

Introducción a la seguridad Informática.

1

Según la Real Academia Española, seguridad es "la cualidad de seguro, es decir, de estar libre y exento de todo daño, peligro o riesgo". En informática, como en tantas facetas de la vida, la seguridad entendida según la definición anterior es prácticamente imposible de conseguir, por lo que se ha relajado acercándose más al concepto de fiabilidad; se entiende un sistema seguro como aquel que se comporta como se espera de él.

De los sistemas informáticos, ya sean sistemas operativos, servicios o aplicaciones, se dice que son seguros si cumplen las siguientes características:

- ✓ **Confidencialidad.** Requiere que la información sea accesible únicamente para las entidades autorizadas.
- ✓ **Integridad.** Requiere que la información sólo pueda ser modificada por las entidades autorizadas. La modificación incluye escritura, cambio, borrado, creación y reenvío de los mensajes transmitidos.
- ✓ **No repudio.** Ofrece protección a un usuario frente a otro que niegue posteriormente que se realizó cierta comunicación. El no repudio de origen protege al receptor de que el emisor niegue haber enviado el mensaje, mientras que el no repudio de recepción protege al emisor de que el receptor niegue haber recibido el mensaje. Las firmas digitales constituyen el mecanismo más empleado para este fin.
- ✓ **Disponibilidad.** Requiere que los recursos del sistema informático estén disponibles para las entidades autorizadas cuando los necesiten.

El tratamiento de los posibles incidentes de seguridad exige que toda organización tenga definida una política de seguridad, cuya función sea establecer las responsabilidades y reglas necesarias para evitar amenazas y minimizar los efectos de éstas. Lo primero que se debe realizar es identificar qué queremos proteger.

Desde el punto de vista informático, existen tres tipos de elementos que pueden sufrir amenazas:

- **hardware,**
- **software y**
- **datos.**

El más sensible, y en el que se basa casi toda la literatura sobre seguridad, son los datos, ya que es el único elemento que depende exclusivamente de la organización. Es decir, tanto el hardware como el software, si en la peor de las situaciones se pierden, siempre se pueden adquirir y/o instalarlos; pero los datos pertenecen a la organización y nadie puede, en el caso de pérdida, proporcionarlos. Sin embargo, la seguridad se debe entender a todos los niveles y aplicarla a todos los elementos.

1. Amenazas de seguridad.

Se entiende por amenaza una condición del entorno del sistema de información (por ejemplo: persona, máquina) que, dada una oportunidad, podría dar lugar a que se produjese una violación de la seguridad (confidencialidad, integridad, disponibilidad o uso legítimo).

Las amenazas de seguridad pueden caracterizarse modelando el sistema como un flujo de información; desde una fuente, como por ejemplo un fichero o un equipo, a un destino, como por ejemplo otro fichero o un usuario.

Tal y como puede ver en la figura las cuatro categorías generales de amenazas o ataques son las siguientes:

- ✓ **Interrupción.** Un recurso del sistema es destruido o deja de estar disponible. Éste es un ataque contra la disponibilidad. Un ejemplo de ataque es la destrucción de un elemento hardware, como un disco duro, cortar una línea de comunicación o deshabilitar el sistema de gestión de ficheros.
- ✓ **Intercepción.** Una entidad no autorizada consigue acceso a un recurso. Éste es un ataque contra la confidencialidad. La entidad no autorizada puede ser una persona, un programa o un ordenador. Un ejemplo de este ataque es escuchar una línea para registrar los datos que circulen por la red y la copia ilícita de ficheros o programas (intercepción de datos), o bien la lectura de las cabeceras de paquetes para desvelar la identidad de uno o más de los usuarios implicados en la comunicación observada ilegalmente (intercepción de identidad).
- ✓ **Modificación.** Una entidad no autorizada no sólo consigue acceder a un recurso, sino que es capaz de manipularlo. Éste es un ataque contra la integridad. Un ejemplo de este ataque es alterar un programa para que funcione de forma diferente, modificar el contenido de un fichero o de un mensaje transferido por red.
- ✓ **Fabricación.** Un ataque contra la autenticidad tiene lugar cuando una entidad no autorizada inserta objetos falsificados en el sistema. Un ejemplo de este ataque es la inserción de mensajes espurios (mensajes basura) en una red o añadir registros a un fichero.

2. Ataques.

Los ataques se pueden, clasificar de forma útil en términos de ataques pasivos y ataques activos.

- **Ataques pasivos.** En los ataques pasivos, el atacante no altera la comunicación, sino que únicamente la escucha o monitoriza para obtener información de lo que está siendo transmitido. El principal uso que se suele realizar de los ataques pasivos es para ver el tráfico de un equipo, obtener las contraseñas de red, etc.

Los ataques pasivos son muy difíciles de detectar, ya que no provocan ninguna alteración de los datos. Sin embargo, es posible evitar su éxito mediante el cifrado de la información.

- **Ataques activos.** Estos ataques implican algún tipo de modificación del flujo de datos transmitido o la creación de un falso flujo de datos. Algunas formas puede ser: suplantación de identidad (el intruso se hace pasar por un usuario legítimo), reactuación (se reenvían mensajes legítimos), modificación de mensajes, denegación de servicio (se satura un servicio para que no pueda ser utilizado correctamente), etc.

3. Vulnerabilidades en el software.

Debido al auge de Internet, las vulnerabilidades de software son uno de los mayores problemas de la informática actualmente. Las aplicaciones actuales cada vez tratan con más datos (personales, bancarios, etc.) y por eso, cada vez es más necesario que las aplicaciones sean seguras.

Una vulnerabilidad de software puede definirse como un fallo o hueco de seguridad detectado en algún programa o sistema informático que puede ser utilizado para entrar en los sistemas de forma no autorizada.

De una forma sencilla, se trata de un fallo de diseño de algún programa, que puede tener instalado en su equipo, y que permite a los atacantes realizar una acción maliciosa. Si nos centramos en una aplicación, es posible que ese fallo de seguridad permita a un usuario que acceda o manipule a una información a la que no esté autorizado.

Pero el gran problema de los agujeros de seguridad reside en la forma en que son detectados. Cuando una aplicación tiene una cierta envergadura, se encarga el trabajo de detectar fallos de seguridad a usuarios

externos para que utilicen la aplicación normalmente y/o a expertos de seguridad para que analicen a fondo la seguridad de la aplicación.

