ISSN: 2594 - 2905

PROCESO DE SEGURIDAD PARA EVITAR LA INFILTRACIÓN DE INYECCIÓN SQL (SQL INJECTION)

SECURITY PROCESS TO AVOID THE INFILTRATION OF SQL INJECTION (SQL INJECTION)

M.C. Juan Manuel Bernal Ontiveros[1], Dr. Francisco Zorrilla Briones[2], MSL Noé Ramón Rosales Morales[3],Ing. Margarita Bailón Estrada[4], M.C. Marisela Palacios Reyes[5]

[1]Maestría en Ciencias en Ingeniería Industrial, [2]Doctorado en Ciencias en Ingeniería Industrial, [3]Maestría en SoftwareLibre, [4]Ingeniería en Sistemas Computacionales en Programación, [5]Maestría en Ciencias en Ciencias de la Computación

[1,2,3,4,5]Instituto Tecnológico de Ciudad Juárez, Departamento de Sistemas y Computación, Avenida Tecnológico No.1340 Fraccionamiento El Crucero Código Postal 32500, Teléfono (656) 6882500, Ciudad Juárez, Chihuahua México
jbernal@itcj.edu.mx[1], fzorrilla@itcj.edu.mx[2], nrosales@itcj.edu.mx[3], mbailon@itcj.edu.mx[4], mpalacios@itcj.edu.mx[5]

Resumen – Las bases de datos son por demás importantes y utilizadas extensamente en los procesos productivos y de servicios en muchas organizaciones; estas bases de datos contienen, en muchos casos, datos sensibles para el(los) dueño(s) de los mismos y se almacenan en sistemas gestores de bases de datos, tales como Oracle, Microsoft SQL Server, entre otros. Atacar una base de datos es uno de los objetivos favoritos para los criminales cibernéticos. Estas bases de datos se han convertido, desde hace tiempo, en un gran blanco para los atacantes que buscan información de interés. Uno de los usos de las bases de datos es el almacenar una gran cantidad de información valiosa, por tanto, un número creciente de organizaciones utilizan metodologías para auditar sus sistemas y evaluar su vulnerabilidad a ataques y robos; metodologías tales como OWASP, OSSTMM, entre otras. La inyección SQL es la segunda amenaza más utilizada, atacando sitios web robando y alterando la información de las bases de datos.

Palabras Clave: Seguridad, Bases de Datos, Inyección, Hacker, Ataque Cibernético, Vulnerabilidades.

Abstract -- Databases have become important and widespread in the production of organizations, since the vast majority of sensitive data is stored in database management systems, such as Oracle, Microsoft SQL Server, among others. Attacking a database is one of the favorite targets for criminals. They have long since become a major target for attackers looking for information of interest. One of the uses of databases is to store a large amount of valuable data, therefore a growing number of regulations have led organizations to audit access to sensitive data, protecting themselves from attacks and abuses. In such a way, automation in the auditing of databases has emerged, identifying malicious activities and fraud. SQL injection is the second most common threat, attacking websites overloading databases.

Key words – Security, Databases, Injection, Hacker, Cyber Attack, Vulnerabilities.

INTRODUCCIÓN

Con el tiempo, las bases de datos se han convertido velozmente en el almacén primordial de búsqueda y localización de las informaciones que la sociedad precisa, lo que ha venido a desatar que el comportamiento de la seguridad sea preocupante más que nunca. Además, el manejo a través de la conexión a Internet ha añadido otras amenazas que siempre habían padecido del acceso remoto desde los más recónditos lugares [9]. Este trabajo pretende de alguna manera fortalecer la seguridad de la que pueden ser vulnerables las bases de datos por todo tipo de amenaza, llámese hacker, infiltrador, usuario no autorizado etc., que cometa el delito, fraude, robo o cualquier infiltración no permitida o autorizada.

La Inyección SQL es una de las vulnerabilidades más devastadoras que afectan a un negocio o empresa, ya que pueden conducir a la exposición de toda la información confidencial almacenada en una aplicación de base de datos, incluyendo información útil, tales como nombres de usuarios, contraseñas, nombres, direcciones, números telefónicos, y detalles de tarjetas de crédito.

La inyección SQL se define a la vulnerabilidad que resulta cuando a un atacante se le da la posibilidad de infiltración y manipulación en las consultas SQL (lenguaje de consulta estructurado), que una aplicación pasa a un final de proceso en una base de datos no visible.

Por esta razón, se puede ser capaz de influir en el contenido de la información que puede ser transferida a la base de datos, el atacante puede aprovechar la sintaxis y capacidades de SQL, así como la potencia y flexibilidad de soporte de la funcionalidad tanto de la

IPSUMTEC
ISSN: 2594 - 2905

base de datos como del sistema operativo disponible para la misma base de datos (plataforma).
La mayoría de las aplicaciones web desarrolladas hoy en día hacen uso de una base de datos para ofrecer páginas dinámicas y almacenar información de los usuarios como de la propia herramienta, datos a los que se accede por medio del lenguaje SQL, lenguaje para interaccionar con las bases de datos relacionales. La inyección SQL no es una vulnerabilidad que afecta exclusivamente a las aplicaciones Web; cualquier código que acepta una entrada a partir de una fuente no confiable y luego esa entrada es usada para formar sentencias SQL dinámicas que podrían ser vulnerables (aplicaciones fast client en una arquitectura cliente/servidor) [3].

Las vulnerabilidades del tipo Client-Site SQL Injection no son muy diferentes a cualquier vulnerabilidad de la inyección SQL tradicional; sólo cambia el contexto o forma de ejecución. Esto quiere decir que la principal fuente de problemas relacionados con este tipo de vulnerabilidades se encuentra en la incorrecta o insuficiente validación de los datos utilizados para conformar las consultas. La mejor forma de prevenir este tipo de errores, es validar los datos de entrada y utilizar la jerarquía de usuarios, en lugar de concatenar los valores de entrada con la cadena de la consulta SQL [1].

En el pasado, la inyección SQL fue típicamente usada más contra el servidor del lado de bases de datos. Sin embargo, con la actual especificación del lenguaje HTML5, un atacante podría igualmente ejecutar un código en lenguaje JavaScript u otros códigos, con el fin de interactuar del lado de la base de datos del cliente para robar información. Esto ocurre del mismo modo con las aplicaciones móviles como la plataforma Android, códigos maliciosos y/o códigos (scripts) del lado del cliente que son aplicaciones que aprovechan formas similares.

