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

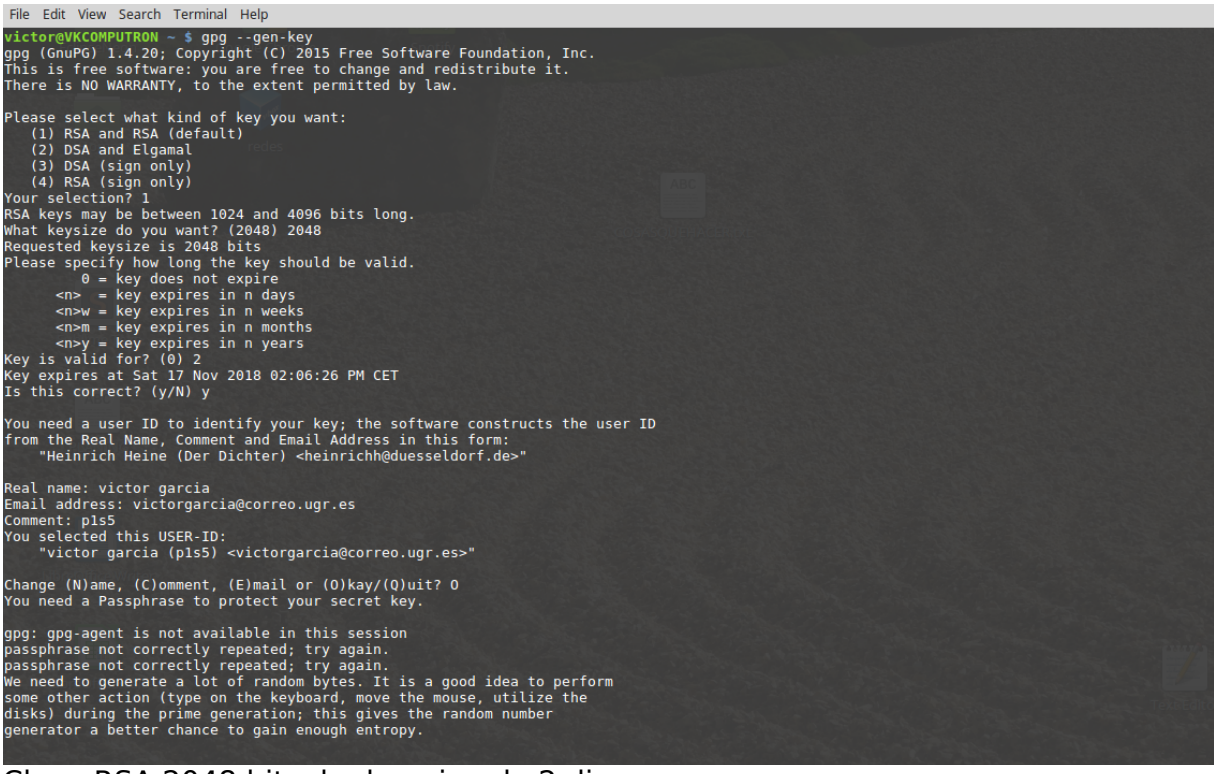
Práctica 1. Administración de la seguridad en Linux

Sesión 5. Cifrado de archivos

Autor[1]: Víctor García Carrera

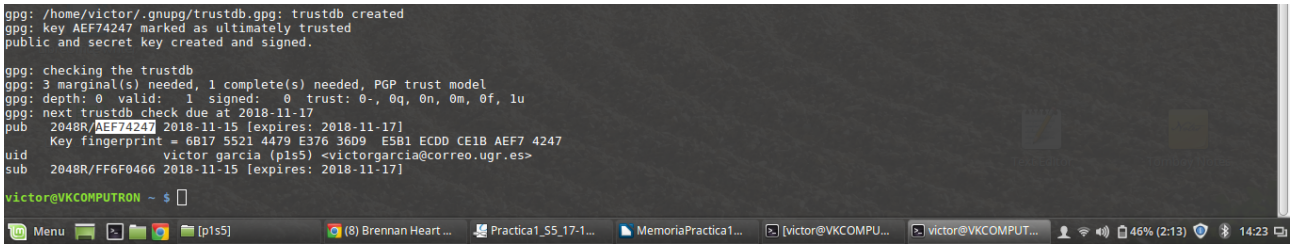
Ejercicio 1.

