EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Tlaxiaco

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TLAXIACO

SEGURIDAD Y VIRTUALIZACIÓN

Práctica 4 - Inyección SQL

CARRERA:
INGENIERIA EN SISTEMAS COMPUTACIONALES

INTEGRANTES:
SANDRA YOLOTZIN REYES GARCÍA – 19620079
LUZ KARINA REYES LÓPEZ – 21620184
JERONIMA ROQUE CABALLERO – 21620206

DOCENTE
OSORIO SALINAS EDWARD

Tlaxiaco, Oax., Octubre de 2024.

INDICE

Contenido

TABLA DE ILUSTRACIONES

INTRODUCCIÓN

La inyección de código es una de las vulnerabilidades más comunes y peligrosas en el desarrollo de aplicaciones web, particularmente en la manipulación de bases de datos. Esta técnica consiste en insertar comandos maliciosos dentro de las consultas a bases de datos, aprovechándose de una falta de validación adecuada en los campos de entrada de datos. Las bases de datos vulnerables a este tipo de ataque permiten a un atacante ejecutar comandos no deseados, como la extracción de información sensible, la modificación o eliminación de registros, o incluso la toma de control total del sistema.

El objetivo de esta práctica es demostrar cómo una base de datos mal diseñada y mal protegida puede ser explotada mediante inyección de código, particularmente utilizando el ataque conocido como SQL Injection. Se analizará el proceso completo, desde la identificación de una vulnerabilidad hasta la ejecución de consultas maliciosas que comprometan la integridad y confidencialidad de los datos almacenados.

OBJETIVO

Crear una base de datos vulnerables a inyección de código y demostrar como se puede explotar esta vulnerabilidad.

DESARROLLO

1. Cree una base de datos con una tabla que contenga al menos 3 registros.
2. Cree una aplicación web que permita buscar un registro por su id, nombre o descripción.

- Esta aplicación debe ser vulnerable a inyección de código, esto significa que si el usuario ingresa un valor malicioso en el campo de búsqueda, la aplicación debe mostrar información que no debería ser accesible o permitir realizar acciones que no deberían ser posibles.