Descripción del Problema

- Toda infiltración por parte de cualquier amenaza sobre las bases de datos, rompe la confidencialidad de la información de la cual las empresas y negocios dependen de su funcionamiento en la toma de decisiones y control de los recursos, por lo que se deben de tomar en cuenta con suma importancia, aspectos que pueden ser atacados, fracturando la seguridad interna de los datos. Estos aspectos son:

- Secuestro del sistema (Ransomware);
- Confidencialidad de la información;
- Fraudes informáticos;
- Robo de datos con algún interés malicioso (extracción de datos);
- Costos económicos que implican las amenazas, al romper la seguridad;
- Pérdida de identidad por la infiltración a las bases de datos, o lo que es lo mismo, la suplantación de usuarios que al poder acceder al sistema de credenciales, es posible que un atacante obtenga las credenciales de otro usuario y realice acciones con la identidad robada, spoofeada a otro usuario;
    - Anomalías de modificación de las bases de datos (pérdida de integridad o corromper la información),
    - son inconsistencias que se introducen cuando se realizan cambios en el contenido de una base de datos;
    - Destrucción de datos (pérdida de datos);
    - Alteración de los privilegios (elevación de privilegios) de acceso a la base de datos;
    - Denegación de servicio: La modificación de comandos SQL puede llevar a la ejecución de acciones destructivas como el borrado de datos, objetos o el cese de servicios con comandos de apagado y arranque de los sistemas. Asimismo, se pueden inyectar comandos que generen un alto cómputo en el motor del gestor de base de datos que haga que el servicio no responda en tiempos útiles a los usuarios legales.

La inyección SQL, tiene ciertas características que deben ser tomadas en cuenta para conocer contra que nos enfrentamos, y en las que se encuentran:

- Está disponible para las plataformas de sistemas operativos Windows, Linux y Unix;
- Soporta SSL (Secure Sockets Layer), que es la capa de conexión segura. Esta proporciona autentificación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía;
- Realiza la técnica de inyección ciega de SQL, comparando las respuestas verdaderas y falsas de las páginas o de los resultados de las cookies y los retrasos de tiempo.

Soporta los motores de los gestores de bases de datos Microsoft SQL Server, Oracle, MySQL, Sybase Adaptive y DB2.

Revisión de Literatura

Los atacantes en las aplicaciones Web sacan ventaja de las organizaciones y de los usuarios de sus contenidos, tales como multimedia, música, entretenimiento, información de todo tipo; esto dado que hay un valor comercial en esa información. Cuando los atacantes logran vulnerar estos sitios, obtienen ciertos beneficios

y es una tarea que cada vez mejoran más estos atacantes, una de las técnicas más utilizada con buenos resultados, es la inyección SQL[10].

Uno de los ataques más conocidos es la inyección de código SQL, que está definido como "Un método de infiltración de código intruso, que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas, para realizar consultas a una base de datos" [11]. Lo que se interpreta como acciones en las que se van introduciendo instrucciones de código SQL en campos de texto donde se validan datos que proporciona el usuario, para tener acceso a la información contenida en la base de datos mediante la aplicación, que es la intermediaria entre los usuarios y la base de datos.

El autor Carlos Tori en su libro Hacking Ético, menciona que el descuido que permite inyectar código SQL (Structured Query Language) va a perdurar siempre por error humano y es tan interesante que se necesita un estudio especializado. En la actualidad, un gran número de sitios y aplicaciones web interactúan con bases de datos, ya que a través de éstas, se maneja información de diferentes niveles críticos, que deben ser almacenados, consultados o modificados, es decir, gestionados de algún modo. Esta información, sensible o no, es accedida a través de sentencias SQL, insertadas (embebidas) desde el mismo código fuente de la página o scripts incluidos [12].

La interacción ocurre más o menos de este modo:

- Cualquier usuario introduce datos en un formulario o hace clic en un link del tipo http://sitioweb/producto.asp?id=25.
- El sitio web podría estar programado en .asp o .php (entre otros lenguajes) y a su vez contendrá, en su código fuente, strings SQL (sentencias).
- Éstas, junto con los datos suministrados por el visitante, irán directamente a la base de datos para lograr un resultado a partir de lo que ideó e interpretó el programador o analista: consulta, almacenamiento, modificación, borrado, ejecución, etcétera.
- El resultado puede mostrarse al usuario o no, o bien puede dar un error de sistema.

Justin Clarke en su libro "SQL Injection Attacks and Defense", hace mención que en la búsqueda de inyección SQL el código de inyección SQL puede estar presente en cualquier aplicación front-end (lado del cliente), que acepte la entrada de datos de un sistema o usuario, y que luego se usa para obtener acceso a un servidor de base de datos. El centro es el entorno Web, ya que este es el escenario más común. En el entorno web, el explorador Web es un cliente que actúa como front-end (lado del cliente) solicitando datos al usuario y enviándolos al servidor remoto que creará consultas SQL utilizando los datos enviados. El objetivo principal en esta etapa es identificar anomalías en la respuesta del servidor y determinar si son generadas por una vulnerabilidad de inyección SQL [3].

Hay que pensar de esta manera: Alguien puede enseñar cómo agregar dos números, pero no es necesario (o práctico) cubrir todas las posibilidades; siempre que sepa cómo agregar dos números, puede aplicar ese conocimiento a cada escenario que implique la adición.

La inyección SQL es la misma. Se necesita entender los cómos y los porqués, por lo que el resto será simplemente una cuestión de práctica. Rara vez se tendrá acceso al código fuente de la aplicación, y por lo tanto, se necesita probar por deducción. Poseer una mentalidad analítica es muy importante para comprender y ejecutar un ataque. Se debe tener mucho cuidado al comprender las respuestas del servidor para tener una idea de lo que podría estar sucediendo en el lado del servidor. Probar por inferencia o deducción es más fácil de lo que se piensa. Se trata de enviar solicitudes al servidor y detectar anomalías en la respuesta. Es posible encontrar vulnerabilidades de inyección SQL, se trata de enviar valores aleatorios al servidor, pero una vez que se comprenda la lógica y los fundamentos del ataque, se convierte en un proceso sencillo y emocionante según Clarke [3].