En este ejercicio vamos a simular el proceso de transmisión de un mensaje o fichero entre usuarios utilizando el cifrado de clave pública con la heramienta GnuPG o gpg. Para ello creamos un primer usuario **victor** y su par de claves RSA con un tamaño de clave de 2048 bits, las cuales tienen una validez de 2 días. Utilizamos una passphrase o frase contraseña para identificar a este usuario y sus claves en futuras transmisiones. La siguiente imagen refleja el proceso de creación de claves descrito:



Clave RSA 2048 bits de duracion de 2 dias

passphrase: *randompassphrase*



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 ..."

Podemos visualizar mediante el comando gpg –list-keys las claves publicas existentes en nuestro sistema, apareciendo la recien creada de **victor**. También podemos exportar en un archivo estas claves.

```
victor@VKCOMPUTRON ~ $ gpg --list-keys
/home/victor/.gnupg/pubring.gpg
------------------------------
pub   2048R/AEF74247 2018-11-15 [expires: 2018-11-17]
uid                  victor garcia (p1s5) <victorgarcia@correo.ugr.es>
sub   2048R/FF6F0466 2018-11-15 [expires: 2018-11-17]

victor@VKCOMPUTRON ~ $
```

Exportar clave publica de **victor**:

gpg --armor --export -output vgcpubkey.gpg victor

Exportar clave privada de **victor**

gpg --armor --export-secret-key -output vgcprivkey.gpg victor

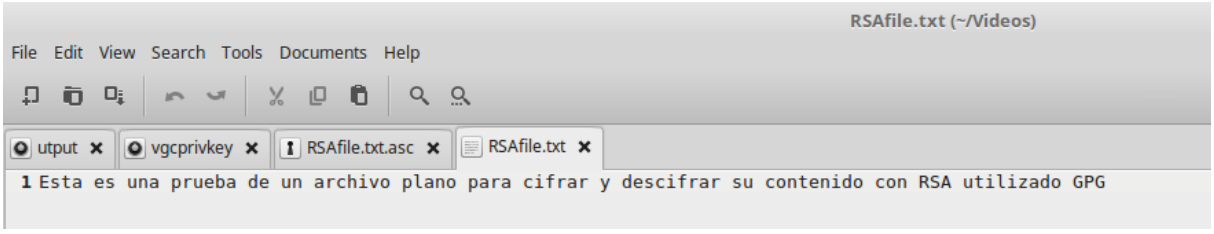
A continuación se detallan los comandos necesarios para cifrar y descifrar un fichero de texto plano *RSAfile.txt* de y para victor.

Cifrar mensaje para **victor**:

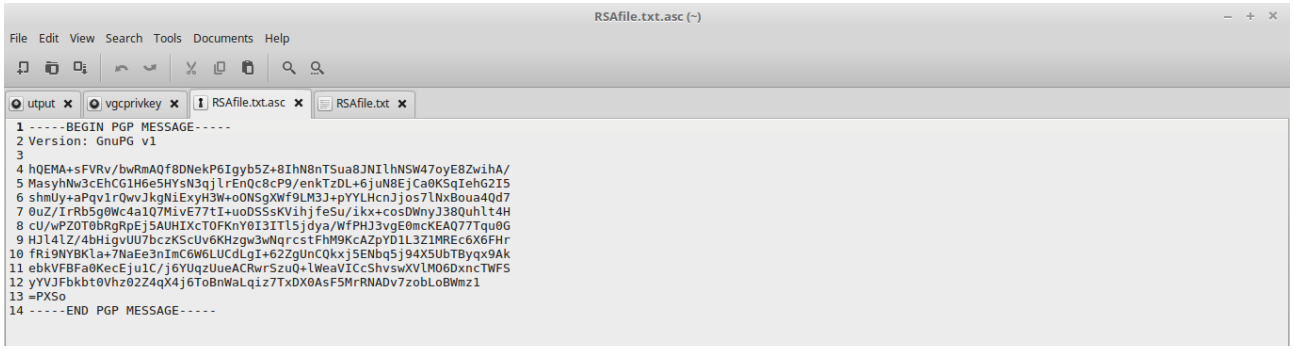
gpg --armor --recipient victorgarcia@correo.ugr.es --encrypt RSAfile.txt

