1.- Introducción a la seguridad Informática.

1.1.- Amenazas de seguridad.

1.2.- Ataques.

1.3.- Vulnerabilidades en el software.

2.- Programación segura.

2.1.- Validación de entradas.

3.- Políticas de seguridad.

4.- Criptografía.

4.1.- Encriptación de la información.

4.2.- Criptografía de clave privada o simétrica.

4.3.- Criptografía de clave pública o asimétrica.

## 1.- Introducción a la seguridad Informática.

Los sistemas informáticos 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.

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.

### 1.1.- 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 cuatro categorías generales de amenazas de seguridad son:

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.

**1.2.- Ataques.**

Los ataques se clasifican en 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 pasa 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.

**1.3.- 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:

Seguridad interna de aplicación. A la hora de realizar la aplicación se debe programar de una forma robusta para que comporte tal y como esperamos de ella. Algunas de las técnicas que podemos implementar, 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.

**2.- Programación segura.**

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. 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 clausulas 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
}
```

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
}
```

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.

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) 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

### 2.2.- 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.
A modo de ejemplo, a continuación se muestran varios ejemplos de expresiones:
Teléfono (formato 000-000000)
pat=Pattern.compile("[0-9]{3}-[0-9]{6}");
DNI
pat=Pattern.compile("[0-9]{8}-[a-zA-Z]");
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 }
A continuación, vamos a ver un ejemplo validarEntrada.java en que se valida el texto de entrada para que cumpla el formato de DNI.

**3.- Políticas de seguridad.**

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 trabamos 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.

**4.- Criptografía.**

El término criptografía viene del griego "cripto" (secreto) y "grafía" (escritura), por lo que su significado es "escritura secreta". La finalidad de la criptografía es enmascarar o codificar una información original, utilizando alguna técnica, de manera que el resultado sea ininteligible para las personas no autorizadas.

La criptografía se ha usado a lo largo de los años para mandar mensajes confidenciales o privados. Básicamente, cuando alguien quiere mandar información confidencial de forma secreta, actúa de la siguiente manera:

Aplica técnicas criptográficas para poder "enmascarar" el mensaje.

Manda el mensaje "enmascarado" por una línea de comunicación que se supone insegura y puede ser interceptada.

Solo el receptor autorizado podrá leer el mensaje "enmascarado".

Otra disciplina relacionada con la criptografía es el criptoanálisis, que analiza la robustez de los sistemas criptográficos y comprueba si realmente son seguros. Para ello, se intenta romper la seguridad que proporciona la criptografía, deshaciendo el sistema y accediendo de esta forma a la información secreta en su formato original. Esta disciplina

es una herramienta muy poderosa que permite mejorar los sistemas de criptografía constantemente y ayuda a desarrollar otros nuevos más efectivos.
Por último, y para terminar de recorrer estas disciplinas, debes conocer el término criptología, que es la ciencia que engloba tanto las técnicas de criptografía como las de criptoanálisis.

### 4.1.- Encriptación de la información.

La encriptación o cifrado de la información es el proceso por el cual la información o los datos a proteger son traducidos o codificados como algo que parece aleatorio y que no tiene ningún significado (datos encriptados).

La desencriptación o descifrado es el proceso inverso, en el cual los datos encriptados son convertidos nuevamente a su forma original.

A continuación, te indicamos los conceptos o términos asociados con la encriptación:

Texto llano o claro: es la información original, la que no está cifrada o encriptada.

Criptograma o texto cifrado: es la información obtenida tras la encriptación.

Algoritmo criptográfico o algoritmo de cifrado: es un conjunto de pasos u operaciones, normalmente una función matemática, usado en los procesos de encriptación y desencriptación. Asociado al algoritmo hay una clave o llave (un número, palabra, frase, o contraseña).

La clave controla las operaciones del algoritmo dentro del proceso de cifrado y descifrado, de manera que usando el mismo algoritmo con llaves diferentes, se obtienen textos cifrados o criptogramas diferentes.