En las debilidades en el uso de SQL, el principal problema que conduce a la inyección de código, y obviamente también a la inyección SQL, es la forma en que los lenguajes de programación (y consulta) funcionan inherentemente según lo menciona Galluccio et al. [13]. Dado que los comandos son solo cadenas de caracteres que se interpretan como código, y la entrada del usuario está hecha de texto, podríamos, en principio, insertar sintaxis de código dentro de la entrada del usuario. Si no se valida correctamente y simplemente se acepta sin que se aplique ningún control, este código inyectado podría resultar en la ejecución de comandos arbitrarios que han sido insertados manualmente por un usuario malicioso.

Esto se debe a que un lector de cadenas ingenuo no hace ninguna distinción entre texto y código ya que se trata esencialmente de datos binarios codificados como texto, lo mismo se hace desde el punto de vista de un programa de computadora o una aplicación. Por lo general, para inyectar instrucciones específicas u objetos de código, se utilizan caracteres específicos para engañar al analizador en el componente de software encargado de leer la entrada de texto, para que interprete el código insertado como comandos no

deseados. Tradicionalmente, la forma más trivial de inyectar código es insertando el carácter de terminación de línea, el punto y coma en la mayoría de los lenguajes de programación, de modo que, además de la operación prevista, la nueva se considera como una instrucción completamente diferente. Se pueden usar otros caracteres para manipular el comportamiento de la aplicación, como el separador de comentarios, que se utiliza para excluir por completo partes del código siguiendo las instrucciones [13].

En referencia a instrucciones que conllevan a un ataque, Gabriel Sánchez Cano autor del libro Seguridad Cibernética [14] define a una inyección de código como la introducción deliberada de éste a una aplicación vulnerable, para cambiar el curso de la implementación del programa, a menudo de forma maliciosa. Las fugas de seguridad de la inyección de código ocurren cuando una aplicación envía datos no validados a un interpretador de scripts. Frecuentemente, los piratas informáticos ingresan estos datos manualmente, aunque también se hace automáticamente.

Estos tipos de fugas de seguridad normalmente se encuentran en consultas (por ejemplo en SQL, LDAP, XPath y NoSQL) o en comandos que son parte del sistema operativo (OS), analizadores XML, encabezados SMTP y argumentos de programas. Muchos de estos tipos de fugas de seguridad son fáciles de detectar al examinar el código fuente o las pruebas. Los escáneres y los fuzzers (una técnica para ingresar datos de forma masiva fuzz que provoca un bloqueo) pueden ayudar a encontrar estas fugas.

Los riesgos asociados con el ataque de inyección SQL, muestran que la inyección SQL es dañina y los riesgos asociados con ella proporcionan motivación para que los atacantes ataquen la base de datos. Las principales consecuencias de estas vulnerabilidades son los ataques a las siguientes características mencionado en la investigación de Alwan y Younis [15]:

a) Autorización: Los datos críticos que se almacenan en una base de datos SQL vulnerable pueden ser alterados por un SQLIA exitoso.
b) Autenticación: Si no hay un control adecuado sobre los campos de entrada dentro de la página de autenticación, puede ser posible iniciar sesión en un sistema como un usuario normal sin conocer al usuario autenticado.
c) Confidencialmente: Por lo general, las bases de datos consisten en datos confidenciales como información personal, números de tarjetas de crédito y / o números sociales. Por lo tanto, la pérdida de confidencialidad es un problema terrible con la vulnerabilidad de inyección SQL.
d) Integridad: Por un SQLIA exitoso no solo un atacante lee información confidencial, sino que también, es posible cambiar o eliminar esta información privada.
e) Huellas dactilares de la base de datos: el atacante puede determinar el tipo de base de datos que se utiliza en el backend (lado del servidor) para que pueda utilizar ataques específicos de la base de datos que correspondan a debilidades en un sistema de gestión de bases de datos en particular; Elmasri y Navathe [16].

En los ataques de Inyección de SQL, Chicaiza et al. [17] clasifica dos tipos de categorías, aunque los ataques realmente varían de unos a otros, estas categorías son:

a) a) Datos en Exfiltración: Exfiltración de datos a través de la inyección de SQL es lo que ha contribuido a algunas de las violaciones de datos más grandes hasta la fecha. Los atacantes encuentran una vulnerabilidad que les permite a la lista de todas las tablas y volcar todas las cuentas de usuario, correos electrónicos y contraseñas.
b) Código de Inyección: No se ve esto muy a menudo, lo que se basan en algunas vulnerabilidades iniciales de pre-pruebas que se bloquean automáticamente a través de un Sitio Web Firewall por lo que es mucho más difícil de grabar.

Gómez [18] describe que un ataque por inyección de código SQL se produce cuando no se filtra de forma adecuada la información enviada por el usuario. Lo que un usuario malicioso podría incluir y ejecutar textos que representan nuevas sentencias SQL, que el servidor no debería aceptar. Este tipo de ataque es independiente del sistema de bases de datos, ya que solo depende únicamente de una inadecuada validación de los datos de entrada. Como consecuencia de estos ataques y, dependiendo de los privilegios del usuario de base de datos bajo el cual se ejecutan las consulta, se podría acceder no sólo a las tablas relacionales con la operación de la aplicación del servidor Web. También pueden propiciar la ejecución de comandos arbitrarios del sistema operativo del equipo del servidor Web.

Análisis metodológico del atacante por Inyección SQL

Es importante explicar cómo funciona la inyección SQL para tener las soluciones a este problema que se ha venido incrementado, puesto que los hackers no éticos (Hackers de sombrero negro) buscan algún punto débil para poderlo explotar y cometer el acto ilícito. Por ello, a continuación se explican algunos ejemplos usando las técnicas de inyección SQL. Existen cuatro tipos de

ataques de inyección SQL contra bases de datos Oracle:

1. Manipulación SQL;
2. Código de Inyección;
3. Función de llamada a Inyección;
4. Desbordamiento de memoria búfer (Buffer Overflows).

Las categorías de manipulación de SQL y código de Inyección son bien conocidas cómo los ataques que más comúnmente han sido documentados para todos los tipos de bases de datos (incluyendo SQL Server, MySQL, DB2 PostgreSQL, y Oracle) [7]. La manipulación SQL típicamente se involucra en sentencias SQL a través de asignación de operaciones llamadas unión, o alterando la cláusula WHERE de la condición de una consulta para retornar un resultado diferente. Muchos ataques de inyección SQL son de este tipo. El ataque más conocido es el de modificar la cláusula WHERE de alguna sentencia de autentificación de usuario, por lo tanto, la cláusula siempre resulta en verdadero (TRUE). La inyección del código resulta cuando un atacante inserta nuevas sentencias en la consulta o comandos de base de datos en la misma instrucción.