Creación de la base de datos

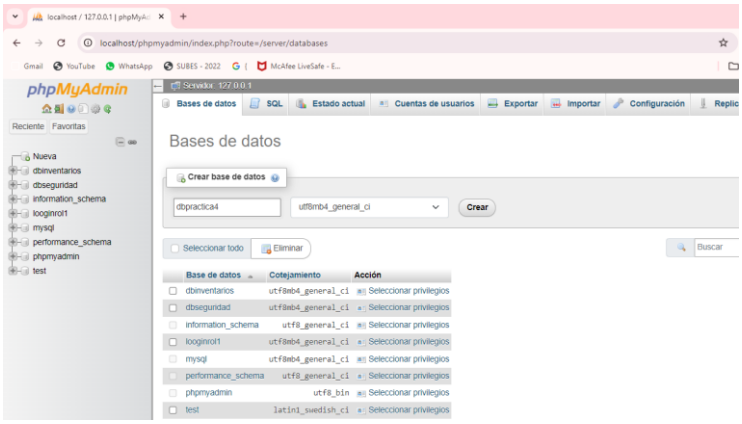

Ilustración 1

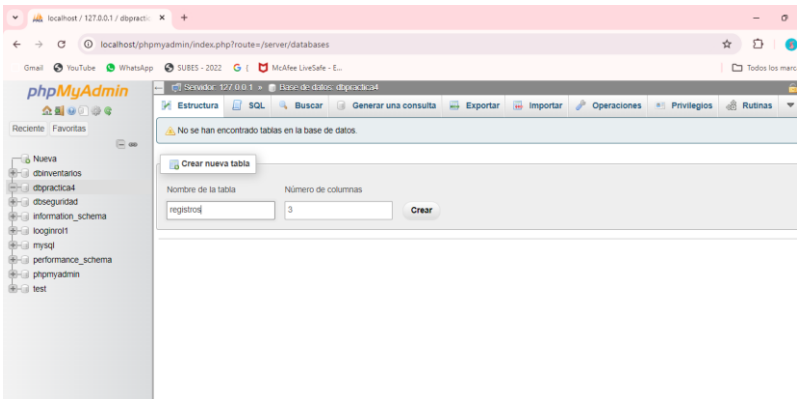

Ilustración 2

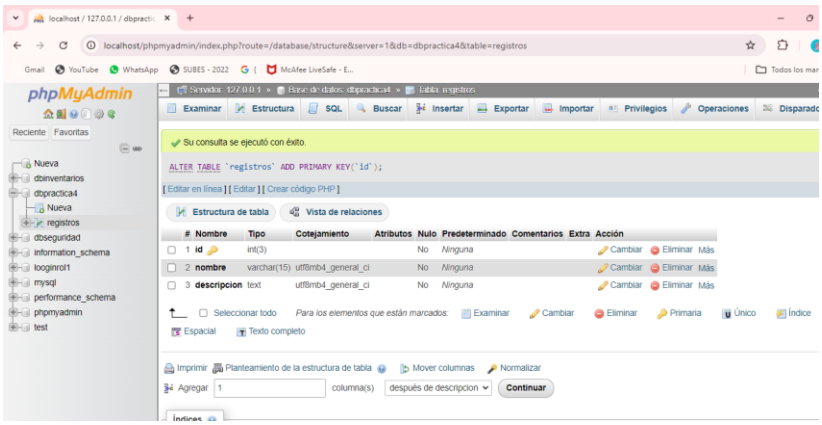

Ilustración 3

Se agregan datos a la tabla registros.

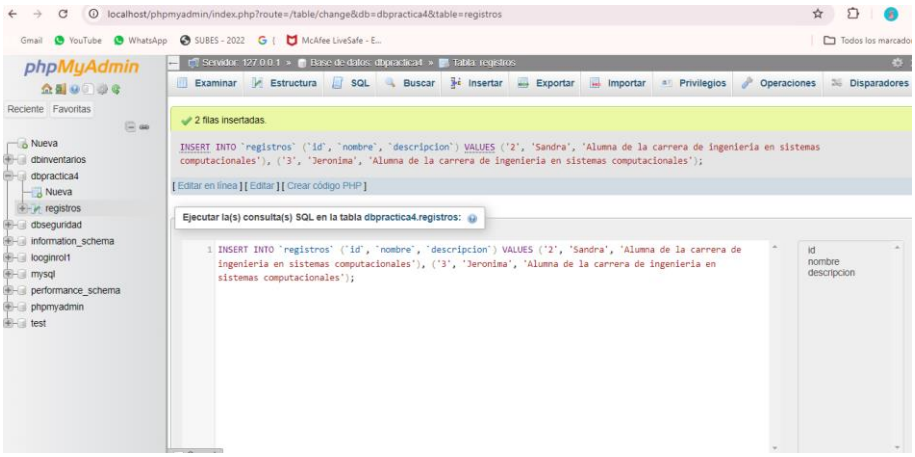

Ilustración 4

Archivo .php en donde se realizó la conexión

```php
<?php
// Datos de conexión a la base de datos
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "dbpractica4";

// Crear la conexión a la base de datos
$conn = new mysqli($servername, $username, $password, $dbname);

// Verificar la conexión
if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}
```

Ilustración 5

Esta parte del código se encarga de establecer una conexión con una base de datos MySQL. Utiliza la clase mysqli en PHP para conectarse a un servidor de base de datos con un conjunto de credenciales (nombre de servidor, nombre de usuario, contraseña) y una base de datos específica. Este paso es fundamental para que las consultas SQL puedan ejecutarse, ya que una vez que se establece la conexión, el script PHP puede interactuar con la base de datos para realizar operaciones como insertar, actualizar, eliminar o consultar datos.

