SEGURIDAD EN SISTEMAS OPERATIVOS
4º Grado en Informática – Complementos de Ing. del Software
Curso 2018-19

Práctica 1. Encriptación/desencriptación en Linux

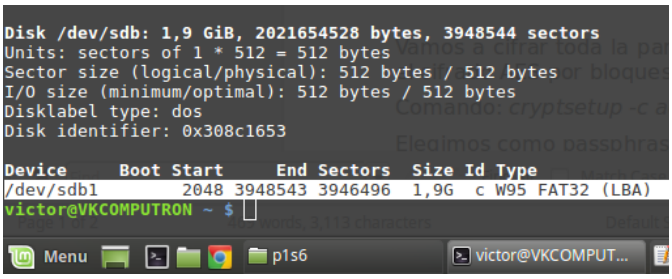
Sesión 6. Cifrado de sistemas de archivos. Esteganografía y estegoanálisis.

Autor[1]: Víctor García Carrera

Ejercicio 1.

Comenzamos la realización de esta práctica utilizando la herramienta *Cryptsetup* para encriptar todo el sistema de archivos de un USB, de forma que cuando se monta el pendrive y se da la clave correspondiente podemos leer/escribir archivos y que se encripten/desencripten al vuelo.

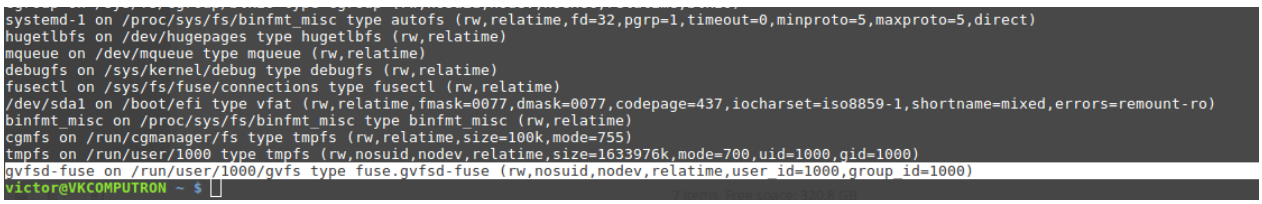
Una vez hemos descargado *Cryptsetup*, conectamos el pendrive a cifrar y utilizamos el comando *fdisk -l* para visualizar todos los dispositivos conectados al sistema. En nuestro caso, podemos observar que se trata de un pendrive de 1,9GB formateado en W95 FAT32 (LBA). Ubuntu le ha asignado a esta partición la dirección física */dev/sdb1*.



Vamos a encriptar toda la partición del pendrive. Con la herramienta *Cryptsetup* elegimos el cifrado AES por bloques utilizando como función hash sha256. Para ello nos pide una passphrase o contraseña, eligiendo como passphrase *randpass654*.

Comando: *cryptsetup -c aes -h sha256 -y -s 256 luksFormat /dev/sdb1*

Tras ejecutar el comando y dar la contraseña, nos dice que no puede formatear el dispositivo porque aun está en uso. Utilizamos el comando *umount /dev/sdb1* para desmontar la unidad. Repetimos el proceso y tras dar la passphrase no produce salida alguna (en principio parece que ha realizado correctamente el cifrado del pendrive). El comando *mount* refleja los sistemas de archivos accesibles en el sistema al estar ligados o montados en el gran arbol de jerarquía de archivos. A continuación se visualiza parte de la salida de este comando donde NO aparece el pendrive puesto que en esta parte del proceso de cifrado lo hemos desmontado:



1 Como autor declaro que los contenidos del presente documento son originales y elaborados por mi. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la "Normativa de evaluación y de calificaciones de los estudiantes de la Universidad de Granada" esto "conllevará la calificación numérica de cero … independientemente del resto de calificaciones que el estudiante hubiera obtenido ..."

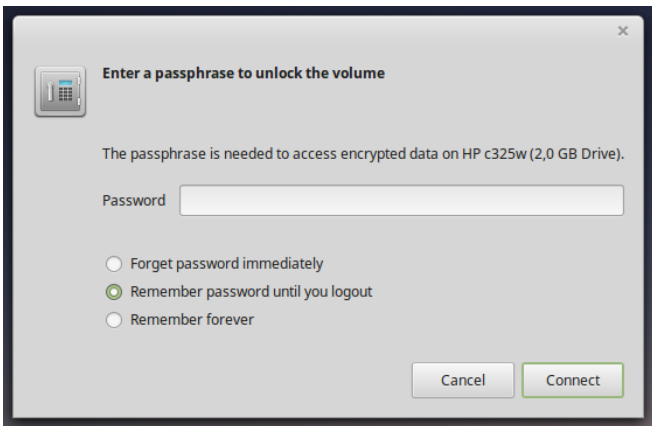
Abrimos el dispositivo con *Cryptsetup* mediante el comando *sudo cryptsetup luksOpen /dev/sdb1 test*. Tras ejecutar el comando, ya tenemos acceso al dispositivo como un dispositivo normal, pero actuamos como un dispositivo mapeado que no es real para la configuración. Utilizamos el comando *mkfs /dev/mapper/test* para crear un sistema de archivos Linux y mapear el USB en la carpeta *test*.