Descifrar mensaje para **victor**:

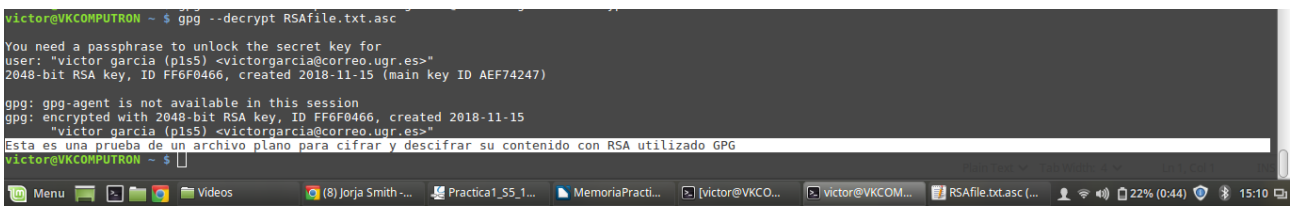
gpg --decrypt RSAfile.txt.asc



Fichero a cifrar y descifrar



Fichero cifrado para **victor**



Fichero descifrado con el passphrase de **victor**

Si queremos cifrar y descifrar archivos en cualquier formato diferente a texto plano: (caso **victor**)

gpg --no-armor --recipient victorgarcia@correo.ugr.es --encrypt rsafile2.pdf

gpg -no-armor --decrypt rsafile2.pdf.gpg

De forma predeterminada aparece un archivo con nombre *-armor*

Vamos a crear otro perfil de claves RSA para simular el envio de mensajes entre dos usuarios distinos. Creamos el perfil **antonio** de correo antonioperez@correo.ugr.es siguiendo el mismo proceso que con **victor**, generando una clave RSA de 2048 bits de longitud con validez durante 2 dias y passphrase=*passphrase*

```
gpg: key A4E2CE2A marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: next trustdb check due at 2018-11-17
pub   2048R/A4E2CE2A 2018-11-15 [expires: 2018-11-17]
      Key fingerprint = 6F4C 776F 914F 0A38 16D0  CA69 1A27 1D30 A4E2 CE2A
uid                  antonio perez <antonioperez@correo.ugr.es>
sub   2048R/7781333D 2018-11-15 [expires: 2018-11-17]

victor@VKCOMPUTRON ~ $
```

```
victor@VKCOMPUTRON ~
File  Edit  View  Search  Terminal  Help
victor@VKCOMPUTRON ~ $ gpg --list-keys
/home/victor/.gnupg/pubring.gpg
-------------------------------
pub   2048R/AEF74247 2018-11-15 [expires: 2018-11-17]
uid                  victor garcia (p1s5) <victorgarcia@correo.ugr.es>
sub   2048R/FF6F0466 2018-11-15 [expires: 2018-11-17]

pub   2048R/A4E2CE2A 2018-11-15 [expires: 2018-11-17]
uid                  antonio perez <antonioperez@correo.ugr.es>
sub   2048R/7781333D 2018-11-15 [expires: 2018-11-17]

victor@VKCOMPUTRON ~ $
```

