

Universidad ORT Uruguay
Facultad de Ingeniería

Aspectos de Seguridad - Obligatorio

Entregado como requisito para la obtención del crédito Aspectos de Seguridad

Segundo Semestre 2019

Marcela Ferraz - 200112

Mauricio Zito - 111725

Docentes: Santiago Paz, Felipe Sotuyo

Tabla de contenido

Introducción 3

 Tecnología utilizada..... 3

 Parte 1 – Sistema seguro Bell Lapadula 4

 1.1 Desarrollo 4

 1.2 Instalación 7

 1.3 Prueba 9

 1.4 Conclusión 11

 Parte 2 – Canal Encubierto 12

 Introducción 12

 2.1 Desarrollo 12

 2.2 Instalación 18

 2.3 Pruebas..... 19

 2.4 Conclusión 22

Conclusión Final 23

Introducción

En el marco de la materia Aspectos de seguridad, se solicita un trabajo práctico en caracter de obligatorio donde se implementará un modelo de seguridad basado en el planteado por la dupla de Billy Elliott Bell y Len LaPadula, consiste en dividir el permiso de acceso de los usuarios a la información en función de etiquetas de seguridad. Por ejemplo, en sistemas militares norteamericanos, categorizándola en 4 niveles: no clasificado, confidencial, secreto y ultrasecreto.

El modelo define 2 reglas de control de acceso mandatorio (MAC):

Propiedad de seguridad simple: Un sujeto de un determinado nivel de seguridad no puede leer un objeto perteneciente a un nivel de seguridad más alto.

Propiedad *: Un sujeto de un determinado nivel de seguridad no puede escribir un objeto perteneciente a un nivel de seguridad más bajo. (También llamada propiedad de confinamiento).

Tecnología utilizada

El obligatorio fue implementado utilizando:

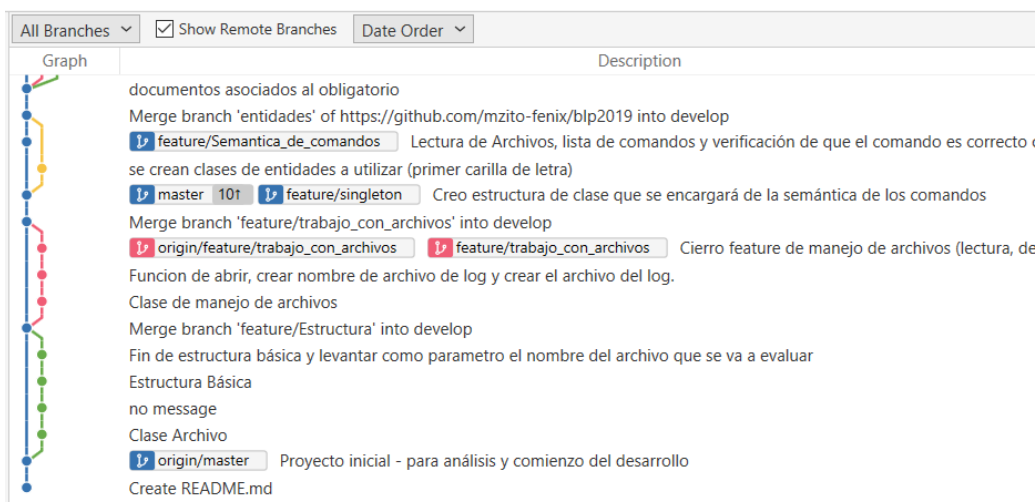
- Lenguaje Java
- IDE Netbeans versión 8.0.2

Version de Java

- JDK 1.8

Control de versiones, desarrollo compartido y organización de la tarea

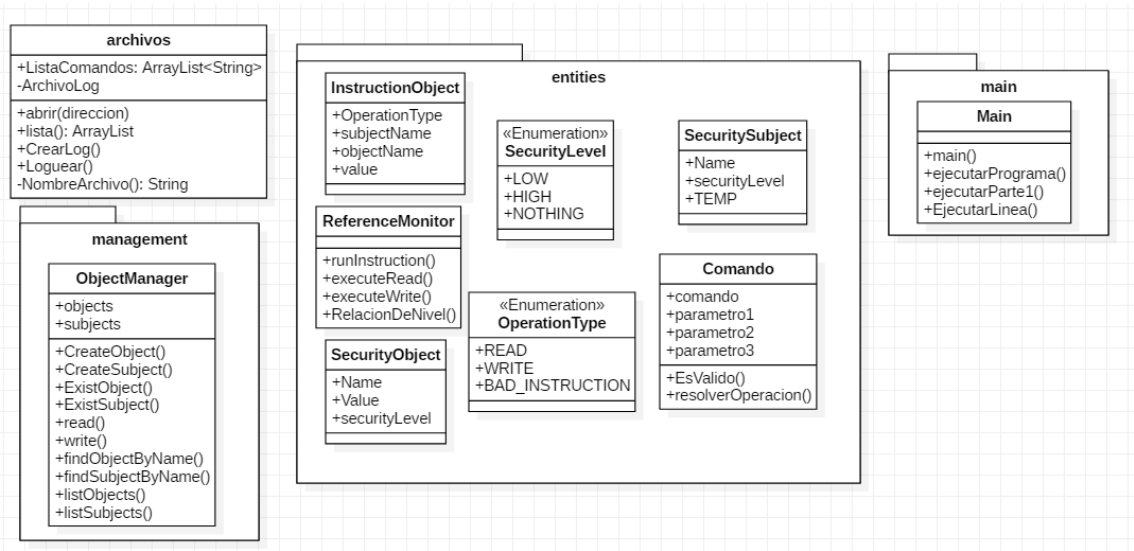
- Utilizamos github y sourcetree para poder trabajar en conjunto