En esta parte se muestra la vulnerabilidad

```php
// Consulta vulnerable a inyección SQL
$sql = "SELECT * FROM registros WHERE id LIKE '%$search%' OR nombre LIKE '%$search%' OR descripcion LIKE '%$search%'";

// Ejecutar la consulta
$result = $conn->query($sql);
```

Ilustración 6

```php
<body>
    <center><h1>Búsqueda de Registros</h1></center>
    <form action="conexion.php" method="GET">
        <label for="search">Buscar por ID, Nombre o Descripción:</label>
        <input type="text" id="search" name="search" required>
        <button type="submit" name="buscar" class="btn pulse-effect">Buscar</button>
    </form>
```

Ilustración 7

¿Por qué es vulnerable?

En este código:

- La variable $search es tomada directamente del usuario sin ninguna validación o filtrado.
- El valor ingresado por el usuario es **directamente concatenado** dentro de la consulta SQL.

- Esto permite que cualquier entrada maliciosa pueda modificar la consulta y ejecutar código SQL adicional.

Ejemplo de inyección maliciosa:

Si un atacante ingresa 1 OR '1'='1' en el campo de búsqueda, la consulta quedaría así:

SELECT * FROM registros WHERE id LIKE '%1 OR '1'='1'%' OR nombre LIKE '%1 OR '1'='1'%' OR descripcion LIKE '%1 OR '1'='1'%'

Este código simula cómo un atacante podría manipular el sistema para obtener información no autorizada de la base de datos.

Interfaz de la página web

Ilustración 8

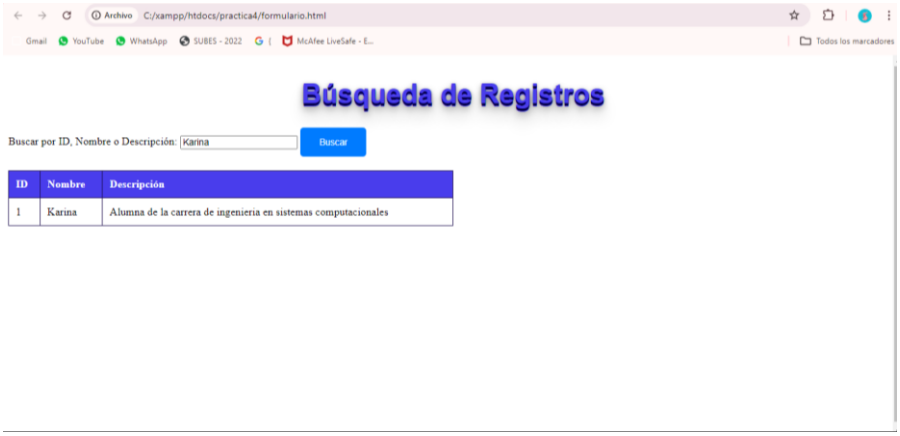

Ilustración 9

Inyección maliciosa

El atacante introduce una condición maliciosa en el campo de búsqueda, manipulando cómo se construye la consulta SQL. Lo que hace es aprovecharse de una vulnerabilidad en el sistema que permite concatenar código SQL no esperado.

Consecuencias de esta vulnerabilidad:

- Un atacante podría acceder a todos los datos almacenados en la tabla, independientemente de lo que esté buscando el sistema originalmente.
- Si la base de datos contiene información sensible, el atacante podría visualizar, modificar o eliminar datos.

Prevención:

- **Consultas preparadas**: Usar consultas preparadas con parámetros que previenen la inserción de código SQL malicioso.
- **Filtrado de entradas**: Validar y limpiar todas las entradas de usuario para evitar que se introduzcan caracteres maliciosos como ' o --.

Esta vulnerabilidad es un ejemplo claro de por qué es crucial proteger las aplicaciones web contra inyecciones SQL mediante buenas prácticas de programación y seguridad.

