SSII

PSI-VERIN

El presente informe documenta los procedimientos seguidos y por consiguiente los resultados obtenidos para resolver las cuestiones del proyecto PSI-Verin, el cual está destinado a garantizar la integridad de los archivos contenidos en uno o varios directorios.

Fco. Javier Delgado Vallano

20 de Febrero del 2015

ÍNDICE

HISTORIAL DE VERSIONES

Versión	Fecha	Descripción
1.0	20/02/2015	Creación y estructuración del documento.
1.1	25/02/2015	Desarrollo de planificación, introducción y tarea 1.
1.2	26/02/2015	Desarrollo de reto.
1.3	01/03/2015	Desarrollo de tarea 2 y de la conclusión y finalización del documento.

PLANIFICACIÓN

Tareas	Tiempo estimado	Tiempo total	Tiempo desviación
Informe	6 h.	5 h.	1 h.
Tarea 1	10 h.	13 h.	3 h.
Tarea 2	1 h.	1 h.	0 h.
Reto	2 h.	2 h.	0 h.

Introducción

PSI-Verin es un proyecto que busca el garantizar la integridad de un conjunto de archivos que están contenidos en uno o varios directorios. Para ello se ha optado por realizar un programa en *Python*, mediante el cual se intenta garantizar dicha integridad haciendo uso de las técnicas de *hashing*. Esto se hace posible mediante la comparación periódica de los *hash* de los archivos con otros que deben ser cargados inicialmente en una base de datos *SQLite3*.

En caso de encontrar algún cambio en dichos archivos, el programa lo notificará tanto por consola como por correo electrónico a la persona especificada en el archivo de configuración mediante el envío de este desde una cuenta de Gmail.

Tarea 1

La primera tarea a realizar consiste en desarrollar o configurar un verificador de la integridad para un conjunto de directorios de un sistema o aplicación informática.

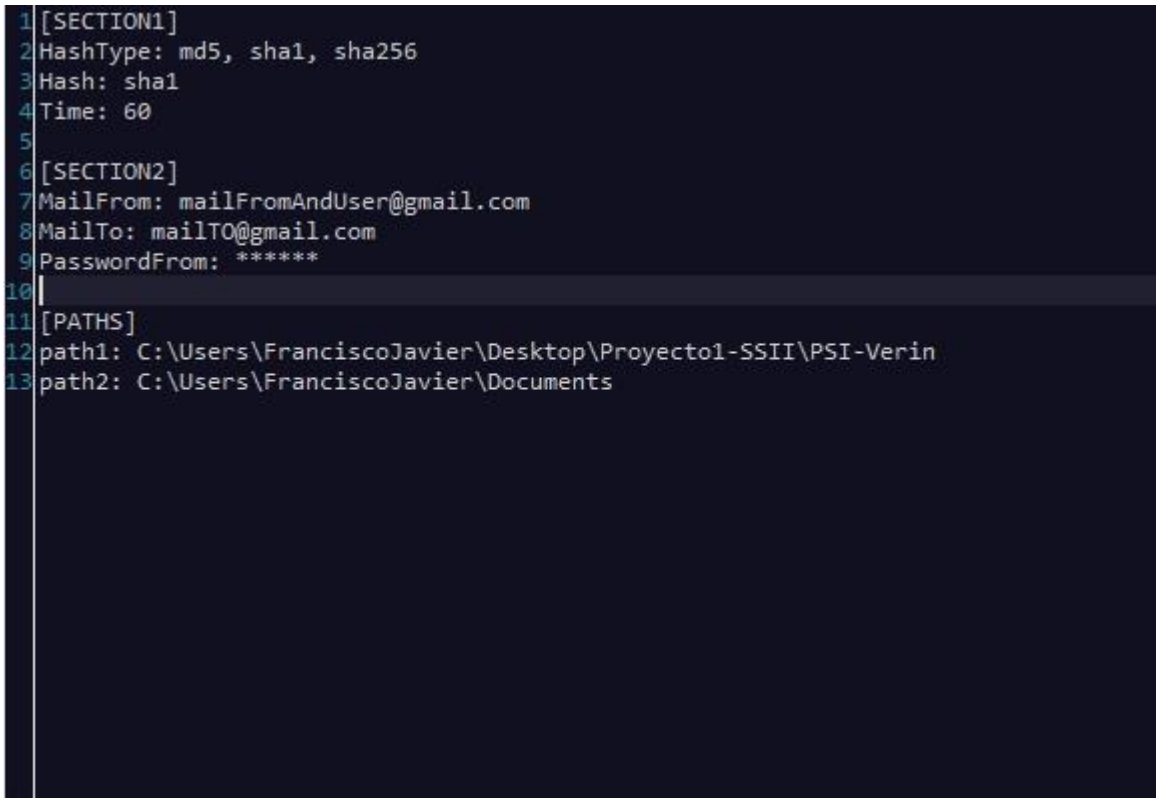
Tras haber realizado un análisis de las aplicaciones ya existentes más conocidas como pueden ser TripWire o VeriSys se decidió realizar un programa por el hecho de que de ésta manera la solución aportada se ajusta en mayor grado a las peticiones del cliente, además de ser ampliable en una nueva iteración para añadir nuevas funciones o modificar las actuales.