A pesar de que antes de lanzar una aplicación al mercado se intentan descubrir y solucionar todos sus fallos de seguridad, en muchas ocasiones se descubren después y por lo tanto, es necesario informar a los usuarios y permitir que se actualicen a la última versión que se encuentre libre de fallos.

Para poder garantizar la seguridad de una aplicación en Java nos vamos a centrar en dos pilares:

3

- **Seguridad interna de aplicación.** A la hora de realizar la aplicación se debe programar de una forma robusta para que se comporte tal y como esperamos de ella. Algunas de las técnicas que podemos implementar, y que veremos en el segundo apartado, es la gestión de excepciones, validaciones de entradas de datos y la utilización de los ficheros de registro.
- **Políticas de acceso.** Una vez que tenemos una aplicación "segura" es importante definir las políticas de acceso para determinar las acciones que puede realizar la aplicación en nuestro equipo. Por ejemplo, podemos indicar que la aplicación pueda leer los ficheros de una determinada carpeta, enviar datos a través de Internet a un determinado equipo, etc. De esta forma, aunque un usuario malicioso utilice la aplicación de forma incorrecta, se limita el impacto de su ataque. Así, si hemos indicado que sólo puede leer los ficheros de la carpeta c:/datos, el atacante nunca podrá modificar esos datos o leer los ficheros de otro directorio.

Programación segura.

1. Excepciones

Una excepción es un evento que ocurre durante de la ejecución de un programa e interrumpe el flujo normal de las instrucciones.

Por ejemplo, si desea crear una aplicación que lea un archivo y lo envíe por la red, algunas de las posibles excepciones son: que el dispositivo donde se almacena el fichero no esté disponible, que no tenga permisos de lectura sobre el fichero, que el fichero no exista, que la red no esté disponible, etc.

Es importante gestionar las posibles excepciones que puedan ocurrir en el programa para evitar cualquier tipo de fallos en su ejecución. En Java existen varios tipos fundamentales de excepciones:

- **Error.** Excepciones que indican problemas muy graves, que suelen ser no recuperables y no deben casi nunca ser capturadas.
- **Exception.** Excepciones no definitivas, pero que se detectan fuera del tiempo de ejecución.
- **RuntimeException.** Excepciones que se dan durante la ejecución del programa.

De forma general, para incluir el manejo de excepciones en un programa hay que realizar los siguientes pasos:

- ✓ Dentro de un bloque try inserte el flujo normal de las instrucciones.
- ✓ Estudie los errores que pueden producirse durante la ejecución de las instrucciones y compruebe que cada uno de ellos provoca una excepción.
- ✓ Capture y gestione las excepciones en bloques catch.

Cuando el programador ejecuta un código que puede provocar una excepción (por ejemplo: una lectura o escritura de un fichero), debe incluir este fragmento de código dentro de un bloque try:

```
try { // Código posiblemente problemático }
```

Pero lo importante es cómo controlar qué hacer con la posible excepción que se genera. Para ello se utilizan las cláusulas catch que permiten especificar la acción a realizar cuando se produce una determinada excepción.

```

try {
    // Código posiblemente problemático
} catch( tipo_de_excepcion e) {
    // Código para solucionar la excepción e
} catch( tipo_de_excepcion_mas_general e) {
    // Código para solucionar la excepción e
}

```

4

Como puede ver en el ejemplo, se pueden anidar sentencias catch para ir gestionando, de forma diferente, diferentes errores. Es conveniente hacerlo indicando en último lugar las excepciones más generales (es decir, que se encuentren más arriba en el árbol de herencia de excepciones), porque el intérprete Java ejecuta el bloque de código catch cuyo parámetro es el tipo de una excepción lanzada.

Si desea realizar una acción común a todas las excepciones se utiliza la sentencia **finally**. *Este código se ejecuta tanto si se trata una excepción (catch) como sino.*

```

try {
    // Código posiblemente problemático
} catch( Exception e ) {
    // Código para solucionar la excepción e
} finally {
    // Se ejecuta tras try o catch
}

```

Ejemplo I.

A continuación, para aprender a utilizar las excepciones, vamos a ver un ejemplo en el que como se puede ver en el siguiente código, *se genera un vector de 100 elementos con valores aleatorios y posteriormente, se almacenan los valores en el fichero Salida.txt.*

EscribirNumeros.java

// Nota: Este código no funciona. Es simplemente para uso educativo

```

import java.io.*;
import java.util.Vector;
import java.util.Random;

class EscribirNumeros {
    private Vector<Integer> numeros;
    private static final int SIZE = 100;

    public EscribirNumeros () {
        // Generamos el vector con números aleatorios
        numeros = new Vector<Integer>(SIZE);
        Random randomGenerator = new Random();

        for (int i = 0; i < SIZE; i++) {
            Integer num=randomGenerator.nextInt(100);
            numeros.addElement(new Integer(num));
        }

        // Guardamos el vector en un fichero
        PrintWriter out = null;
        System.out.println("Entrando a la zona Try");
        out = new PrintWriter(new FileWriter("Salida.txt"));

        for (int i = 0; i < SIZE; i++)
            out.println("Valor de: " + i + " = " + numeros.elementAt(i));

        out.close();
    }

    public static void main( String[] arg ) {
        new EscribirNumeros ();
    }
}

```

}

En la primera parte del código, donde se generan los valores aleatorios no se puede producir ningún tipo de excepción. Pero a la hora de guardar los valores en el fichero, es posible que no se pueda abrir el fichero y se produzca una excepción de entrada/salida (`IOException`). Otro posible fallo es al ejecutar la sentencia

```
numeros.elementAt(i)
```

5

es que el número de elemento solicitado (*i*) esté fuera del rango del vector (que es de 0 a 99). En el caso de intentar obtener un elemento que se encuentre fuera del vector se genera la excepción **`IndexOutOfBoundsException`**.

En el siguiente código vamos a tratar las dos excepciones comentadas para permitir que el código siempre se ejecute de una forma segura.