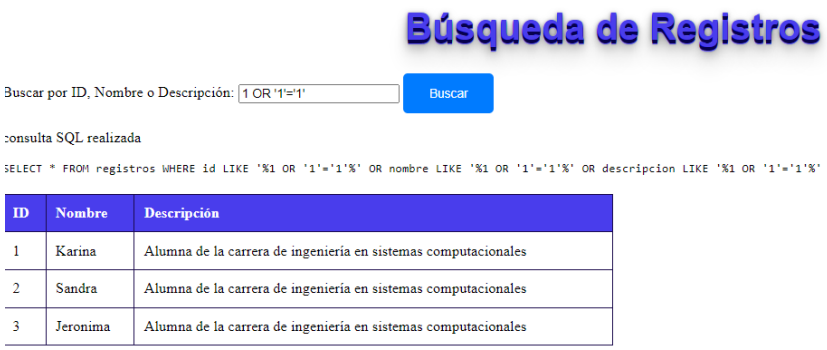

Ilustración 10

INVESTIGACIÓN

INYECCIÓN SQL

La inyección SQL (SQLi) sigue siendo una de las amenazas más frecuentes para las aplicaciones web, las bases de datos y la integridad de los datos en ciberseguridad. Este vector de ataque aprovecha las vulnerabilidades del software de una aplicación inyectando sentencias SQL maliciosas en los campos de entrada, lo que provoca el acceso no autorizado y la manipulación de la información de la base de datos.

Ilustración 11 INYECCIÓN SQL

¿Qué es una inyección SQL (SQLi)?

La inyección SQL (SQLi) es un tipo de ciberataque en el que un atacante inserta o manipula consultas SQL en campos de entrada de aplicaciones web para ejecutar comandos SQL maliciosos. Esto permite al atacante interferir en las consultas que una aplicación hace a su base de datos. Mediante la inyección SQL, los atacantes pueden obtener acceso no autorizado a los datos, manipular el contenido de la base de datos o ejecutar operaciones administrativas en ella.

Ilustración 12 ¿Qué es una inyección SQL (SQLi)?

Cómo funciona la inyección SQL
Punto de inyección: El atacante identifica un campo de entrada o parámetro en una aplicación web que interactúa con la base de datos, como formularios de inicio de sesión, cuadros de búsqueda o parámetros de URL.

Ilustración 13 Cómo funciona la inyección SQL

Entrada maliciosa: El atacante introduce sentencias SQL especialmente diseñadas en el campo de entrada, con el objetivo de manipular la consulta SQL ejecutada por la aplicación.

Ejecución de consultas: La aplicación procesa la entrada y construye una consulta SQL, que incluye el código malicioso. A continuación, la consulta se envía al servidor de la base de datos para su ejecución.

Respuesta de la base de datos: Dependiendo de la naturaleza del SQL inyectado, el atacante puede recuperar, alterar o borrar datos, ejecutar comandos del sistema o escalar privilegios dentro de la base de datos.

Ejemplo de inyección SQL
Consideremos un sencillo formulario de acceso a una aplicación web que utiliza la siguiente consulta SQL para autenticar a los usuarios:

SELECT * FROM usuarios WHERE nombre_usuario = 'user_input' AND contraseña = 'user_password';

Un atacante podría entrar OR '1'='1 como nombre de usuario y contraseña. La consulta SQL resultante tendría el siguiente aspecto:

SELECT * FROM usuarios WHERE nombre de usuario = '' O '1'='1' AND contraseña = '' OR '1'='1';

Ilustración 14 Ejemplo de inyección SQL

Esta consulta siempre devuelve verdadero porque "1=1" es siempre verdadero, eludiendo así la autenticación y otorgando potencialmente al atacante acceso no autorizado a la aplicación.

