Reporte análisis estático de código

**Informe detallado de resultados: WebGoat**

Control de cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Fecha** | **Apartado** | **Resumen de cambios** |
| 1.0 | 02/06/2021 | N/A | Versión inicial |
|  |  |  |  |
|  |  |  |  |

Índice

[1. Introducción 4](#_Toc77068460)

[2. Información de la aplicación 5](#_Toc77068461)

[2.1. Datos generales de la aplicación 5](#_Toc77068462)

[2.2. Datos de calidad 5](#_Toc77068463)

[3. Nivel y modelo de calidad 6](#_Toc77068464)

[3.1. Listado de reglas incumplidas 7](#_Toc77068465)

[3.1.1. Security Hotspots 7](#_Toc77068466)

[3.1.2. Vulnerability 7](#_Toc77068467)

[4. Tendencias e histórico 15](#_Toc77068468)

[5. Análisis de la aplicación 16](#_Toc77068469)

[5.1. Hot Spots 16](#_Toc77068470)

[5.1.1. Credentials should not be hard-coded 16](#_Toc77068471)

[5.1.2. Command-injection 17](#_Toc77068472)

[5.1.3. SQL Injection 18](#_Toc77068473)

[5.1.4. Denial of Service (DOS) 19](#_Toc77068474)

[5.1.5. Weak-cryptography 20](#_Toc77068475)

[5.1.6. Object-injection 22](#_Toc77068476)

[5.1.7. Make sure that environment variables are used safely 23](#_Toc77068477)

[5.2. Vulnerabilidades 24](#_Toc77068478)

[5.2.1. Throwable.printStackTrace(...) should not be called 24](#_Toc77068479)

[5.2.2. Mutable fields should not be "public static" 24](#_Toc77068480)

[5.2.3. Return values should not be ignored when they contain the operation status code 25](#_Toc77068481)

[5.2.4. XML parsers should not be vulnerable to XXE attacks 26](#_Toc77068482)

[5.2.5. Links with "target=\_blank" should prevent phishing attacks 27](#_Toc77068483)

[5.3. Duplicidad 28](#_Toc77068484)

[6. Siguientes pasos 29](#_Toc77068485)

1. Introducción

Los resultados expuestos en este documento tratan de reflejar el estado de la aplicación WebGoat analizada desde el punto de vista de la calidad de su código fuente.

El modelo de calidad aplicado responde a las necesidades planteadas y trata de minimizar el impacto que supone la implantación de un servicio de análisis estático de código en un equipo de desarrollo. La herramienta en la que se apoya el servicio, SonarQube, establece una clasificación en el tipo de problemas detectados atendiendo a la característica de calidad con la que están relacionados y este informe se referirá a ellos en sus mismos términos. De esta forma, surgen los siguientes tipos de problemas:

* ***Vulnerability***. Problemas relacionados con la seguridad de la aplicación y que pueden suponer un riesgo para la integridad de la misma.
* ***Bug***. Problemas relacionados con la fiabilidad de la aplicación, su capacidad para recuperarse de errores inesperados y mantener el servicio.
* ***Code smell.*** Este tipo de problema se relaciona con la mantenibilidad del código fuente, su capacidad para cambiar de una forma eficiente y efectiva.

Así mismo, SonarQube emplea una terminología propia para ciertos aspectos internos que se explica a continuación:

* ***Quality profile.*** Conjunto de reglas de calidad de código que se aplican a un lenguaje en concreto.
* ***Quality gate.*** Métrica o conjunto de métricas asociadas a ciertos valores límite en los cuales se basa SonarQube para valorar el resultado de un escaneo.

De forma adicional a estos conceptos, durante el informe se hará mención a otros aspectos técnicos de la calidad de código que se explican a continuación:

* ***Problema***. Un problema hace referencia a un componente (proyecto, modulo, archivo, bloque de código, etc.) que no cumple con la especificación de una regla de calidad.
* ***Regla cumplida.*** Hace referencia a reglas de calidad de código que pertenecen al modelo de calidad y que no generan ningún problema en el análisis actual.
* ***Regla incumplida.*** Son reglas de calidad de código pertenecientes al modelo de calidad de código aplicado y que generan al menos un problema.
* ***Duplicidad.*** Métrica de calidad de código que recoge el porcentaje de código duplicado que se ha encontrado al analizar la aplicación.
* ***Bloques duplicados.*** Un bloque duplicado es un conjunto de líneas de código que se encuentran al menos dos veces escritos en la aplicación.

1. Información de la aplicación

Esta sección muestra la información de partida para conocer la aplicación que se está valorando, su tamaño y los lenguajes implicados en la misma.