El ataque de inyección de código clásico trata de agregar un comando EXECUTE a una instrucción SQL vulnerable. La Inyección de código sólo funciona cuando se admiten múltiples sentencias SQL por petición de la base de datos. Los gestores SQL Server y PostgreSQL tienen esta capacidad, y a veces es posible inyectar múltiples sentencias en la consulta con Oracle. Las vulnerabilidades de código de inyección de Oracle implican la ejecución dinámica de SQL en PL/SQL(Programming Language SQL).

Las dos últimas categorías son ataques más específicos contra bases de datos Oracle y no son bien conocidas o documentadas. En la mayoría de las auditorías que se llevan a cabo se han encontrado aplicaciones vulnerables a estos dos tipos de ataques. La función llamada inyección es la inserción de las funciones de base de datos Oracle o funciones personalizadas en una instrucción de una consulta vulnerable; estas llamadas de función pueden utilizarse para realizar llamadas de sistemas operativos o manipular información en la base de datos. La inyección SQL de desbordamiento de búfer es un subconjunto de una función llamada inyección. En varios manejadores de bases de datos comerciales y código abierto (open-source), las vulnerabilidades existen en algunas funciones de las bases de datos que pueden resultar en un desbordamiento de búfer, por tanto, los parches existentes están disponibles para la gran mayoría de estas vulnerabilidades, pero otras bases de datos de producción permanecen sin parches.

Una aplicación es vulnerable a una inyección SQL por la razón de que la cadena de entrada de un usuario no está debidamente validada y pasa a una instrucción de una consulta SQL dinámica, sin ninguna validación alguna. La cadena de entrada es usualmente pasada directamente a la consulta. Sin embargo, la entrada de usuario puede ser almacenada en la base de datos, y más tarde pasada a una instrucción SQL dinámica, contemplada como una inyección de SQL de segundo orden. En muchas aplicaciones web es común escribir información en las bases de datos o almacenarla cuando se utilizan otros medios entre páginas web. Este tipo de ataque indirecto es mucho más complejo, y frecuentemente requiere un profundo conocimiento de la aplicación.

Las sentencias o instrucciones usando variables enlazadas están generalmente protegidas de la inyección SQL, haciéndolas no vulnerables, tal como las bases de datos de Oracle, por lo que usarán el valor de las variables de enlace y no se podrá interpretar los contenidos de la variable de ninguna manera. PL/SQL y JDBC (java database connectivity) permiten enlazar variables. Dichas variables enlazadas serán extensivamente usadas por razones de seguridad y rendimiento [7].

Manipulación SQL

El tipo más común de ataque de inyección SQL es la manipulación de comando del lenguaje SQL. El atacante intenta modificar la instrucción o consulta existente, añadiendo elementos a la cláusula WHERE o extiende la instrucción de la consulta asignando operadores de Unión (UNION), intersección (INTERSECT),o menos (MINUS). Hay otras posibles variaciones, pero estas son las más significativas. La clásica manipulación SQL bien conocida se realiza durante la autenticación de inicio de sesión de acceso (Login). En una aplicación web simple, se puede comprobar la autentificación del usuario por la ejecución de la siguiente consulta, y se puede ser comprobada para ver si las filas o registros fueron retornadas.

```
SELECT * FROM usuarios WHERE usuario='juan' and PASSWORD = 'mipassword'
```

El atacante intenta manipular la instrucción SQL para ejecutar la consulta como se muestra:

```
SELECT * FROM usuarios WHERE usuario = 'juan' and PASSWORD = 'mipassword' or 'a' = 'a'
```

IPSUMTEC
ISSN: 2594 - 2905

Basado en precedencias de los operadores, la cláusula *WHERE* es válida para cada fila y el atacante gana acceso a la aplicación. La asignación del operador *UNION* es frecuentemente usado en ataques de inyección SQL. El objetivo es manipular la instrucción en regresar filas de otra tabla. Un formulario Web puede ejecutar la siguiente consulta para devolver una lista de productos disponibles.

```
SELECT producto_nom FROM productos  WHERE producto_nom like '%Sillas%'
```

El atacante intenta manipular la sentencia para ejecutar la consulta como se muestra:

```
SELECT producto_nom FROM productos WHERE producto_nom like '%Sillas' UNION SELECT usuario FROM dba_usuarios WHERE usuario like '%'
```

La lista devuelta al formulario Web incluirá todos los productos seleccionados, pero también todos los usuarios de la base de datos.

Inyección de Código

Los ataques de código de inyección intentan agregar comandos o sentencias adicionales a la instrucción SQL existente. Este tipo de ataques es utilizado con frecuencia contra las aplicaciones Microsoft SQL Server, pero rara vez se trabaja con una base de datos de Oracle.

La instrucción *EXECUTE* en SQL Server es un blanco frecuente de ataques de inyección, no hay ninguna instrucción correspondiente en Oracle. En PL/SQL y Java, Oracle no soporta múltiples sentencias SQL por petición de la base de datos. Por lo tanto, el siguiente ataque de inyección común no funcionará contra una base de datos Oracle a través de una aplicación Java o PL/SQL. La siguiente instrucción dará lugar a un error.

```
SELECT * FROM usuarios WHERE usuario='juan' and PASSWORD = 'mipassword'; DELETE FROM usuarios WHERE usuario = 'admin';
```

Sin embargo, algunos lenguajes de programación o APIs pueden permitir múltiples sentencias SQL para ser ejecutadas. Aplicaciones Java y PL/SQL dinámicamente pueden ejecutar bloques anónimos PL/SQL, los cuáles son vulnerables a la inyección de código. El siguiente es un ejemplo de un bloque PL/SQL ejecutado en una aplicación Web.

```
BEGIN ENCRYPT PASSWORD('juan', 'mipassword'); END;
```