Por tanto, para el desarrollo de dicho programa se han barajado las opciones de *Python* y *Java* debido a su numerosa lista de librerías y gran comunidad que tienen a sus espaldas, pero finalmente fue el primero el elegido debido a ventajas como son:

- Código dinámico.
- Mayor eficiencia con respecto a *Java*.
- No necesita clases predefinidas, por lo que el número de líneas de código se ve reducido.

El programa en cuestión se compone de dos *scripts*, “start.py” encargado de realizar una búsqueda y carga inicial de los archivos contenidos en los directorios a examinar junto son sus respectivos códigos *hash*, “compruebaIntegridad.py” que se encarga de realizar las verificaciones periódicas de la integridad de estos; y por último un archivo de configuración “config.cfg”.

Antes de ponerlo en funcionamiento debemos dirigirnos a dicho archivo de configuración para definir los parámetros necesarios para el correcto funcionamiento, como podemos observar a continuación.



En éste archivo podemos observar 3 secciones claramente diferenciadas:

- "SECTION1":
  - Hash: Tipo de *hashing* a usar.
  - Time: Tiempo en minutos, en el que se volverá a realizar la verificación.
- "SECTION2":
  - MailFrom: Dirección de correo de Gmail origen y por tanto usuario de la misma, desde la que se realizaría el envío de un email en caso de que se viera violada la integridad.
  - MailTo: Dirección de correo destino.
  - PasswordFrom: Contraseña de la cuenta asociada al atributo MailFrom.
- "PATHS":
  - pathX: Ruta del directorio en el que se desee verificar la integridad. (X = número de ruta)

Como nota aclaratoria sobre las rutas, estas deben ser creadas siguiendo el mismo esquema mostrado en la imagen anterior.

Tras haber configurado correctamente el programa debemos proceder a ejecutar el script “start.py” por consola, para realizar el análisis y la carga de datos inicial en la base de datos, dirigiéndonos a la ruta en la que se encuentran dichos scripts y ejecutándolos. Esta base de datos almacenará tanto el nombre del archivo como su código hash.

En caso de que el resultado sea satisfactorio se mostrará un mensaje en la consola como el siguiente:

BBDD creada correctamente

A continuación se muestra parte del código del *script* anteriormente comentado que hace posible esta operación.

```
def archivosEnDirectorio(dir,algoritmo):


    con = sqlite3.connect(DirBBDD)
    cursor = con.cursor()
    msgFinal = 'BBDD creada correctamente'

    if (len(dir) > 0):

        for directorios in dir :

            ficheros = os.walk(directorios)

            for (root, dirs, files) in ficheros:
                for f in files:

                    if (f.decode("ISO-8859-1") == "integrity.db"):
                        continue
                    if algoritmo == opciones.get(1):
                        cod = hashlib.md5()
                    elif algoritmo == opciones.get(2):
                        cod = hashlib.sha1()
                    else:
                        cod = hashlib.sha256()

                    h = open(os.path.join(root, f), "r", encoding = "ISO-8859-1")

                    contenido = h.read()

                    cod.update(contenido.encode("utf-8"))

                    digest = cod.hexdigest()

                    #Insertar en la BBD
                    params = (f, digest)
                    cursor.execute('INSERT INTO files (fileName, code) VALUES ( ?, ?)', params)
                    con.commit()

                    del cod
                    h.close()

    else:

        msgFinal = "Introduzca al menos un directorio!!"


    print(msgFinal)

    con.close()
```