* 1. Datos generales de la aplicación

|  |  |  |
| --- | --- | --- |
| Nombre del proyecto | [WebGoat](http://localhost:9000/dashboard?id=webgoat) | |
| Análisis base | 03-06-2021 07:52 | |
| Líneas de código | 76495 LOC | |
| Lenguajes | CSS | 6.644 LOC (8,69 %) |
| Java | 14.897 LOC (19,47 %) |
| JavaScript | 43.812 LOC (57,27 %) |
| HTML | 5.467 LOC (7,15 %) |
| XML | 5.675 LOC (7,42 %) |
| Nivel reglas SonarQube | Nivel 1 | |
| Quality gates | No se han establecido | |

* 1. Datos de calidad

A continuación, se muestran las métricas de calidad más relevantes.

|  |  |
| --- | --- |
| Vulnerabilities | 187 |
| Hotspots | 149 |
| Bugs | 477 |
| Code smells | 1.605 |
| Duplicidad | 12,8 % |
| Bloques duplicados | 253 |

1. Nivel y modelo de calidad

WebGoat se encuentra actualmente en el **Nivel 1 del modelo de calidad** acordado entre los equipos de desarrollo y calidad. Dentro de este nivel, se muestran a continuación la relación de reglas cumplidas e incumplidas (reglas que generan al menos un problema):

Además, se muestra a continuación los problemas encontrados categorizados por tipo y lenguaje.

* 1. Listado de reglas incumplidas

Las reglas incumplidas y el número de problemas que ocasiona cada una, separadas por tipo, son las siguientes:

* + 1. Security Hotspots

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| auth | 'PASSWORD' detected in this expression, review this potentially hard-coded credential. | 23 |
| command-injection | Make sure that command line arguments are used safely here. | 8 |
| sql-injection | Ensure that string concatenation is required and safe for this SQL query. | 15 |
| dos | Make sure that sockets are used safely here. | 57 |
| weak-cryptography | Make sure that hashing data is safe here. | 22 |
| object-injection | Make sure deserializing objects is safe here. | 4 |
| others | Make sure that environment variables are used safely here. | 20 |
| Total |  | 149 |

* + 1. Vulnerability

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| Java | Throwable.printStackTrace(...) should not be called | 25 |
| Java | Mutable fields should not be "public static" | 5 |
| Java | Return values should not be ignored when they contain the operation status code | 2 |
| Java | XML parsers should not be vulnerable to XXE attacks | 2 |
| HTML | Links with "target=\_blank" should prevent phishing attacks | 4 |
| Total |  | 38 |

**Bug**

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| CSS | Properties should not be duplicated | 5 |
| CSS | Font declarations should contain at least one generic font family | 3 |
| CSS | Selectors should be known | 1 |
| Java | Resources should be closed | 66 |
| Java | Double Brace Initialization should not be used | 12 |
| Java | Method parameters, caught exceptions and foreach variables' initial values should not be ignored | 8 |
| Java | "Random" objects should be reused | 8 |
| Java | Strings and Boxed types should be compared using "equals()" | 2 |
| Java | "PreparedStatement" and "ResultSet" methods should be called with valid indices | 2 |
| Java | Optional value should only be accessed after calling isPresent() | 2 |
| Java | Conditionally executed code should be reachable | 2 |
| Java | Null pointers should not be dereferenced | 1 |
| JavaScript | Special identifiers should not be bound or assigned | 33 |
| JavaScript | Non-empty statements should change control flow or have at least one side-effect | 11 |
| JavaScript | Properties of variables with "null" or "undefined" values should not be accessed | 3 |
| JavaScript | All branches in a conditional structure should not have exactly the same implementation | 1 |
| HTML | "html" element should have a language attribute | 63 |
| HTML | "strong" and "em" tags should be used | 59 |
| HTML | "title" should be present in all pages | 53 |
| HTML | "table" tags should have a description | 42 |
| HTML | Tables should have headers | 35 |
| HTML | "th" tags should have "id" or "scope" attributes | 28 |
| HTML | Image, area and button with image tags should have an "alt" attribute | 21 |
| HTML | "!DOCTYPE" declarations should appear before "html" tags | 8 |
| HTML | "fieldset" tags should contain a "legend" | 5 |
| HTML | Elements deprecated in HTML5 should not be used | 2 |
| HTML | "li" and "dt" item tags should be in "ul", "ol" or "dl" container tags | 1 |
| Total |  | 477 |