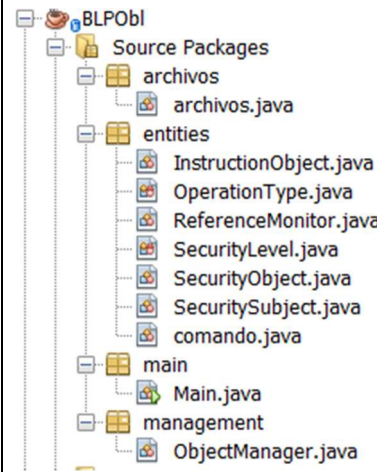


Parte 1 – Sistema seguro Bell Lapadula

1.1 Desarrollo

Vista de descomposición



	archivos	Archivos	Clase que controla la lectura y escritura de archivos (lectura de instrucciones y escritura de log), interacción con el sistema operativo.
	Entities	InstructionObject	Clase que resume información para la ejecución del comando
		OperationType	Enumeración de tipos de operaciones. READ, WRITE, BAD_INSTRUCTION
		ReferenceMonitor	Control de instrucciones, control de relación de permiso y ejecución.
		SecurityLevel	Enumeración de tipos de niveles de seguridad LOW, HIGH, NOTHING

		SecurityObject	Representación de un Objeto del modelo Bell Lapadula
		SecuritySubject	Representación de un Sujeto del modelo Bell Lapadula
		Comando	Encargado de la verificación de correctitud y separación de las operaciones y parámetros para su posterior ejecución
	Main	Main	Programa principal, funcion Main.
	Managemen t	ObjectManager	Controlador que contiene el comportamiento de una base de datos para registrar el listado de sujetos, objetos y sus estados

La función principal

```
public static void main(String[] args) {  
    try {  
        ejecutarPrograma(args);  
    } catch (Exception e) {  
        System.out.println("Error");  
    }  
}
```

La función “ejecutarPrograma” recibe los parámetros recibidos por línea de comandos directo del main.

```
private static void ejecutarPrograma(String[] args) throws IOException {
```

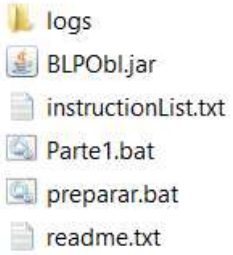
Varias de las funciones fueron implementadas con el patrón “Singleton”, de ésta forma es posible utilizarlas de forma transversal desde diferentes secciones del proyecto.

```
    archivos archivoLog =archivos.getInstance();  
    archivoLog.CrearLog();  
  
    ObjectManager sys=ObjectManager.getInstance();  
  
    //Si no se reciben parametros, finaliza la ejecución  
    if (args.length <1) {  
        System.out.println("Debe ingresar el nombre del archivo a evaluar (con dirección absoluta)");  
        System.exit(0);  
    }  
}
```

```
private static void ejecutarPartel(String Archivo) throws IOException{  
    ObjectManager OM=ObjectManager.getInstance();  
    ArrayList<String> listaComandos = new ArrayList<String>();  
    archivos archivoLog =archivos.getInstance();  
    archivoLog.Loguear("Abriendo archivo->" + Archivo);  
    int lineas=archivoLog.abrir(Archivo);  
    if(lineas>0){  
        listaComandos=archivoLog.ListaComandos;  
        int n=0;  
        String lineaActual;  
        String comandoActual;  
        comando Comando=new comando();  
        for(n=0;n<lineas;n++){  
            lineaActual=listaComandos.get(n);  
            if(Comando.Separar(lineaActual)){  
                comandoActual=Comando.getComando().toString();  
                if(Comando.EsValido(Comando.getComando().toString(),Comando.getParametro1(),Comando.getParametro2(),Comando.getParametro3()))  
                {  
                    Ejecutar(Comando);  
                    archivoLog.Loguear(comandoActual );  
                }  
                else  
                {  
                    archivoLog.Loguear("BAD_INSTRUCTION");  
                }  
            }  
            else  
            {  
                archivoLog.Loguear("BAD_INSTRUCTION");  
            }  
        }  
    }  
    else  
    {  
        archivoLog.Loguear("El archivo estaba vacio");  
    }  
}
```

1.2 Instalación

Para realizar la prueba, solo hay que copiar la carpeta “test” que se encuentra dentro del proyecto a cualquier carpeta del equipo destino.

	En la misma carpeta debe estar el archivo “InstructionList.txt” que es donde se encuentran las instrucciones a ejecutar.
---	--

Se debe usar **parte1.bat** para ejecutarlo

1.2.1 Sobre el Log

Se implementó una clase para realizar el log del sistema, utilizando las siguientes librerías.

```
import java.io.BufferedReader;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
```

Se compone de 2 partes

- La creación del archivo de log

En la subcarpeta “logs” del proyecto se genera un archivo bajo el siguiente patrón de nombre

```
String respuesta="";
Calendar fecha = Calendar.getInstance();
int año = fecha.get(Calendar.YEAR);
int mes = fecha.get(Calendar.MONTH) + 1;
int dia = fecha.get(Calendar.DAY_OF_MONTH);
int hora = fecha.get(Calendar.HOUR_OF_DAY);
int minuto = fecha.get(Calendar.MINUTE);
int segundo = fecha.get(Calendar.SECOND);
```

Se compone de la palabra “Log_” seguido de Año, Mes, Día, Hora, minuto y segundo y le asigna la extensión “.log” de ésta forma nos aseguramos que los logs se puedan acumular y no se pierdan registros de cada ejecución independiente.


Crear Log

La función CrearLog se encarga de crear un archivo de registro con nombre único y por única vez para toda la ejecución.

```

public void CrearLog() throws IOException
{
    String path = new File(".").getAbsolutePath();
    path=path.substring(0,path.length()-1);
    archivoLog=path + "\\logs\\"+ NombreArchivo();
    System.out.println(archivoLog);
    FileWriter fichero=null;
    fichero=new FileWriter(archivoLog);
}

```

 Log_2019128121848.log	08/12/2019 12:18
---	------------------

b. El Append de nueva linea al archivo de log

Es la función de la clase que se encarga de hacer “Append” al archivo utilizado y designado por la instancia de ejecución actual.

