TECNOLOGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

INTEGRANTES:

JULISSA MIGDALIA JOSE CRUZ

LUZ MARIA MENDOZA CORTES

ANGELES GONZÁLEZ MARTÍNEZ

ELISAHADAD MAYTE LEÓN DE JESÚS

DOCENTE:

ING.EDWARD OSORIO SALINAS

MATERIA:

SEGURIDAD Y VIRTUALIZACION

PRACTICA:

4 (INYECCION SQL)

GRUPO: 7US **SEMESTRE:** SEPTIMO

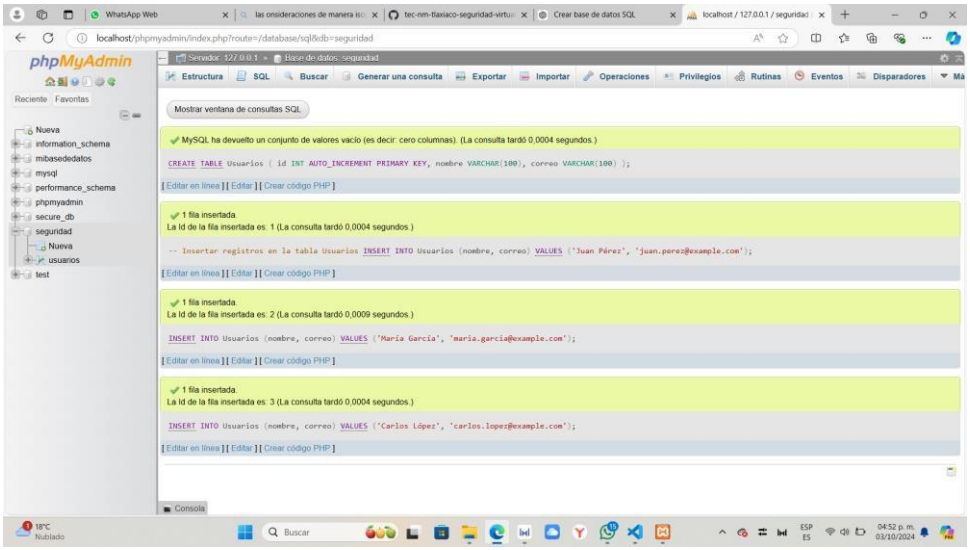
FECHA: 2 DE OCTUBRE DEL 2024

INTRODUCCION:

La inyección SQL es una técnica de ataque que permite a un atacante interferir en las consultas realizadas por una aplicación web a su base de datos. Este tipo de vulnerabilidad se produce cuando una aplicación permite que un usuario inserte datos en una consulta SQL sin la debida validación o saneamiento. Como resultado, un atacante puede manipular la consulta SQL para obtener acceso no autorizado a la información, modificar datos, ejecutar comandos en la base de datos o incluso comprometer el servidor en el que se aloja la aplicación. La inyección SQL se clasifica en varias categorías, siendo las más comunes: **Inyección SQL Clásica**: Consiste en añadir código SQL malicioso a una consulta existente, lo que permite al atacante ejecutar comandos arbitrarios. **Inyección SQL Basada en Tiempo**: Permite al atacante inferir información sobre la base de datos mediante el uso de retrasos en las respuestas del servidor. **Inyección SQL Ciega**: Se utiliza cuando el atacante no puede ver los resultados de sus consultas,pero puede hacer preguntas para obtener información a través de respuestas de verdadero/falso. La práctica de inyección SQL es fundamental para los profesionales de la seguridad, ya que les permite identificar y mitigar estas vulnerabilidades en aplicaciones web. Durante esta práctica, se explorarán técnicas de inyección SQL, se aprenderán métodos para prevenir estos ataques y se realizarán ejercicios prácticos para aplicar los conocimientos adquiridos. Es crucial que los desarrolladores comprendan cómo se producen estas inyeccionesy las medidas que deben implementarse para proteger sus aplicaciones, garantizando así la integridad y la confidencialidad de la información almacenada en las bases de datos.