Finalmente queda ejecutar de la misma manera que el *script* anterior, el *script* “compruebaIntegridad.py”, el cual se encargará de realizar la verificación periódica de los archivos que pretendemos mantener íntegros.

Este *script* se compone de varios métodos que hacen posible la correcta ejecución del mismo.

En primer lugar, tenemos el método encargado de cargar los datos almacenados previamente en la base de datos en una lista. Esta lista será posteriormente recorrida para realizar las comparaciones que verificarán si se cumple la integridad. En la siguiente imagen se muestra dicho método:

```
64 #Metodo para cargar los datos de la BBDD a una lista
65 def cargaDatos(DirBBDD):
66
67     con = sqlite3.connect(DirBBDD)
68     cursor = con.cursor()
69
70     BBDD = []
71
72     cursor.execute('SELECT id, fileName, code from files')
73     for i in cursor:
74         BBDD += i;
75
76     con.close()
77     return BBDD
78
```

En segundo lugar, nos centramos en el método que hace posible el envío de la notificación, mediante correo electrónico, de que la integridad se ha visto violada en determinados archivos, señalando de este modo el número de archivos totales y sus respectivos nombres.

```
42 #Metodo para enviar email de notificacion
43 def enviarMail (mensaje):
44
45     msg = MIMEText(mensaje)
46
47     MTo = config.get("SECTION2", "MailTo")
48     MFrom = config.get("SECTION2", "MailFrom")
49     password = config.get("SECTION2","PasswordFrom")
50
51     msg['Subject'] = "Pérdida de integridad"
52     msg['From'] = MFrom
53     msg['To'] = MTo
54
55     mailServer = SMTP('smtp.gmail.com',587)
56     mailServer.ehlo()
57     mailServer.starttls()
58     mailServer.ehlo()
59     mailServer.login(MFrom,password)
60
61     mailServer.sendmail(MFrom, MTo, msg.as_string())
62
63     mailServer.close()
64
```

A continuación nos centramos en otra función del *script* que nos ocupa, el cual podríamos nombrarlo como el núcleo de todo, ya que es este el que comprobará si la integridad se mantiene, y en caso contrario lo notificará como se ha descrito anteriormente, además de mostrar por consola los mismo mensajes enviados mediante dicho correo.

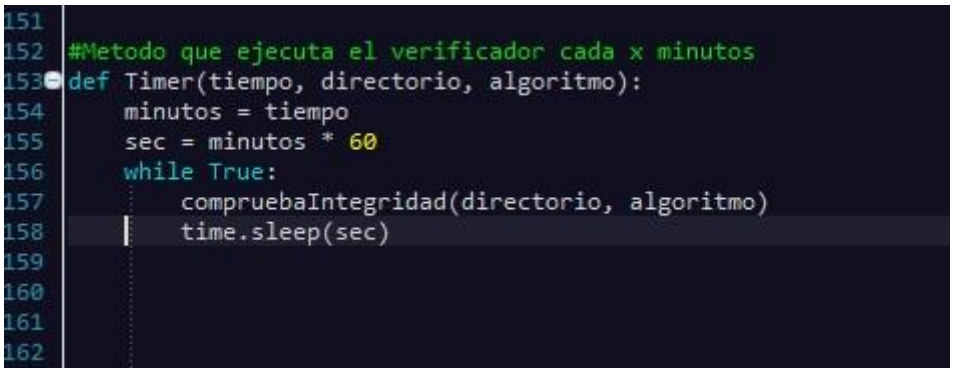
Su funcionamiento consiste en realizar la comparación de los datos almacenados en la base de datos con los recogidos ahora en un nuevo recorrido. En caso de ser el resultado favorable obtendremos como salida el mensaje:

OK

Mientras que en caso contrario, obtendremos lo anteriormente descrito. A continuación podemos comprobar el código del método en cuestión:

```
def compruebaIntegridad( dir , algoritmo ):

    con = sqlite3.connect(DirBBDD)
    cursor = con.cursor()
    noCumplen = []
    BBDD = cargaDatos(DirBBDD)

    if (len(dir) > 0):

        for directorios in dir :

            ficheros = os.walk(directorios)

            for (root, dirs, files) in ficheros:
                for f in files:

                    if (f.decode("utf-8") == "integrity.db"):
                        continue
                    if algoritmo == opciones.get(1):
                        cod = hashlib.md5()
                    elif algoritmo == opciones.get(2):
                        cod = hashlib.sha1()
                    else:
                        cod = hashlib.sha256()

                    h = open(os.path.join(root, f), "r", encoding = "ISO-8859-1")
                    contenido = h.read()
                    cod.update(contenido.encode("utf-8"))
                    digest = cod.hexdigest()

                    if ( f in BBDD and digest in BBDD ): #Si no ha habido ninguna modificación se procede a comprobar el siguiente archivo
                        pass

                    elif( f in BBDD and digest not in BBDD ): #Si el hash es modificado este notifica cual es el cambiado
                        noCumplen.append(f)
                        print("El archivo " + f.decode("utf-8") + "ha sido modificado")

                    elif ( f not in BBDD and digest not in BBDD ): #Si es un archivo nuevo lo introduce en la BBDD
                        params = (f, digest)
                        cursor.execute('INSERT INTO files (fileName, code) VALUES ( ?, ?)', params)
                        con.commit()
                        print("Se ha añadido un nuevo archivo con nombre : " + f.decode("utf-8") + "\n")

                    del cod
                    h.close()
    else:
        print("Introduzca al menos un directorio")

    numNoCumplen = len(noCumplen)

    if( numNoCumplen > 0 ):
        #Envio de email notificando que hay archivos modificados
        print("Hay " + str(numNoCumplen) + " archivos que han sido modificados." + "\n" + "\n")

        #Mensaje envidado por email
        mensaje = "Hola! \n \nEn la verificación de la integridad realizada: " + time.ctime() + "\n \nExisten " + str(numNoCumplen) + " archivos que han sido modificados." + "\n \n" + "

        enviarMail(mensaje);

        print("Email enviado correctamente \n")

    if(numNoCumplen == 0):
        return print("OK \n")
    con.close()
```

Por último, tenemos el método encargado de que la verificación se realice cada cierto tiempo, manteniendo el programa a la espera hasta que se alcance dicho tiempo para volver a realizar la verificación de la integridad. Este tiempo se especifica en el archivo de configuración.



```
151
152 #Metodo que ejecuta el verificador cada x minutos
153 def Timer(tiempo, directorio, algoritmo):
154     minutos = tiempo
155     sec = minutos * 60
156     while True:
157         compruebaIntegridad(directorio, algoritmo)
158         time.sleep(sec)
159
160
161
162
```

Tarea 2

La segunda tarea propuesta consiste en aportar un *feedback* a la entidad hospitalaria en la que se aporten posibles mejoras que se puedan realizar al producto entregado, es decir, al verificador de la integridad.

Algunas de las mejoras aplicables pueden ser:

- Desarrollar una interfaz de usuario que permita realizar la configuración inicial del programa de manera más visual, cómoda y sencilla para cualquier usuario.
- Implementar una función que permita reestablecer el contenido del o de los archivos que hayan sido modificados por los originales, los cuales son los almacenados en la base de datos.
- Modificar la opción de notificar mediante correo electrónico permitiendo la entrada de más de un email tanto como emisor como receptor. Además de permitir la opción de utilizar otro servidor que no sea el establecido por Google.
- Almacenar los datos en la base de datos de modo que se unan los métodos de *Hashing* y *salting* para así aumentar la seguridad de los mismos.
- Implementar la opción de registrar a uno o varios usuarios con permisos para controlar el verificador.
- Cifrar y almacenar como archivo oculto del sistema el archivo de base de datos.
- Modificar el verificador de la integridad para que se inicie automáticamente con el inicio del sistema.

