B B4 AD 8B 75 C2 17 BC DE 15 EE 43 61 0F 1C B7 4A
00000060 60 BE F3 BA 48 45 C8 D9 67 3E 45 DB FD 1D 8A 90 2F 2B B2 08 EF B1 3F CF
00000078 AF 74 41 DA 11 05 28 59 6A 1F 10 22 81 4D 4B F9 3F 63 13 01 BF EE E6 E3
00000090 D8 9E B8 2F 19 26 26 EE C9 1A 76 C7 EC 47 02 70 C8 66 26 94 F2 CA 03 40
000000a8 B8 24 3D D7 9D 78 55 6A 15 AA 4A 74 3E B4 6A 00 F9 17 35 2B A2 80 5A 51
000000c0
```

Figure 7: Contents of out2.bin



```
out1.bin  out2.bin
00000000 73 65 63 72 65 74 20 6D 65 73 73 61 67 65 29 0A 73 65 63 72 65 74 20 6D
00000018 65 73 73 61 67 65 29 0A 73 65 63 72 65 74 20 6D 65 73 73 61 67 65 29 0A
00000030 73 65 63 72 65 74 20 6D 65 73 73 61 67 65 29 0A E3 46 33 0D AF D6 91 0F
00000048 D6 FE E0 00 9C 2B D7 9B B4 AD 8B F5 C2 17 BC DE 15 EE 43 61 0F 1C B7 4A
00000060 60 BE F3 BA 48 45 C8 D9 67 3E 45 DB FD 9D 8A 90 2F 2B B2 08 EF B1 3F CF
00000078 AF 74 41 5A 11 05 28 59 6A 1F 10 22 81 4D 4B F9 3F 63 13 01 BF EE E6 E3
00000090 D8 9E B8 AF 19 26 26 EE C9 1A 76 C7 EC 47 02 70 C8 66 26 94 F2 CA 03 40
000000a8 B8 24 3D D7 9D F8 54 6A 15 AA 4A 74 3E B4 6A 00 F9 17 35 AB A2 80 5A 51
000000c0
```

The differences between the two files are displayed in Table 1. Note a pattern in the changes: flip the first bit of the byte in out1.bin to compute the value of in out2.bin. The only exception to this rule is at offset 0xAE, where the last bit is flipped.

Table 1: Differences in out1.bin and out2.bin

offset	out1.bin	out2.bin
0x53	75	F5
0x6D	1D	9D
0x7B	DA	5A
0x93	2F	AF
0xAD	78	F8
0xAE	55	54
0xBB	2B	2A

Task 2

Task 2 investigates the suffix extension property of MD5. That is, for two distinct inputs m and n (with hash(m) = hash (n)), is it true that hash($m||t$) = hash($n||t$)? The following experiment shows this to be the case. Figure 8 shows the construction of a hash collision.

Figure 8: Constructing initial messages

```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ md5collgen -p prefix.txt -o m.bin n.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'm.bin' and 'n.bin'
Using prefixfile: 'prefix.txt'
Using initial value: d783743b3bcd61737433b73141bec313

Generating first block: .......
Generating second block: S10.....
Running time: 10.7939 s

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ diff m.bin n.bin
Binary files m.bin and n.bin differ

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ md5sum m.bin
33d1330b821c6819d4492f75ba53791a  m.bin

