Buenas Prácticas para el Desarrollo de Código Seguro

Pinilla Castañeda Julian Alejandro
Julianpinilla88@gmail.com
Universidad Piloto de Colombia

Abstract— Today is increasing the need for companies to provide security in their products, they are opting to implement the techniques or methodologies that generate secure software development. Because the portfolio of services offered software companies creates greater dependence on applications for normal operation. It is because of this that these techniques generate secure software must be used in each life cycle software (requirements, analysis, design, development and testing), that is, is to identify, analyze and evaluate earliest form possible vulnerabilities, risks or threats that create future problems in applications.

Resumen— En la actualidad es cada vez mayor la necesidad que tienen las empresas de brindar seguridad en sus productos, que están optando por implementar técnicas o metodologías que generen desarrollo de software seguro. Debido a que el portafolio de servicios de software que brindan las compañías genera una mayor dependencia de aplicaciones para su normal operación, es por ello que estas técnicas de generación de software seguro deben estar aplicadas en cada ciclo de vida del software (requisito, análisis, diseño, desarrollo y pruebas), es decir, se busca identificar, analizar y evaluar de forma más temprana posibles vulnerabilidades, riesgos o amenazas que generen problemas futuros en las aplicaciones.

Índice de Términos—ciclo de software, desarrollo, software, aplicativo, seguridad, riesgos, ataque

I. INTRODUCCIÓN

En la actualidad tanto las personas como las organizaciones en el desarrollo normal de sus actividades diarias ejecutan millones de líneas de códigos debido a la gran cantidad de información que circula a través de diferentes plataformas, aplicativos, contenidos web, y muchos otros, que la seguridad en los sistemas, se ha convertido en un tema de vital importancia y hoy, no basta con asegurar la calidad en los productos si no que también se hace necesario contar con la seguridad, de los mismos ya que cada vez es mayor el riesgo de sufrir un ataque informático. Esta situación, en buena parte se presenta desde el momento de codificar el software porque el desarrollador no es consciente de los riesgos de seguridad a los que pueden estar expuestas las aplicaciones. Además de esto la mayoría no cuenta con técnicas o habilidades que les permitan desarrollar códigos seguro, por esto, que las entidades deben adoptar políticas y metodologías que permitan cumplir con las exigencias de seguridad que sean pertinentes.

El ciclo de vida de un software se puede definir como un proceso interactivo e incremental. Esto quiere decir, que se encuentra en constante evolución y cuenta con varias etapas del modelo de cascada el cual cuenta con 5 fases del ciclo de vida del software.,

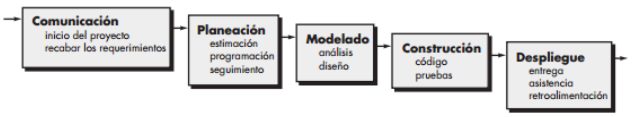

Fig. 1. Ciclo de vida del software. Fuente: Ingeniería de Software un Enfoque Práctico

Además de este modelo de vida de software que se considera el modelo clásico, existen una variedad de modelos como CMMI (Modelo de Madurez de la Capacidad Integrado) que es un conjunto de buenas prácticas para el desarrollo de software, esta metodología en si está enfocada más hacia la calidad del producto y por esto realiza cubrimiento del ciclo de vida del software desde su concepción hasta la entrega y mantenimiento; esta metodología no contempla la seguridad como parte de su entorno y por eso deben implementarse procesos alternos de seguridad.

Finalmente se puede definir un producto seguro como "un producto que protege la confidencialidad, integridad y disponibilidad de la información de los clientes, y la integridad y disponibilidad de los recursos de procesamiento, bajo el control de propietario o administrador del sistema"[1] . Esta es una definición importante debido a que la seguridad no debe quedar relegada a un segundo plano y debe estar presente desde la concepción misma del desarrollo, también debe estar enfocada a proteger los activos críticos. Los cuales salen a partir de un correcto análisis de vulnerabilidades y los desarrollos generados debe contemplar la perspectiva del agresor. Este artículo estará enfocado en entender en que consiste el desarrollo de código seguro y su aplicación a lenguajes de programación como JAVA.