El bloque PL/SQL del ejemplo anterior ejecuta una aplicación de un procedimiento almacenado de otra aplicación que codifica (encripta) y guarda la contraseña del usuario. Un atacante intentará manipular el bloque PL/SQL a ejecutarlo de la siguiente manera:

```
BEGIN ENCRYPT PASSWORD('juan', 'mipassword'); DELETE FROM usuarios WHERE upper(usuario) = upper('admin'); END;
```

Función de una Llamada a Inyección

La función de una llamada a inyección es la inserción de funciones de base de datos Oracle o funciones personalizadas en una instrucción SQL vulnerable. Estas llamadas de función pueden utilizarse para realizar llamadas de un sistema operativo o manipular datos en la base de datos. En el gestor de base de datos Oracle permite funciones o funciones en paquetes para ser ejecutadas como parte de una instrucción SQL. Oracle provee muchas funciones en varios paquetes de base de datos estándar, aunque sólo una fracción de estas funciones pueden utlizarse en un ataque de inyección SQL. Algunas de estas funciones realizan una comunicación de red, la cuál puede ser explotada para el beneficio de un intruso no autorizado. Cualquier función personalizada o función que reside en una librería personalizada también se puede ejecutar en una instrucción SQL.

Las funciones ejecutadas como parte de una instrucción *SELECT* de SQL no pueden realizar cambios en la base de datos a menos que la función está marcada como "*PRAGMA TRANSACTION*" (Transacción Autónoma).

Muy pocas de las funciones estándar de Oracle se ejecutan como transacciones autónomas. Las funciones ejecutadas en *INSERT*, *UPDATE* o *DELETE* están disponibles para modificar datos en la base de datos. Al usar las funciones estándares de Oracle, un atacante puede enviar información de la base de datos a un equipo remoto o ejecutar otros ataques desde el servidor de base de datos.

Muchas aplicaciones basadas en Oracle aprovechan las bibliotecas de las bases de datos, que pueden ser explotadas por un atacante. Estas librerías personalizadas pueden incluir funciones para cambiar las contraseñas o realizar otras transacciones de aplicaciones vulnerables. El problema con la inyección de una llamada de función es que cualquier instrucción SQL generada dinámicamente es vulnerable, incluso las más sencillas instrucciones de SQL se pueden explotar con eficacia. El siguiente ejemplo demuestra que incluso la más simple sentencia SQL puede ser vulnerable. Los desarrolladores de aplicaciones a veces utilizarán las funciones de las bases de datos en lugar de códigos nativos (por ejemplo, Java) para realizar

IPSUMTEC
ISSN: 2594 - 2905

tareas comunes. No existe equivalencia directa de la función de traducción (*TRANSLATE*) de la base de datos en Java, así que el programador decide utilizar una instrucción SQL.

```
SELECT TRANSLATE('user input',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
', '0123456789') FROM dual;
```

Esta instrucción SQL no es vulnerable a otros tipos de ataques de inyección, pero es fácilmente manipulable a través de una función de ataque de inyección. El atacante intenta manipular la instrucción SQL para ejecutarla como a continuación se muestra:

```
SELECT TRANSLATE('' ||
UTL_HTTP.REQUEST('http://192.168.1.1/') || '',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
', '0123456789') FROM dual;
```

La instrucción SQL cambiante solicitará una página desde un servidor Web. El atacante podría manipular la cadena y el URL para incluir otras funciones con el fin de recuperar información útil de la base de datos del servidor y enviarla al servidor web en la URL. Desde que el servidor de base de datos Oracle pueda ser vulnerable es más probable que esté detrás de una aplicación de cortafuegos (*Firewall*), también podría ser utilizado para atacar a otros servidores en una red interna. Las funciones personalizadas y funciones de librería personalizadas también pueden ser ejecutadas. Un ejemplo sería que una aplicación personalizada tiene la función *ADDUSER* en la librería personalizada *MYAPPADMIN*.

El desarrollador marcaría la función como "Transacción Autónoma" (*PRAGMA TRANSACTION*), así que podría ser ejecutada bajo cualquier circunstancia especial en la que la aplicación podría encontrar. Desde que está marcada la "Transacción Autónoma", puede escribir sobre la base de datos incluso en una instrucción *SELECT*. Corriendo esta sentencia SQL, el atacante está habilitado para crear nuevas aplicaciones de usuario.

```
SELECT TRANSLATE('' ||
myappadmin.adduser('admin', 'newpass') || '',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
', '0123456789') FROM dual;
```

Desbordamiento de Búfer (Buffer Overflows)

Un número de funciones estándar del gestor de base de datos Oracle son susceptibles a desbordamientos de búfer, que pueden ser infiltradas a través de un ataque de inyección SQL en una base de datos que no tiene parches. El desbordamiento de búfer conocido existe en las bibliotecas de bases de datos estándares, así como en las funciones estándares tales como TZ_OFFSET, TO_TIMESTAMPTZ, BFILENAME, FROM_TZ, NUMTOYMINTERVAL, y NUMTODSINTERVAL. El ataque de desbordamiento de búfer mediante TZ_OFFSET, TO_TIMESTAMP_TZ, BFILENAME, FROM_TZ, NUMTOYMINTERVAL, o NUMTODSINTERVAL es ejecutado usando la función de métodos de inyección descritos previamente.

Explotando el desbordamiento de búfer a través del ataque de inyección SQL, el acceso remoto no autorizado al sistema operativo puede ser logrado y la información adicional está ampliamente disponible en la ejecución, así como la prevención de ataques de desbordamiento de búfer. Además, algunas aplicaciones y servidores Web no manipulan la pérdida de una conexión de base de datos debido a un desbordamiento de búfer. Usualmente, el proceso de una sesión Web durará hasta que la conexión con el usuario es terminada, entre tanto potencialmente se está fraguando un efectivo ataque de denegación de servicio.