Para finalizar, debemos montar el USB para poder trabajar con el sistema de archivos. Para esto ya no nos referimos al pendrive con su dirección real */deb/sdb1*, sino con la unidad mapeada anteriormente */dev/mapper/test*, y ejecutamos el comando *sudo mount /dev/mapper/test /home/prueba* para montarlo en el directorio */home/prueba* con previos permisos 777 (completos). Comprobamos mediante el comando *mount* que ahora aparece este nuevo sistema de archivos:

```
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=38,pgrp=1,timeout=0,minproto=5,maxproto=5,direct)
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
/dev/sda1 on /boot/efi type vfat (rw,relatime,fmask=0077,dmask=0077,codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,relatime)
cgmfs on /run/cgmanager/fs type tmpfs (rw,relatime,size=100k,mode=755)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=1633976k,mode=700,uid=1000,gid=1000)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
/dev/mapper/test on /home/prueba type ext2 (rw,relatime,block_validity,barrier,user_xattr,acl)
victor@VKCOMPUTRON /dev/mapper $
```

Una vez que el sistema de archivos está montado, creamos y copiamos con permisos de root diversos archivos en el USB (un directorio, un fichero de extensión pdf y otro de extensión txt). Tras esto, procedemos a cerrar el pendrive, para lo cual ejecutamos el comando *cryptsetup luksClose /dev/mapper/test*.

Extraemos el dipositivo. Cuando volvemos a insertar el USB, nos pide la clave o passphrase para poder acceder al sistema de archivos. Si no damos la contraseña correcta, el sistema de archivos del pendrive permanece inaccesible y cifrado.



Ejercicio 2.

Dejamos a un lado el cifrado de sistemas de archivos para ver un concepto y una técnica de ocultación de información muy interesante: la *esteganografía*. Ésta se basa en ocultar información (mensajes, objetos...) dentro de otros (portadores) de forma que pase inadvertida tanto su existencia como envío al no diferir en nada (a simple vista) el objeto portador original y el objeto que porta el secreto. A diferencia del cifrado, la información a ocultar, si bien no resulta percibible, puede ser o no comprensible (puede estar cifrada o no). Tras ver diversas técnicas utilizadas en esteganografía como *LSB (Least Significant Byte)* o *inyección*, y múltiples herramientas con diferentes extensiones y métodos, vamos a trabajar con la herramienta StegHide.

Tras descargarla e instalarla, contamos con 2 ficheros de texto, *ocultar.txt* y *ocultar2.txt*, que queremos ocultar en la imagen *portadora_original.jpg*. La diferecia entre *ocultar.txt* y *ocultar2.txt* es que este último contiene un mensaje bastante más grande que el primero. Duplicamos la imagen en *portadora.jpg* y *portadora2.jpg* para conservar la imagen original y utilizar estas 2 imagenes para ocultar los 2 textos y tratar de observar diferencias.

ocultar.txt

```
1 ESTE ES EL MENSAJE A OCULTAR DENTRO DE UNA IMAGEN(PORTADOR) CON LA HERRAMIENTA STEGHIDE
2 COMPROBAMOS SI FUNCIONA Y SI NO SE PERCIBE LA DIFERENCIA ENTRE LAS IMAGENES ORIGINAL Y LA PORTADORA
```

Utilizando el comando *steghide embed -cf portadora.jpg -ef ocultar.txt* comprimimos y ocultamos el archivo de texto en la imagen, para lo cual nos pide una contraseña: *secret27*. Tras esto, eliminamos el mensaje de texto original. Repetimos el proceso con *ocultar2.txt* y *portadora2.jpg*. A continuación las imágenes portadoras resultantes:

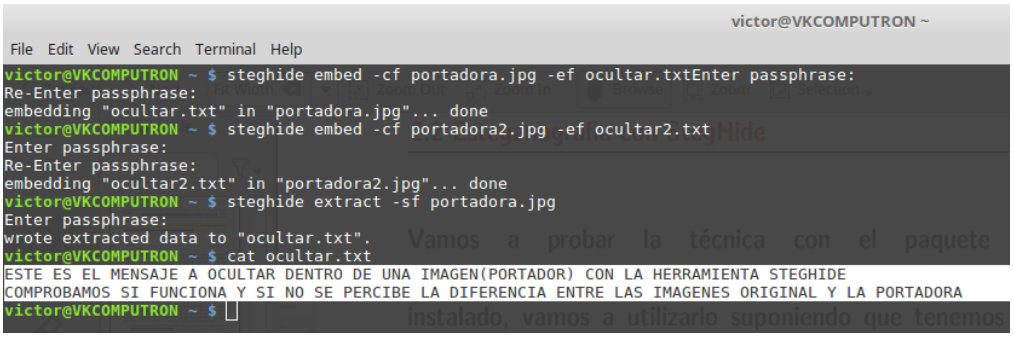
IMAGEN ORIGINAL

IMAGEN PORTADORA

IMAGEN PÒRTADORA 2

A pesar de que cuesta un poco observarlo, hay una diferencia mínima en la nitidez de las imagenes original y portadoras, siendo más nítida la primera. La imagen original tiene un tamaño de 117,3 kB mientras que las portadoras ocupan 90,4 kB. Además, al observar desde la terminal gráfica las propiedades de cada imagen, la original cuenta con muchos más datos tales como el modelo de la cámara, la apertura, la fecha... Datos que no aparecen en las imagenes portadoras (seguramente por la compresión realizada en ellas). Resulta interesante que, a pesar de que el mensaje oculto en *portadora2.jpg* sea mucho más grande que el de *portadora.jpg*, no se percibe diferencia alguna entre ambas imagenes.

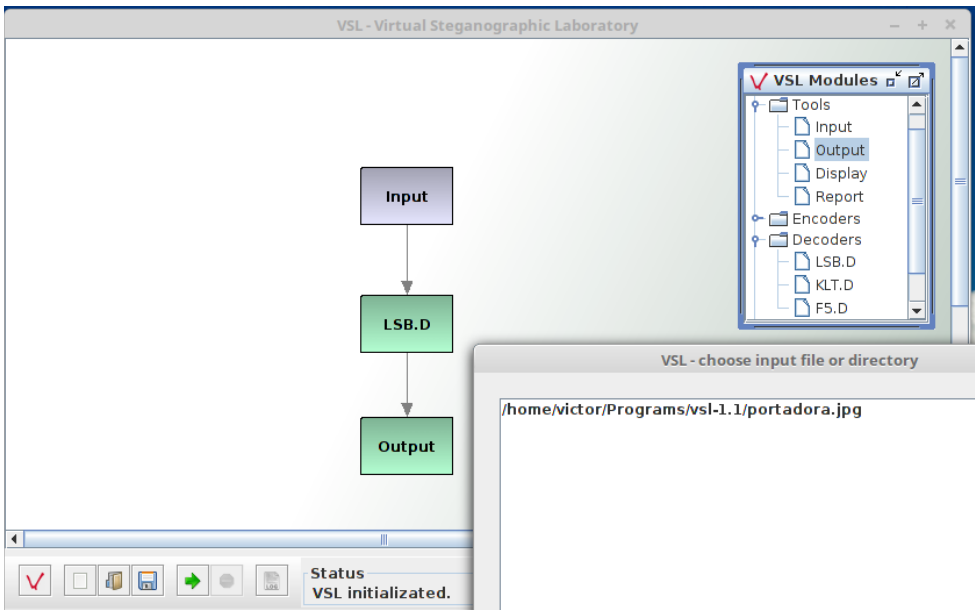
Con el comando *steghide extract -sf portadora.jpg* y dando la clave utilizada anteriormente, podemos recuperar el contenido original del mensaje a ocultar.



Ejercicio 3.

Para finalizar esta práctica, haremos uso de la herramienta VSL para analizar la imagen esteganográfica generada anteriormente (*portadora.jpg*) y comprobar si detecta información oculta. Con ella, seleccionamos como input la imagen portadora, como descodificador LSB (la técnica empleada anteriormente que resulta bastante frecuente en la esteganografía), y como output el archivo *vsl_result.txt.*

Tras correr la herramienta, el contenido de *vsl_result.txt* es el mensaje original oculto.



vsl_result.txt

```
1 ESTE ES EL MENSAJE A OCULTAR DENTRO DE UNA IMAGEN(PORTADOR) CON LA HERRAMIENTA STEGHIDE
2 COMPROBAMOS SI FUNCIONA Y SI NO SE PERCIBE LA DIFERENCIA ENTRE LAS IMAGENES ORIGINAL Y LA PORTADORA
```