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ md5sum n.bin
33d1330b821c6819d4492f75ba53791a  n.bin
```

Figure 9 shows a number of steps. It first shows the creation of the concatenated files (the first two commands). Then, using the diff command, it is proven that the two files are different. Finally, the MD5 hashes are computed for each concatenated file. Viewing the last two commands in Figure 9, it can be seen that their hashes are identical. The odds of this occurring by chance are quite slim so it is safe to assume that concatenations preserve existing hash collisions.

Figure 9: Adding suffix and showing hash

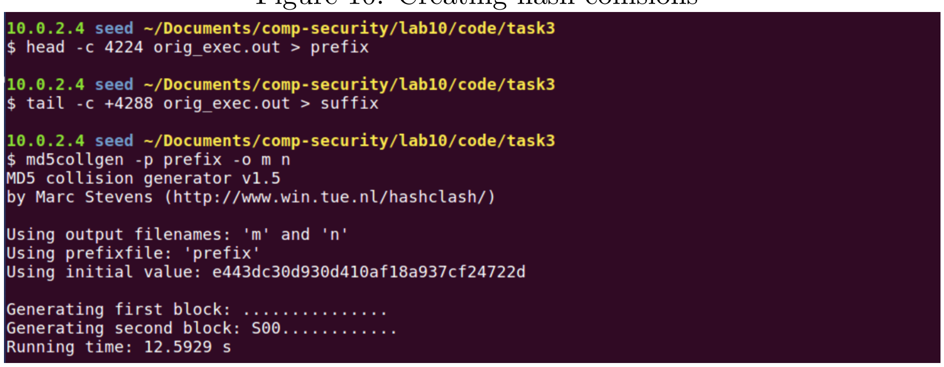
```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ cat m.bin suffix.txt > m-suff.bin

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ cat n.bin suffix.txt > n-suff.bin

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ diff m-suff.bin n-suff.bin
Binary files m-suff.bin and n-suff.bin differ

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ md5sum m-suff.bin
4f6dd3e7bfaf3c90536f099fd6892b2f  m-suff.bin

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task2
$ md5sum n-suff.bin
4f6dd3e7bfaf3c90536f099fd6892b2f  n-suff.bin
```

Task 3

Task 3 wonders whether two executable files can be constructed to have the same hash. The goal is to produce two programs with different source code, with the same function, and identical hashes. The program used is compiled from C source code in “print_array.c” on the course repository. In this program, there is a 200 byte array filled with the hex for A. When compiled, the location of the array will be evident (using a hex editor) and changes can be made. Note that the array is 200 bytes long, so a portion of it can be overwritten with the 128 bytes needed to generate a hash collision for MD5.

After compiling “print_array.c,” a hex editor can be used to find that the beginning of the array is at offset 1040 in hex. This translates to 4160 in decimal. Padding might cause an error in the program so the input size must be a multiple of 64. The smallest multiple of 64 (that is greater than 4160) is chosen: 4224. Figure 10 shows the use of the head/tail tools to copy the compiled binary into an appropriate prefix and suffix file. Then the hash collision is generated and placed into files m and n.

Figure 10: Creating hash collisions



From there, the distinct binary files m and n can be concatenated with the suffix to generate the valid programs “exec1.out” and “exec2.out.” This is shown in Figure 11.

Figure 11: Creating executables

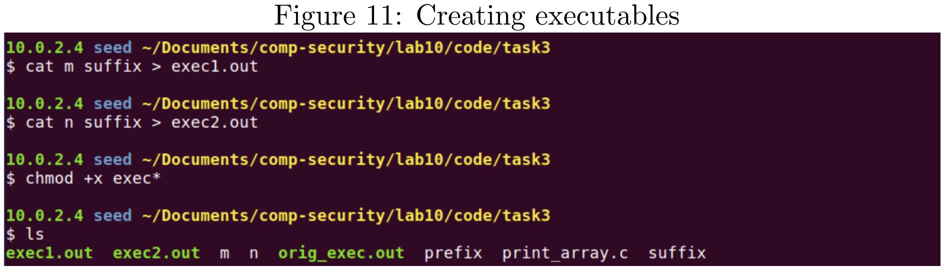


Figure 12 shows two results. First, that both binary files are functional programs (ie they print out the array). Second, it is also shown that the two binary files differ. The only remaining question is whether their hashes are identical. It is expected that they are; m and n already had a hash collision and the same binary was tacked on as a suffix.

Figure 12: Showing that the executables are functional

```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/task3
$ ./exec1.out
41414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414
14141414141414141414141414141414141dc3bb777f794d52ad24436df7c8be56acf6473ca6471cc593aae3976e9c1b0
f90ceee36aaf2d647e6de33dcaf447dff9284506469f520b920b467118545a3ed1c42ccacbbd26099aee5c5a7ce07e5
2b793ff2b94056c9fb14c555615da2536957e25d7682728583c42fc3856ea8ef274706bab84dfd49ffd6b801cb66741
41414141414141