Entorno de Explotación del Ataque

Las condiciones necesarias para que se pueda dar en una aplicación web la vulnerabilidad de inyección de comandos SQL son los siguientes [2]:

- *Fallo en la comprobación de parámetros de entrada:* Se considera parámetro de entrada cualquier valor que provenga desde un usuario. En este entorno se debe asumir “un ataque inteligente”, por lo que cualquiera de estos parámetros pueden ser enviados con “malicia”. Se debe asumir también que cualquier medida de protección implementada en el usuario puede fallar. Como ejemplos de parámetros a considerar donde se suelen dar estos fallos, tendríamos:
  - Campos de formularios: Utilizados en métodos de llamadas de la variable de ambiente *POST*.
  - Campos de llamada de la variable de ambiente *GET* pasados por variables.
  - Parámetros de llamadas a funciones JavaScript.
  - Valores en cabeceras http a través del URL.
  - Datos almacenados en *cookies*.
- *Utilización de parámetros en la construcción de llamadas a bases de datos:* El problema no reside en la utilización de los parámetros en las sentencias SQL, sino en la utilización de aquellos parámetros que no han sido comprobados correctamente.
- *Construcción no fiable de sentencias:* Existen diversas formas de crear una sentencia SQL

IPSUMTEC
ISSN: 2594 - 2905

dinámica dentro de una aplicación web, que dependen del lenguaje utilizado. A la hora de crear una sentencia SQL dinámica dentro de una aplicación web, existen, dependiendo del lenguaje utilizado, diversas maneras. El problema se genera con la utilización de una estructura de construcción de instrucciones SQL basada en la concatenación de cadenas de caracteres, es decir, el programador toma la sentencia como una cadena alfanumérica a la que va concatenando el valor de los parámetros recopilados, lo que implica que tanto los comandos como los parámetros tengan el mismo nivel dentro de la cadena. Cuando se acaba de construir el comando no se puede diferenciar qué parte ha sido introducida por el programador y qué parte procede de los parámetros.

Metodología propuesta para la prevención de ataques
En esta sección se presenta la metodología que sirve como guía, permitiendo seguir el orden necesario en las posibles pruebas a realizar. Antes que todo, se mencionan dos de las más importantes, entre las que se encuentran:

- OWASP (Open Web Application Security Project), sobre todo a nivel Web.
- A nivel general OSSTMM que también es muy importante, ya que abarca todos los campos.

Una metodología de seguridad consiste en la ejecución de determinados pasos a seguir, con el fin de determinar la mayor cantidad de amenazas que puedan afectar a una organización, y evaluar cuáles pueden ser las vulnerabilidades con su respectivo nivel de riesgo y sus posibles efectos en las diferentes áreas de la organización.

En la actualidad existen gran cantidad de estándares, normas y procedimientos tanto públicos como privados. Estos permiten generar una fuerte defensa contra ataques informáticos. Una detección temprana de un intrusión o ataque, y así como medidas de contingencias de rápido desarrollo, permiten que la organización sea lo menos vulnerable posible, ante cualquier ataque [6].

Durante un proceso de pruebas de seguridad en una aplicación web, se puede tener la falsa concepción de que la revisión automatizada es eficiente y efectiva, es así como muchos usuarios (testers [probadores] y pentesters [probador de técnicas de penetración]) especialmente los menos experimentados suelen anteponer y priorizar los resultados devueltos por aplicaciones que escanean (scanners) las vulnerabilidades en aplicaciones web sobre la inspección manual. Con esto no se pretende de ningún modo desestimar la labor que desempeñan dichas herramientas como scanners y frameworks (un esquema {un esqueleto o un patrón} para el desarrollo y/o la implementación de una aplicación) de penetración, solamente que es necesario comprender que dichas herramientas tienen sus propias limitaciones y que no se puede esperar que una utilidad que ha sido creada para aplicaciones genéricas funcione del mismo modo para aplicaciones con un alto nivel de personalización [8].

Dadas estas características, está claro que el uso de las herramientas no es suficiente para afrontar el reto que implica desplegar una aplicación Web segura, es necesario que la persona responsable y el equipo involucrado tenga presentes como mínimo las siguientes técnicas sobre el testing (prueba) de una aplicación web.

Defensa
La defensa no es demasiado complicada, consiste en asegurarse de que los datos de entrada son seguros. Como el lenguaje que se usa en estos casos suele ser PHP, es el que utilizará a través de este trabajo, pero los conceptos son los mismos. La forma de filtrar la entrada cambia según el tipo de datos, si es un número, simplemente hay que interpretarlo como tal y desechar el resto, por ejemplo, “65535 or 1=1” pasaría a ser “65535”. Esto se puede hacer con la siguiente instrucción:

```
$variable_segura = intval($variable_insegura);
O, si es un número con coma flotante:
$variable_segura = floatval($variable_insegura);
```

Si es un dato alfanumérico, solo hay que usar la función, que sanitiza la cadena. Lo que hace la función mysql_real_escape_string es escapar los caracteres, haciendo que no tengan ningún significado especial, por ejemplo, un ' significa el final de una variable alfanumérica, pero con un \ antes (escapado), no significa el final, es solo una comilla en la variable.

```
mysql_real_escape_string(cadena)
$variable_segura=mysql_real_escape_string
($variable_insegura);
```

Si $variable_insegura es ' or '1'='1, la variable segura sería \' or \'1\'=\'1 , como se puede ver, todo lo que tiene algún significado en SQL tiene un \ antes, así que se convierte en un carácter normal. Eso es todo, tomando estas medidas de seguridad, el código está seguro contra las inyecciones SQL.

Protección contra la Inyección SQL
Ataques de inyección SQL pueden ser derrotados fácilmente con simples cambios de programación, sin embargo, los desarrolladores deben ser disciplinados lo

ISSN: 2594 - 2905

suficiente como para aplicar los métodos para cada procedimiento y función Web accesible. Cada sentencia SQL dinámica debe ser protegida. Una sola sentencia SQL sin protección puede resultar en componerse de la aplicación, datos o servidor de base de datos.

Enlace de Variables

La más poderosa protección contra ataques de inyección SQL es el uso de enlace (Bind) de variables. El uso de enlace de variables también mejorará el rendimiento de la aplicación. En la aplicación de estándares de codificación es necesario el uso de enlace de variables en todas las sentencias SQL. Ninguna declaración SQL debe ser creada mediante la concatenación de cadenas juntas y el paso de parámetros. Los enlaces de variables deben ser utilizadas para cada sentencia SQL sin importar dónde ni cuándo se ejecuta la instrucción SQL.

Un ataque de inyección SQL muy complejo posiblemente podría aprovechar una aplicación para almacenar una cadena (de caracteres) de ataque en la base de datos, que más tarde sería ejecutado por una instrucción SQL dinámica (contemplada como un ataque de segundo orden), tanto en el lenguaje PL/SQL como en la aplicación de interface de programación (API) JDBC demostraron cómo se debe utilizar con eficacia el enlace de variables para eliminar vulnerabilidades de inyección SQL, el uso de las mismas es simple, pero requiere por lo menos una línea más de código por variable. Puesto que una instrucción SQL típica puede estar utilizando los valores de 10 a 20, por lo que el esfuerzo de codificación adicional puede ser sustancial. Hay algunas raras ocasiones cuando un desarrollador debe crear dinámicamente una instrucción SQL.