La clave puede ser simétrica (misma llave para cifrar y descifrar) o asimétrica (llaves diferentes para cifrar y descifrar).

### 4.2.- Criptografía de clave privada o simétrica.

Este método de encriptación utiliza una clave secreta, conocida solo por emisor y receptor, para encriptar la información. Se la denomina también criptografía simétrica porque la clave utilizada para la encriptación y desencriptación es la misma. Es adecuada para garantizar confidencialidad.

A continuación te indicamos los principales aspectos de la criptografía de clave privada o simétrica:

La clave es privada o secreta, solo la conocen las partes involucradas.

Se utiliza la misma clave para el cifrado y descifrado. Así, por ejemplo, cuando A le

envía información a B, utiliza la clave privada para cifrar el mensaje, que solo podrá descifrar B si también conoce la clave. Por tanto, A debe hacer llegar a B la clave privada.

Ventaja:

Son algoritmos muy rápidos y que no aumentan el tamaño del mensaje; por tanto adecuados para cifrar grandes volúmenes de datos.

Inconvenientes:

El problema de la distribución de claves: el receptor debe conocer la clave que se va a utilizar, lo que implica que el emisor se la debe enviar y no es posible utilizar medios inseguros para el intercambio de claves.

Otro inconveniente es el gran número de claves que se necesitaría si el grupo de personas que puede comunicarse de forma privada es muy grande, pues se necesitaría una clave diferente cada dos personas del grupo.

Algunos ejemplos de algoritmos de este tipo son: AES o Rijndael (Nuevo estándar mundial), DES, 3-DES, DES-X, IDEA y RC5.

**4.3.- Criptografía de clave pública o asimétrica.**

La criptografía de clave pública surge para solucionar el problema de distribución de claves que plantea la criptografía de clave privada, permitiendo que emisor y receptor puedan acordar una clave en común sobre canales inseguros. Se la denomina también criptografía asimétrica porque las claves utilizadas para la encriptación y desencriptación son diferentes.

Es adecuada para garantizar además de confidencialidad, la autenticación de los mensajes y el no repudio.

Características:

Cada parte posee una pareja de claves, una pública, conocida por todos, y su inversa privada, conocida solo por su poseedor.

Cada pareja de claves, son complementarias: lo que cifra una de ellas, solo puede ser descifrado por su inversa.

Esa pareja de claves sólo se puede generar una vez, de modo que se puede asumir que no es posible que dos personas hayan obtenido casualmente la misma pareja de claves.

Al cifrar un mensaje con la clave pública, tan solo podrá descifrarlo quien posea la clave privada inversa a esa clave pública. Así, por ejemplo, cuando A le envía información a B, utiliza la clave pública de B para cifrar el mensaje, que solo podrá descifrar B con su clave privada.

Conocer la clave pública no permite obtener ninguna información sobre la clave

privada, ni descifrar el texto que con ella se ha cifrado.

Cifrar un mensaje con la clave privada equivale a demostrar la autoría del mensaje, autenticación, nadie más ha podido cifrarlo utilizando esa clave privada. El mensaje cifrado con una clave privada podrá descifrarlo todo aquel que conozca la clave pública inversa. Por tanto, si A enviase un mensaje cifrado con su clave privada, podrá descifrarlo todo el que conozca la clave pública de A y además sabrá que el mensaje proviene de A.

Ventaja:

No existe el problema de distribución de claves, ya que cada parte posee su juego de claves.

Inconvenientes:

Son más lentos que los algoritmos simétricos.

Garantizar que la clave pública es realmente de quien dice ser su poseedor, lo que se conoce como el problema del 'ataque del hombre en el medio' o man in the middle. (Es un ataque en el que el atacante es capaz de observar e interceptar mensajes entre las dos partes sin que ninguna de ellas sepa que el enlace entre ellos ha sido violado).

Algunos ejemplos de algoritmos de este tipo son: DSA, RSA(estándar de facto), algoritmo de Diffie-Hellman.

Generalmente se utiliza una combinación de ambas: criptografía asimétrica para negociar una clave privada con la que después se comunicarán los datos.