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task3
$ ./exec2.out
41414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414
14141414141414141414141414141414141dc3bb777f794d52ad24436df7c8be56acf64734a6471cc593aae3976e9c1b0
f90ceee36aaf2d647e6de33dc2f457dff9284506469f520b920b4e7118545a3ed1c42ccacbbd26099aee5c5a7ce07e5
2b79bff2b94056c9fb14c555615da2536957e25d7682728583c42fc356ea8ef274706bab84dfd49ffdeb801cb667414
1414141414141

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task3
$ diff exec1.out exec2.out
Binary files exec1.out and exec2.out differ
```

In fact, their hashes are identical! This is shown in Figure 13.

Figure 13: Computing hashes

```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/task3
$ md5sum exec1.out
6e659a75208e0a0fadeb863449623760  exec1.out

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task3
$ md5sum exec2.out
6e659a75208e0a0fadeb863449623760  exec2.out
```

Task 4

In Task 4, two programs with different behavior are generated, despite having the same hash. This is done using the knowledge gained in the previous tasks. The final product will have two arrays that it compares, if the arrays are identical it will perform a benign action. But if they differ, it will perform a shifty action. But how is the program constructed? The original binary file looks like the following:

header	array 1	mid	array 2	suffix

In each executable, the prefix will be formed from the header and will populate array 1. The "mid" section will be left alone and array 2 should be populated with one of the 128 bytes generated from creating the hash collision. Then the suffix should be the same for both programs.

Figure 14 forms the hash collision and prepares the rest of the binary to be edited (by placing one of the collision tags in array 2).

Figure 14: Forming hash collision

```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ head -c 4224 orig.out > prefix

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ md5collgen -p prefix -o b-head s-head
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'b-head' and 's-head'
Using prefixfile: 'prefix'
Using initial value: 070caa665d6db0e04ccb78bb0f97d2c4

Generating first block: .
Generating second block: W.......
Running time: 0.739736 s

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ tail -c +4225 b-head > b-tail

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ tail -c +4225 orig.out > orig-tail
```

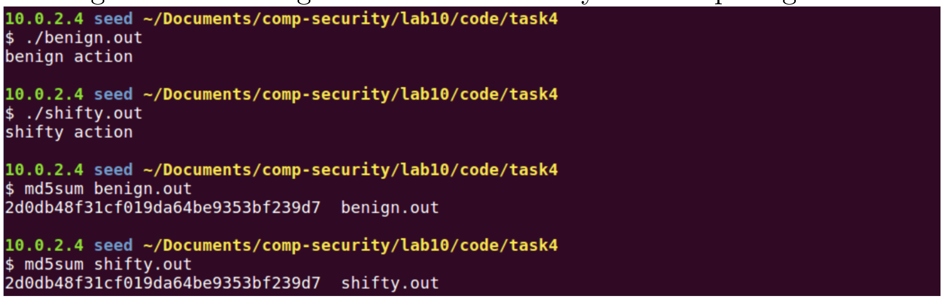
Figure 15 shows the creation of the binary files "benign.out" and "shifty.out."

Figure 15: Creating executables

```
10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ tail -c +4353 orig.out > orig-tail

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ head -c 96 orig-tail > mid

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ tail -c +225 orig-tail > suffix

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ cat b-head mid b-tail suffix > benign.out

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ cat s-head mid b-tail suffix > shifty.out

10.0.2.4 seed ~/Documents/comp-security/lab10/code/task4
$ chmod +x benign.out shifty.out
```

Finally, Figure 16 shows two results. The two programs perform different actions, but also have identical hashes. This is a major security flaw in the MD5 hash function.

Figure 16: Showing different functionality and computing hashes



For hash functions to be effective, it is necessary that they are collision resistant. Without that property, a malicious coder may be able to get code certified but then publish different code with an identical hash. This makes users vulnerable to attackers.