BLIND SQL INJECTION

El ataque de inyección de SQL ciego es un tipo específico de ataque de inyección de SQL en el que un atacante busca explotar las vulnerabilidades de una aplicación web relacionadas con el manejo inadecuado de las consultas SQL. A diferencia de los ataques de inyección de SQL normales, en un ataque de inyección de SQL ciego, el atacante no recibe directamente los resultados de sus consultas maliciosas. En cambio, debe deducir información sobre la base de datos y su contenido a través de observaciones indirectas, como cambios en el comportamiento de la aplicación o en el tiempo de respuesta.

Ilustración 15 BLIND SQL INJECTION

Hay dos tipos principales de inyección de SQL ciego:

- Inyección de SQL ciego basada en contenido: El atacante infiere información de la base de datos observando las diferencias en el contenido de la aplicación web como resultado de sus consultas maliciosas.
- Inyección de SQL ciego basada en tiempo: El atacante infiere información de la base de datos observando las diferencias en el tiempo de respuesta de la aplicación web como resultado de sus consultas maliciosas.

Ilustración 16 Hay dos tipos principales de inyección de SQL ciego:

Para proteger las aplicaciones web de los ataques de inyección de SQL ciego, los desarrolladores pueden tomar las siguientes medidas:

- Validar y filtrar adecuadamente todos los datos de entrada del usuario antes de procesarlos en consultas SQL para evitar la inclusión de consultas maliciosas o no permitidas.
- Utilizar consultas parametrizadas o procedimientos almacenados para separar los datos de entrada de las consultas SQL, reduciendo así el riesgo de inyección de SQL.
- Implementar mecanismos de autenticación y autorización sólidos para restringir el acceso no autorizado a los datos y sistemas de la aplicación.
- Aplicar el principio de mínimo privilegio para limitar los permisos y el acceso a los datos y sistemas de la aplicación.

Ilustración 17 Los ataques de inyección de SQL ciego

SQL INJECTION BASADA EN ERRORES

Las inyecciones de SQL basadas en errores aprovechan los mensajes de error de la base de datos para revelar información sensible. Los atacantes que están detrás de estos ataques envían intencionadamente consultas SQL malformadas, haciendo que la base de datos genere mensajes de error que contienen información valiosa.

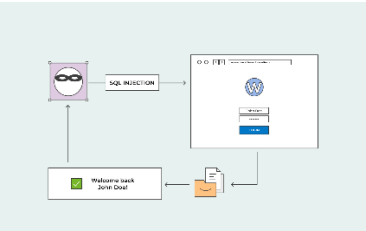

Ilustración 18 SQL INJECTION BASADA EN ERRORES

Analizando detenidamente estos mensajes, los ciberdelincuentes pueden conocer el funcionamiento interno del sistema e identificar sus (de los sistemas) posibles puntos débiles.

Cómo Funciona

Vulnerabilidad en la entrada de datos: El ataque generalmente se da en aplicaciones que no validan adecuadamente la entrada del usuario. Por ejemplo, si un formulario en una página web permite que un usuario introduzca un nombre de usuario sin sanitizar los datos, es un posible vector para SQL Injection.

Ilustración 19 Cómo Funciona

Inyección de comandos maliciosos: El atacante puede introducir una cadena de texto diseñada para interrumpir la consulta SQL. Por ejemplo, si la consulta SQL original es algo como:

SELECT * FROM usuarios WHERE username = 'usuario';

Un atacante podría modificar la entrada para que se convierta en:

' OR '1'='1

Esto podría transformar la consulta original en una que siempre sea verdadera, permitiendo al atacante acceder a todos los registros.

Generación de errores: Al inyectar código malicioso, el atacante intenta forzar a la base de datos a producir un error. Por ejemplo, puede usar una sintaxis SQL incorrecta que genere un mensaje de error. Este mensaje puede contener información invaluable sobre la estructura de la base de datos, como nombres de tablas, columnas, etc.