Tiene la doble funcionalidad de dejar registrado en un archivo la salida y tambien de mostrar en pantalla (por consola) lo que el programa va realizando.


```

public void Loguear(String linea) throws IOException
{
    FileWriter fichero=null;
    fichero=new FileWriter(archivoLog,true);
    fichero.write(linea);
    fichero.write("\r\n");
    fichero.close();
    System.out.println(linea);
}

```

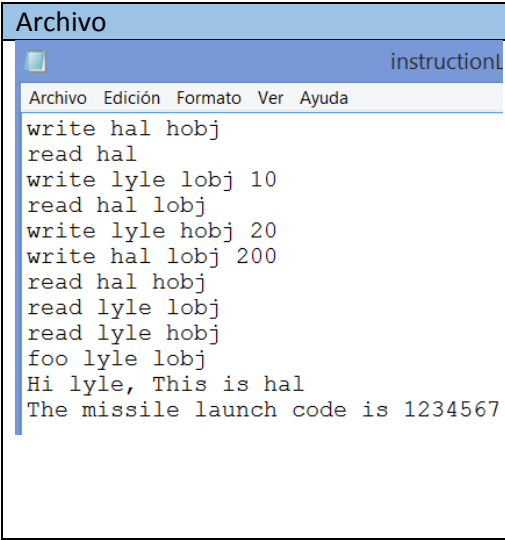
1.2.2 Ejecutar

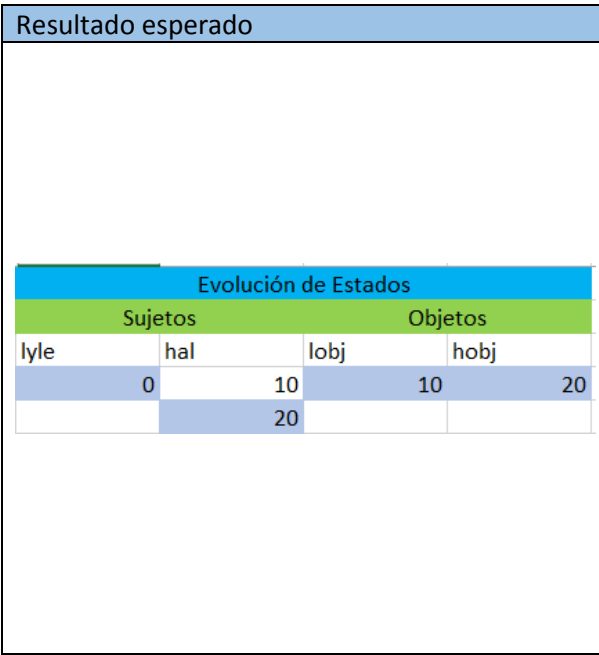
Para correr el programa se debe ejecutar utilizando el archivo “Parte1.bat” que tiene el siguiente contenido.

	ejecutar.bat: Bloc de notas
Archivo Edición Formato Ver Ayuda	
java -jar "BLPObl.jar" ".\instructionList.txt"	
Pause	

1.3 Prueba

Prueba 1 – Verificación según Instruction list de letra del obligatorio

Archivo	Resultado de cada linea esperada																										
	<table><tr><th>Comandos</th><th>Resultado</th></tr><tr><td>write hal hobj</td><td>BAD_INSTRUCCION</td></tr><tr><td>read hal</td><td>BAD_INSTRUCCION</td></tr><tr><td>write lyle lobj 10</td><td>OK</td></tr><tr><td>read hal lobj</td><td>OK</td></tr><tr><td>write lyle hobj 20</td><td>OK</td></tr><tr><td>write hal lobj 200</td><td>BAD_INSTRUCCION</td></tr><tr><td>read hal hobj</td><td>OK</td></tr><tr><td>read lyle lobj</td><td>OK</td></tr><tr><td>read lyle hobj</td><td>PROBLEMA PERMISOS</td></tr><tr><td>foo lyle lobj</td><td>BAD_INSTRUCCION</td></tr><tr><td>Hi lyle, This is hal</td><td>BAD_INSTRUCCION</td></tr><tr><td>The missile launch code</td><td>BAD_INSTRUCCION</td></tr></table>	Comandos	Resultado	write hal hobj	BAD_INSTRUCCION	read hal	BAD_INSTRUCCION	write lyle lobj 10	OK	read hal lobj	OK	write lyle hobj 20	OK	write hal lobj 200	BAD_INSTRUCCION	read hal hobj	OK	read lyle lobj	OK	read lyle hobj	PROBLEMA PERMISOS	foo lyle lobj	BAD_INSTRUCCION	Hi lyle, This is hal	BAD_INSTRUCCION	The missile launch code	BAD_INSTRUCCION
Comandos	Resultado																										
write hal hobj	BAD_INSTRUCCION																										
read hal	BAD_INSTRUCCION																										
write lyle lobj 10	OK																										
read hal lobj	OK																										
write lyle hobj 20	OK																										
write hal lobj 200	BAD_INSTRUCCION																										
read hal hobj	OK																										
read lyle lobj	OK																										
read lyle hobj	PROBLEMA PERMISOS																										
foo lyle lobj	BAD_INSTRUCCION																										
Hi lyle, This is hal	BAD_INSTRUCCION																										
The missile launch code	BAD_INSTRUCCION																										

Resultado esperado	Resultado tras corrida
	##### Estado Final LISTA DE SUJETOS Name :LYLE Nivel:LOW Temp:0 ##### Name :HAL Nivel:HIGH Temp:20 ##### LISTA DE OBJETOS Name :LOBJ Nivel:LOW Valor:10 ##### Name :HOBJ Nivel:HIGH Valor:20 #####