**Code Smell**

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| CSS | Selectors should not be duplicated | 14 |
| CSS | Empty blocks should be removed | 11 |
| CSS | CSS files should not be empty | 1 |
| Java | Source files should not have any duplicated blocks | 204 |
| Java | Parameters should be passed in the correct order | 96 |
| Java | Local variable and method parameter names should comply with a naming convention | 94 |
| Java | Unused "private" fields should be removed | 80 |
| Java | Unnecessary imports should be removed | 65 |
| Java | String literals should not be duplicated | 47 |
| Java | Standard outputs should not be used directly to log anything | 40 |
| Java | Raw types should not be used | 38 |
| Java | Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used | 29 |
| Java | Try-catch blocks should not be nested | 24 |
| Java | Array designators "[]" should be on the type, not the variable | 23 |
| Java | Sections of code should not be commented out | 22 |
| Java | Generic exceptions should never be thrown | 19 |
| Java | Track uses of "TODO" tags | 19 |
| Java | Generic wildcard types should not be used in return parameters | 17 |
| Java | Boolean expressions should not be gratuitous | 16 |
| Java | Only static class initializers should be used | 12 |
| Java | Unused local variables should be removed | 11 |
| Java | Method names should comply with a naming convention | 10 |
| Java | String function use should be optimized for single characters | 10 |
| Java | Unused assignments should be removed | 10 |
| Java | Case insensitive string comparisons should be made without intermediate upper or lower casing | 10 |
| Java | Redundant pairs of parentheses should be removed | 10 |
| Java | Constant names should comply with a naming convention | 6 |
| Java | "toString()" should never be called on a String object | 6 |
| Java | "StandardCharsets" constants should be preferred | 6 |
| Java | Boolean literals should not be redundant | 6 |
| Java | Local variables should not be declared and then immediately returned or thrown | 6 |
| Java | The diamond operator ("") should be used | 5 |
| Java | A field should not duplicate the name of its containing class | 5 |
| Java | Lambdas should be replaced with method references | 4 |
| Java | Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList" | 4 |
| Java | "@Deprecated" code should not be used | 4 |
| Java | Multiple variables should not be declared on the same line | 4 |
| Java | Collection.isEmpty() should be used to test for emptiness | 4 |
| Java | Boxed "Boolean" should be avoided in boolean expressions | 4 |
| Java | Return of boolean expressions should not be wrapped into an "if-then-else" statement | 4 |
| Java | Local variables should not shadow class fields | 4 |
| Java | Parsing should be used to convert "Strings" to primitives | 3 |
| Java | Deprecated elements should have both the annotation and the Javadoc tag | 3 |
| Java | Utility classes should not have public constructors | 3 |
| Java | Unused "private" methods should be removed | 3 |
| Java | Deprecated code should be removed | 3 |
| Java | "Stream" call chains should be simplified when possible | 2 |
| Java | URIs should not be hardcoded | 2 |
| Java | Methods should not have too many parameters | 2 |
| Java | "close()" calls should not be redundant | 2 |
| Java | Constants should not be defined in interfaces | 2 |
| Java | Strings should not be concatenated using '+' in a loop | 2 |
| Java | "@Override" should be used on overriding and implementing methods | 2 |
| Java | Empty arrays and collections should be returned instead of null | 2 |
| Java | Unused method parameters should be removed | 2 |
| Java | Null checks should not be used with "instanceof" | 2 |
| Java | Catches should be combined | 2 |
| Java | Modifiers should be declared in the correct order | 2 |
| Java | Instance methods should not write to "static" fields | 2 |
| Java | Empty statements should be removed | 2 |
| Java | Cognitive Complexity of methods should not be too high | 2 |
| Java | "throws" declarations should not be superfluous | 2 |
| Java | Try-with-resources should be used | 1 |
| Java | "java.nio.Files#delete" should be preferred | 1 |
| Java | Loops should not contain more than a single "break" or "continue" statement | 1 |
| Java | Anonymous inner classes containing only one method should become lambdas | 1 |
| Java | Unused "private" classes should be removed | 1 |
| Java | The default unnamed package should not be used | 1 |
| Java | Methods should not return constants | 1 |
| Java | Two branches in a conditional structure should not have exactly the same implementation | 1 |
| Java | Throwable and Error should not be caught | 1 |
| Java | Collapsible "if" statements should be merged | 1 |
| JavaScript | Variables should not be shadowed | 172 |
| JavaScript | Variables and functions should not be redeclared | 134 |
| JavaScript | Unused assignments should be removed | 57 |
| JavaScript | Unused local variables and functions should be removed | 43 |
| JavaScript | Nested blocks of code should not be left empty | 8 |
| JavaScript | Variables should be declared explicitly | 7 |
| JavaScript | Multiline string literals should not be used | 4 |
| JavaScript | Switch cases should end with an unconditional "break" statement | 4 |
| JavaScript | Assignments should not be redundant | 3 |
| JavaScript | Function returns should not be invariant | 3 |
| JavaScript | Return of boolean expressions should not be wrapped into an "if-then-else" statement | 3 |
| JavaScript | Redundant pairs of parentheses should be removed | 3 |
| JavaScript | Comma operator should not be used | 3 |
| JavaScript | Boolean checks should not be inverted | 3 |
| JavaScript | Equality operators should not be used in "for" loop termination conditions | 2 |
| JavaScript | Sparse arrays should not be declared | 2 |
| JavaScript | Two branches in a conditional structure should not have exactly the same implementation | 2 |
| JavaScript | "switch" statements should have at least 3 "case" clauses | 2 |
| JavaScript | Extra semicolons should be removed | 2 |
| JavaScript | Function call arguments should not start on new lines | 2 |
| JavaScript | A conditionally executed single line should be denoted by indentation | 1 |
| JavaScript | Arguments to built-in functions should match documented types | 1 |
| JavaScript | "indexOf" checks should not be for positive numbers | 1 |
| JavaScript | Multiline blocks should be enclosed in curly braces | 1 |
| JavaScript | Boolean expressions should not be gratuitous | 1 |
| HTML | Attributes deprecated in HTML5 should not be used | 52 |
| HTML | Sections of code should not be commented out | 18 |
| HTML | "aria-label" or "aria-labelledby" attributes should be used to differentiate similar elements | 7 |
| HTML | Meta tags should not be used to refresh or redirect | 1 |
| XML | Sections of code should not be commented out | 2 |
| Total |  | 1.605 |