Ilustración 20 Atacante acceder a todos los registros

Recopilación de información: Los mensajes de error devueltos por la base de datos pueden revelar detalles que ayudan al atacante a planear ataques más sofisticados. Por ejemplo, un mensaje de error que indica que una tabla o columna específica no existe puede llevar a un atacante a deducir cómo están organizados los datos en la base de datos.

SQL INJECTION BASADA EN TIEMPO

SQL es una abreviatura para “lenguaje de consulta estructurado”, (o Structured Query Language en inglés), es un lenguaje diseñado para manipular y administrar datos en una base de datos; los hackers que hacen uso de SQL inyectan códigos maliciosos en elementos SQL existentes en la infraestructura de una página para engañar a los sistemas y hacer que estos les den acceso, al implementar esta

táctica logran interceptar datos o ubicar credenciales de administrador, lo cual les ayuda a obtener control total sobre un sistema o red.

Ilustración 21 SQL INJECTION BASADA EN TIEMPO

¿Cómo funcionan los ataques de inyección SQL?
Los ataques de inyección SQL son llevados a cabo a través de páginas web o entradas de aplicaciones, generalmente a través de formularios de entrada tales como cuadros de búsqueda, páginas de formulario o parámetros específicos de una URL.

Ilustración 22 ¿Cómo funcionan los ataques de inyección SQL?

Para llevar a cabo un ataque SQLI, los actores maliciosos primero deben encontrar vulnerabilidades en un sistema o red para luego proceder a inyectar cargas útiles maliciosas que pueden ejecutar acciones perniciosas y perjudiciales para el dueño del dominio, tal como otorgar acceso a los datos que están a salvaguardo en la página.

SQL INJECTION EN PROCEDIMIENTOS ALMACENADOS
Este capítulo explica cómo utilizar los Procedimientos Almacenados en SQL Server para evitar ataques SQL Injection, comparando las dos alternativas posibles de uso: utilizar procedimientos almacenados sin consultas parametrizadas, y utilizar procedimientos almacenados con consultas parametrizadas (opción recomendada). Para ello se incluye código ASP de ejemplo (llamadas ADODB.Connection, ADODB.Recordset, ADODB.Command) sobre el que se explican los ejemplos de ataques SQL Injection, razonando su funcionamiento y consecuencias.

Ilustración 23 SQL INJECTION EN PROCEDIMIENTOS ALMACENADOS

Quizás el principal método de protección contra SQL Injection sea la utilización de Procedimientos Almacenados para realizar todos los accesos a base de datos (al menos en el caso de SQL Server y SQL Injection). Sin embargo, no es suficiente con utilizar Procedimientos Almacenados, ya que en función de cómo utilicemos los Procedimientos Almacenados en nuestra aplicación conseguiremos protegernos de SQL Injection o seguiremos al descubierto ante ataques de SQL Injection.

¿Cómo puede ocurrir SQL Injection en procedimientos almacenados?
Concatenación de cadenas: Si los procedimientos almacenados construyen dinámicamente sentencias SQL utilizando concatenación de cadenas con entradas no validadas del usuario, pueden ser susceptibles a inyecciones SQL.

En este ejemplo, si un usuario malintencionado introduce un @username como user'; DROP TABLE Users; --, el procedimiento ejecutará comandos no deseados.

Ilustración 24 ¿Cómo puede ocurrir SQL Injection en procedimientos almacenados?

Uso de parámetros incorrectos: Aunque los procedimientos almacenados generalmente aceptan parámetros que pueden ayudar a prevenir inyecciones, si los parámetros se utilizan incorrectamente (por ejemplo, en comparación de texto sin parametrizar o filtros que dejen espacio para construcciones maliciosas), siguen existiendo riesgos.

