Lab 2

The goal of this lab is to learn how to work with various encryption and hashing technologies.

Lab Report Submission

Submit a PDF with screenshots of each command that was used, and a short description of the screenshots containing what the command does and what its output means.

Setup Environment

If you have Labtainer from lab 1, you can run all of the commands for task 1 in the "labtainer file-integrity" environment. Open the VM and run the command "labtainer file-integrity" to start the environment.

If you cannot use the Labtainer VM for task 1, you can run the lab from any bash shell or similar that has the following commands available:

- sha1sum
- cd
- mkdir
- cat
- wc
- echo
- find
- diff
- Any file editor (nano, vim, etc)

Task 2 requires age: https://age-encryption.org/

Task 3 requires GPG: https://gnupg.org/

Task 1: Simple Integrity Scheme with Hashing

This section introduces a simple integrity scheme using sha1sum. The goal is to be able to identify modifications to files after an attack. This task is modified tasks from the Labtainer file-integrity lab, designed to work on more systems.

Hashing a single file

1. Create a temporary file and hash it
   - `echo tempfile content > tempfile.txt`
   - `sha1sum tempfile.txt`
2. Save the hash to a file, then view it

- `sha1sum tempfile.txt > hash.txt`
- `cat hash.txt`

3. Modify the file, and hash it, manually compare it to the old, saved hash. Note the difference between the two hashes after the modification
   - `nano tempfile.txt` (Any method to edit the file is fine. Make sure to save the changes)
   - `sha1sum tempfile.txt`
   - `cat hash.txt`

Hashing all files in a folder

4. Hash a folder of files. Create a folder of at least 3 temporary files, and cd into it, then run sha1sum
   - `mkdir filestohash`
   - `cd filestohash`
   - `echo file 1 > one.txt`
   - `echo file 2 > two.txt`
   - `echo file 3 > three.txt`
   - `sha1sum ./* > hashes.txt`
   - `cat hashes.txt`
5. Verify integrity of files using --check option
   - `sha1sum --check hashes.txt`
6. Modify a file and check the hashes again to see it fail a check
   - `nano one.txt` (Any method to edit the file is fine. Make sure to save the changes)
   - `sha1sum --check hashes.txt`
7. Count the number of files hashed with wc
   - `wc hashes.txt`

Verify integrity after a file has been added

Sometimes attackers add files instead of modifying files. This section shows a simple method of how to detect newly added files to a system using find and diff

8. Create a list of files from the “filestohash” folder. First, go back to the folder that “filestohash” was created in using the cd command. Build the list with the find command, then, view myFiles.txt with cat
   - `cd ..`
   - `find ./filestohash/* -print > myFiles.txt`
   - `cat myFiles.txt`
9. Create a new file in “filestohash” folder
   - `echo “this is a bad new file” > ./filestohash/badnewfile.txt`
10. Create a new database now that a file has been created
    - `find ./filestohash/* -print > allFiles.txt`
11. Compare the difference between the directory structure:
    - `diff myFiles.txt allFiles.txt`

Task 2: Asymmetric Encryption using age

This task introduces age, a simple command line encryption tool written in Go. We will focus on age's implementation of the X25519 protocol to encrypt a file using a public key. This task and task 3 should and be completed outside of the Labtainer environment if you used it for task 1.

You can download age using most package managers, or as an executable for windows from their website. See age-encryption.org for more info.

1. Download age. This command might be different depending on what operating system you have
   - `sudo apt install age`
2. Generate a public key. Save it to a file, then view the key
   - `age-keygen -o key.txt`
   - `cat key.txt`
3. Create a file and encrypt it using the public key from key.txt.
   - `echo testfilecontents > test_file_to_encrypt.txt`
   - `age --encrypt --armor -r PUBLIC_KEY_HERE test_file_to_encrypt.txt > ciphertext.txt`
      i. Copy the text after "Public key:" in key.txt and replace PUBLIC_KEY_HERE with it.
      ii. The command should look like:
      iii. `age --encrypt --armor -r age1su9ukywuxv528y6kr5ufdd33mck5qdzr33yyz6ntt94ql2a2yuhqaapgmu test_file_to_encrypt.txt > ciphertext.txt`
4. View the ciphertext
   - `cat ciphertext.txt`
5. Decrypt the file and view it
   - `age --decrypt -i key.txt ciphertext.txt > decrypted.txt`
   - `cat decrypted.txt`

Task 3: Symmetric Encryption using GPG

GPG (Gnu Privacy Guard) is an open source encryption and signing tool. It is more widely used than age. This task is to use GPG to create an AES symmetric key, then encrypt and decrypt a file with it.

1. Download GPG. This command might be different depending on what operating system you have
   - `sudo apt install gpg`
2. Generate a 256-bit AES key.
   - `gpg --gen-random --armor 2 32 > key_aes.txt`
      i. The 2 indicates that gpg should use a large amount of entropy when creating the key, and 32 is the number of bytes (which is 256 bits)
3. Create a file. Encrypt it and view the cipher text. If prompted to enter a passphrase, you can use "password" or anything else

- `echo filecontents > file.txt`
- `gpg --symmetric --cipher-algo AES256 --passphrase-file key_aes.txt --output encrypted_file.txt file.txt`
- `cat encrypted_file.txt`

4. Decrypt and view the plaintext file
   - `gpg --decrypt --passphrase-file key_aes.txt --output decrypted_file.txt encrypted_file.txt`
   - `cat decrypted_file.txt`