EscribirNumeros.java

```
import java.io.*;
import java.util.Vector;
import java.util.Random;

class EscribirNumeros {
    private Vector<Integer> numeros;
    private static final int SIZE = 100;

    public EscribirNumeros () {
        // Generamos el vector con números aleatorios
        numeros = new Vector<Integer>(SIZE);
        Random randomGenerator = new Random();

        for (int i = 0; i < SIZE; i++) {
            Integer num=randomGenerator.nextInt(100);
            numeros.addElement(new Integer(num));
        }

        // Guardamos el vector en un fichero
        PrintWriter out = null;

        try {
            System.out.println("Entrando a la zona Try");
            out = new PrintWriter(new FileWriter("Salida.txt"));

            for (int i = 0; i < SIZE; i++)
                out.println("Valor de: " + i + " = " + numeros.elementAt(i));

        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Caught ArrayIndexOutOfBoundsException: "
+e.getMessage());
        } catch (IOException e) {
            System.err.println("Caught IOException: " + e.getMessage());
        } finally {
            if (out != null) {
                System.out.println("Cerrando el fichero");
                out.close();
            } else {
                System.out.println("NO se ha abierto el fichero");
            }
        }
    }

    public static void main( String[] arg ) {
        new EscribirNumeros ();
    }
}
```

2. Validación de entradas.

Una vía importante de errores de seguridad y de inconsistencia de datos dentro de una aplicación se produce a través de los datos que introducen los usuarios. Un fallo de seguridad muy frecuente, consiste en los errores basados en *buffer overflow*, *se producen cuando se desborda el tamaño de una determinada variable*, vector, matriz, etc. y se consigue acceder a zonas de memoria reservadas. Por ejemplo, si reservamos memoria para el nombre de usuario que ocupa 20 caracteres y, de alguna forma, el usuario consigue que la aplicación almacene más datos, se está produciendo un *buffer overflow* ya que los primeros 20 valores se almacenan en un lugar correcto, pero los restantes valores se almacenan en zonas de memoria destinadas a otros fines.

La validación de datos permite:

- **Mantener la consistencia de los datos.** Por ejemplo, si a un usuario le indicamos que debe introducir su DNI éste debe tener el mismo formato siempre.
- **Evitar desbordamientos de memoria *buffer overflow*.** Al comprobar el formato y la longitud del campo evitamos que se produzcan los desbordamientos de memoria.

Para llevar un riguroso control, sobre los datos de entrada de los usuarios hay que tener en cuenta la validación del formato y la validación del tamaño de la entrada.

Java incorpora una potente y útil librería (`import java.util.regex`) para utilizar la clase **Pattern** que nos permite definir expresiones regulares. Las expresiones regulares permiten definir exactamente el formato de la entrada de datos.

Para utilizar las expresiones regulares debemos realizar los siguientes pasos:

1. Importamos la librería:
`import java.util.regex.*;`
2. Definimos Pattern y Matcher:
`Pattern pat=null;`
`Matcher mat=null;`
3. Compilamos el patrón a utilizar.
`pat=Pattern.compile(patron);`

donde el patrón a comprobar es la parte más importante ya que es donde tenemos que indicar el formato que va a tener la entrada.

Elementos para la validación entradas

Elemento	Comentario.
x	El carácter x.
[abc]	Los caracteres a, b o c.
[a-z]	Una letra en minúscula.
[A-Z]	Una letra en mayúscula.
[a-zA-Z]	Una letra en minúscula o mayúscula.
[0-9]	Un número comprendido entre el 0 y el 9.
[a-zA-Z0-9]	Una letra en minúscula, mayúscula o un número.

Operadores para la validación entradas

Operador	Comentario
[a-z]{2}	Hay que introducir 2 letras en minúsculas.
[a-z]{2,5}	Hay que introducir de 2 a 5 letras en minúsculas.
[a-z]{2,}	Hay que introducir más de 2 letras en minúsculas.
hola adios	Es la operación OR lógica y permite indicar que se introduzca el texto "hola" o "adios".
XY	Es la operación AND lógica y permite indicar que se deben introducir dos expresiones X seguida de Y.
e(n l) campo	Los delimitadores () permite hacer expresiones más complejas. En el ejemplo, el usuario debe introducir el texto "en campo" o "el campo".

A modo de ejemplo, a continuación se muestran varios ejemplos de expresiones:

1. Teléfono (formato 000-000000)

```
pat=Pattern.compile("[0-9]{3}-[0-9]{6}");
```

2. DNI

```
pat=Pattern.compile("[0-9]{8}-[a-zA-Z]");
```

3. Provincias andaluzas

```
pat=Pattern.compile("Almería","Granada","Jaén","Málaga","Sevilla",
"Cádiz","Córdoba","Huelva");
```

4. Le pasamos al evaluador de expresiones el texto a comprobar.

```
mat=pat.matcher(texto_a_comprobar);
```

5. Comprobamos si hay alguna coincidencia:

```
if(mat.find()){
    // Coincide con el patrón
}else{
    // NO coincide con el patrón
}
```

Ejemplo II.

A continuación, vamos a ver un ejemplo en que se valida el texto de entrada para que cumpla el formato de DNI.

ValidarEntrada.java

```
import java.io.*;
import java.util.regex.*;

class ValidarEntrada {

    public ValidarEntrada() {
        String dni_cliente=new String();
        Pattern pat=null;
        Matcher mat=null;

        // para leer del teclado
        BufferedReader reader=new BufferedReader(new
        InputStreamReader(System.in));

        try{
            System.out.println("Introduce tu DNI (Formato 00000000-A):");
            dni_cliente=reader.readLine();

            pat=Pattern.compile("[0-9]{8}-[a-zA-Z]");
            mat=pat.matcher(dni_cliente);

            if(mat.find()){
                System.out.println("Correcto!! "+dni_cliente);
            }else{
                System.out.println("El DNI esta mal "+dni_cliente);
            }

        } catch (Exception e) {
            System.out.println( e.getMessage() );
        }

    }

    public static void main( String[] arg ) {
        new ValidarEntrada();
    }
}
```


}

}

Compilamos el código:

```
javac
ValidarEntrada.java
```

y lo ejecutamos:

```
java ValidarEntrada
```

Podemos ver un ejemplo del resultado de la ejecución de la aplicación.

```
C:\WINDOWS\system32\cmd.exe
C:\ejemplos>javac ValidarEntrada.java
C:\ejemplos>java ValidarEntrada
Introduce tu DNI <Formato 00000000-A>:
34545678-H
Correcto!! 34545678-H
C:\ejemplos>java ValidarEntrada
Introduce tu DNI <Formato 00000000-A>:
34345654H
El DNI esta mal 34345654H
C:\ejemplos>
```

3. *Ficheros de registro.*

Los ficheros de registro o logs, permiten almacenar información sobre las distintas acciones que realiza la aplicación. Estos ficheros permiten realizar un seguimiento detallado de lo que ocurre en el sistema.