RETO

En este apartado se abordará el reto propuesto para el presente proyecto. Este consiste en conseguir descifrar una contraseña dada en código hexadecimal, debido a alguno de los tipos existentes de *Hashing*, la cual es la siguiente:

fdfba0e18f6c257f9c2e48575ed84b7568b3fceb680ece9620524df1d14603f2a6c d8130dd7a8c13da5314c68cc5655a

En primer lugar, tras un breve análisis del enunciado del mismo, dicha contraseña no cumple la Política de Seguridad ya que al ser una palabra perteneciente al diccionario de la Real Academia Española esta no puede contener ni números ni símbolos, además de saber que dicha contraseña no puede ser contiene un combinación de letras en mayúsculas o minúsculas por lo que o está de una manera o de otra, por tanto el problema se ve proporcionalmente acotado. Por tanto, como solución se propone implementar una función que mediante la comprobación del número de caracteres de la cadena hexadecimal, se pueda averiguar el tipo de *Hash* utilizado para esta contraseña, tal y como se muestra en la imagen que viene a continuación:

```
6  #Metodo para identificar el metodo hashing utilizado
7  def tipoHash(password):
8      tam = len(password)
9      if(tam == 32):
10         hash = hashlib.md5();
11     elif(tam == 40):
12         hash = hashlib.sha1()
13     elif(tam == 56):
14         hash = hashlib.sha224()
15     elif(tam == 64):
16         hash = hashlib.sha256()
17     elif(tam == 96):
18         hash = hashlib.sha384()
19     else:
20         hash = hashlib.sha512()
21
22     return hash
```

Una vez tengamos en posesión dicho método de *hashing*, el cual es *sha384*, pasamos a realizar un ataque diccionario para comprobar que efectivamente esta contraseña no cumple las políticas establecidas, el cual consiste en una comparación entre el código *hash* de cada palabra contenida en el diccionario, una a una; y el código dado hasta comprobar cuál es la palabra a la que pertenece este.

A continuación podemos observar el código utilizado para realizar esta comprobación:

```
24
25 #Metodo utilizado para leer archivo qu contiene las palabras del diccionario
26 def leerDesdeArchivo(archivo):
27     dic = []
28     f = open(archivo, "r" , encoding="utf8")
29     for line in f.readlines():
30         line = line.replace("\n","")
31         dic.append(line)
32     f.close()
33     return dic
34
35
36 #Metodo que comprueba la contraseña utilizada
37 def hack(passw):
38
39     dic = leerDesdeArchivo("words.txt")
40     hash = tipoHash(passw)
41
42
43     for contenido in dic:
44         hash = tipoHash(passw)
45
46         hash.update(contenido.encode("utf-8"))
47
48         digest = hash.hexdigest()
49
50         if (passw == digest):
51             print(contenido)
52             break
53
54         del hash
55
56
```

Finalmente, se observa que efectivamente no se ha cumplido con las Políticas de Seguridad y que la palabra utilizada como contraseña es:

badulaque

Por tanto, se propone imponer la obligación de almacenar contraseñas que cumplan estrictamente dichas políticas, la utilización de un *nonce* o alguna técnica similar para aumentar la complejidad de ataque a estas contraseñas. Por otra parte, la entidad debe mantener íntegro y aumentar la seguridad en el almacenamiento de las mismas, además de mantener un control sobre las personas que tienen acceso a los archivos/directorios que se quieren controlar, ya sea dando permisos a un grupo determinado de personas y/o establecer un log de la persona que accede a estos junto con el momento del mismo.

CONCLUSIÓN

La integridad en el almacenamiento debe ser controlada y cuidada, para así, garantizar que la información almacenada en los sistemas informáticos permanezca completa, y en el caso de que sea modificada, sólo sea así por personas autorizadas. Esto es importante ya que actualmente existe mucha información que puede resultar confidencial y que debe ser salvaguardada para evitar su uso malintencionado.

Cabe destacar el aprendizaje de los numerosos métodos existentes para asegurar dicha integridad, así como su funcionamiento como también su implementación.