Validación de Entrada

Cada parámetro de cadena pasada, debe validarse. Muchas aplicaciones Web utilizan campos ocultos y otras técnicas que también deben ser validadas. Si no se utiliza un enlace de variable, caracteres especiales de base de datos se deben retirar o aplicar la función de escape. Para bases de datos Oracle, el único elemento en cuestión es una comilla simple.

El método más sencillo es aplicar la función de escape a todas las comillas simples, Oracle interpreta comillas consecutivas como una comilla simple literal. El uso de enlace de variables y la función de escape de comillas simples no deberían hacerse de la misma cadena. Un enlace de variable almacenará la cadena exacta de entrada en la base de datos y la función de escape cualquier comilla simple resultará en comillas dobles siendo almacenadas en la base de datos.

La Función de Seguridad

Las funciones de base de datos estándares y personalizadas pueden explotar en ataques de inyección SQL. Muchas de estas funciones pueden utilizarse con eficacia en un ataque. Oracle se entrega con cientos de funciones estándar y por defecto puede tener concesiones al usuario. La aplicación puede tener funciones adicionales que realizan operaciones como cambiar contraseñas o creación de usuarios que podrían ser aprovechados. Todas las funciones que no sean absolutamente necesarias para la aplicación deben ser restringidas.

Mensajes de Error

Si un atacante no puede obtener el código fuente de una aplicación, mensajes de error se convierten en críticamente importantes para un ataque exitoso. Muchas aplicaciones Java no devuelven mensajes de error detallados, pero deben realizarse análisis y pruebas para determinar si la aplicación devuelve mensajes de error detallados. En lugar de devolver mensajes de error de base de datos detallada para el usuario, esta información debe ser escrita en un archivo de registro.

Puerto de Acceso PL/SQL (PL/SQL Gateway)

El puerto de acceso (Gateway) PL/SQL puede configurarse para mostrar diferentes niveles de mensajes de error. Cuanta más información devuelta en un mensaje de error, es más útil el mensaje a un atacante. Todas las aplicaciones de PL/SQL Gateway deben diseñarse para devolver una aplicación generando una página de error cuando se detecta el mismo en lugar de permitir que el puerto de entrada devuelva una aplicación de mensaje de error. Sin embargo, algunos errores como procedimientos no encontrados deben ser devueltos por la puerta de entrada. La configuración de ERROR_STYLE en el archivo de configuración "wdbsvr.app" determina el nivel de información que devuelve al usuario.

Puesto que estos tipos de errores son probablemente causados por un atacante más que los errores, en el procesamiento de las aplicaciones normales, solo una mínima o ninguna información deben ser devueltos. El parámetro ERROR_STYLE debe establecerse a "Servidor" en lugar de "Gateway" o "GatewayDebug". El parámetro ERROR_STYLE puede ser ajustado a nivel global o nivel de progenitor, así que ambas secciones del archivo de configuración deben revisarse.

Revisión de Código

Como su nombre lo indica, es una técnica que debe de ser aplicada por una persona o equipo que tenga conocimientos sobre la plataforma, framework y/o

lenguaje de programación que se emplea para el desarrollo de la aplicación, debe ser llevada a cabo durante el proceso de desarrollo y despliegue con el fin de encontrar cualquier tipo de vulnerabilidad en fases previas al despliegue de la aplicación en un entorno productivo.

La revisión "estática" del código (que es como le conoce a la inspección manual) se debe ser medida en función a los requerimientos de negocio, problemas intrínsecos relacionados con la plataforma o lenguaje de programación, la lista de verificación (Checklist) de elementos que deben medirse dependiendo de las necesidades organizacionales (en este punto la guía de desarrollo de OWASP [4], cuenta con listas bastante completas que pueden ser empleadas para verificar detalles concretos del desarrollo de una aplicación web).

Modelado de Amenazas y Riegos

Dentro de los requerimientos de seguridad se necesita tomar en consideración muchos "marcos" de amenazas posibles y el riesgo de que estas se puedan reproducir con éxito. El modelado de amenazas y riesgos en cualquier tipo de entorno, normalmente suele asignar un nivel de criticidad a cada una de las amenazas detectadas en un análisis previo, aunque está claro que no es posible medir todas las amenazas que pueden afectar a un sistema, tener una lista de estas amenazas y un nivel criticidad a cada una permite a los equipos de desarrollo y testing tener un "punto fijo" que debe tenerse en cuenta en el proceso de desarrollo para evitar que se puedan producir.

Este estudio normalmente da paso (o viene acompañado) de lo que se conoce como "Base de Conocimiento de Vulnerabilidades" el cual es simplemente un repositorio donde se plasman las amenazas y sus correspondientes niveles de riesgo, que evidentemente dependen de las actividades de la organización, nivel de disponibilidad de las aplicaciones y otros factores de negocio relacionados.

Ciclo de Vida de Desarrollo de Software con la Seguridad Como Objetivo Primordial

Aunque suene un poco repetitivo, es importante que en el ciclo de vida de desarrollo (SDLC por sus siglas en ingles Software Develop Life Cicle) se encuentre apoyado por expertos en seguridad y que los desarrolladores de la aplicación tengan conocimientos avanzados al respecto, además de esto, deben de ser conscientes de las implicaciones que pueden acarrear algunas decisiones técnicas a la hora de escribir código, dado que posiblemente estas decisiones, aunque en apariencia intenten resolver un problema, pueden representar un riesgo para la seguridad de la aplicación.

Controles y Medidas de Seguridad Comunes

Los controles y medidas de seguridad habituales deben tenerse en cuenta en todo momento, que se trata de las medidas habituales tales como la correcta instalación de parches y actualizaciones, Firewalls, AV, IDS, NIDS, etc. Con la entrada anterior y estos párrafos queda explicado de forma muy general, los conceptos básicos sobre el desarrollo de aplicaciones web y el rol que juegan los auditores y probadores (testers) en seguridad.