1. **CREAR UNA BASE DE DATOS CON UNA TABLA QUE CONTENGA AL MENOS 3 REGISTROS.**

   Principalmente aquí se creó lo que es las tablas con los registros en el apartado de XAMMP, como se muestra a continuación en la siguiente imagen:



2. Crear 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.

Posteriormente realizamos lo que es la configuración para la conexión, donde se muestra lo que es el **(localhost)**, y en la parte de **$db='SEGURIDAD';** lo que nos servirá para establecer la configuración de la base de datos y tenga una conexión para que nos muestre correctamente los datos, como se muestra a continuación en la siguiente imagen:

```php
// Configuración de conexión
$host = 'localhost';
$db = 'seguridad';
$user = 'root';
$pass = '';
```

Después se realiza lo que es la '**Verificacion de conexión'** donde en esta parte verifica si la conexión a la base de datos ha fallado. Si es así, se muestra un mensaje de error y se detiene la ejecución del script.

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

Después realizamos lo que es la consulta de búsqueda donde, Si el usuario envía el formulario de búsqueda, se captura el término introducido en el campo de texto **($_GET['termino']).** Luego, se construye una consulta SQL que busca registros en la **tabla Usuarios** donde el **id, nombre** o **correo** coincidan (total o parcialmente) con el término ingresado:

```php
if (isset($_GET['buscar'])) {
    $termino = $_GET['termino'];

    // Consulta SQL vulnerable
    $sql = "SELECT * FROM Usuarios WHERE id = '$termino' OR nombre LIKE '%$termino%' OR correo LIKE '%$termino%'";

    echo "<p>Consulta ejecutada: $sql</p>"; // Para confirmar la consulta
    $resultado = $conn->query($sql);

    if (!$resultado) {
        die("Error en la consulta SQL: " . $conn->error);
    }
```

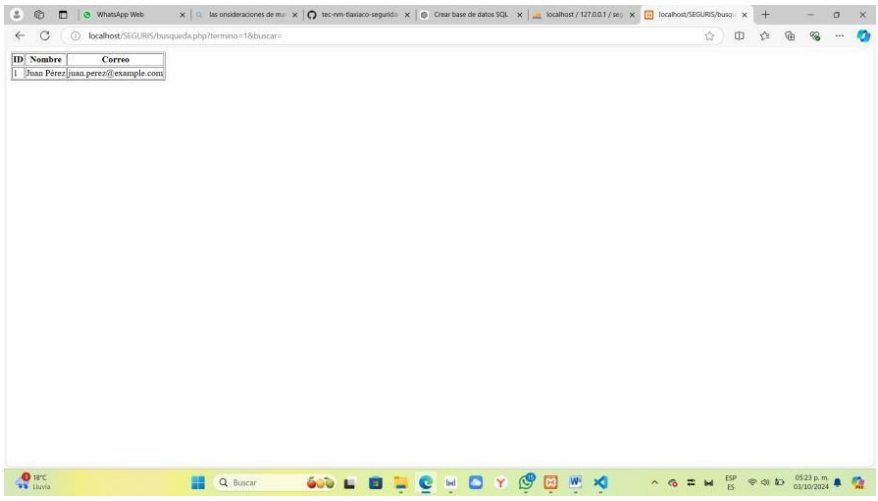
En esta parte declaramos los parámetros para mostrar los resultados, donde Si la consulta devuelve resultados, se muestra una tabla HTML con los nombres de las columnas (ID, Nombre y Correo). Si no hay resultados, se muestra un mensaje indicando que no se encontraron coincidencias.

```php
if ($resultado->num_rows > 0) {
    echo "<table border='1'><tr><th>ID</th><th>Nombre</th><th>Correo</th></tr>";
    while ($fila = $resultado->fetch_assoc()) {
        echo "<tr><td>" . $fila['id'] . "</td><td>" . $fila['nombre'] . "</td><td>" .
    }
    echo "</table>";
} else {
    echo "No se encontraron resultados.";
}
```