1. Tendencias e histórico

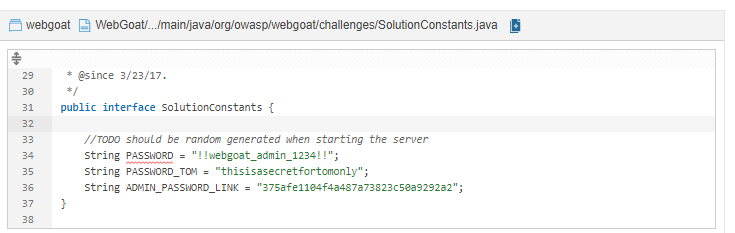
No aplica debido a que es el primer análisis realizado de WebGoat y no existen datos suficientes para realizar la comparativa. En esta sección se realizará un análisis de la evolución de las métricas más destacadas y una valoración de la calidad a nivel histórico, hitos planteados e hitos conseguidos.

1. Análisis de la aplicación

Después de someter la aplicación WebGoat a un análisis más detallado basado en los resultados obtenidos por SonarQube y teniendo en cuenta que este informe, por ser el primero, tiene como objetivo presentar una primera radiografía de la aplicación y establecer un punto de partida sobre el que comenzar a trabajar en la mejora de la calidad del código fuente, se exponen las siguientes conclusiones.

* 1. Hot Spots
     1. Credentials should not be hard-coded

Debido a que es fácil extraer cadenas del código fuente de una aplicación o binario, las credenciales no deben codificarse en código (hard-coded). Esto es particularmente grave en aplicaciones distribuidas o de código abierto.



En el caso de utilizar usuarios y password de acceso a servicios, dentro del código, estas deben ir cifradas como ***“Secure strings”***, para ello se pueden seguir las siguientes recomendaciones:

* Aplicar Recomendaciones [OWASP ESAPI](https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API#tab=Home)

Para todas las versiones de Java. Esta es la recomendación que siguen actualmente empresas como Renfe, BBVA, Visa, Mastercard, Etc.

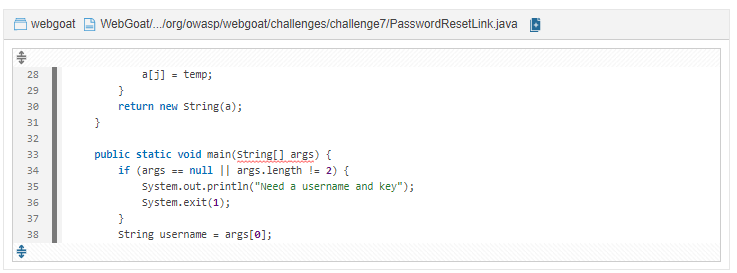
* [Reglas encriptación Maven](https://maven.apache.org/guides/mini/guide-encryption.html)

Estas reglas se pueden utilizar para las versiones de java >1.8

* + 1. Command-injection

La falta de verificación de los parámetros de entrada puede provocar la ejecución de comandos de sistema no deseadas.