II. . DIRECTRICES DE PROGRAMACIÓN SEGURA

Existen numerosas directrices y consejos que un programador puede implementar para ayudar en la prevención de los errores de seguridad comunes en las aplicaciones. Muchos de estos pueden ser aplicados a cualquier lenguaje de programación, y otros pueden estar generados especificamente para un lenguaje de programacion como por ejemplo java,php,perl entre otros. Es por esto, que en este cápitulo se desprenden una serie de directrices que deberian ser tomadas en cuenta por los desarrolladores a la hora de iniciar a codificar una aplicacion:

A. Validaciones de entrada

Los datos de entrada de una aplicación se puede definir como el conjunto de parámetros que van hacer digitados y posteriormente capturados para ser manipulados en codificación y para esto, en términos de implementación, validación de entrada se debe definir un conjunto de caracteres válidos los cuales deben ser validado cada caracter de entrada y si no se encuentra un cáracter dentro de esta seleccion de caracteres validos se debe generar un error o ser manejado por medio de excepciones, las cuales deben traducir estos errores a un lenguaje comprensible para el usuario. En ninguno de los casos se debe arrojar la línea del código ni tampoco se deben notar la clase, evento o método ya que esto puede dar indicios al presunto atacante sobre las tecnologías implementadas.

B. Sentencias SQL.

Cuando las consultas se construyen directamente con los datos del usuario entre líneas o concatenan directamente con el texto de la consulta, en lugar de utilizar los parámetros de vinculación de tipo seguro, esto puede permitir al atacante realizar un ataque de inyeccion de sql y consiste en una debilidad de programación en que la aplicación construye dinámicamente consultas SQL utilizando la concatenación de cadenas de datos, esto puede permitir que el atacante pueda modificar consultas como por ejemplo para saltarse el proceso de autenticación de usuario, un ejemplo de cómo deberían ejecutar consultas sql para evitar esto sería el siguiente:

```
String firstname = req.getParameter("firstname");
String lastname = req.getParameter("lastname");
// FIXME: do your own validation to detect attacks
String query = "SELECT id, firstname, lastname FROM authors WHERE forename = ? and surname = ?";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, firstname );
pstmt.setString( 2, lastname );
try
{
       ResultSet results = pstmt.execute( );
}
```

Fig. 2. Ejecución de sentencias SQL de forma segura. Fuente: OWASP

C. Código Comentado

Todo código comentado dentro de la aplicación debe ser eliminado antes de ser lanzado en producción debido a que el código comentado puede generar el uso accidental en producción que puede alterar el correcto funcionamiento de la aplicación, como también puede llegar a contar con código quemado. Es por esto que por ningún motivo puede salir a producción, en la vida real es un proceso muy común encontrar código comentado, debido a que muchas veces en el proceso de desarrollo se usa para la

[1] Writing Secure Code Second Edition Michael Howard and David Leblanc

generación de pruebas pero casi nunca se eliminan en el momento de liberar el código.

D. Mensajes de Error

Todos los mensajes de error que lleguen al usuario final no deben contener información confidencial como sistema operativo, red, lenguaje de programación, línea del error, motor de base de datos, estos errores deben darse tratamiento y ser estandarizados de forma genérica y comprensible para el desarrollador pero que sea entendible para el usuario como por ejemplo "Se presentó el error 1281 no se pudo registrar el usuario, póngase en contacto con el proveedor de servicio."

E. Contenido URL

En el contenido url nunca se debe mostrar información sobre la url como passwords, nombre de servidores, direcciones ip, lenguaje de programación o las rutas del sistema de archivos que revelan la estructura de los directorios del servidor web, ya que para un atacante esto se convierte en información valiosa ya que le da una visión sobre objetivo.

F. *Valores Quemados*

En muchas ocasiones los desarrolladores de software tienden a utilizar HARDCODE (Código Quemado) que consiste en dejar variables definidas por defecto en el código, esto puede configurar un problema de seguridad ya que algunas de estos valores preestablecidos como puede ser un usuario, contraseña, dirección ip entre muchas otras, pueden salir a un ambiente de producción y un atacante puede aprovechar esta vulnerabilidad en el código para conocer el entorno y explotar fallas de seguridad como por ejemplo.

```
import java.sql.*;

public class JDBCDemo {


static String query =
        "SELECT customerno, name " +
        "FROM customers;";


public static void main(String[]args)
{
    String username = "cowboy";
    String password = "123456";
    try
    {
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
    }catch(SQLException ex){System.out.println("Exception! " + ex);}

}

}
```

Fig. 3. Hardco de Fuente: stackOverflow

Este es un ejemplo muy diciente del riesgo de seguridad al que se puede ver expuesto una aplicación si no se controla debidamente este error, es por esto que las buenas prácticas de software están enfocadas a la no utilización de código quemado y se recomienda usar unas clases de constantes en las que se definan y se almacenen estos valores. Usar constantes además permite mejorar la calidad del código generado por la organización.

G. Cifrado de Contraseñas.

Las Contraseñas que sean utilizadas en la codificación de software deben estar cifradas ya sea en Sha1, MD5 el algoritmo de cifrado que la organización considere conveniente ya que de no implementar una política de contraseñas, estas pueden viajar en texto plano y puede ser capturada y explotada por un atacante.

H. Menor Privilegio

Este es un principio fundamental de seguridad ya que para las aplicaciones desarrolladas se debe garantizar que la información sea visible solamente para los que la requieren y para esto la organización debe implementar una política de roles y privilegios ya que todos los usuarios del sistema no requieren la misma información

III. JAVA

En la actualidad JAVA es uno de los lenguajes de programación más populares, debido a que Java funciona con las principales plataformas de hardware y sistemas operativos. Entre las características más relevantes se encuentran: Que es simple, orientado a objetos, robusto y la característica que más nos interesa en este artículo es que es seguro. La máquina virtual de java (JVM) es capaz de interpreter y ejecutar instrucciones expresadas en un código binario especialmente desarrollado por java el cual se denomina (bytecode), este verifica todo El código Java pasara evitar errores en la compilación de la aplicación, esta funcionalidad permite detectar fragmentos de código ilegal -código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto.

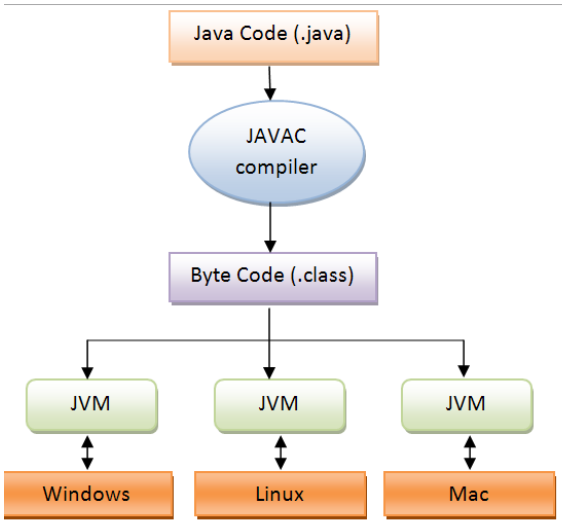

Fig. 4. Java Virtual Machine. Fuente: javapapers

A continuación se mencionan buenas practicas a la hora de codificar en este lenguaje de programación:

A. *Impresión de mensajes de salida*

La impresión de mensajes de consola es ampliamente utilizada en el proceso de codificación de software debido a que le permite al desarrollador realizar seguimiento a funcionalidades y navegar por métodos, clases dejando simplemente mensajes alrededor del código como por ejemplo:

```
class Aplicacion
{
   public static void main( String args[] )
   {
      System.out.println( "Mi primera aplicación Java" );
   }
}
```

Fig. 5. Java Virtual Machine. Fuente: Microsoft

El problema de esta práctica es que en la mayoría de las ocasiones se omite eliminar estos mensajes y salen al ambiente de producción y pueden ser de gran ayuda a un atacante por que puede dar nombres de clases, métodos, la tecnología usada, consultas a la base de datos entre algunas otras, es recomendable como buena práctica realizar una inspección de código por parte de un líder o un par que permita detectar esta falla antes de liberar a producción un desarrollo.

B. *Encapsulación*

En JAVA si se declara una clase, método o campo sin un modificador de acceso (privado, público y protegido) luego estos paquetes por defecto tienen acceso a ninguna clase dentro del mismo paquete, se debe tener en cuenta que si bien esto proporciona encapsulación para el paquete, esto sólo sucede si cada trozo de código que se carga en el paquete es controlado por persona (s) autorizada. Un usuario malintencionado puede insertar sus propias clases y tener pleno acceso a todas las clases, métodos y campos en el paquete.

La idea de usar correctamente la encapsulación es poder ocultar detalles de implementación tales como variables internas que mantienen el estado de un objeto y los mecanismos internos como son algoritmos entre otros.

C. *Manejo de Excepciones*

El manejo de excepciones es fundamental para lograr un correcto desarrollo seguro de código ya que cuando un código java viola restricciones semánticas del lenguaje se produce un error que es lanzado por la máquina virtual de java, existen muchas clases de errores que generan excepciones y nos dan a entender fallos como desbordamiento de búfer, desbordamiento de memoria, intentar acceder a un vector fuera de sus límites, entre otras. Y si no se capturan adecuadamente le puede dar indicios a un

atacante sobre fallos en el código, nombre de la clase o método que genera el problema y le puede dar las bases para un futuro ataque. Es por esto que las buenas prácticas solicitan que estas excepciones sean capturadas adecuadamente y lanzadas al cliente con mensajes como por ejemplo "consulta no exitosa" o "no se pudo procesar la solicitud "

IV. Herramientas de Análisis de Código

El uso de un analizador de código fuente es considerado como una buena práctica en el desarrollo de código seguro ya que tiene como objetivo principal garantizar que el código fuente cumpla con un cierto nivel de seguridad establecido por una serie de reglas establecidas por las necesidades del negocio. Existen varias formas de realizar esta actividad, se puede realizar una inspección de código manual realizada por un par del proceso de desarrollo que puede ser el líder de desarrollo el cual debe verificar si el código cumple con las reglas establecidas por la organización, el problema que tiene este proceso es que se pueden pasar por alto algunos errores no evidentes. Estos errores pueden conducir a errores de seguridad significativos en las aplicaciones, es por esto que es bueno automatizar estos procesos por medio de herramientas analizadoras de código automáticas.
Las cuales deben contar con algunos de los siguientes atributos:

- Flexibilidad: las herramientas de inspección de código debe contar con varias opciones para ejecutar esta actividad entre las cuales sobresalen: análisis estatice de código fuente como checkstyle, pmd y findbugs entre otros.
- Precisión: el objetivo de estas herramientas es poder encontrar el máximo de errores que permitan depurar y otorgar calidad y confianza sobre el código fuente que se va a liberar.
- Facilidad de uso: debido a la utilidad que brinda esta clase de herramientas en el ciclo de vida del software debe ser fácil de usar e intuitiva debido a que no solo personal de T.I. pueda operarlas
- Reportar Hallazgos: estas herramientas deben reportar los errores encontrados en el escaneo de código fuente para permitir dar solución a estos problemas

A continuación se mencionan algunas herramientas analizadoras de código fuente:

1. ConQAT: es un motor de análisis altamente configurable calidad del software. ConQAT se basa en una arquitectura de tuberías y filtros que permiten realizar configuraciones complejas de análisis de una manera muy apligable debido a su entorno grafico. Esta herramienta esta soportada para los lenguajes de programacion Java, C#, C++, JavaScript entre otros.

2. Parasoft: es una herramienta bastante poderosa en la prevencion de defectos de desarrollo, se encuenta soportada para los lenguajes de programacion c,c++ y .net

Finalmente existe una herramienta que he podido comprobar la funcionalidad y su versatilidad y se trata de SONAR que es una plataforma para gestionar la calidad del código y como tal abarca 7 ejes de la calidad del código.

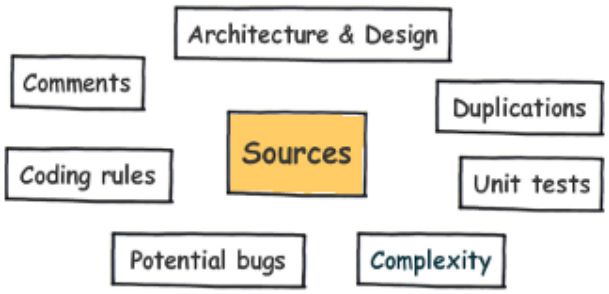

Fig. 6. Estructura Sonar. Fuente: SONAR

Una de sus mayores ventajas es la de ser una herramienta de código libre que permite configurar una gran cantidad de reglas de codificación como están las mencionadas anteriormente en este ensayo, impresiones en consolas, métodos mal definidos, clases sin modificadores de acceso entre muchos otros. Y su funcionalidad se basa en cargar previamente el proyecto a desarrollar y una vez este se encuentra arriba, el desarrollador procede a

realizar la codificación requerida y finalizado este se corre nuevamente el proyecto arrojando esta vez el número de errores inyectados al código, su criticidad catalogándola como alta, media, baja y bloqueante entre otras. Posteriormente a este análisis el desarrollador tiene las herramientas necesarias para mejorar la calidad de sus entregables.

V. CONCLUSIONES

- La seguridad debe están implícita en cada fase del desarrollo de software ya que esto genera una mayor robustez sobre el código desarrollado.

- Se debe buscar una mayor comprensión de los desarrolladores sobre los riesgos de seguridad que se pueden encontrar en la codificación del software ya que existe un gran desconocimiento sobre ellos debido a que en la mayoría de casos prima la solución generada sin importar el cómo se logra.

- Una correcta estrategia para generar código seguro debe contar con un apoyo incondicional por parte de la gerencia ya que para lograr estos objetivos es necesario tiempo, dinero y constante capacitación del personal ya que el mundo de la seguridad es muy dinámico y lo que hoy puede generar una solución quizás mañana no.

- Para una correcta generación de código seguro se debe realizar un análisis minucioso por parte de la gerencia de T.I para definir las reglas de codificación o las buenas prácticas aplican a las reglas de negocio de la organización ya que esto es fundamental para el éxito de la implementación.

REFERENCIAS

[1] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013
[2] Writing secure code Second Edition (Michel Howard y and David)
[3] Ingeniería del Software. Un Enfoque Práctico 7 Edición Roger Pressman
[4] http://stackoverflow.com/
[5] http://www.oracle.com/technetwork/java/seccodeguide-139067.html.
[6] http://howtodoinjava.com/java-best-practices