Prueba 2 – Verificación según Instruction list de Aulas

Archivo	Resultado de cada línea esperada																																								
<div> <div>instructionList.txt: ...</div> <div> <div>Archivo Edición Formato Ver Ayuda</div> <div> <pre> write lyle lobj 10 write hal hobj 90 como venimos por ahora? read lyle lobj read hal lobj write lyle lobj 20 write hal lobj 200 read hal hobj write lyle hobj 150 read hal hobj read lyle lobj read lyle hobj write lyle hobj 50 leer lyle lobj write lyle lucas 70 read hal hobj write hal lobj write hal hobj 200 Final de las instrucciones </pre> </div> </div> </div>	<table> <tr> <th>Comandos</th><th>Resultado</th></tr> <tr><td>write lyle lobj 10</td><td>OK</td></tr> <tr><td>write hal hobj 90</td><td>OK</td></tr> <tr><td>como venimos por ahora?</td><td>BAD_INSTRUCTION</td></tr> <tr><td>read lyle lobj</td><td>OK</td></tr> <tr><td>read hal lobj</td><td>OK</td></tr> <tr><td>write lyle lobj 20</td><td>OK</td></tr> <tr><td>write hal lobj 200</td><td>PROBLEMA DE PERMISOS</td></tr> <tr><td>read hal hobj</td><td>OK</td></tr> <tr><td>write lyle hobj 150</td><td>OK</td></tr> <tr><td>read hal hobj</td><td>OK</td></tr> <tr><td>read lyle lobj</td><td>OK</td></tr> <tr><td>read lyle hobj</td><td>PROBLEMA DE PERMISOS</td></tr> <tr><td>write lyle hobj 50</td><td>OK</td></tr> <tr><td>leer lyle lobj</td><td>BAD_INSTRUCTION</td></tr> <tr><td>write lyle lucas 70</td><td>NO EXISTE LUCAS</td></tr> <tr><td>read hal hobj</td><td>OK</td></tr> <tr><td>write hal lobj</td><td>PROBLEMA DE PERMISOS</td></tr> <tr><td>write hal hobj 200</td><td>OK</td></tr> <tr><td>Final de las instrucciones</td><td>BAD_INSTRUCTION</td></tr> </table>	Comandos	Resultado	write lyle lobj 10	OK	write hal hobj 90	OK	como venimos por ahora?	BAD_INSTRUCTION	read lyle lobj	OK	read hal lobj	OK	write lyle lobj 20	OK	write hal lobj 200	PROBLEMA DE PERMISOS	read hal hobj	OK	write lyle hobj 150	OK	read hal hobj	OK	read lyle lobj	OK	read lyle hobj	PROBLEMA DE PERMISOS	write lyle hobj 50	OK	leer lyle lobj	BAD_INSTRUCTION	write lyle lucas 70	NO EXISTE LUCAS	read hal hobj	OK	write hal lobj	PROBLEMA DE PERMISOS	write hal hobj 200	OK	Final de las instrucciones	BAD_INSTRUCTION
Comandos	Resultado																																								
write lyle lobj 10	OK																																								
write hal hobj 90	OK																																								
como venimos por ahora?	BAD_INSTRUCTION																																								
read lyle lobj	OK																																								
read hal lobj	OK																																								
write lyle lobj 20	OK																																								
write hal lobj 200	PROBLEMA DE PERMISOS																																								
read hal hobj	OK																																								
write lyle hobj 150	OK																																								
read hal hobj	OK																																								
read lyle lobj	OK																																								
read lyle hobj	PROBLEMA DE PERMISOS																																								
write lyle hobj 50	OK																																								
leer lyle lobj	BAD_INSTRUCTION																																								
write lyle lucas 70	NO EXISTE LUCAS																																								
read hal hobj	OK																																								
write hal lobj	PROBLEMA DE PERMISOS																																								
write hal hobj 200	OK																																								
Final de las instrucciones	BAD_INSTRUCTION																																								

Resultado esperado	Resultado tras corrida																												
<table><tr><th colspan="4">Evolución de Estados</th></tr><tr><th colspan="2">Sujetos</th><th colspan="2">Objetos</th></tr><tr><th>lyle</th><th>hal</th><th>lobj</th><th>hobj</th></tr><tr><td>10</td><td></td><td>10</td><td>90</td></tr><tr><td>0</td><td></td><td>90</td><td>20</td></tr><tr><td></td><td></td><td>150</td><td>50</td></tr><tr><td></td><td>50</td><td></td><td>200</td></tr></table>	Evolución de Estados				Sujetos		Objetos		lyle	hal	lobj	hobj	10		10	90	0		90	20			150	50		50		200	<pre>#### Estado Final LISTA DE SUJETOS Name :LYLE Nivel:LOW Temp:0 ***** Name :HAL Nivel:HIGH Temp:50 ***** LISTA DE OBJETOS Name :LOBJ Nivel:LOW Valor:20 ***** Name :HOBJ Nivel:HIGH Valor:200 *****</pre>
Evolución de Estados																													
Sujetos		Objetos																											
lyle	hal	lobj	hobj																										
10		10	90																										
0		90	20																										
		150	50																										
	50		200																										

1.4 Conclusión

La propuesta de la implementación de un sistema seguro con el concepto básico planteado por Bell y Lapadula fomenta la comprensión del método y las decisiones que se deben tomar para que la información vaya en un sentido seguro.

Si bien en ésta ocasión se trabajaba con números enteros, es posible implementar con documentos completos e incluso encriptados.

La implementación del modelo propuesto originalmente por sus creadores nos ofrecieron una mayor comprensión sobre la potencia que le ofrece éste modelo al concepto de confidencialidad.

Parte 2 – Canal Encubierto

Introducción

Un canal encubierto (del inglés covert channel), es un canal que puede ser usado para transferir información desde un usuario de un sistema a otro, usando medios no destinados para este propósito por los desarrolladores del sistema.

En ésta segunda parte del obligatorio se solicita el desarrollo de un canal encubierto para intentar transferir información entre 2 sujetos de diferentes niveles sorteando las reglas del modelo de Bell Lapadula.

