Assignment x
SEED Lab: MD5 Collision Attack Lab

Name
October 1, 2019

Introduction

This assignment followed the "MD5 Collision Attack Lab" from the SEED Labs project.[1]

In performing this lab, I explored how the MD5 hash function works, and saw how easy it was to create collisions for this hash function. The first two tasks were basic exploratory activities, learning about the MD5 hash function and the program "`md5collgen`" which can produce collisions for MD5. In the last two tasks, I used this knowledge to create two executable programs that do very different things and yet have the same MD5 hash. This shows that the MD5 hash function is not collision resistant (in fact, it allows even more powerful attacks than collision finding, since we can create collisions of meaningful files with a specific malicious purpose).

The setup for this lab involved using VirtualBox to create a virtual machine from the SEED Labs Ubuntu 16.04 image. This was straightforward given the instructions on the SEED Lab site, and I ended up increasing the virtual machine's RAM size to 2GB instead of using the recommended 1GB minimum, just to be safe. Once the virtual machine was set up, it was easy to run from VirtualBox, and it started with a desktop logged in as the `seed` user. For this lab using a virtual machine wasn't really necessary, but it was convenient since it had the necessary software (specifically `md5sum`, `md5collgen`, and `bless`) already installed.

Tasks

This lab included four specific tasks to perform, and in the sections that follow I give a summary of what I did, what I saw, and what I learned from each task.

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

Actions and Observations: In task 1, I experimented with the "`md5collgen`" program to see how to create different files with the same MD5 hash. In particular, I created a file named `prefix.txt` that just contains the string "hello world" and then ran the command:

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

This created two files that are different (as reported by the diff command), but have the same md5 hash value (as determined by the md5sum command). The screenshot below shows the process of running both diff and md5sum, as well as a hex dump of out1.bin as produced by the

1 http://www.cis.syr.edu/~wedu/seed/

"xxd" command:

```
[09/17/19]seed@srtate-vm:~/lab1$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[09/17/19]seed@srtate-vm:~/lab1$ md5sum out?.bin
7100fcad405a43547bc2e7e5deadfdf8  out1.bin
7100fcad405a43547bc2e7e5deadfdf8  out2.bin
[09/17/19]seed@srtate-vm:~/lab1$ xxd out1.bin
00000000: 6865 6c6c 6f20 776f 726c 640a 0000 0000  hello world.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 02e9 5e81 9508 1364 40bf a457 8c91 ace8  ..^....d@..W....
00000050: bcc9 5919 5db9 0b50 0be6 79a6 081e 3fc3  ..Y.]..P..y...?.
00000060: 0723 bac7 fe25 fe39 e719 cbdb c567 0877  .#...%.9.....g.w
00000070: d9fe c301 1412 bc8e 51a0 5aab cb31 2862  ........Q.Z..1(b
00000080: b692 b2d9 3e37 cd7f 615f 521f 357f 8a8e  ....>7..a_R.5...
00000090: bae5 6d12 1c37 a11e d537 16c1 d059 79a2  ..m..7...7...Yy.
000000a0: 7f79 9fb6 579f b1d7 7578 9ccf 9d42 72aa  .y..W...ux...Br.
000000b0: caea 9f1d 672f a266 abe3 4d72 422e 0dc5  ....g/.f..MrB...
[09/17/19]seed@srtate-vm:~/lab1$
```

From the hex dump it appears that the prefix.txt data was first padded with zeros to bring it up to 64 bytes, and then random-looking data is appended to that. The out2.bin file has the same initial 64 bytes (the prefix and padding), but differs slightly in the remaining bytes.

To explore what happens with different prefix sizes, I created a prefix file that was 92 bytes, and tried this experiment again. The hex dump shows that the padding bytes are used to make the prefix 128 bytes. Then I carefully edited the prefix file so that it was exactly 64 bytes long, and in this case the md5collgen program did not add any padding at all. These experiments provided all the information to answer the questions asked in the lab write-up.

Discussion and Questions: The SEED lab asks 3 specific questions for this task, which are answered below.

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

In all three tests I performed, the prefix was padded with zero bytes until the size was a multiple of 64.

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

With the 64-byte prefix, no zero-bytes are added for padding. The prefix file is immediately followed by the random-looking data that was created for the collision.