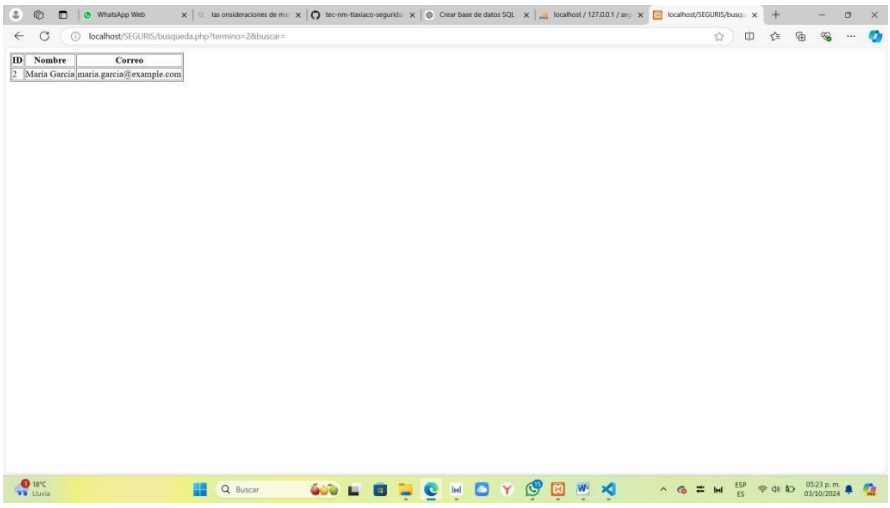
Finalmente, se define un formulario HTML que permite al usuario introducir el término de búsqueda. Al enviar el formulario, se envía un parámetro buscar junto con el valor del término, lo cual activa la consulta en el código PHP.

```
<form method="GET" action="busqueda.php">
    <label for="termino">Buscar por ID, Nombre o Correo:</label>
    <input type="text" name="termino" id="termino" required>
    <button type="submit" name="buscar">Buscar</button>
```

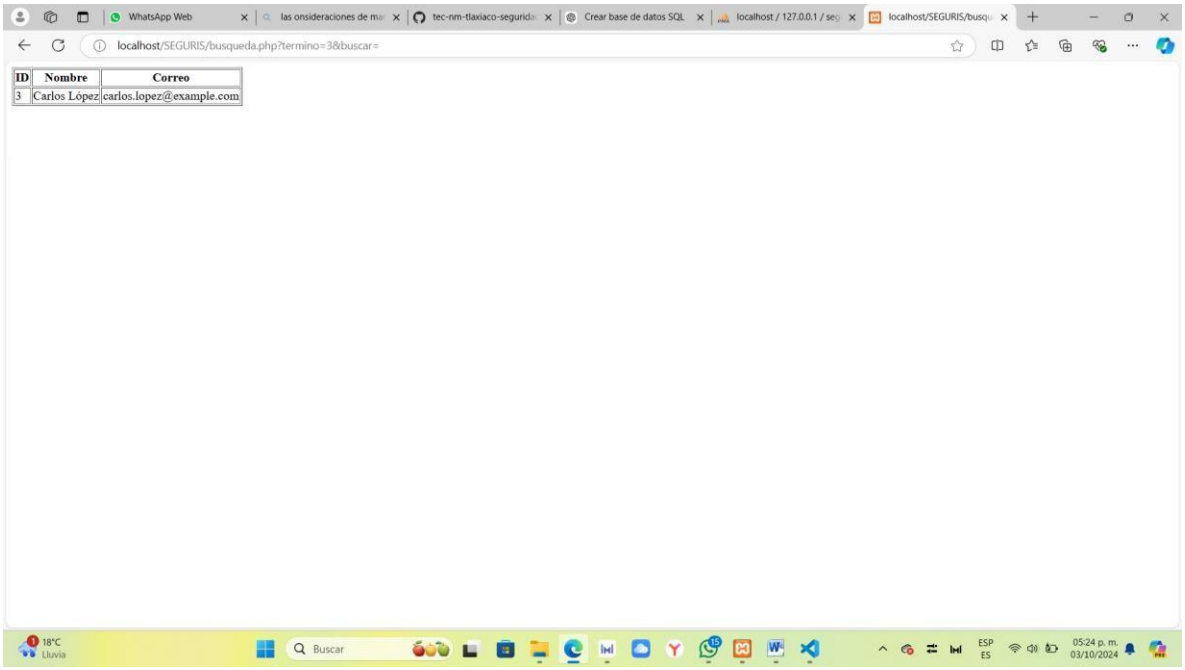
Posteriormente realizamos lo que es la prueba para ver el registro y nos muestra correctamente el usuario, como se puede observar a continuación en la siguiente imagen:



Posteriormente realizamos lo que es la prueba para el segundo usuario de Maria y vemos que está funcionando correctamente como se puede observar acontinuación en la siguiente imagen:



Posteriormente realizamos lo que es la otra prueba de donde colocamos lo que es el otro usuario de Carlos López y vemos que nos muestra correctamente, como se puede ver a continuación en la siguiente imagen:



3. Realizar pruebas de inyección de código en la aplicación web.

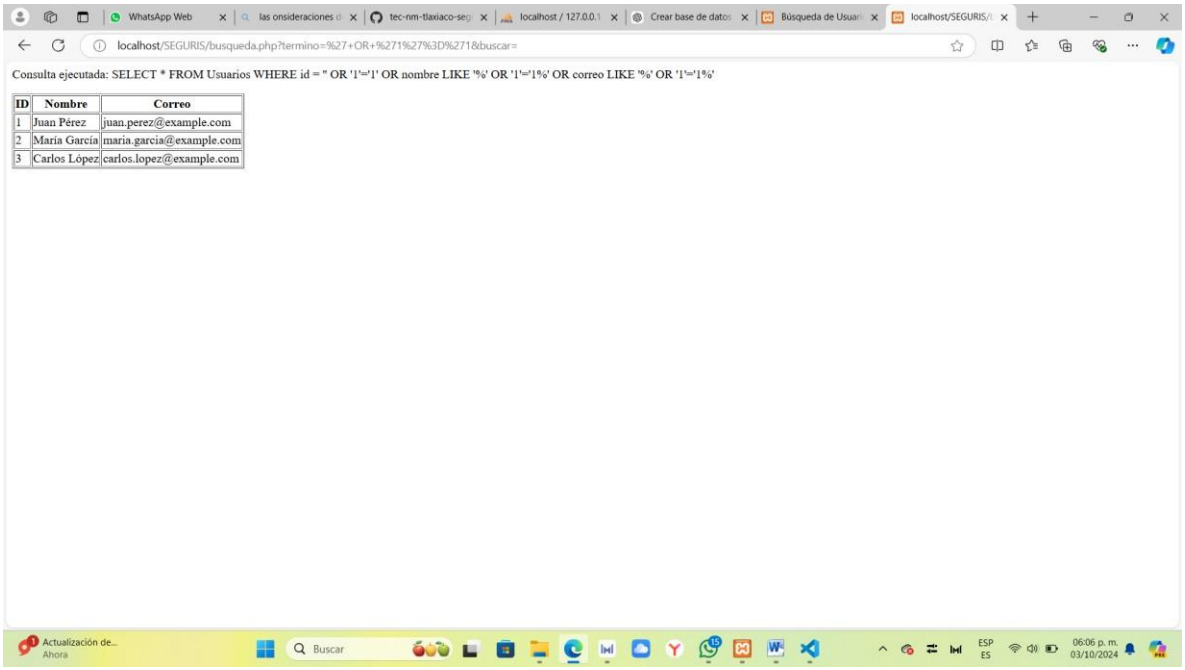
Posteriormente realizamos lo que es realizar en el código la parte de la inyección , con los parámetros de conexión:

```php
// Configuración de conexión
$host = 'localhost';
$db = 'seguridad';
$user = 'root';
$pass = '';

$conn = new mysqli($host, $user, $pass, $db);

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

Posteriormente una vez realizando cambios en el código , se guarda, y se muestra lo que es cuando se ejecuta la consulta y nos muestra correctamente lo que es la **“iyeccion en el SQL**”, como se puede apreciar en la siguiente imagen:



Investiga y describe los siguientes conceptos:

- Inyección de SQL
- Blind SQL Injection
- SQL Injection basada en errores
- SQL Injection basada en tiempo
- SQL Injection en procedimientos almacenados
- SQL Injection en ORM
- Herramientas para detectar y prevenir SQL Injection

INYECCIÓN DE SQL:

La **inyección de SQL** es una técnica de ataque que explota vulnerabilidades en las aplicaciones que interactúan con bases de datos. Se produce cuando unatacante inserta o "inyecta" sentencias SQL maliciosas en un campo de entrada, con el fin de manipular las consultas que realiza la aplicación hacia la base de datos. Este ataque puede permitir a los atacantes eludir mecanismos de autenticación, acceder a datos no autorizados, modificar o borrar información, e incluso ejecutar comandos en el servidor de la base de datos.

¿Cómo funciona la inyección de SQL?

El ataque se produce cuando una aplicación web permite que los usuarios ingresen datos en campos de formulario o URL sin validar o sanitizar adecuadamente la entrada. El atacante aprovecha esto para inyectar sentencias SQL maliciosas, que se mezclan con las consultas legítimas generadas por la aplicación. Esto da como resultado una manipulación directa de las consultas que interactúan con la base de datos.

Tipos de inyección de SQL

Existen varias variantes de ataques de inyección de SQL, cada una con diferentes enfoques y objetivos:

Inyección de SQL Clásica:

Es el tipo más básico de inyección de SQL. Los atacantes envían consultas SQL maliciosas a través de campos de entrada de usuario, intentando modificar o ejecutar consultas en la base de datos.

Inyección de SQL Ciega (Blind SQL Injection):