Java incorpora la librería **java.util.logging** que permite utilizar la clase **logger** para llevar un control de todos los eventos que ocurren a lo largo de la ejecución de la aplicación

Para trabajar con ficheros de registro en Java debemos realizar los siguientes pasos:

1. Importamos la librería:

```
import java.util.logging.*;
```

2. Buscar o crear el logger que queremos utilizar.

```
Logger logger = Logger.getLogger("MyLog");
```

3. Una vez definido el logger debemos asociarlo a un fichero log:

```
FileHandler fh = new FileHandler("c:\\MyLogFile.log", true);
```

donde `MyLogFile.log` es el fichero donde se guardan el registro de la aplicación. Y la variable `true` indica que los registros se añadan a los ya existentes. Si establece `false`, el fichero se reinicia.

4. Establecer si queremos visualizar los mensajes de log por pantalla. En caso de no querer ver los mensajes por pantalla lo indicamos de la siguiente forma:

```
logger.setUseParentHandlers(false);
```

5. Establecer el formato del fichero. Los tipos de formato pueden ser:

1. **Fichero de texto simple** (SimpleFormatter).

```
SimpleFormatter formatter = new SimpleFormatter();
fh.setFormatter(formatter);
```

2. **Fichero en formato XML** (XMLFormatter).

```
XMLFormatter formatter = new XMLFormatter();
```

6. Establezco el nivel de seguridad de las actividades que quiero registrar.

Niveles de seguridad de los registros

Nivel	Importancia
SEVERE	7 (Máxima)
WARNING	6
INFO	5
CONFIG	4
FINE	3
FINER	2
FINEST	1 (Mínima)

Por ejemplo, si deseo registrar sólo los registros más graves ejecutamos

```
logger.setLevel(Level.SEVERE);
```

Además, de los niveles anteriores, si queremos registrar todos los eventos utilizaremos:

```
logger.setLevel(Level.ALL);
```

Y si no queremos registrar ningún evento, ejecutaremos:

```
logger.setLevel(Level.OFF);
```

9

Una vez inicializado correctamente el logger, el siguiente paso es ir registrando en la aplicación los diferentes registros. Por ejemplo, para añadir un registro de nivel de importancia medio (WARNING) lo creamos de la siguiente forma:

```
logger.log(Level.WARNING, "Mi primer log");
```

Ejemplo III.

A continuación, vamos a ver un ejemplo completo para crear un fichero de registro sencillo de una aplicación.

MyLogger.java

```
import java.io.*;
import java.util.logging.*;
public class MyLogger {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger("MyLog");
        FileHandler fh;

        try {
            // Configuro el logger y establezco el formato
            fh = new FileHandler("c:\\MyLogFile.log", true);
            logger.addHandler(fh);
            logger.setLevel(Level.ALL);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);

            // Añado un mensaje al log
            logger.log(Level.WARNING, "Mi primer log");

        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

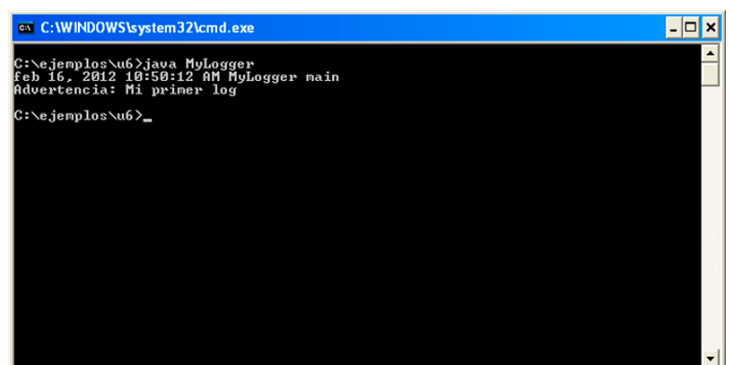
Compilamos el código:

```
javac MyLogger.java
```

y lo ejecutamos:

```
java MyLogger
```

En la siguiente figura podemos ver como la aplicación nos muestra el registro.



Y si abrimos el fichero de registro, en el ejemplo `c:/MyLogFile.log` podemos ver su contenido donde se van guardando los registros.

```
Feb 16, 2012 10:50:12 AM MyLogger main
Advertencia: Mi primer log
```

Políticas de seguridad.

10

1. Modelo de seguridad de java.

La plataforma Java 2 permite un gran conjunto de características de seguridad: basado en políticas de acceso, fácilmente configurable, control de acceso, servicio criptográfico y gestión de claves, etc.

En la siguiente unidad nos centraremos en la utilización del servicio criptográfico y ahora vamos a centrarnos en las políticas de seguridad de las aplicaciones.

Modelos de seguridad en java

Versión	Esquema de seguridad	Descripción
JDK 1.0		Inicialmente, el modelo de seguridad de Java en su primera versión era un modelo muy básico ya que las aplicaciones locales tenían total acceso a todos los recursos del sistema y las aplicaciones remotas no tenían acceso total al sistema.
JDK 1.1		En la siguiente versión, se permitió que las aplicaciones firmadas digitalmente puedan acceder a todos los recursos del sistema.
JDK 1.2		Se incrementa la seguridad del sistema permitiendo establecer las políticas de acceso tanto a las aplicaciones locales o remotas. Una vez que la aplicación es autorizada, se envían las peticiones al Security Manager para poder acceder a los recursos del sistema.

Las políticas de seguridad se pueden establecer utilizando los siguientes elementos:

- **Origen** (usuario o ruta de acceso de la aplicación).
- **Permisos** (por ejemplo, se puede indicar si tiene permisos de lectura/escritura sobre un fichero, si puede enviar o no información).
- **Destino**. Destino al que afecta el permiso. Por ejemplo si trabajamos con ficheros el destino es su ubicación.

- **Acción.** Las acciones que se pueden realizar sobre el fichero. Por ejemplo, en un fichero los permisos son lectura o escritura.

Aunque existen muchos tipos de permisos que se pueden especificar en las políticas de seguridad, en la siguiente tabla se pueden ver las más importantes.

Permiso	Descripción
PropertyPermission.	Permite establecer los permisos de acceso sobre información del sistema (por ejemplo: sistema operativo, versión de java, directorio home del usuario actual)
Documentación oficial (SocketPermission).	Permite establecer los permisos sobre las comunicaciones de red. Los permisos se pueden establecer sobre una determinada dirección IP o URL y/o el puerto de comunicaciones al que puede acceder.
Documentación oficial (FilePermission).	Permite establecer los permisos de acceso al disco de una aplicación. Los permisos se pueden establecer a un determinado fichero o carpeta y, permite establecer los permisos de lectura o escritura.

11

2. Asegurando las aplicaciones

Como hemos comentado anteriormente, en las aplicaciones que se ejecutan localmente, el Security Manager se encuentra deshabilitado. A continuación, vamos a ver cómo aplicar las políticas de seguridad a las aplicaciones para poder controlar los recursos a los que pueden acceder.

GetProps.java

