Lab - Hashing Things Out

Objectives

Part 1: Hashing a Text File 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 function 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 virtual machine

Instructions

Part 1: Hashing a Text File with OpenSSL

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

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. From the terminal window, issue the command below to hash the text file. The command will use SHA-2-256 as the 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 sha256 letter_to_grandma.txt
SHA256(letter_to_grandma.txt)=
deff9c9bbece44866796ff6cf21f2612fbb77aa1b2515a900bafb29be118080b
```

Notice the format of the output. OpenSSL displays the hashing algorithm used, SHA-256, followed by the name of file used as input data. The SHA-256 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 SHA-2-256 hash of the file.

```
[analyst@secOps lab.support.files]$ openssl sha256 letter_to_grandma.txt
SHA256(letter_to_grandma.txt)=
43302c4500b7c4b8e574ba27a59d83267812493c029fd054c9242f3ac73100bc
```

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

g. A hashing algorithm with longer bit-length, such as SHA-2-512, can also be used. To generate a SHA-2-512 hash of the letter_to_grandma.txt file, use the command below:

```
[analyst@secOps lab.support.files]$ openssl sha512 letter_to_grandma.txt
SHA512(letter_to_grandma.txt)=
7c35db79a06aa30ae0f6de33f2322fd419560ee9af9cedeb6e251f2f1c4e99e0bbe5d2fc32ce5
01468891150e3be7e288e3e568450812980c9f8288e3103a1d3
[analyst@secOps lab.support.files]$
```

h. Use **sha256sum** and **sha512sum** to generateSHA-2-256 and SHA-2-512 hash of the letter_to_grandma.txt file:

```
[analyst@secOps lab.support.files]$ sha256sum letter_to_grandma.txt
43302c4500b7c4b8e574ba27a59d83267812493c029fd054c9242f3ac73100bc
letter_to_grandma.txt


[analyst@secOps lab.support.files]$ sha512sum letter_to_grandma.txt
7c35db79a06aa30ae0f6de33f2322fd419560ee9af9cedeb6e251f2f1c4e99e0bbe5d2fc32ce5
01468891150e3be7e288e3e568450812980c9f8288e3103a1d3  letter_to_grandma.txt
```

Do the hashes generated with **sha256sum** and **sha512sum** match the hashes generated in items (f) and (g), respectively? Explain.

Note: SHA-2 is the recommended standard for hashing. While SHA-2 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-2.

SHA-3 is the newest hashing algorithm and eventually be the replacement for SHA-2 family of hashes.

Note: The CyberOPS Workstation 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 hash that wascomputed 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 downloaded without errors? Explain.

Note: While comparing hashes is a relatively robust method to 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.