Se produce cuando la aplicación no devuelve errores de base de datos ni muestra mensajes evidentes de éxito o fracaso, pero aún permite deducir la existencia de una vulnerabilidad. Esto se realiza evaluando las respuestas que la aplicación proporciona mediante demoras (inyección basada en tiempo) o resultados booleanos (verdadero/falso).

Inyección de SQL Basada en Errores (Error-Based SQL Injection):

Este tipo de ataque se basa en los errores que devuelve la base de datos cuando una consulta SQL falla. Los mensajes de error pueden revelar información valiosa, como la estructura de la base de datos, los nombres de tablas y columnas, lo que facilita al atacante refinar su ataque.

Inyección de SQL Basada en Unión (Union-Based SQL Injection):

Aquí, el atacante utiliza la sentencia SQL UNION para combinar los resultados de una consulta legítima con los resultados de una consulta controlada por el atacante, lo que puede permitirle recuperar información adicional de la base de datos.

Inyección de SQL Basada en Tiempo (Time-Based SQL Injection):

Es una forma de inyección ciega en la que el atacante introduce funciones que causan retrasos (como SLEEP() en MySQL) y mide el tiempo de respuesta de la aplicación para deducir si una consulta SQL es verdadera o falsa. Esto es útil cuando la aplicación no devuelve errores ni mensajes explícitos.

Inyección de SQL en Procedimientos Almacenados:

Los procedimientos almacenados, que son rutinas SQL predefinidas en el servidor de base de datos, pueden ser vulnerables si aceptan entradas de usuario sin validarlas. Si un atacante logra manipular un procedimiento almacenado, puede ejecutarlo con parámetros maliciosos y afectar la integridad de la base de datos.

Inyección de SQL en ORMs (Object-Relational Mapping):

Aunque los ORMs están diseñados para evitar inyecciones de SQL, pueden ser vulnerables si los desarrolladores utilizan funciones que permiten consultas dinámicas o manipulan directamente cadenas de SQL con entradas no controladas.

Consecuencias de la inyección de SQL

Las consecuencias de una inyección de SQL exitosa pueden ser devastadoras:

Robo de datos: Los atacantes pueden obtener información confidencial, como credenciales de usuario, números de tarjetas de crédito, información personal, etc.

Alteración de datos: Los atacantes pueden modificar los registros de la base de datos, corrompiendo los datos o alterando información clave.

Eliminación de datos: En casos extremos, los atacantes pueden eliminar tablas enteras, causando pérdida de datos y posiblemente interrumpiendo el funcionamiento de la aplicación.

Acceso no autorizado: Los atacantes pueden acceder a cuentas protegidas o a áreas restringidas de la aplicación.