Observaciones

La parte débil de una aplicación Web para que se permita ataques de inyección SQL es la nula validación de las variables de entrada. Se debe reconocer que la seguridad es un componente fundamental de cualquier producto de software y deben incluirse buenas prácticas de programación en el desarrollo de software. Establecer la seguridad en un producto es mucho más fácil y más rentable que cualquier intento posterior a la eliminación o acción correctiva al recomponer los defectos descubiertos por los intrusos. Siendo esta una de las vulnerabilidades que genera mayor impacto ya que permite el acceso a la información confidencial de una empresa, institución u organización. En los diferentes gestores de bases de datos, así como en los lenguajes de programación, se contemplan funciones y técnicas para protegerse de este tipo de ataques, aunque la responsabilidad siempre debe recaer en el desarrollador, quien debe verificar, corregir y validar adecuadamente la entrada de datos.

CONCLUSIONES

Con este trabajo podemos decir que quizá no es suficiente obtener una seguridad muy férrea puesto que los atacantes siempre están buscando nuevas formas de infiltración para cometer el delito computacional. Aunque en esta investigación no se presentó la técnica de Minería de Datos (Data Mining), que pudiera reflejar en la seguridad informática sobre un sistema de detectores de intrusos y análisis de Log, por lo que podría contribuir hacia otras áreas de seguridad. Se determinó que los Firewalls Semánticos de Aplicaciones Web y de Bases de datos son la mejor opción para mitigar inyecciones SQL, a nivel de aplicación. Aunque son tecnologías que han surgido recientemente, han tenido muy pocas publicaciones y muy poco conocimiento, lo que proporciona una gran novedad a la metodología de la seguridad de la inyección en SQL. Podemos decir que cada proceso de metodología, contribuye de forma decisiva, en encauzar la definición de un modelo para mitigar inyecciones SQL.

En la literatura revisada, se ha encontrado una descripción exhaustiva de las formas y métodos de

ataque de los delincuentes cibernéticos, sin embargo, es muy escasa la literatura que propone medios de mitigación y/o solución a estos problemas, de ahí la importancia de esta sencilla pero eficiente contribución.

BIBLIOGRAFÍA

[1] Adastra (2012), "Seguridad y Técnicas de Hacking", http://thehackerway.com/author/jdaanial/.
[2] Alonso Cebrián J. M. (2014), Guzmán Sacristán A., Laguna Durán P., Martín Bailón A., Ataques a BB. DD., SQL Injection, UOC (Universitat Oberta de Catalunya), España.
[3] Clarke J. (2012), SQL Injection Attacks and Defense, 2nd Edition, Syngress-Elsevier USA ISBN: 978-1-59749-963-7.
[4] Fundación Owasp,"Guía De Pruebas Owasp", licencia Creative Commons Attribution-ShareAlike Version 3.0, 2008.
[5] Gutiérrez Pedro (2012), Evita los ataques de inyección de SQL, Genbeta: http://www.genbetadev.com/seguridad-informatica/evita-los-ataques-de-inyeccion-de-sql.
[6] Gómez González I. C. (2012), Diseño de Metodología para Verificar la Seguridad en Aplicaciones Web Contra Inyecciones SQL. Universidad Militar Nueva Granada, Bogotá Colombia, Mayo 2012.
[7] Kost S. (2007), An Introduction to SQL Injection Attacks for Oracle Developers, Integrity trademark of Integrity Corporation. (Marzo 2007).
[8] Montero V. H.(2005), Técnicas de Penetration Testing, CYBSEC Security System, Buenos Aires Argentina, Septiembre 2005.
[9] Ribagorda A. (2001), Seguridad en Bases de Datos, fundación Dintel,(2001) ISBN: 84-931933-9-9.
[10] Delgado Caballero Gerson Geovanny (2013), Metodología de Pruebas de Inyección SQL Para Entornos Web, http://apuntesdeinvestigacion.bucaramanga.upbbga.edu.co/ Revista Digital Apuntes de Investigación, Universidad Pontificia Bolivariana – Seccional Bucaramanga Colombia Vol 7 – Septiembre 2013, ISSN 2248-7875.
[11] Tovar Valencia Orlando (2015), Inyección SQL, Tipos de Ataques y Prevención en ASP.NET-C#, http://repository.unipiloto.edu.co/handle/20.500.12277/2948, Repositorio Institucional Universidad Piloto de Colombia, Bogotá Colombia.
[12] Tori Carlos (2008), Haking Ético, 1era Edición, Mastroianni Impresiones Buenos Aires Argentina, Mayo del 2008, ISBN 978-987-05-4364-0.
[13] Galluccio Ettore, Caselli Edoardo, Lombari Gabriele (2020), SQL Injection Strategies: Practical techniques to secure old vulnerabilities against modern attacks, Packt Publishing Ltd, Birmingham ISBN 978-1-83921-564-3.
[14] Sánchez Cano Gabriel (2018), Seguridad Cibernética: Hackeo Ético y Programación defensiva, 1era Edición, Alfaomega Grupo Editor, S.A. de C.V. México ISBN: 978-607-538-294-4.
[15] Alwan Zainab S., Younis Manal F. (2017), Detection and Prevention of SQL Injection Attack: A Survey, International Journal of Computer Science and Mobile Computing, Vol.6 Issue.8, August-2017 ISSN 2320–088X.
[16] Elmasri R., Navathe S.B. (2011), Fundamentals of Database Systems, 6th edition, Addison-Wesley, United States of America, 2011.
[17] Chicaiza Giovanny, Ponce Luis, Velásquez Campos Gabriela (2020), Inyección de SQL, Caso de Estudio Owasp, Universidad de las Fuerzas Armadas ESPE Ecuador.https://www.academia.edu/11491488/inyecci%c3%93n_de_sql_caso_de_estudio_owasp.
[18] Gómez Vieites Álvaro (2011), Enciclopedia de la Seguridad Informática, 2da Edición Actualizada, AlfaOmega Ra-ma Editorial, Madrid España ISBN 978-84-9964-036-5.

ROLES DE CONTRIBUCIÓN

Rol de contribución	Autor (es)
Administración del proyecto Conceptualización	M.C. Juan Manuel Bernal Ontiveros
Redacción	Dr Francisco Zorrilla Briones
Supervisión	MSL Noé Ramón Rosales Morales
Metodología	Ing. Margarita Bailón Estrada
Redacción	M.C. Marisela Palacios Reyes