Vamos a cifrar el fichero *RSAfile.txt* para enviarselo al recien creado usuario **antonio** (modificamos el contenido del fichero para este ejemplo). Utilizamos la clave pública del receptor, en este caso de **antonio**. A continuación se muestra este proceso:

```
victor@VKCOMPUTRON ~
File  Edit  View  Search  Terminal  Help
victor@VKCOMPUTRON ~ $ gpg --armor --recipient antonioperez@correo.ugr.es --encrypt RSAfile.txt
victor@VKCOMPUTRON ~ $ ls
antonioprivkey  Desktop     Downloads            Instagram  Pictures  Public      RSAfile.txt.asc  vgcprivkey  Videos
antoniopubkey   Documents   google-chrome.desktop  Music      PRUEBAS   RSAfile.txt  Templates        vgcpubkey   workspace
victor@VKCOMPUTRON ~ $ gpg --decrypt RSAfile.txt.asc

You need a passphrase to unlock the secret key for
user: "antonio perez <antonioperez@correo.ugr.es>"
2048-bit RSA key, ID 7781333D, created 2018-11-15 (main key ID A4E2CE2A)

gpg: gpg-agent is not available in this session
gpg: encrypted with 2048-bit RSA key, ID 7781333D, created 2018-11-15
      "antonio perez <antonioperez@correo.ugr.es>"
MENSAJE PARA ANTONIO:
Este mensaje está cifrado exclusivamente para ti, Antonio, con tu clave pública, pues sólo tu posees la clave privada.
victor@VKCOMPUTRON ~ $
```

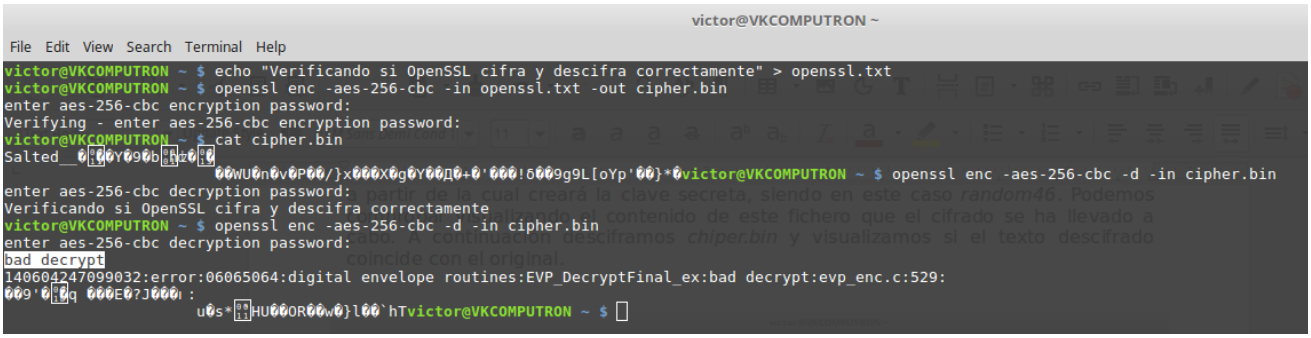
Si intentamos descifrar el fichero como **victor** (con la clave privada de **victor**), veremos que el mensaje resultante no es el mensaje original.

Ejercicio 2.

OpenSSL es una biblioteca de código abierto que implementa las operaciones criptográficas básicas utilizada de manera extensa por diversos servicios de internet. Vamos a probar a cifrar y descifrar un archivo de texto plano mediante el algoritmo de cifrado simétrico AES en modo CBC (cifrado de bloque) con un tamaño de bloque de 256 bits (AES-256 CBC), uno de los más utilizados hoy en día. De hecho, en la práctica los cifrados presentan un esquema híbrido con firma digital donde se suele utilizar

este algoritmo de cifrado simétrico para encriptar el mensaje, y posteriormente cifrando esta clave simétrica o clave de sesión con RSA.

Creamos el fichero *openssl.txt* y lo ciframos con OpenSSL utilizando el algoritmo AES-256 CBC, resultando en un fichero binario *chiper.bin*. Para ello nos pide una **contraseña** a partir de la cual creará la clave secreta, siendo en este caso *random46*. Podemos comprobar visualizando el contenido de este fichero que el cifrado se ha llevado a cabo. A continuación desciframos *chiper.bin* y visualizamos si el texto descifrado coincide con el original. Si introducimos en el descifrado la **contraseña** incorrecta, el mensaje resultante carece de similitud con el original.