2.1 Desarrollo

El mensaje recibido originalmente mediante el archivo (mensaje.txt), se debe descomponer en caracteres (bytes) y luego en bits (1 y 0) ya que la transferencia se hace en “goteos de bits” para rearmar le mensaje en el destinatario, conociendo la secuencia de ejecución (o participación) de un usuario y el otro mediante un archivo (secuencia.txt)

Investigación:

Como pasar un array de bytes en un array de bits:

```
public class Main {
    public static String toBitString(final byte[] b) {
        final char[] bits = new char[8 * b.length];
        for(int i = 0; i < b.length; i++) {
            final byte byteval = b[i];
            int bytei = i << 3;
            int mask = 0x1;
            for(int j = 7; j >= 0; j--) {
                final int bitval = byteval & mask;
                if(bitval == 0) {
                    bits[bytei + j] = '0';
                } else {
                    bits[bytei + j] = '1';
                }
                mask <<= 1;
            }
        }
        return String.valueOf(bits);
    }

    public static void main(String[] argv){
        System.out.println(toBitString(new byte[]{1,2,3}));
    }
}
```

Se confeccionó una función para tales efectos, logrando una cadena de bits desde el texto original.

Fuente:

http://www.java2s.com/Tutorials/Java/Data_Type/Array_Convert/Convert_byte_array_to_bit_string_in_Java.htm

Vaciar contenido de archivo:

Cuando vamos a escribir en un archivo, primero procedemos a eliminar el contenido “viejo” del mismo para luego escribir los resultados de la nueva ejecución.

En el caso de no existir el archivo no hace nada, de lo contrario elimina su contenido dejándolo en blanco.



```
File inputFile = new File("myFile.txt");
File tempFile = new File("myTempFile.txt");

BufferedReader reader = new BufferedReader(new FileReader(inputFile));
BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

String lineToRemove = "bbb";
String currentLine;

while((currentLine = reader.readLine()) != null) {
    // trim newline when comparing with lineToRemove
    String trimmedLine = currentLine.trim();
    if(trimmedLine.equals(lineToRemove)) continue;
    writer.write(currentLine + System.getProperty("line.separator"));
}
writer.close();
reader.close();
boolean successful = tempFile.renameTo(inputFile);
```

Fuente:

<https://stackoverflow.com/questions/1377279/find-a-line-in-a-file-and-remove-it>

Proceso de resolución del problema:

- 1) Se crean los sujetos involucrados
- 2) Se lee el archivo que se va a transferir
- 3) Se lee la secuencia de lectura
- 4) Se traduce el archivo a transferir a bytes y luego a bits
- 5) La lectura del archivo de secuencia indica en que orden se deben ejecutar las intervenciones de los diferentes sujetos.
- 6) Si es el turno de HAL y el objeto existe, debe eliminarlo y ejecutar el proceso de “dejar” el dato creado y con un estado legible por LYLE.
- 7) Si es el turno de LYLE, se ejecuta el proceso de “leer” el dato creado y con un estado legible
- 8) LYLE va armando el mensaje en base a los bits recibidos que va leyendo. En caso de no poder leerlo (por su nivel de seguridad) se acumula en la variable “sendedBits” un 0, en caso de poder leerlo se concatena al mismo un 1.
- 9) Generando de ese modo una acumulacion de bits que luego se traducen a un char de ASCII.

```
public void ExecuteREADAction(String objectName, String subjectName) throws IOException {
    int valueRead = 0;
    String actionLog = "";
    SecuritySubject subject = objectManager.findSubjectByName(subjectName);
    SecurityObject object = objectManager.findObjectByName(objectName);
    if (objectManager.dominates(subject.getSecurityLevel(), object.getSecurityLevel())) {
        subject.setTEMP(objectManager.read(subjectName, objectName));
        valueRead = subject.getTEMP();
        actionLog = "READ " + subjectName + " " + objectName;
        logFile.InsertLogLine(actionLog);
    } else {
        subject.setTEMP(0);
        valueRead = subject.getTEMP();
    }

    sendedBits += valueRead;
}
```

- 10) Por ultimo lyle siempre ejecuta la secuencia RUN. Una vez que el proceso de secuencia finalice, se arman paquetes de 8 bits, se transforma a bytes y se pasa a Ascii nuevamente creándose el archivo mensaje.salida.txt y escribiendo en el mismo los correspondientes caracteres.

```
public void transferDataToOutput() throws IOException {
    byte[] byteToStore = new byte[1];
    byteToStore[0] = Byte.parseByte(sendedBits, 2);

    try {
        fileRecordName = fileStreamManager.getFileRecord();
        RecordFile.getInstance("test//" + fileRecordName).record(byteToStore, fileRecordName);
    } catch (Exception e) {
        File file = new File("test//" + fileRecordName);
        file.createNewFile();
        RecordFile.getInstance("test//" + fileRecordName).record(byteToStore, fileRecordName);
    }
}
```

```

public void ExecuteRUNAction(String subjectName) throws IOException {
    String actionLog = "";
    //la instruccion RUN de lyle le permite hacer lo que tenga que hacer
    //para registrar en su estado interno, agregar al byte que se acaba de crear,
    //y sacar el byte si se ha recibido los 8 bits para ese byte.
    if (subjectName == "lyle" && sendeBits.length() == 8) {
        transferDataToOutput();
        sendeBits = "";
    }

    actionLog = "RUN " + subjectName;
    logFile.InsertLogLine(actionLog);
}

```

En el caso de completar los bits necesarios, entonces procede a grabar el carácter en el nuevo archivo output. Por último, vuelve a dejar la variable sendeBits vacía para poder hacer el siguiente procesamiento del próximo carácter que se desee transmitir.

Librerías adicionales

En esta ocasión fue necesario utilizar la librería javatuples-1.2.jar (<http://www.java2s.com/Code/Jar/j/Downloadjavatuples12jar.htm>), la cual fue utilizada para apoyar el análisis del archivo importado, tercerizando dicho control y separación. En caso de que ocurra algún inconveniente, dicha herramienta se encarga de cargar el error y ofrecer información sobre el mismo, nombre del archivo desde donde se va a leer el mensaje.