SQL INJECTION EN ORM
SQL Injection es una técnica de ataque en la que un atacante inserta o "inyecta" código SQL malicioso en una consulta, con el objetivo de manipular la base de datos de una aplicación. Aunque el uso de un ORM (Object-Relational Mapping) normalmente proporciona una capa de abstracción que puede ayudar a prevenir este tipo de vulnerabilidades, no siempre es infalible. Aquí te explico cómo se relacionan ambos conceptos y algunas consideraciones a tener en cuenta.

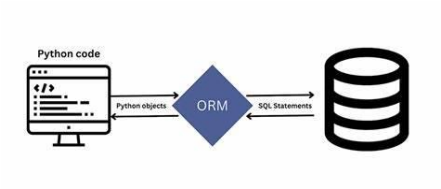

Ilustración 25 SQL INJECTION EN ORM

Una aplicación es vulnerable a estos tipos de ataque cuando:

- Los datos proporcionados por el usuario no son validados, filtrados ni sanitizados por la aplicación.
- Se invocan consultas dinámicas o no parametrizadas, sin codificar los parámetros de forma acorde al contexto.
- Se utilizan datos dañinos dentro de los parámetros de búsqueda en consultas Object-Relational Mapping (ORM), para extraer registros adicionales sensibles.
- Se utilizan datos dañinos directamente o se concatenan, de modo que el SQL o comando resultante contiene datos y estructuras con consultas dinámicas, comandos o procedimientos almacenados.

Prevención

Ilustración 26 Prevención

Para proteger tu aplicación de inyecciones SQL incluso al usar un ORM, considera las siguientes prácticas:

- Usa consultas parametrizadas: Siempre que sea posible, usa los métodos que permiten consultas parametrizadas o preparadas, donde los parámetros son enlazados y no concatenados directamente en la consulta.
- Validación de entrada: Realiza una validación sólida de la entrada del usuario para asegurarte de que cumpla con las expectativas (por ejemplo, tipo, longitud, formato).
- Evita construir consultas SQL dinámicas manualmente: Utiliza las funciones del ORM para construir las consultas en vez de hacerlo a mano.
- Actualiza tu ORM: Mantén tu ORM actualizado para asegurarte de que tienes las últimas correcciones de seguridad.
- Auditoría y pruebas de seguridad: Realiza auditorías de seguridad y pruebas de penetración en tu aplicación para identificar vulnerabilidades.

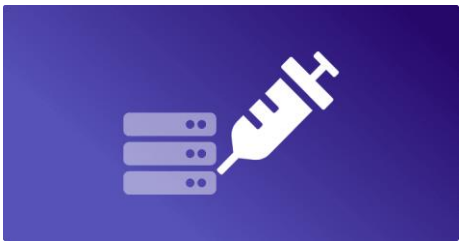

Ilustración 27 Inyecciones SQL incluso al usar un ORM

El uso adecuado de un ORM puede ayudar a mitigar los riesgos de SQL Injection, pero nunca debe ser considerado como una solución única y definitiva. Las mejores prácticas de codificación y seguridad son esenciales.

HERRAMIENTAS PARA DETECTAR Y PREVENIR SQL INJECTION

La detección y explotación de SQL Injection puede ser una tarea compleja y requiere conocimientos técnicos avanzados. Sin embargo, existen diversas herramientas diseñadas para facilitar este proceso tanto para los profesionales de la seguridad como para los atacantes malintencionados. En esta sección del artículo, exploraremos diferentes herramientas utilizadas para la detección y explotación de SQL Injection, desde escáneres automáticos hasta herramientas de inyección manual de código.

Ilustración 28 HERRAMIENTAS PARA DETECTAR Y PREVENIR SQL INJECTION

La detección y explotación de inyección SQL puede llevar mucho tiempo y requerir un análisis exhaustivo de la aplicación objetivo. Para agilizar este proceso, se han desarrollado herramientas especializadas que automatizan tareas como la detección de vulnerabilidades, la identificación de puntos de inyección y la explotación de SQL Injection. Estas herramientas pueden ser utilizadas tanto con fines legítimos, para evaluar la seguridad de una aplicación, como con fines maliciosos por parte de los atacantes. A continuación, se describen algunas de las herramientas más utilizadas en este contexto.