Ejercicio 3.

Finalmente tratamos con la vulnerabilidad Heartbleed. Esta vulnerabilidad se detectó en abril de 2014 y afecta a la librería criptográfica OpenSSL, utilizada de manera extensa en Internet para crear una comunicación segura y privada en diversos servicios web, de mensajería y algunas VPNs. Esta vulnerabilidad es grave pues permite que cualquiera acceda y lea la memoria del sistema. Esto compromete las claves secretas utilizadas para identificar los proveedores de servicio y encriptar el tráfico, además de nombres y contraseñas de usuarios. Todo esto rompe la comunicación segura.

Esta vulnerabilidad se solventa con las versiones de OpenSSL a partir de la 1.0.1g. Aquellas versiones entre 1.0.1 y 1.0.1f son vulnerables. La rama de 1.0.0 y la de 0.9.8 no se ven afectadas. El fallo que permite esta vulnerabilidad se debe a un problema en la implementación de la librería de OpenSSL (su código), no en el protocolo SSL/TLS en el que se basa esta librería. Heartbleed es una vulnerabilidad famosa, y esto se debe al gran número de sistemas a los que afectó (y afecta), pues tardó algo más de 2 años en descubrirse y dejó expuestas una gran cantidad de claves y secretos. Afectó a más de la mitad de los servidores mundiales, pues los servidores de código abierto como Apache o nginx utilizan OpenSSL.

El comportamiento del bug es el siguiente: El RFC 6520 es la extensión Heartbeat para los protocolos TLS y DTLS utilizado para probar los enlaces de comunicación segura TLS/DTLS y mantenerlos vivos sin la necesidad de renegociar la conexión cada vez. Permite que un ordenador en un extremo de la comunicación, nuestro emisor, envíe una solicitud “Heartbeat Request”, consistente en una carga útil, típicamente una cadena de texto, junto con un entero de 16-bits que refleja la longitud de dicha carga. El receptor responde enviando la misma carga exacta de vuelta al emisor. Las versiones afectadas por esta vulerabilidad asignan un búffer de memoria para el mensaje de respuesta de tamaño igual al que indica el campo longitud del mensaje de solicitud del emisor, sin tener en cuenta el tamaño real de la carga útil de ese

mensaje. Debido a esto, el mensaje de respuesta consta de la carga útil seguida de cualquier otra información que esté asignada en ese buffer de memoria. De esta manera, con cada hearbeat o latido de corazón, se está filtrando información no deseada.

Podemos clasificar en diferentes niveles de importancia los secretos comprometidos. El nivel más importante es el de las claves de cifrado, que permiten a quien las posea descifrar cualquier comunicación pasada y futura, además de usurpar la identidad del afectado. Para solventar esto, es necesario parchear la vulnerabilidad (descargar una nueva versión de OpenSSL), revocar las claves comprometidas y volver a crear y distribuir las nuevas claves. Aun así, el trafico pasado sigue vulnerable de ser desencriptado.

Para solventar las perdidas relativas a las credenciales de usuario (username, password) en diversos servicios, en primer lugar los propietarios del servicio en cuestión, los administradores de sistemas, deben recuperar la confianza del servicio, reemitiendo los CA o certificados de autentificación (certificados de seguridad) y revocando el certificado anterior. Después de esto, los usuarios ya pueden (y deben) cambiar sus contraseñas y posibles claves de encriptación utilizadas por el servicio.

El problema principal de este bug es que no deja traza diferenciable ninguna en los logs, por lo que no es posible detectar si alguien ha explotado esta vulnerabilidad en nuestro sistema. Aun así, existen herramientas que intentan comprobar si una dirección IP o dominio ha sido afectado, como las 2 siguientes:

https://pentest-tools.com/network-vulnerability-scanning/openssl-heartbleed-scanner

https://filippo.io/Heartbleed/