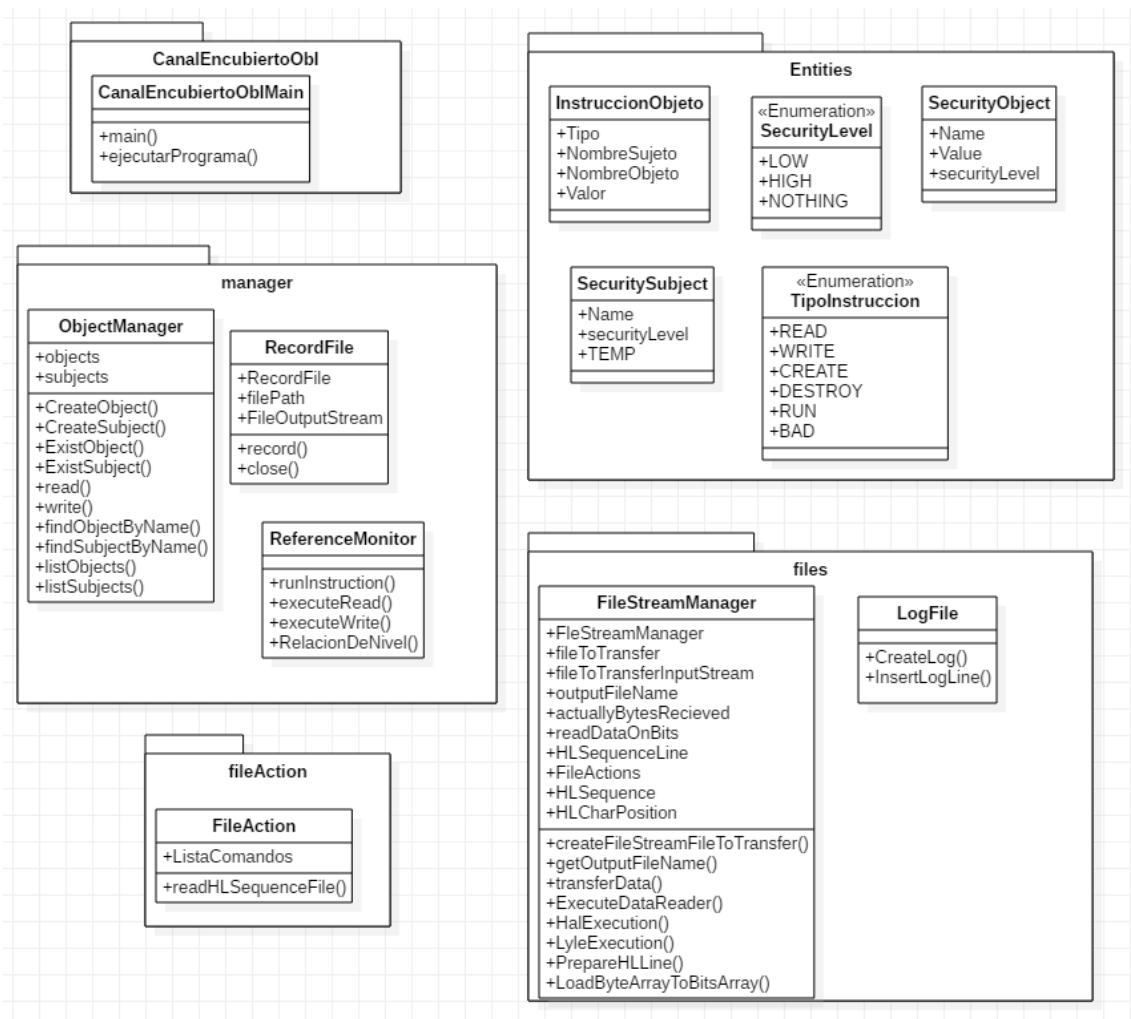
Se adjunta una imagen donde se utilizó la herramienta:

```

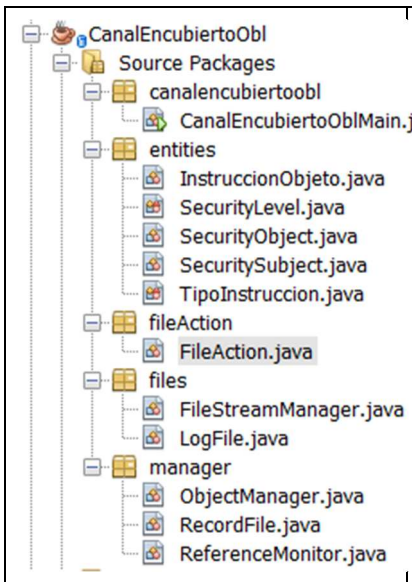
public static Triplet<String, File, InputStream> createFileStreamFileToTransfer() {
    String messageError = "";
    try {
        String message = "Ingrese nombre de archivo que contiene el mensaje junto con su extension (mensaje.txt) : ";
        System.out.println(message);
        Scanner scanner = new Scanner(System.in);
        String inputPath = scanner.nextLine();
        fileToTransfer = new File("test/" + inputPath);
        fileToTransferInputStream = new FileInputStream(fileToTransfer);
    } catch (Exception e) {
        messageError = "Error en nombre de archivo";
        return new Triplet<String, File, InputStream>(messageError, fileToTransfer, fileToTransferInputStream);
    }
    return new Triplet<String, File, InputStream>("", fileToTransfer, fileToTransferInputStream);
}

```

Vista de descomposición



Detalle de clases, agrupación y función










	canalencubiertool ob1	CanalEncubiertoObl Main	Clase Main, principal, inicial.
		Entities	InstruccionObjeto

		SecurityLevel	Enumeración de tipos de niveles de seguridad LOW, HIGH, NOTHING
		SecurityObject	Representación de un Objeto del modelo Bell Lapadula
		TipoInstruccion	Enumeración de tipos de operaciones. READ, WRITE, CREATE, DESTROY, RUN, BAD_INSTRUCTION
	fileAction	FileAction	Encargada de consultar al usuario, leer y cargar la secuencia de ejecución en un buffer.
	files	FileStreamManager	Organiza y administra la ejecución dependiendo del turno en el archivo de secuencia
		LogFile	Manager del log de la aplicación, crea y adiciona lineas.
	manager	ObjectManager	Controlador que contiene el comportamiento de una base de datos para registrar el listado de

			sujetos, objetos y sus estados
		RecordFile	Encargado de la creación del archivo de salida
		ReferenceMonitor	Control de instrucciones, control de relación de permiso y ejecución.