```
// Download from
http://docs.oracle.com/javase/tutorial/security/tour2/examples/GetProps.java

import java.lang.*;
import java.security.*;

class GetProps {

    public static void main(String[] args) {
        /* Test reading properties w & w/out security manager */
        String s;

        try {
            System.out.println("About to get os.name property value");

            s = System.getProperty("os.name", "not specified");
            System.out.println("  The name of your operating system is: " + s);

            System.out.println("About to get java.version property value");

            s = System.getProperty("java.version", "not specified");
            System.out.println("  The version of the JVM you are running is: " + s);

            System.out.println("About to get user.home property value");

            s = System.getProperty("user.home", "not specified");
            System.out.println("  Your user home directory is: " + s);

            System.out.println("About to get java.home property value");

            s = System.getProperty("java.home", "not specified");
            System.out.println("  Your JRE installation directory is: " + s);

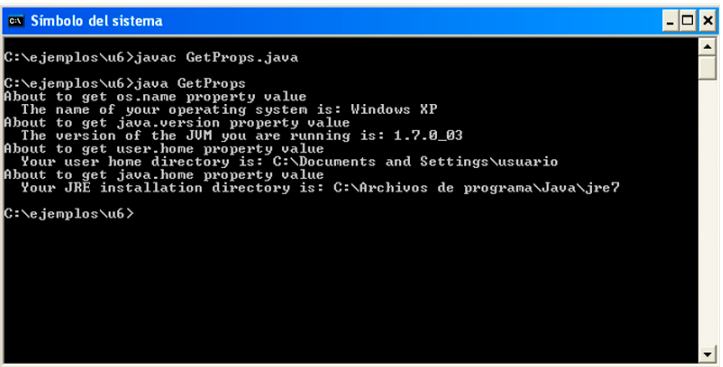
        } catch (Exception e) {
            System.err.println("Caught exception " + e.toString());
        }
    }
}
```

Compila y ejecuta el código:

```
javac GetProps.java
```

```
java GetProps
```

Como puedes ver, la aplicación tiene acceso a las propiedades del sistema.



```

C:\ejemplos\6>javac GetProps.java
C:\ejemplos\6>java GetProps
About to get os.name property value
The name of your operating system is: Windows XP
About to get java.version property value
The version of the JVM you are running is: 1.7.0_03
About to get user.home property value
Your user home directory is: C:\Documents and Settings\usuario
About to get java.home property value
Your JRE installation directory is: C:\Archivos de programa\Java\jre7
C:\ejemplos\6>

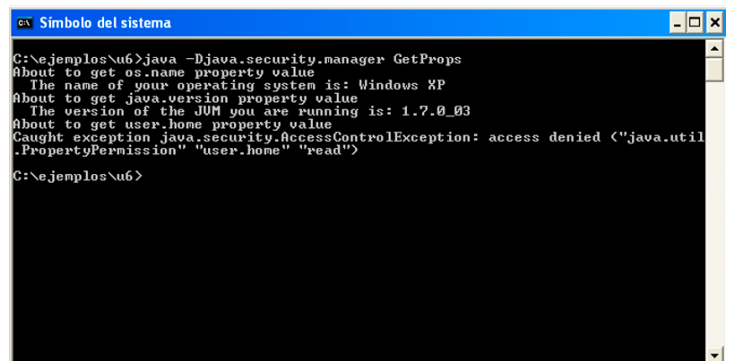
```

Para poder aplicar las políticas de seguridad, en una aplicación local, debemos activar el Security Manager utilizando la opción `-Djava.security.manager`, por lo que ejecutaremos:

```
java -Djava.security.manager GetProps
```

Las políticas de seguridad que se cargan por defecto, permiten al código a acceder a datos que no son sensibles y no son un riesgo de seguridad como la versión del sistema operativo (`os.name`) y la versión de java (`java.version`). En cambio, en los datos más sensibles como el directorio home del usuario (`user.home`) o el directorio de trabajo de java (`java.home`) el gestor de seguridad no permite el acceso de lectura y por eso se produce la excepción de que no tiene permisos (`AccessControlException`).

Tal y como podemos ver, se ha denegado el acceso a ("java.util.PropertyPermission" "user.home" "read"). Estos datos son necesarios para poder establecer correctamente las políticas de acceso.