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

Looking at the hex dumps of the final two output files shows that relatively few bytes are different between the files. Careful examination shows that only 8 bytes are different (but the outputs are 192 bytes, not 128 bytes as stated in the question). The following are the two hex dumps, with the differing bytes underlined.

out1.bin hex dump:

```
00000000: 6865 6c6c 6f20 776f 726c 640a 7468 6973  hello world.this
00000010: 2069 7320 6120 3634 2d62 7974 6520 7072   is a 64-byte pr
00000020: 6566 6978 0a6c 6120 6465 2064 6120 2d20  efix.la de da -
00000030: 6669 6c6c 696e 6720 6974 206f 7574 2e0a  filling it out..
00000040: 59ee ac7c 9936 4fbc 13a0 d0f6 5a1e 546e  Y..|.6O.....Z.Tn
00000050: e950 19cf ac26 d2d4 f87d 8256 6fa9 a3cc  .P...&...}.Vo...
00000060: bebb 07ba ca04 3798 e761 d07a 1fd6 3030  ......7..a.z..00
00000070: d7a7 f828 f11d 962a f323 f7a3 6e5c 18d9  ...(...*.#..n\..
00000080: 247c b112 9bbf e79c 715b f959 3c80 5375  $|......q[.Y<.Su
00000090: 91ff 1151 4c97 223a 8db3 0080 0760 48d7  ...QL.":.....`H.
000000a0: d3ed 5e68 97c2 ec13 760a ca5b c243 f1e3  ..^h....v..[.C..
000000b0: 34ab d838 078c e3e8 b866 48b6 58f2 b499  4..8.....fH.X…
```

out2.bin hex dump:

```
00000000: 6865 6c6c 6f20 776f 726c 640a 7468 6973  hello world.this
00000010: 2069 7320 6120 3634 2d62 7974 6520 7072   is a 64-byte pr
00000020: 6566 6978 0a6c 6120 6465 2064 6120 2d20  efix.la de da -
00000030: 6669 6c6c 696e 6720 6974 206f 7574 2e0a  filling it out..
00000040: 59ee ac7c 9936 4fbc 13a0 d0f6 5a1e 546e  Y..|.6O.....Z.Tn
00000050: e950 194f ac26 d2d4 f87d 8256 6fa9 a3cc  .P.O.&...}.Vo...
00000060: bebb 07ba ca04 3798 e761 d07a 1f56 3130  ......7..a.z.V10
00000070: d7a7 f828 f11d 962a f323 f723 6e5c 18d9  ...(...*.#.#n\..
00000080: 247c b112 9bbf e79c 715b f959 3c80 5375  $|......q[.Y<.Su
00000090: 91ff 11d1 4c97 223a 8db3 0080 0760 48d7  ....L.":.....`H.
000000a0: d3ed 5e68 97c2 ec13 760a ca5b c2c3 f0e3  ..^h....v..[....
000000b0: 34ab d838 078c e3e8 b866 4836 58f2 b499  4..8.....fH6X...
```

As a final observation, while 8 bytes changed, the changes are small: each byte that changed only has a single bit different between the outputs.

Task 2: Understanding MD5's Property

Actions: For this part, I designed the following experiment to explore the structure of the MD5 hash algorithm, and the use of iterated compression functions. To accomplish this task, I had to create specific binary files, and so I learned about the "`bless`" hex editor to create the necessary files. Since the … (and so on...)

Observations: Running the files created as described above through the "`md5sum`" program to compute hash values, I observed that...

Discussion: From this task, I saw that has values of longer files are just extensions of the hash value of shorter prefixes of the file. This means that...

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

…. and so on ….

Conclusion

This SEED lab explored how the MD5 hash function works, and methods for creating collisions for this hash function. Collisions could be made quickly for files with very structured format, and as the final task we created two executable programs that performed very different tasks (one good and one malicious) and yet had the same hash.

This lab showed that the MD5 hash function cannot provide the necessary security for using in an anti-virus system that uses the hash function to detect when files change. In particular, a malicious software producer could produce a harmless but attractive program (like a game, which is distributed for free), and the later replace it with a malicious program which has the same MD5 hash value as the game. Since it has the same hash value, the anti-virus software would not be able to detect that the file had changed!

While this is one specific attack against MD5, I conclude from this that the MD5 algorithm has some serious weaknesses and should not be used.