2.2 Instalación

Para realizar la prueba, solo hay que copiar la carpeta “test” que se encuentra dentro del proyecto a cualquier carpeta del equipo destino.

 lib  test  CanalEncubiertoObl.jar  Parte2.bat	El contenido completo de la carpeta
 javatuples-1.2.jar	Dentro de la carpeta lib debe estar la librería que es necesaria
 log.log  mensaje.salida.txt  mensaje.txt  secuencia.txt	<p>En la carpeta test deben estar los archivos que utilizará y donde quedará el archivo mensaje.salida.txt con el resultado del texto recibido.</p> <p>Mensaje.txt = mensaje a transmitir</p> <p>Secuencia.txt = archivo que indica la secuencia de ejecución (H=HAL, L=Lyle)</p> <p>Log.log = Archivo donde quedará el paso a paso de lo que sucedió</p> <p>Mensaje.salida.txt = archivo generado en base a lo transmitido + secuencia de ejecución e interpretación.</p>

Se debe usar “Parte2.bat” para probarlo, el sistema solicitará el archivo a transmitir (ingrese mensaje.txt) y el archivo de secuencia (ingrese secuencia.txt) a continuación se ejecutará el proceso de transferencia.

Al terminar, en el archivo mensaje.salida.txt de la carpeta test quedara el resultado.

2.3 Pruebas

A Continuación se muestran 3 pruebas realizadas de transferencia de datos, utilizando la metodología de canal encubierto propuesta por el obligatorio y que fue implementada.

Prueba 1 – Verificación según el mensaje y la secuencia ofrecida por el obligatorio

Mensaje enviado	Secuencia
Mensaje recibido	

Resultado: OK

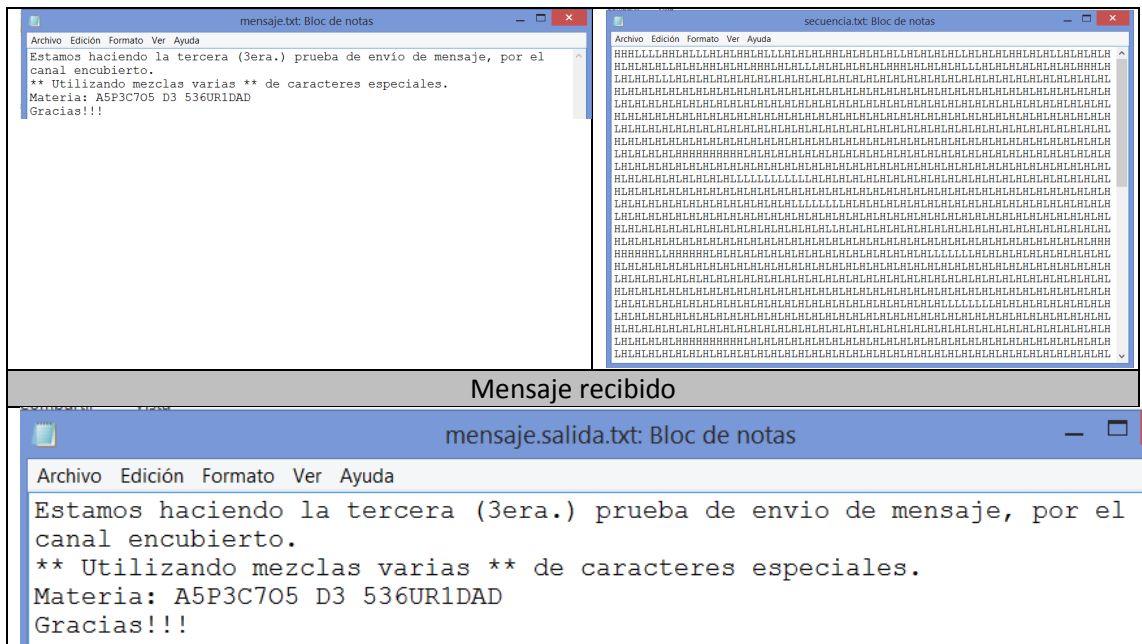
Prueba 2 – Como la prueba 1, pero cambiando el archivo de secuencia

Mensaje enviado	Secuencia
Mensaje recibido	

Resultado: OK

Prueba 3 – Como la prueba 1, pero cambiando el mensaje a transmitir

Mensaje enviado	Secuencia
-----------------	-----------



Resultado: OK

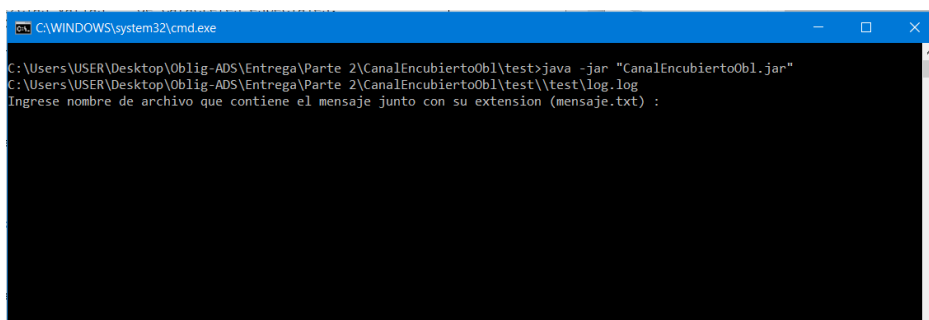
Bug encontrado:

Cuando se intenta pasar un mensaje con un carácter con tilde (á, é, í, ó, ú) , se produce un error. Finalizando con la ejecución. Una posible solución es limpiar el mensaje de caracteres con tilde antes de la transferencia, respetando el contenido del mensaje, aunque con faltas de ortografía.

Pruebas desde ejecutable:

Instalacion:

- Acceder a la carpeta Entrega/Parte 2/CanalEncubiertoObl/test/
Abrir el ejecutable "Parte2.bat", donde se abra la consola para ejecutar el proyecto.