Sentencias de este estilo deben incluir la verificación de todos los parámetros ante entradas no deseadas:

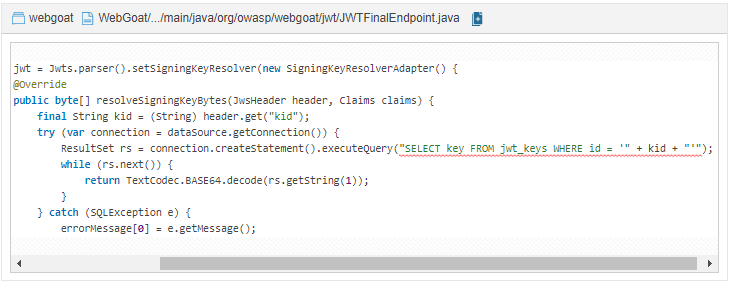


Para más información:

* [OWASP Top 10 2017 Category A1](https://www.owasp.org/index.php/Top_10-2017_A1-Injection) - Injection
* [MITRE, CWE-88](https://cwe.mitre.org/data/definitions/88.html) - Argument Injection or Modification
* [MITRE, CWE-214](https://cwe.mitre.org/data/definitions/214.html) - Information Exposure Through Process Environment
* [SANS Top 25](https://www.sans.org/top25-software-errors/#cat1) - Insecure Interaction Between Components
  + 1. SQL Injection

La concatenación de cadenas en sentencias de SQL debe de ser erradicada ya que puede provocar fácilmente la inclusión de sentencias SQL no deseadas.

La siguiente secuencia de código incluye una consulta SQL donde en la concatenación del parámetro puede incluirse una sentencia SQL distinta que al ejecutarse puede ejecutar una consulta SQL distinta a la esperada.



Este tipo de consultas concatenadas debe ser modificadas para la inclusión de parámetros, de esta forma la sentencia:

query = “SELECT key FROM jwt\_keys WHERE id =?”;

pstmt = con.prepareStatement(query);

pstmt.setString(1, kid);

ResultSet rs = pstmt.executeQuery();

* + 1. Denial of Service (DOS)

Se han detectados dos posibles fuentes que pueden provocar que la aplicación no responda correctamente (DoS):

* El uso incorrecto de sockets
* Inclusión de expresiones regulares genéricas que hacen uso intensivo de la CPU

El uso de sockets puede ser vulnerable de múltiples formas:

* Permiten que un software interactúe con el mundo exterior. Como este mundo está lleno de atacantes, es necesario verificar que no puedan recibir información sensible o inyectar información peligrosa.
* El número de sockets es limitado y se puede agotar. Lo que hace que la aplicación no responda a los usuarios que necesitan sockets adicionales.

Para una correcta utilización de los sockets hay que tener en consideración los siguientes puntos:

* En muchos casos no es necesario el uso de sockets, pueden ser sustituidos por librerías o protocolos existentes que realicen la misma función.
* Cifre todos los datos enviados si son confidenciales. Por lo general, es mejor cifrarlo incluso si los datos no son confidenciales, ya que podrían cambiar más adelante.
* Verificarcualquier entrada que se lea del socket.
* Limitar el número de sockets que puede crear un usuario determinado. Cierre los enchufes lo antes posible.

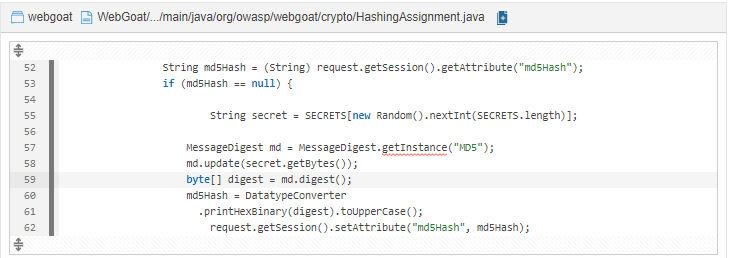
Respecto a las expresiones regulares, la evaluación de expresiones regulares es una tarea que consume mucho CPU. Las expresiones regulares especialmente diseñadas como (a+)+s tardarán varios segundos en evaluar la cadena de entrada ‘aaaaaaaaaaaaaaaaaaaaaaaaaaabs’.

El problema es que con cada carácter adicional que se agrega a la entrada, el tiempo requerido para evaluar la expresión regular se duplica. Sin embargo, la expresión regular equivalente, a + s (sin agrupamiento) se evalúa de manera eficiente en milisegundos y se escala linealmente con el tamaño de entrada.