```

C:\ejemplos\6>java -Djava.security.manager GetProps
About to get os.name property value
The name of your operating system is: Windows XP
About to get java.version property value
The version of the JVM you are running is: 1.7.0_03
About to get user.home property value
Caught exception java.security.AccessControlException: access denied ("java.util.PropertyPermission" "user.home" "read")
C:\ejemplos\6>

```

A continuación se va a configurar las políticas de seguridad para permitir el acceso de la aplicación para que pueda leer las propiedades en la que no había acceso (`user.home` y `java.home`).

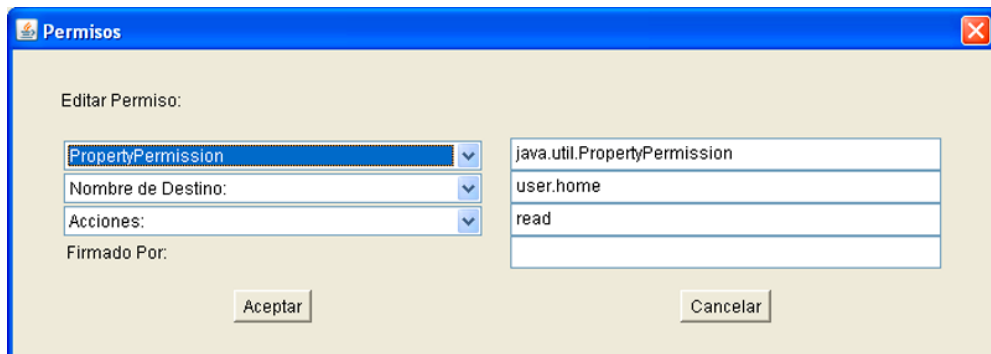
Los pasos que tenemos que realizar para establecer los permisos son:

1. Ejecutamos el comando `policytool` para iniciar la herramienta que nos permite administrar las políticas de acceso.
2. Nos vamos al menú **Archivo -> Abrir** y seleccionamos el fichero `java.policy` que se encuentra en `c:\Archivos de programa\Java\jre7\lib\security` (hay que tener cuidado porque la ubicación puede cambiar dependiendo de la versión de java).
3. Una vez abierto el fichero de políticas de acceso, tenemos que añadir una entrada para permitir el acceso de nuestro código. Para ello, realizamos los siguientes pasos:



- Pulsamos el botón **Agregar Entrada de Política** y podemos indicar el origen del fichero por su ubicación (`codebase`) o por su firma (`SignedBy`). En nuestro caso, vamos a identificar la aplicación por su ubicación por lo que escribimos `file:/C:/ejemplo/` (`C:/ejemplo` es la ubicación donde estamos trabajando. Si utilizas otro directorio cámbialo). Si vamos a establecer los permisos en una aplicación en la red sería `http://direccion`.

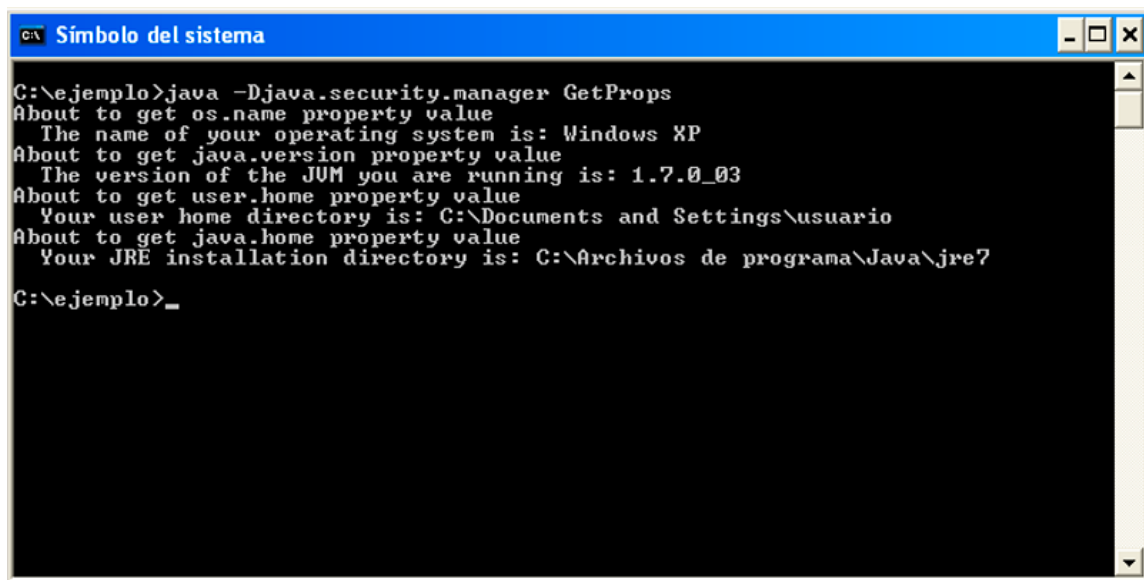
- Pulsamos en Agregar Permiso y añadimos un permisos con los siguientes datos:
- Añadimos una nueva política de acceso con los siguientes datos:
 - **PropertyPermission** = `java.util.PropertyPermission`.
 - **Nombre de Destino** = `user.home`.
 - **Acciones** = `read`
- Pulsamos en Agregar Permiso para añadir el segundo permiso con los siguientes datos:
 - **PropertyPermission** = `java.util.PropertyPermission`.
 - **Nombre de Destino** = `java.home`.
 - **Acciones** = `read`



4. Finalmente, guardamos todos los cambios.

Una vez configurada la política de acceso, volvemos a ejecutar la aplicación y ya podemos comprobar que la aplicación se ejecuta correctamente.

```
java -Djava.security.manager GetProps
```



3. Firmando ficheros Jar.

Los ficheros **JAR** (ARchive of Java) contienen componentes de Java (como archivos de clase) y, opcionalmente, recursos de soporte (como imágenes o sonidos).

Como hemos visto antes, las políticas de seguridad se pueden especificar por su ubicación o por la persona que firma el fichero `jar`. A continuación, vamos a aprender a crear un fichero `jar` y firmarlo digitalmente para, más adelante, establecer correctamente sus políticas de acceso.

Los pasos que debemos realizar son:

1. Utiliza siguiente código fuente o utiliza cualquier otro código que estimes conveniente.

GetProps.java

```
// Download from
http://docs.oracle.com/javase/tutorial/security/tour2/examples/GetProps.java

import java.lang.*;
import java.security.*;

class GetProps {

    public static void main(String[] args) {
        /* Test reading properties w & w/out security manager */
        String s;

        try {
            System.out.println("About to get os.name property value");

            s = System.getProperty("os.name", "not specified");
            System.out.println("  The name of your operating system is: " + s);

            System.out.println("About to get java.version property value");

            s = System.getProperty("java.version", "not specified");
            System.out.println("  The version of the JVM you are running is: " + s);

            System.out.println("About to get user.home property value");

            s = System.getProperty("user.home", "not specified");
            System.out.println("  Your user home directory is: " + s);

            System.out.println("About to get java.home property value");

            s = System.getProperty("java.home", "not specified");
            System.out.println("  Your JRE installation directory is: " + s);

        } catch (Exception e) {
            System.err.println("Caught exception " + e.toString());
        }
    }
}
```

2. Compila el código fuente:

```
javac GetProps.java
```

3. Crea el fichero jar ejecutando:

```
jar cvf GetProps.jar GetProps.class
```

4. Genera las claves. A continuación vamos a crear un par de claves (una clave privada y una clave pública) para firmar el fichero. La clave privada la vamos a utilizar para firmar el fichero y la clave pública la van a utilizar los demás usuarios para comprobar que el fichero es correcto.

Para generar las claves ejecutamos:

```
keytool -genkey -alias firmar -keypass hola00 -keystore DAM -storepass
distancia
```

donde:

- **-alias firmar.** Indica el alias que se va a utilizar para referirnos al keystore, que es donde se van a almacenar las llaves generadas.
- **-keypass hola00.** Especifica la contraseña de la llave privada.
- **-keystore DAM.** Indica el nombre del keystore que se esta creando o utilizando.
- **-storepass distancia.** Indica el password del keystore.

Tal y como se puedes ver, hay que introducir los datos del certificado: nombre, organización, localidad, etc.

15

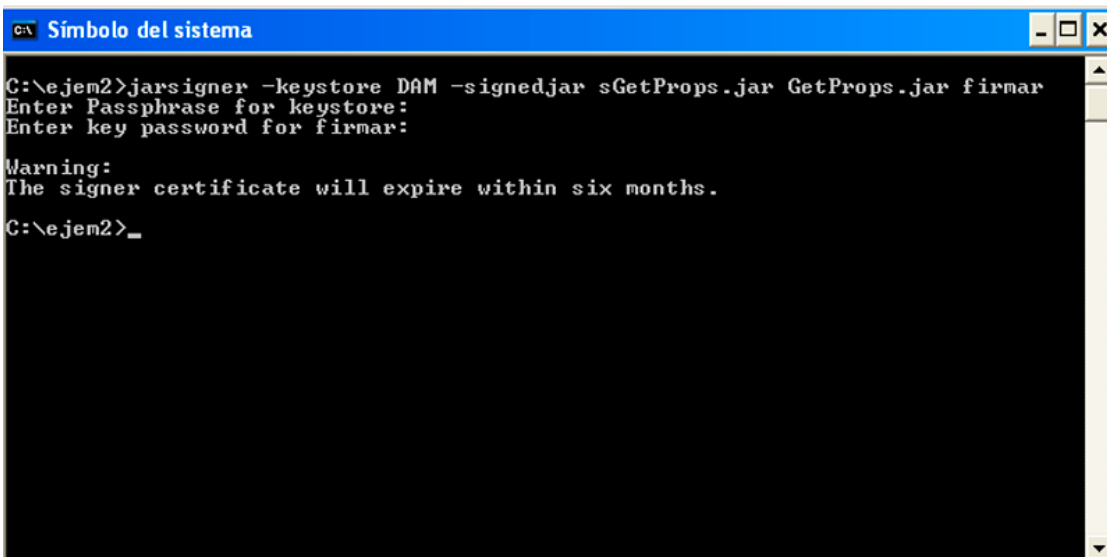
```
C:\ejem2>keytool -genkey -alias firmar -keypass hola000 -keystore DAM -storepass
distancia
¿Cuáles son su nombre y su apellido?
[Unknown]: DAM Distancia
¿Cuál es el nombre de su unidad de organización?
[Unknown]: Andalucía
¿Cuál es el nombre de su organización?
[Unknown]: MEC
¿Cuál es el nombre de su ciudad o localidad?
[Unknown]: Almería
¿Cuál es el nombre de su estado o provincia?
[Unknown]: Almería
¿Cuál es el código de país de dos letras de la unidad?
[Unknown]: SP
¿Es correcto CN=DAM Distancia, OU=Andalucía, O=MEC, L=Almería, ST=Almería, C=SP?
[no]: si
```

5. Finalmente, firmamos el fichero jar utilizando el certificado creado

```
jarsigner -keystore DAM -signedjar sGetProps.jar GetProps.jar firmar
```

donde:

- -keystore DAM. Es el keystore creado anteriormente.
- -signedjar sGetProps.jar. Indica el fichero jar firmado.
- GetProps.jar Es el fichero jar a firmar.
- firmar. Es el alias que se ha creado anteriormente.



```
C:\ejem2>jarsigner -keystore DAM -signedjar sGetProps.jar GetProps.jar firmar
Enter Passphrase for keystore:
Enter key password for firmar:

Warning:
The signer certificate will expire within six months.
C:\ejem2>_
```

Exportamos la llave pública del certificado ejecutando:

```
keytool -export -keystore DAM -alias firmar -file Javier.cert
```

Ahora ya tenemos todos los archivos necesarios para enviarlos a los diferentes clientes que quieran utilizar nuestra aplicación: el fichero **jar** firmado (sGetProps.jar) y el certificado (Javier.cert).

Una vez que tenemos el fichero jar firmado (**sGetProps.jar**) y el certificado de seguridad (**Javier.cert**), podemos intentar ejecutar directamente la aplicación, pero veremos que al utilizar el **SecurityManager** nos da problemas de seguridad.

```
java -cp sGetProps.jar GetProps
java -Djava.security.manager -cp sGetProps.jar GetProps
```

```
C:\ejem2>java -cp sGetProps.jar GetProps
About to get os.name property value
The name of your operating system is: Windows XP
About to get java.version property value
The version of the JUM you are running is: 1.7.0_03
About to get user.home property value
Your user home directory is: C:\Documents and Settings\usuario
About to get java.home property value
Your JRE installation directory is: C:\Archivos de programa\Java\jre7

C:\ejem2>java -Djava.security.manager -cp sGetProps.jar GetProps
About to get os.name property value
The name of your operating system is: Windows XP
About to get java.version property value
The version of the JUM you are running is: 1.7.0_03
About to get user.home property value
Caught exception java.security.AccessControlException: access denied ("java.util
.PropertyPermission" "user.home" "read")

C:\ejem2>_
```

Para que la aplicación funcione correctamente, primero debemos importar el certificado ejecutando:

```
keytool -import alias Javier -file Javier.cert -keystore DAM
```

donde:

- **-import alias Javier**. Indica que se va a importar el certificado con el alias **Javier**.
- **-keystore DAM**. Indica el keystore donde se guarda el certificado.

Una vez que tenemos el alias, tenemos que configurar las políticas de acceso para que el usuario **Javier** pueda ejecutar correctamente la aplicación. Para ello, debemos realizar los siguientes pasos:

1. Ejecutamos el comando **policytool** para iniciar la herramienta que permite administrar las políticas de acceso.
2. Nos vamos al menú **Archivo -> Abrir** y seleccionamos el fichero **java.policy** que se encuentra en **c:\Archivos de programa\Java\jre7\lib\security\java.policy** (hay que tener cuidado porque la ubicación puede cambiar dependiendo de la versión de java).
3. Una vez abierto el fichero de políticas de acceso tenemos que indicar el keystore que vamos a utilizar. Para ello, en el menú **Almacen de claves** seleccionamos **Editar** y en el campo **URL de almacén de claves** indicamos la ubicación de las claves (que en nuestro caso es **file:\c:\ejemplo\DAM**).
4. Añadimos una entrada para permitir el acceso de nuestro código. Para ello, realizamos los siguientes pasos:
5. Pulsamos el botón **Agregar Entrada de Política** y podemos indicar el origen del fichero por su ubicación (codebase) o por su firma (SignedBy). En esta ocasión, en el campo **SignedBy** escribimos nuestro usuario **Javier**.
6. Pulsamos en **Agregar Permiso** y añadimos un permisos con los siguientes datos.
7. añadimos una nueva política de acceso con los siguientes datos:
 - **PropertyPermission = java.util.PropertyPermission**

- Nombre de Destino = `user.home`
- Acciones = `read`

8. Pulsamos en *Agregar Permiso* para añadir el segundo permiso con los siguientes datos:

- `PropertyPermission = java.util.PropertyPermission`
- Nombre de Destino = `java.home`
- Acciones = `read`

9. Finalmente, guardamos todos los cambios.

Una vez configurada la política de acceso, volvemos a ejecutar la aplicación y ya podemos comprobar que la aplicación se ejecuta correctamente.

```
java -Djava.security.manager -cp sGetProps.jar GetProps
```

4. Herramientas de seguridad.

El JDK proporciona herramientas que se utilizan para implementar las normas de seguridad de Java. Las herramientas que podemos utilizar son:

- **policytool.** *Es una herramienta gráfica que permite administrar las normas de seguridad de una instalación local de JDK.* Para utilizar la herramienta debemos ejecutar:

```
policytool
```

Una vez ejecutada la herramienta **policytool**, selecciona *Open* en el menú *File* y luego selecciona el archivo de normas que desea modificar. La ruta predeterminada es **c:\Archivos de programa\Java\jre7\lib\security\java.policy**.

- **keytool.** *Permite administrar la base de datos de claves y certificados del sistema.* Para utilizarla la herramienta ejecuta:

```
keytool [comandos]
```

Si ejecutas **keytool -help** podrás ver la lista de órdenes y sus respectivas opciones.

- **jarsigner.** Permite firmar y verificar la firma de los ficheros jar. Su sintaxis es:

```
jarsigner [opciones] Fichero.jar alias
```

donde

- **Fichero.jar** Es el fichero **jar** que se firma o verifica
- **Alias.** Es el alias de entrada al almacén de claves que contiene la clave privada que se utiliza para generar la firma. El almacén de claves es una base de datos de claves y certificados que mantiene la herramienta **keytool**.
- **Opciones.** Son las opciones que utiliza **jarsigner** y que podemos ver al ejecutar simplemente **jarsigner**.

Anexo I.- Ataques.

18

Nombre	Descripción
Sistemas	
Explotar bugs del software	Utilizar fallos de seguridad en el software para atacar un sistema.
Romper contraseñas	Fuerza bruta o ataques basados en diccionarios que permiten obtener las contraseñas del sistema o de un determinado servicio.
Red	
Barridos de ping	Utilización del protocolo ICMP para determinar los equipos activos de una red.
Confianza transitiva	Aprovechar la confianza entre usuarios o equipos para tomar sus privilegios.
DNS spoofing	Falsificación de una entrada DNS que apunta a un servidor no autorizado
DoS	Ataque de denegación de servicio que consiste en saturar un sistema para impedir la utilización correcta del sistema.
Hijacking	Permite a un usuario robar una conexión de un usuario que ha sido autenticado en el sistema.
Man in the middle	A través del ataque ARPspoofing el atacante se sitúa en medio de la comunicación entre varios equipos para realizar otros ataques, como sniffer, spoofing, phishing etc.
Mensajes de control de red o enrutamiento fuente	Se envían paquetes ICMP para hacer pasar los paquetes por un router comprometido.
Navegación anónima	No se considera directamente un ataque, pero la suelen utilizar los atacantes para realizar sus fechorías. Se denomina "navegación anónima" cuando un usuario utilizar diferentes servidores proxy para ocultar su dirección IP.
Phising	Ataque informático que consiste en falsificar un sitio web para poder obtener las contraseñas de sus usuarios.
Reenvío de paquetes	Retransmisión de paquetes para engañar o duplicar un mensaje (p.ej. una transferencia).
Sniffer	Programa o equipo que registra todo el tráfico de una red. Se utiliza especialmente para obtener las contraseñas de los sistemas.
Spoofing	El atacante envía paquetes con una dirección fuente incorrecta. Las respuestas se envían a la dirección falsa. Pueden usarse para: Acceder a recursos confiados sin privilegios Para DoS directo como indirecto o recursivo.
Servidores web	
Inyección SQL	Ataque que consiste en modificar las consultas SQL de un servidor web para poder realizar consultas SQL maliciosas.
LFI	Ataque informático que consiste en hacer que un servidor ejecute un script que está alojado en el mismo servidor.
RFI	Ataque informático que consiste en hacer que un servidor ejecute un script que está alojado en una máquina remota.
XSS	Consiste en engañar al servidor web para que ejecute un script malicioso en el navegador del cliente que visita una determinada página.
Aplicaciones	
Crack	Software que permite romper la protección de una aplicación comercial.
Keylogger	Software o hardware que registra todas las pulsaciones de teclado que se realizan en el sistema.
Rootkit	Software que se instala en un sistema y oculta toda la actividad de un usuario (el atacante).
Troyano	Software que se instala en el ordenador atacado que permite al atacante hacerse con el control de la máquina.
Virus	Software que se copia automáticamente y que tiene por objeto alterar el normal funcionamiento del equipo sin el permiso o el conocimiento del usuario.
Varios	
Ingeniería social	Atacante que consiste en convencer a un usuario legítimo para que facilite información (contraseñas, configuraciones, etc.).
Rubber-hosse	Utilizar soborno o tortura para obtener una determinada información.