Prueba:

Ingresar el nombre del archivo que contiene el mensaje que se desea transmitir, junto con su extension.

Luego colocar el nombre del archivo que contiene la secuencia H y L junto con su extension:

```
C:\WINDOWS\system32\cmd.exe
C:\Users\USER\Desktop\Oblig-ADS\Entrega\Parte 2\CanalEncubiertoObl\test>java -jar "CanalEncubiertoObl.jar"
C:\Users\USER\Desktop\Oblig-ADS\Entrega\Parte 2\CanalEncubiertoObl\test>\test\log.log
Ingrese nombre de archivo que contiene el mensaje junto con su extension (mensaje.txt) :
mensaje.txt
Archivo de salida ya esta limpio





ESPERE...

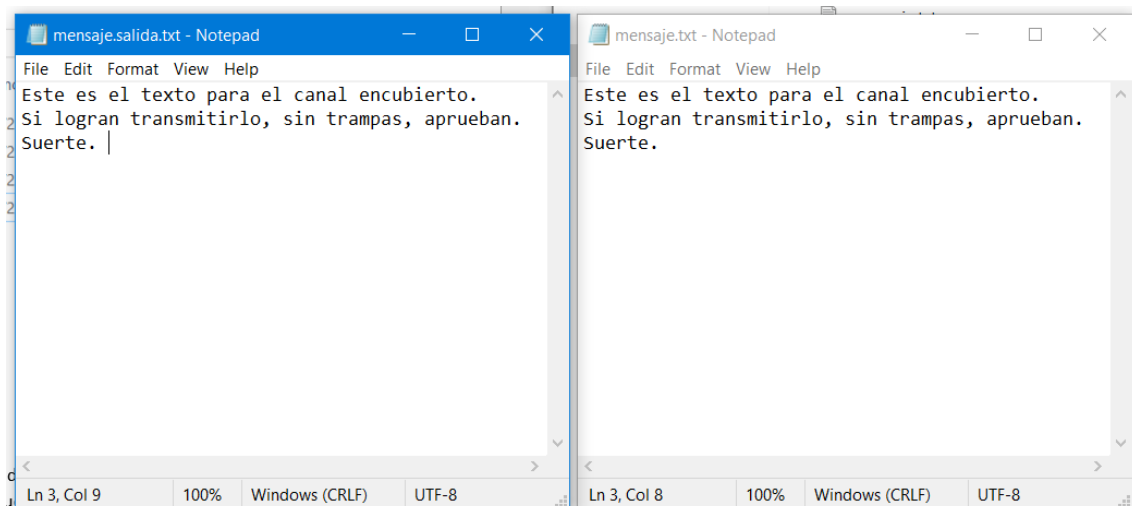
Ingrese nombre de archivo que contiene la secuencia H y L junto con su extension (secuencia.txt):
secuencia.txt
```

Una vez ejecutado estos pasos, se comienza a correr el programa, imprimiendo un log por consola de las instrucciones que se van realizando. En paralelo las mismas se van registrando en el log, en el archivo “log.log” ubicado en la subcarpeta \test.

Por ultimo, se puede ver como se crea un archivo de salida con el mismo nombre del archivo que contiene el mensaje + “.salida”.

Se realizó de esta manera porque dio problemas para crearlo en una nueva carpeta aparte.

Entrega > Parte 2 > CanalEncubiertoObl > test > test				Search test
<input type="checkbox"/> Name	Date modified	Type	Size	
 log.log	12/10/2019 2:36 PM	Text Document		
 mensaje.salida.txt	12/10/2019 2:36 PM	Text Document		
 mensaje.txt	12/10/2019 2:11 PM	Text Document		
 secuencia.txt	12/10/2019 2:11 PM	Text Document		



Evidencia del mensaje de salida junto con el archivo que contiene el mensaje principal a transmitir.

Nota: Para crear pruebas adicionales, se crean archivos .txt en la carpeta “\Entrega\Parte 2\CanalEncubiertoObl\test\test”.

Luego, se ejecuta nuevamente la consola y cuando pide el nombre del archivo que contiene el mensaje, se coloca allí el mismo.

Una vez ejecutado, debería aparecer en la misma carpeta, un nuevo archivo con el nombre “[pruebaAdicional].salida.txt”.

2.4 Conclusión

El canal encubierto se basa en la transmisión de mensajes entre 2 sujetos que pertenecen a niveles de acceso diferentes, es decir que un sujeto con un nivel de acceso superior pueda transmitir un mensaje a un sujeto de nivel de acceso inferior (contrario a la regla estrella de Bell Lapadulla).

Mediante la utilización de la máxima granularidad posible de información (bits) es posible desarmar un mensaje en letras, éstas en bytes y luego en bits, transmitirlo mediante cambios de estados de objetos (respetando las reglas de bell-lapadulla) y volver a armarlo tomando nota de los cambios de estado por parte del receptor.

El mensaje al ser granulado en bits de 0 y 1. Se logra desarmar cada carácter en bits, entonces dependiendo si el sujeto lyle puede leer o no el archivo, se va produciendo en un string “sendedBits” un conjunto de 0s y 1s que al alcanzar 8 bits (si lyle puede leer el archivo entonces se concatena un 1 en “sendedBits”, de lo contrario se concatena un 0) se los convierte a un correspondiente numero, ejemplo: si al completar los 8 bits tengo un “01000101” se convierte en un char “69” que corresponde a la letra E en la tabla ASCII y de esta manera (ya convertido en caracter) se escribe en un archivo de salida el mensaje.

Conclusión Final

En éste trabajo obligatorio, pudimos desarrollar y probar los conceptos del modelo de Bell y Lapadulla así también como el camino para poder transmitir mensajes rompiendo las reglas básicas del mismo.

No hay dudas que es un modelo sólido de seguridad, si es que se tienen precauciones a nivel logístico para evitar la transferencia de información entre sujetos de diferentes niveles.

Mediante el análisis de la solución, la discusión entre los miembros del equipo y la implementación y testing cruzado, pudimos entender el funcionamiento del mismo y su aplicación.