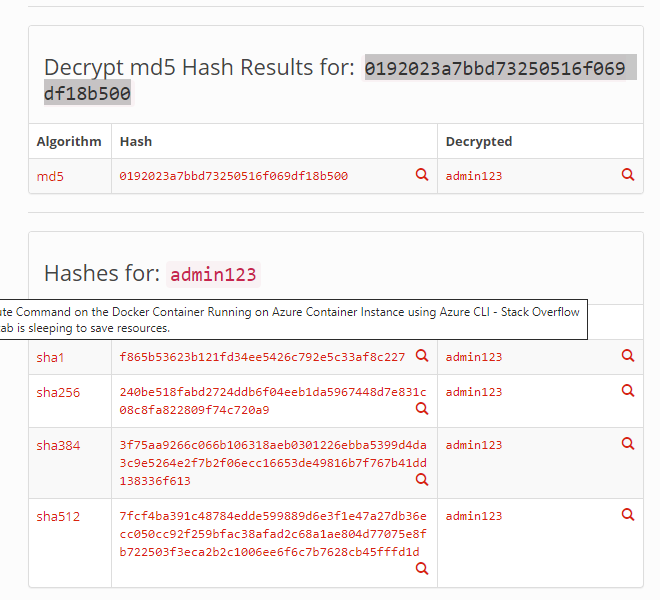
* + 1. Weak-cryptography

Las funciones de hash criptográficas se utilizan para identificar información de forma única sin almacenar su forma original. Cuando no se hace correctamente, un atacante puede robar la información original adivinándola.

Particularmente la función de hash MD5 lleva ya numerosos años siendo insegura



Un hash generado por MD5 pude ser fácilmente revertido. Por ejemplo, si consiguiéramos el hash del password del administrador de un sistema como el siguiente hash ‘0192023a7bbd73250516f069df18b500’ puede ser desencriptado fácilmente como muestra la siguiente imagen:



Por tanto, se debe eliminar el uso de librerías de hash inseguras, como es el caso de MD5, en favor de librerías de hash seguras como puede ser el caso de la librería bcrypt

public class BcryptHashingExample

{

    public static void main(String[] args) throws NoSuchAlgorithmException

    {

        String  originalPassword = "password";

        String generatedSecuredPasswordHash = BCrypt.hashpw(originalPassword, BCrypt.gensalt(12));

        System.out.println(generatedSecuredPasswordHash);

        boolean matched = BCrypt.checkpw(originalPassword, generatedSecuredPasswordHash);

        System.out.println(matched);

    }

}

Output:

$2a$12$WXItscQ/FDbLKU4mO58jxu3Tx/mueaS8En3M6QOVZIZLaGdWrS.pK

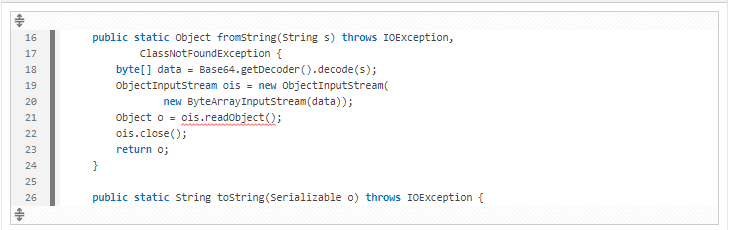
true

* + 1. Object-injection

La deserialización de objetos depende de la seguridad. Por ejemplo, ha dado lugar en el pasado a las siguientes vulnerabilidades:

* [CVE-2018-10654](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-10654): Hazelcast Library: Java deserialization vulnerability
* [CVE-2018-1000058](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1000058): Jenkins Pipeline: arbitrary code execution vulnerability

La deserialización de objetos de una fuente que no son de confianza puede provocar la ejecución inesperada de código. ObjectInputStream no proporciona una forma de aplicar reglas en su argumento InputStream. Sabiendo que todas las clases serializables en el alcance del cargador de clases serán deserializadas, existe la posibilidad de que se ejecute código malicioso durante la fase de deserialización incluso si, al final, se lanza una excepción del tipo ClassCastException.

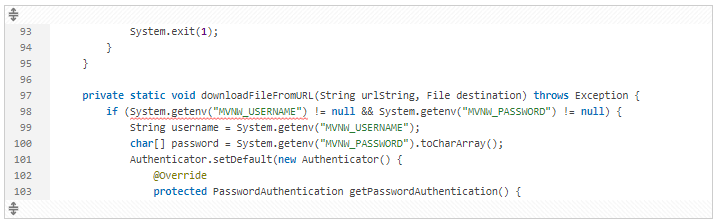


Para evitar la deserialización insegura, debe usar la deserialización anticipada (anterior a Java 9) o un filtro para asegurarse de que se está tratando con el tipo correcto de objeto antes de actuar sobre él.

