Mini tutorial: Comparación de hashes SHA-256 en Linux

1. Objetivo

Aprender a:

- Generar hashes SHA-256 de archivos.
- Guardar hashes en un archivo de referencia.
- Comparar hashes para verificar integridad usando sha256sum -c.
- Detectar modificaciones de archivos (simulación de ataque).

2. Preparar archivos de ejemplo

En tu VM Linux:

```
# Crear archivo de prueba
echo "Este es un mensaje secreto" > mensaje.txt
```

Contenido de mensaje.txt:

```
Este es un mensaje secreto
```

3. Generar hash SHA-256

```
sha256sum mensaje.txt > mensaje.sha256
```

Esto crea un archivo mensaje.sha256 con contenido como:

```
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855  mensaje.txt
```

La primera cadena es el hash del archivo.

4. Verificar integridad del archivo

```
sha256sum -c mensaje.sha256
```

Salida esperada:

```
mensaje.txt: OK
```

Significa que el archivo **no ha sido modificado** desde que se generó el hash.

Nota:

4.1.- Comando usado

```
sha256sum -c mensaje.sha256
```

La opción -c significa **"check"**, es decir, comprobar la integridad.
El argumento mensaje.sha256 **no contiene el archivo en sí**, sino el **hash calculado previamente** y el **nombre del archivo asociado**.

4.2.- Contenido de mensaje.sha256

Si abrimos el archivo:

```
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855  mensaje.txt
```

Primera columna → hash SHA-256 del archivo original (mensaje.txt).

Segunda columna → nombre del archivo a verificar (mensaje.txt).

4.3. Cómo hace la comparación sha256sum -c

sha256sum -c **lee línea por línea** el archivo mensaje.sha256.

Para cada línea:

Toma el **hash almacenado** (primer campo).

Toma el **nombre del archivo** (segundo campo).

Calcula el **hash actual** del archivo con ese nombre en el directorio.

Compara ambos hashes.

Resultado:

Si coinciden → OK

Si no coinciden → FAILED

5. Simular un cambio en el archivo (ataque)

```
echo "Modificación maliciosa" >> mensaje.txt
sha256sum -c mensaje.sha256
```

Salida:

```
mensaje.txt: FAILED
sha256sum: WARNING: 1 computed checksum did NOT match
```

La verificación falla porque el archivo **ya no coincide con el hash original**.

6. Verificación de múltiples archivos

Crear más archivos:

```
echo "Archivo 2" > archivo2.txt
echo "Archivo 3" > archivo3.txt
```

Generar hashes para todos:

```
sha256sum mensaje.txt archivo2.txt archivo3.txt > hashes.sha256
```

Verificar integridad automáticamente:

```
sha256sum -c hashes.sha256
```

Salida posible:

```
mensaje.txt: FAILED
archivo2.txt: OK
archivo3.txt: OK
```

Detecta **cuál archivo fue modificado**.

Nota:

6.1.- Contenido del archivo hashes.sha256

Si generaste los hashes así:

```
sha256sum mensaje.txt archivo2.txt archivo3.txt > hashes.sha256
```

El archivo hashes.sha256 contendrá algo como:

```
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855  mensaje.txt
d4735e3a265e16eee03f59718b9b5d03d1f4a1b2f934a0985fdf7df7d1f3e5f2  archivo2.txt
9c56cc51b3747e8c1a8e9a5b4e2b9c1d3f6a7b8c2d9e1f3a4b5c6d7e8f9a0b1c2  archivo3.txt
```

Cada línea tiene **el hash esperado y el nombre del archivo** correspondiente.

6.2.- Qué hace sha256sum -c

Lee la primera línea: toma el hash e3b0... y compara con el hash actual de mensaje.txt.

Lee la segunda línea: toma el hash d4735... y compara con el hash actual de archivo2.txt.

Lee la tercera línea: compara el hash de archivo3.txt.

Cada comparación es independiente, por eso puede salir algo como:

```
mensaje.txt: FAILED
archivo2.txt: OK
archivo3.txt: OK
```

Significa que **solo mensaje.txt fue modificado**, mientras que los otros archivos permanecen intactos.

sha256sum -c **no está comparando archivos entre sí**, sino **cada archivo con su hash guardado**. Esto es útil para verificar la integridad de muchos archivos enviados o recibidos.

7. Uso combinado con scripts (opcional)

Puedes crear un **script Bash** que haga:

```
#!/bin/bash
# Generar hash y verificar integridad automáticamente
sha256sum "$1" > "$1.sha256"
sha256sum -c "$1.sha256"
```

Ejemplo de ejecución:

```
./verificar.sh mensaje.txt
```

8. Resumen y buenas prácticas

Siempre genera un hash **antes de enviar o compartir un archivo**.

Usa sha256sum -c para **verificar automáticamente** múltiples archivos.

Simula ataques modificando archivos para **verificar que la integridad se rompe**.

9. Subida de la práctica

Realiza una serie de pantallas explicando como has realizado el apartado 6.