Ejecución remota de comandos: En algunos casos, la inyección de SQL puede permitir a los atacantes ejecutar comandos en el servidor subyacente.

Técnicas de mitigación de inyección de SQL

La prevención de la inyección de SQL implica la implementación de buenasprácticas de codificación y uso de herramientas que protejan contra este tipo de ataques. Algunas de las mejores prácticas incluyen:

Consultas preparadas (Prepared Statements):

Usar sentencias SQL preparadas es una de las formas más efectivas de prevenir la inyección de SQL. Las consultas preparadas separan los datos de los comandos SQL, lo que dificulta que las entradas maliciosas se ejecuten como parte de la consulta.

BLIND SQL INJECTION:

La **Blind SQL Injection** (inyección de SQL ciega) ocurre cuando la aplicación no devuelve mensajes de error explícitos que ayuden al atacante a conocer el resultado de su ataque. Sin embargo, el atacante puede seguir manipulando el comportamiento de la aplicación para deducir la información de la base de datos basándose en respuestas booleanas (verdadero/falso) o en el tiempo que tarda en responder.

Características de la Blind SQL Injection

En una inyección SQL estándar, los errores o la respuesta generada por la base de datos se utilizan para extraer información, pero en una Blind SQL Injection la aplicación web sigue funcionando normalmente y no proporciona información directa. Sin embargo, se puede deducir información valiosa de la base de datos basándose en cómo responde la aplicación web a diferentes entradas.

Por ejemplo, un atacante puede realizar consultas que producen diferentes respuestas según si una condición SQL es verdadera o falsa, lo que les permite extraer información de la base de datos sin que se muestren datos o mensajes de error evidentes.

Tipos de Blind SQL Injection

Boolean-Based Blind SQL Injection (Basada en Booleanos):

En este tipo de ataque, el atacante envía una consulta SQL que genera una condición booleana (verdadero o falso) y observa cómo responde la aplicación. Dependiendo de la respuesta, puede inferir si la consulta es correcta o no.

Por ejemplo:

sql

Copiar código

SELECT * FROM usuarios WHERE usuario = 'admin' AND 1=1; -- Devuelve verdadero

SELECT * FROM usuarios WHERE usuario = 'admin' AND 1=2; -- Devuelve falso

Si la primera consulta devuelve un resultado diferente de la segunda, el atacante puede inferir que la condición es verdadera o falsa, y puede continuar haciendo suposiciones iterativas sobre el contenido de la base de datos.

Ejemplo: Un atacante podría intentar determinar el nombre de la primera columna en una tabla de la base de datos, realizando una serie de pruebas booleanas como esta:

sql

Copiar código