Herramientas de detección

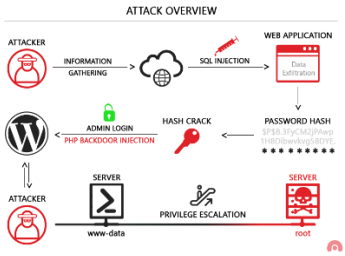

Ilustración 29 Herramientas de detección

- SQLMap: Es una herramienta de código abierto que automatiza el proceso de detección y explotación de vulnerabilidades de inyección SQL. Permite realizar pruebas exhaustivas en aplicaciones web.
- Burp Suite: Esta herramienta incluye un proxy interceptador y varios complementos para detectar vulnerabilidades, incluyendo SQLi. Su escáner de seguridad puede identificar automáticamente problemas de inyección.
- OWASP ZAP (Zed Attack Proxy): Es otra herramienta de código abierto que permite encontrar vulnerabilidades en aplicaciones web, incluyendo inyecciones SQL.
- Acunetix: Esta aplicación realiza escaneos automáticos de seguridad en aplicaciones web y puede detectar SQLi, así como otras vulnerabilidades.
- Nessus: Un escáner de vulnerabilidades que puede identificar una variedad de problemas de seguridad, incluyendo SQL Injection.

CONCLUSIÓN

Es fundamental que los desarrolladores implementen buenas prácticas de seguridad, tales como la utilización de consultas preparadas, la validación estricta de las entradas de usuario y el uso de mecanismos de autenticación y control de accesos robustos. Al abordar proactivamente estas vulnerabilidades, es posible mitigar los riesgos de ataques de inyección y proteger los sistemas y datos de forma eficaz. La comprensión y prevención de esta clase de amenazas no solo mejora la seguridad de las aplicaciones, sino que también garantiza la confianza de los usuarios en los sistemas que gestionan información crítica.

La importancia de proteger las aplicaciones ante estas vulnerabilidades radica en la necesidad de mantener la confidencialidad, integridad y disponibilidad de los datos, pilares esenciales en cualquier infraestructura. No solo se trata de evitar ataques, sino de generar confianza en los usuarios y en los sistemas que interactúan con ellos. Implementar medidas preventivas adecuadas, como el uso de consultas preparadas, la codificación segura y la revisión constante de la arquitectura de seguridad, es esencial para mitigar el riesgo de ataques y reducir la superficie de exposición a posibles vulnerabilidades.

REFERENCIAS

https://es.vectra.ai/topics/sql-injection

https://mineryreport.com/ciberseguridad/glosario/tipos-de-amenazas/termino/ataque-de-inyeccion-de-sql-ciego/#:~:text=El%20ataque%20de%20inyecci%C3%B3n%20de,inadecuado%20de%20las%20consultas%20SQL.

https://nordpass.com/es/blog/what-is-sql-injection/

https://easydmarc.com/blog/es/que-es-una-inyeccion-sql-sqli-y-como-prevenirla/

https://www.guillesql.es/Articulos/SQL_Injection_Procedimientos_Almacenados_SQL_Server_ASP_ADO_Ejemplos.html#:~:text=Este%20cap%C3%ADtulo%20explica%20c%C3%B3mo%20utilizar%20los

https://owasp.org/Top10/es/A03_2021-Injection/#:~:text=Algunas%20de%20las%20inyecciones%20m%C3%A1s%20comunes

https://ciberinseguro.com/sql-injection/#:~:text=Herramientas%20para%20la%20detecci%C3%B3n%20y%20explotaci%C3%B3n%20de%20SQL,de%20interceptaci%C3%B3n%20...%204%20Herramientas%20de%20fuzzing%20