Algunas librerías que ofrecen deserialización anticipada (look-ahead deserialization), son:

* ikkisoft's SerialKiller
* Apache Commons Class IO's ValidatingObjectInputStream
* contrast-rO0's SafeObjectInputStream
  + 1. Make sure that environment variables are used safely

Las variables de pueden ser susceptibles de inyecciones, como cualquier otra entrada



En el pasado el uso de estas variables ha dado lugar a las vulnerabilidades descritas en estos casos:

* [CVE-2014-6278](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6278)
* [CVE-2019-3464](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-3464)
* [CVE-2018-1000402](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1000402)
* [CVE-2016-10530](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-10530)

Hay que tener encuneta que las variables de entorno puedes ser exponerse de múltiples formas, el almacenamiento de información confidencial en ellas debe hacerse con cuidado:

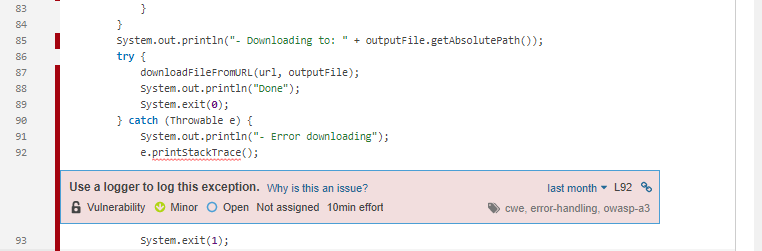
* En sistemas Unix, las variables de entorno de un proceso pueden ser leídas por otro proceso que se ejecuta con el mismo UID.
* Las variables de entorno se pueden reenviar a procesos secundarios.
* La aplicación que se ejecuta en modo de depuración a menudo expone su variable de entorno.

Para evitar estos problemas es recomendable:

* Verificar cada variable de entorno antes de usar su valor.
* Si almacena información confidencial en una variable de entorno, asegúrese de que ningún otro proceso pueda acceder a ella, es decir, el proceso se ejecuta con una cuenta de usuario separada y los procesos secundarios no tienen acceso al entorno de sus padres.
* No ejecute su aplicación en modo de depuración si tiene acceso a información confidencial, incluidas las variables de entorno.
  1. Vulnerabilidades
     1. Throwable.printStackTrace(...) should not be called

Throwable.printStackTrace(...) prints a Throwable and its stack trace to some stream. By default that stream System.Err, which could inadvertently expose sensitive information.

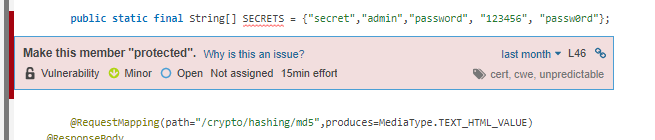
La función printStackTrace (...) imprime una excepción del tipo ***Throwable*** y su seguimiento de pila en algún flujo. De forma predeterminada, ese flujo System.Err, que podría exponer inadvertidamente información confidencial.



El uso de “***printStackTrace***” debe ser sustituido por el uso de ***“logger”***, ya que esto presenta las siguientes ventajas:

* Los usuarios pueden recuperar fácilmente los registros.
* El formato de los mensajes de registro es uniforme y permite a los usuarios navegar por los registros fácilmente.
  + 1. Mutable fields should not be "public static"

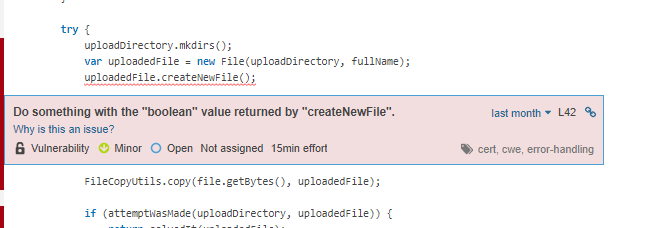
No hay una buena razón para tener un objeto no mutable (***“static***”) como miembro público (por defecto). Estas variables deben trasladarse a clases y reducir su visibilidad.



* + 1. Return values should not be ignored when they contain the operation status code

Cuando el valor de retorno de una llamada de función contiene el código de estado de la operación, este valor debe probarse para asegurarse de que la operación se haya completado correctamente.

El siguiente código lanzara un error sino el comando señalado no se ejecuta correctamente:



Lo correcto sería incluir la siguiente comprobación:

if(¡uploadedFile.createNewFile())

//take appropriate action

)

Esta regla es aplicable a los siguientes objetos:

* java.io.File operations that return a status code (except mkdirs)
* Iterator.hasNext()
* Enumeration.hasMoreElements()
* Lock.tryLock()
* non-void Condition.await\* methods
* CountDownLatch.await(long, TimeUnit)
* Semaphore.tryAcquire
* BlockingQueue: offer, remove
  + 1. XML parsers should not be vulnerable to XXE attacks

Una gestión errónea o demasiado permisiva en la gestión de los archivos XML puede dar lugar a ataques de tipo XML Injection. Estos ataques se basan en la inclusión de referencias externas en ficheros XML que van a ser procesados por la aplicación. Este tipo de vulnerabilidad puede dejar expuesto el sistema a ejecución remota de comandos. Un ejemplo de cómo se lleva a cabo este tipo de ataques es el siguiente:

**<?xml version="1.0" encoding="ISO-8859-1"?>**

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

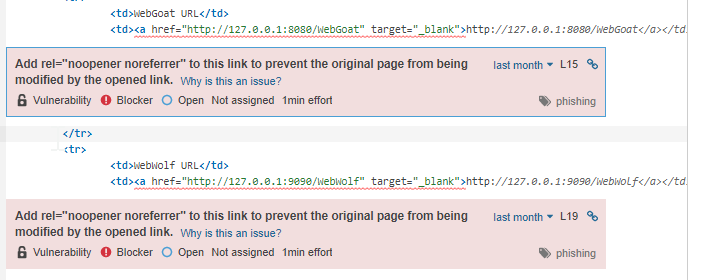
Para evitar este tipo de ataques, debemos desactivar la posibilidad de referenciar entidades externas desde archivos XML siempre que sea posible. A continuación, para complementar y extender la documentación aportada por Sonarqube ([aquí](http://srv-analiza/sonar/coding_rules?languages=java&open=squid%3AS2755&severities=BLOCKER&types=VULNERABILITY)), se añaden referencia a documentación aportada por OWASP sobre este tipo de vulnerabilidades.

* [XML External Entity (XXE) Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)
* [XML External Entity (XXE) Prevention Cheat Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)

Para proteger a los analizadores XML de Java de los ataques XXE, se han implementado [estas propiedades:](https://docs.oracle.com/en/java/javase/13/security/java-api-xml-processing-jaxp-security-guide.html#GUID-94ABC0EE-9DC8-44F0-84AD-47ADD5340477)

* ***ACCESS\_EXTERNAL\_DTD***: Debe establecerse en "" al procesar archivos XML / XSD / XLS (busca DOCTYPE externos)
* ***ACCESS\_EXTERNAL\_SCHEMA***: Debe establecerse en "" al procesar archivos XML/XSD/XLS files (busca schemalocation externos)
* ***ACCESS\_EXTERNAL\_STYLESHEET*** Debe establecerse en "" cuando se procesan ficheros XLS (busca imports, includes, etc.);
  + 1. Links with "target=\_blank" should prevent phishing attacks

Cuando un enlace abre una URL en una nueva pestaña con (***target = "\_ blank"),*** es muy sencillo que la página abierta cambie la ubicación de la página original porque la variable de JavaScript window.opener no es nula y, por lo tanto, "window.opener". La ubicación se puede establecer mediante la página abierta, lo que expone al usuario a ataques de phishing muy simples.



Para evitar que las páginas abusen de window.opener, debemos usar (***rel = noopener***) en <a href=> para forzar que su valor sea nulo en las páginas abiertas. Con esto en su lugar, window.opener es nulo en Chrome 49+, Opera 36+, Firefox 52+, Desktop Safari 10.1+ e iOS Safari 10.3+.

Para navegadores más antiguos, use "noreferrer". En conjunto, ***rel = "noopener noreferrer***" es la forma más segura de mitigar esta vulnerabilidad. Para solventar el problema de la imagen anterior bastaría con la siguiente modificación:

**<a** href="http://127.0.0.1:8080/WebGoat" target="\_blank" rel="noopener noreferrer"**>**

* 1. Duplicidad

La duplicidad para la parte de código Java/JavaScript/HTML se sitúa en 12,8%, en principio, una cifra nada preocupante de cara al mantenimiento de la aplicación.

1. Siguientes pasos

Se recomiendan las siguientes tareas:

* Centrar las actividades del equipo de desarrollo en los problemas de tipo vulnerability.
* Analizar y confirmar los falsos positivos sugeridos por el equipo de CALIDAD en cuanto a bugs se refiere.
* Comprobar y confirmar que las contraseñas encontradas por el analizador no suponen un riesgo para la integridad de los sistemas.

Así mismo el Equipo de Calidad propone:

* Mantener a la aplicación WebGoat en el Nivel 1 del modelo de calidad hasta que se realicen nuevos informes detallados y se analice la evolución derivada de los mismos.
* Mantener los análisis planificados y desatendidos en SonarQube de forma semanal. Tanto el Equipo de Calidad como el Equipo de Desarrollo podrán acceder a los datos actualizados por cada análisis a través de [SonarQube](http://srv-analiza/sonar).
* Realizar el siguiente análisis detallado y generación de informes durante el mes de \_ o en el momento en que se libere una nueva versión si es anterior a esa fecha.
* La instalación del complemento SonarLint en los equipos de los desarrolladores de la aplicación WebGoat. De esta manera se puedan realizar análisis del código fuente en fases tempranas del desarrollo por parte de los integrantes del equipo.

Sobre ESIIAB

Para más información acceda a: [Escuela Superior de Ingeniería Informática (uclm.es)](https://www.esiiab.uclm.es/)

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |