Seguridad en aplicaciones JAVA Empresarial

Yohan Carlos Camacho Santana [1], Fidelio Castillo Romero [2], Miguel Pérez Vasconcelos [2], Eutimio Sosa Silva [2]

[1]Departamento maestría, Tecnológico Nacional de México- Instituto Tecnológico de Villahermosa.
[2]Departamento sistemas, Tecnológico Nacional de México- Instituto Tecnológico de Villahermosa.

Resumen

Por más de dos décadas, la arquitectura Java ha sabido dominar el desarrollo del entorno empresarial y doméstico. Son incontables las veces en que Java Platform Enterprise Edition, anteriormente conocida como J2EE, y más recientemente Jakarta EE, ha sido la plataforma seleccionada por un gran número de bancos, departamentos de defensas y empresas en general. Java EE ha evolucionado dramáticamente con los años, simplificando en gran medida los esfuerzos de desarrollo al momento de construir aplicaciones empresariales complejas. Una de las principales ventajas de Java2EE es la seguridad que es capaz de brindar a las aplicaciones que son desarrolladas bajo estas tecnologías. Logrando establecer altos niveles de seguridad y confiabilidad si se implementan de forma correcta el desarrollo de las soluciones de software.

Abstract

For more than two decades, Java architecture has dominated the development of the business and home environment. There are countless times that the Java Platform Enterprise Edition, formerly known as J2EE, and more recently Jakarta EE, has been the platform of choice for a large number of banks, defense departments, and businesses in general. Java EE has evolved dramatically over the years, greatly simplifying development efforts by creating complex business applications. One of the main advantages of Java2EE is the security that it is capable of providing to applications that are developed under these technologies. Establishing high levels of security and reliability if the development of software solutions is implemented correctly.

Palabras Claves: Java, J2EE, Seguridad, Vulnerabilidad, API
Keywords: Java, J2EE, Security, Vulnerability, API

1. INTRODUCCIÓN

El modelo de aplicación Java EE comienza con el lenguaje de programación Java y la máquina virtual Java. La portabilidad, seguridad y productividad del desarrollador que proporciona la base del modelo de aplicación. El modelo de aplicación Java EE define una arquitectura para implementar servicios como aplicaciones de varios niveles que ofrecen la escalabilidad, accesibilidad y capacidad de administración que necesitan las aplicaciones de nivel empresarial. Este modelo divide el trabajo necesario para implementar un servicio multinivel en las siguientes partes: la lógica comercial y de presentación implementado por el desarrollador y por otra parte los servicios del sistema estándar proporcionados por la plataforma Java EE. (Oracle, 2017) Son varias las tecnologías y APIs (Application Programming Interface) que integran la plataforma de Jakarta EE, haciendo referencia a las mismas mediante la imagen siguiente:

ISSN: 2007-4786

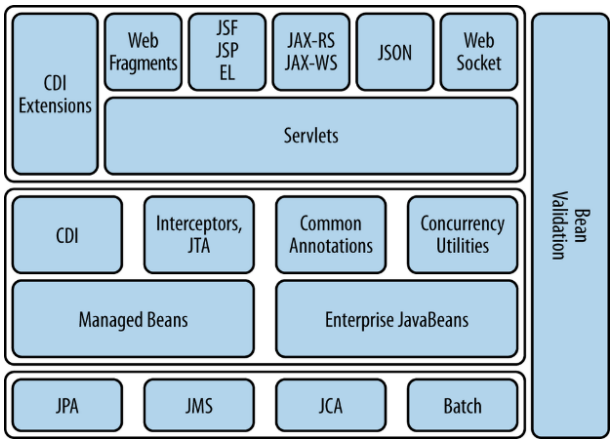

Figura 1. APIs de Jakarta EE

2. DESARROLLO

Unos de los temas cruciales al momento de desarrollar aplicaciones Java EE es la seguridad, ya sea que la aplicación a desarrollar sea consultada a través de una interface web, utilizando Servlet o JSF, que se expongan las funcionalidades de la misma utilizando WebServices del tipo SOA o Rest o que directamente un cliente Swing tenga acceso a un Enterprise Java Beans (EJB) declarado, es necesario establecer los mecanismos que garanticen la seguridad de la información de los sistemas. La seguridad se debe de establecer en la totalidad de las capas (Capa de Presentación, Capa de Negocio y Capa de Acceso a Datos) presentes en el sistema a desarrollar, aunque lo anterior no constituye un estándar si es recomendable su implementación de la forma más conveniente para el desarrollador y cliente.

La seguridad de un sistema se engloba dentro de dos conceptos básicos; la autenticación proceso donde se valida la identidad del usuario, usualmente manejado para dar acceso a un sistema determinado haciendo uso de diferentes datos. Estos últimos son conocidos como credenciales, el segundo concepto es llamado autorización que no son más que los diferentes permisos otorgados al usuario en cuestión para lograr mediante los mismos restringir las acciones que puede ejecutar en el sistema. La autenticación y la autorización están ligadas intrínsecamente a los usuarios, grupos y roles conceptos que conforman la base del esquema de seguridad en un sistema empresarial. Para simplificar la administración los usuarios son divididos por grupos, estos son una partición lógica para la identificación de usuarios. Los roles serán los que decidirán dentro de la aplicación que permisos serán otorgados a los grupos o a los usuarios según sea el caso. Un rol permite relacionar los grupos de una organización en entidades que entienda y procese correctamente la aplicación. De esta forma los roles son una abstracción de los grupos enfocados en el servidor de aplicaciones Java. Comúnmente se realiza por parte del servidor de aplicaciones un mapeo entre grupos y roles de forma automática, siempre que los nombres sean coincidentes. En caso de no cumplirse esto último, se deberá realizar manualmente dicha estructura de compenetración. Esto último se puede realizar mediante configuración en el servidor de aplicaciones Java utilizado o a través de un archivo XML, en el caso de GlassFish se realiza en el archivo *"glassfish-web.xml"* y en WildFly el archivo es llamado *"jboss-web.xml"*.

La seguridad en Java EE está basada en la interfaz de programación de aplicaciones (API) del Servicio de Autenticación y Autorización Java (JAAS). Está enfocado en agregar seguridad en la capa Web y en la capa de Negocios conformada por los EJB (Enterprise Java Beans). Una vez que el usuario se autentifica ante una aplicación, se crea un objeto (Principal) el cual es propagado a lo largo del ciclo de vida de la aplicación. El mismo puede ser consultado en cualquier momento sin importar la capa de negocio en que se encuentre, con el objetivo de evitar el proceso de autenticación por cada una de las capas de manera individual. El objeto Principal está asociado con uno o más roles, por cada acción el sistema de seguridad revisa que se cuente en todo momento con los permisos necesarios para utilizar los recursos. El mismo es enviado de forma transparente entre las diferentes capas de la aplicación donde se desea consultar los privilegios del usuario según sea necesario.

En la capa Web la seguridad es tratada mediante el archivo "*web.xml*" en el cual se realizarán las declaraciones necesarias para implementar el sistema de autenticación deseado. Existen varios tipos de autenticación que se pueden implementar tales como: http-basic, http-form, https-client-cert, siendo los dos primeros los más utilizados. Estas formas son declaradas dentro de la etiqueta <login-config> y dependiendo del método de autenticación escogido se puede declarar un realm-name que representa una abstracción del servidor de aplicaciones, donde se especifican cada una de las políticas de seguridad del sistema en el servidor, pudiendo ser estas declaradas usando una base de datos, archivos o basados en implementaciones del propio sistema Linux como Kerberos o Lightweight Directory Access Protocol (LDAP). Además de la declaración de <login-config> aparece la etiqueta <security-constraint> usada para declarar a que parte de nuestra aplicación se tendrá acceso, así como que roles participaran en la implementación se puede agregar además el tipo de método HTTP permitido en la aplicación: POST, PUT, GET, DELETE, entre otros.

Ejemplo de archivo de configuración web.xml usando el tipo de autenticación a través de formulario:

```
<servlet>
   <display-name>index</display-name>
   <servlet-name>index</servlet-name>
   <jsp-file>/index.jsp</jsp-file>
</servlet>
<login-config>
 <auth-method>FORM</auth-method>
  <form-login-config>
   <form-login-page>/login.jsp</form-login-page>
   <form-error-page>/loginError.jsp</form-error-page>
  </form-login-config>
</login-config>
<security-constraint>
 <web-resource-collection>
   <web-resource-name>TEST</web-resource-name>
    <url-pattern>/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
   <role-name>ROLE_ADMIN</role-name>
```

```
  <role-name>ROLE_GUEST</role-name>
 </auth-constraint>
</security-constraint>
<session-config>
 <session-timeout>20</session-timeout>
</session-config>
```

La pareja de datos usuario-clave no ha cambiado mucho desde que se empezaron a usar los sistemas y aplicaciones, siendo la clave uno de los componentes de esta pareja al cual muchos le dedican más tiempo de implementación y verificación. Algunas estrategias definen la clave por su obligatoria recepción de caracteres alfanuméricos combinados con letras capitales y restringen un máximo y mínimo de dichos datos. Otras en cambio toman parte de esta representación sumando a la ecuación un token, el cual se usará para ser presentado por el servidor como una credencial de autenticación. En todo momento sin importar que estrategia se use por parte del desarrollador la información debe ser enviada por medio de un canal seguro de datos mediante certificación Secure Sockets Layer (SSL) termino el cual se sigue usando en la actualidad a pesar de haber sido reemplazado por TLS (Transport Layer Security) con la opción de cifrado más conveniente ya sea cifrado ECC (Elliptic Curve Cryptography), DSA (Digital Signature Algorithm) o RSA (Ron **R**ivest, Adi **S**hamir, y Leonard **A**dleman nombres de los creadores del algoritmo). (ssl247, 2020)

La clave y los otros datos, aunque la conexión SSL los cifra, es recomendable encriptarlos usando para ello alguno de los tipos de encriptación disponibles en la actualidad. Desde hace ya un tiempo lo más común era disponer del encriptamiento utilizando algoritmos como MD5 (Message-Digest), SHA (Secure Hash Algorithm) y cualquier combinación de sus variantes. Aunque ya no es recomendable usar el MD5 debido a que fue quebrado a través de un mecanismo de generación de colisiones según fue probado por grupo de investigadores europeos:

“Nuestro ataque aprovecha una debilidad en la función de hash criptográfico MD5 que permite la construcción de diferentes mensajes con el mismo hash MD5. Esto se conoce como una "colisión" MD5. El trabajo previo sobre colisiones MD5 entre 2004 y 2007 mostró que el uso de esta función hash en firmas digitales puede conducir a escenarios de ataque teóricos. Nuestro trabajo actual demuestra que al menos un escenario de ataque puede explotarse en la práctica, exponiendo así la infraestructura de seguridad de la web a amenazas realistas.” (Sotirov et al., 2008)

El SHA0 y el SHA1 han presentado también sus problemas de seguridad y es mejor dejar de utilizarlos ampliamente. No siendo así con el SHA256 o 512 que aún se mantienen inquebrantables a los ataques y pruebas de estrés a los que han sido sometidos. Además de estas consideraciones de protección de los datos, formas de validar y autenticar a los usuarios también se hace necesario la exposición de algunas de las vulnerabilidades que frecuentemente son utilizadas, como es el caso de las Inyecciónes. Las fallas de inyección, como la de SQL, NoSQL, OS y LDAP, se producen cuando se envían datos no confiables a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete para que ejecute comandos no deseados o acceda a los datos sin la autorización adecuada. La prevención de la inyección requiere mantener los datos separados de los comandos y consultas. La opción preferida es usar una API segura, que evite el uso del intérprete por completo o proporciona una

ISSN: 2007-4786

interfaz parametrizada, usar herramientas de mapeo relacional de objetos (ORM) es otra opción disponible. Para cualquier consulta dinámica residual, escape caracteres especiales usando la sintaxis de escape específica para el intérprete que se esté utilizando. Se debe utilizar de ser posible el comando LIMIT y otros controles SQL en las consultas para evitar la divulgación masiva de registros en caso de ocurra una inyección SQL. Otra de las brechas de seguridad que puede presentar un software de Java EE es el Cross-Site Scripting XSS, los defectos de XSS ocurren cada vez que una aplicación incluye datos no confiables en una nueva página web sin una validación o escape adecuados, o actualiza una página web existente con datos proporcionados por el usuario utilizando una API de navegador que puede crear HTML o JavaScript. XSS permite a los atacantes ejecutar scripts en el navegador de la víctima que pueden secuestrar sesiones de usuario, desfigurar sitios web o redirigir al usuario a sitios maliciosos. La prevención de XSS requiere la separación de datos no confiables del contenido activo del navegador esto se puede lograr escapando los datos de solicitud HTTP no confiables basados en el contexto en la salida HTML (cuerpo, atributo, JavaScript, CSS o URL). Habilitar una Política de seguridad de contenido (CSP) como un control de mitigación de defensa en profundidad contra XSS. Es efectivo si no existen otras vulnerabilidades que permitan colocar código malicioso a través de archivos locales incluidos. (OWASP, 2020)

3. CONCLUSIONES

Estableciendo adecuadamente las diferentes estrategias de seguridad necesarias para proteger la integridad de los sistemas, creados bajo las tecnologías Java Empresarial, podremos lograr un nivel de seguridad bastante alto en cualquier aplicación a desarrollar. Siempre se debe tener en cuenta las buenas prácticas para establecer dichos niveles y no escatimar en la implementación de las medidas necesarias para proteger el bien más preciado de cualquier empresa, la información. Un sistema completamente seguro es aquel al que nadie tiene acceso físico, ni lógico, haciéndolo por ende inutilizable. Se pretende ilustrar con la frase anterior que nunca debemos fiarnos de la total seguridad implementada en un sistema siempre habrá una forma de vulnerarlo, por eso se debe de pensar como atacante y no como desarrollador a la hora de implementar medidas de seguridad en los sistemas.

REFERENCIAS

[1] Oracle. (2017). *Java EE Application Model*. https://javaee.github.io/tutorial/overview003.html
[2] OWASP. (2020). *OWASP Top Ten Web Application Security Risks | OWASP*. https://owasp.org/www-project-top-ten/
[3] Sotirov, A., Stevens, M., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D. A., & Weger, B. de. (2008, diciembre 30). *MD5 considered harmful today*. Creating a Rogue CA Certificate. https://www.win.tue.nl/hashclash/rogue-ca/
[4] ssl247. (2020). *¿Qué son RSA, DSA y ECC?* SECURING VALUE. https://www.ssl247.es

Correo Electrónico Autor: *yccamacho87@aol.com*