Lab – Hashing Things Out

Objectives

Part 1: Creating Hashes with OpenSSL

Part 2: Verifying Hashes

Background / Scenario

Hash functions are mathematical algorithms designed to take data as input and generate a fixed-size, unique string of characters, also known as the hash. Designed to be fast, hash functions are very hard to reverse; it is very hard to recover the data that created any given hash, based on the hash alone. Another important property of hash functions is that even the smallest change done to the input data yields a completely different hash.

While OpenSSL can be used to generate and compare hashes, other tools are available. Some of these tools are also included in this lab.

Required Resources

- CyberOps Workstation VM
- Internet access

Part 1: Creating Hashes with OpenSSL

OpenSSL can be used as a standalone tool for hashing. To create a hash of a text file, follow the steps below:

Step 1: Hashing a Text File

a. In the CyberOps Workstation virtual machine, open a terminal window.

b. Because the text file to hash is in the /home/analyst/lab.support.files/ directory, change to that directory:

```
[analyst@secOps ~]$ cd /home/analyst/lab.support.files/
```

c. Type the command below to list the contents of the letter_to_grandma.txt text file on the screen:

```
[analyst@secOps lab.support.files]$ cat letter_to_grandma.txt
Hi Grandma,
I am writing this letter to thank you for the chocolate chip cookies you sent me. I
got them this morning and I have already eaten half of the box! They are absolutely
delicious!


I wish you all the best. Love,
Your cookie-eater grandchild.
```

d. Still from the terminal window, issue the command below to hash the text file. The command will use MD5 as hashing algorithm to generate a hash of the text file. The hash will be displayed on the screen after OpenSSL has computed it.

```
[analyst@secOps lab.support.files]$ openssl md5 letter_to_grandma.txt
MD5(letter_to_grandma.txt)= 8a82289f681041f5e44fa8fbeeb3afb6
```

Notice the format of the output. OpenSSL displays the hashing algorithm used, MD5, followed by the name of file used as input data. The MD5 hash itself is displayed after the equal ('=') sign.

e. Hash functions are useful for verifying the integrity of the data regardless of whether it is an image, a song, or a simple text file. The smallest change results in a completely different hash. Hashes can be calculated before and after transmission, and then compared. If the hashes do not match, then data was modified during transmission.

Let's modify the letter_to_grandma.txt text file and recalculate the MD5 hash. Issue the command below to open **nano,** a command-line text editor.

```
[analyst@secOps lab.support.files]$ nano letter_to_grandma.txt
```

Using **nano**, change the first sentence from **'Hi Grandma'** to **'Hi Grandpa'**. Notice we are changing only one character, **'m'** to **'p'**. After the change has been made, press the **<CONTROL+X>** keys to save the modified file. Press **'Y'** to confirm the name and save the file. Press the **<Enter>** key and you will exit out of nano to continue onto the next step.

f. Now that the file has been modified and saved, run the same command again to generate a MD5 hash of the file.

```
[analyst@secOps lab.support.files]$ openssl md5 letter_to_grandma.txt
MD5(letter_to_grandma.txt)= dca1cf6470f0363afb7a65a4148fb442
```

Is the new hash different that hash calculated in item (d)? How different?

Yes, the new hash different that hash calculated in item (d) It is totally different.

g. MD5 hashes are considered weak and susceptible to attacks. More robust hashing algorithms include SHA-1 and SHA-2. To generate a SHA-1 hash of the letter_to_grandma.txt file, use the command below:

```
[analyst@secOps lab.support.files]$ openssl sha1 letter_to_grandma.txt
SHA1(letter_to_grandma.txt)= 08a835c7bcd21ff57d1236726510c79a0867e861
[analyst@secOps lab.support.files]$
```

Note: Other tools exist to generate hashes. Namely, md5sum, sha1sum, and sha256sum can be used to generate MD5, SHA-1 and SHA-2-256 hashes, respectively.

h. Use **md5sum** and **sha1sum** to generate MD5 and SHA-1 hash of the letter_to_grandma.txt file:

```
[analyst@secOps lab.support.files]$ md5sum letter_to_grandma.txt
dca1cf6470f0363afb7a65a4148fb442  letter_to_grandma.txt

[analyst@secOps lab.support.files]$ sha1sum letter_to_grandma.txt
08a835c7bcd21ff57d1236726510c79a0867e861  letter_to_grandma.txt
[analyst@secOps lab.support.files]$
```

Do the hashes generated with `md5sum` and `sha1sum` match the images generated in items (g) and (h), respectively? Explain.

Yes, they do. Because md5 algorithm in openssl is the same as md5sum, and sha1 in openssl is the same as sha1sum, too.

Note: While SHA-1 has not yet been effectively compromised, computers are becoming more and more powerful. It is expected that this natural evolution will soon make it possible for attackers to break SHA-1. In a proactive move, SHA-2 is now the recommended standard for hashing. It is also worth noting that SHA-2 is in fact, a family of hashing algorithms. The SHA-2 family is comprised of six hash functions,

namely SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. These functions generate hash values that are 224, 256, 384 or 512 bits long, respectively.

Note: The CyberOPS VM only includes support for SHA-2-224, SHA-2-256, and SHA-2-512 (**sha224sum**, **sha256sum**, and **sha512sum**, respectively).

Part 2: Verifying Hashes

As mentioned before, a common use for hashes is to verify file integrity. Follow the steps below to use SHA-2-256 hashes to verify the integrity of sample.img, a file downloaded from the Internet.

a. Along with sample.img, sample.img_SHA256.sig was also downloaded. sample.img_SHA256.sig is a file containing the SHA-2-256 computed by the website. First, use the cat command to display the contents of the sample.img_SHA256.sig file:

```
[analyst@secOps lab.support.files]$ cat sample.img_SHA256.sig
c56c4724c26eb0157963c0d62b76422116be31804a39c82fd44ddf0ca5013e6a
```

b. Use SHA256sum to calculate the SHA-2-256 hash of the sample.img file:

```
[analyst@secOps lab.support.files]$ sha256sum sample.img
c56c4724c26eb0157963c0d62b76422116be31804a39c82fd44ddf0ca5013e6a  sample.img
```

Was the sample.img correctly downloaded? Explain.

Yes, it was.

Because the sample.img hasn't been edited.

Note: While comparing hashes is a relatively robust method detect transmission errors, there are better ways to ensure the file has not been tampered with. Tools such as **gpg** provide a much better method for ensuring the downloaded file has not been modified by third parties, and is in fact the file the publisher meant to publish.