```
SELECT * FROM usuarios WHERE usuario = 'admin' AND ASCII(SUBSTRING((SELECT column_name FROM information_schema.columns WHERE table_name='usuarios' LIMIT 1), 1, 1)) = 65;
```

Si la respuesta es diferente a la habitual, el atacante sabe que el primer carácter del nombre de la columna es "A" (ASCII 65).

Time-Based Blind SQL Injection (Basada en Tiempo):

Aquí, el atacante utiliza funciones SQL que introducen retrasos en la respuesta de la aplicación. Si una consulta tarda más tiempo en ejecutarse de lo normal, el atacante puede inferir que la consulta es verdadera.

Procedimiento de Blind SQL Injection

La técnica general de Blind SQL Injection sigue los siguientes pasos:

Identificación de la vulnerabilidad: El atacante detecta que la aplicación es vulnerable a SQL Injection mediante prueba y error en campos de entrada (formularios, parámetros de URL, etc.).

Prueba de respuestas booleanas: El atacante intenta hacer preguntas "sí/no" a la base de datos, basadas en el comportamiento de la aplicación ante diferentes consultas. Por ejemplo, si la aplicación devuelve un resultado diferente cuando se introduce un AND 1=1 (que es verdadero) en comparación con AND 1=2 (que es falso), es probable que la base de datos esté siendo consultada directamente.

Extracción de datos: El atacante realiza una serie de consultas booleanas o basadas en tiempo para inferir datos. Esto puede incluir el uso de funciones como SUBSTRING() y ASCII() para extraer un carácter a la vez, lo que hace que el proceso sea lento pero efectivo.

Herramientas para realizar Blind SQL Injection

SQLMap:

SQLMap es una de las herramientas más populares para detectar y explotar vulnerabilidades de inyección SQL, incluidas las de tipo blind. Automáticamente realiza pruebas booleanas y basadas en tiempo para identificar inyecciones ciegas.

Burp Suite:

Burp Suite permite realizar pruebas manuales y automáticas de aplicaciones web, incluidas vulnerabilidades de Blind SQL Injection, mediante el envío de consultas y la observación de las respuestas de la aplicación.

OWASP ZAP:

Similar a Burp Suite, OWASP ZAP es una herramienta gratuita que puede identificar y explotar Blind SQL Injection mediante técnicas automáticas y manuales.

Consecuencias de la Blind SQL Injection

Aunque la Blind SQL Injection puede ser más lenta y difícil de realizar que otras formas de inyección de SQL, las consecuencias son igualmente devastadoras si se ejecuta con éxito:

Acceso no autorizado: El atacante puede acceder a datos confidenciales almacenados en la base de datos.

Exfiltración de datos sensibles: Utilizando consultas iterativas, el atacante puede recuperar datos como nombres de usuario, contraseñas cifradas, información personal, etc.

Modificación de datos: En casos más avanzados, el atacante puede modificar, insertar o eliminar registros en la base de datos.

Prevención de Blind SQL Injection

Para evitar las vulnerabilidades de Blind SQL Injection, se deben implementar las mismas mejores prácticas de seguridad que para otros tipos de inyección de SQL:

Consultas preparadas (Prepared Statements): Usar sentencias SQL parametrizadas para evitar que el usuario pueda alterar la lógica de las consultas SQL.

Validación y sanitización de entradas: Asegúrate de validar y sanear todas las entradas de usuario, asegurando que solo se acepten valores esperados.

Limitación de privilegios en la base de datos: Usa el principio de privilegios mínimos para que las cuentas de la base de datos solo tengan los permisos necesarios.

Uso de ORM (Object-Relational Mapping): Los ORMs abstraen las consultas SQL y pueden reducir el riesgo de inyección si se usan correctamente.

Monitoreo y auditoría: Implementa sistemas para detectar actividad inusual y posibles intentos de explotación en la base de datos.

SQL INJECTION BASADA EN ERRORES:

La **inyección de SQL basada en errores** ocurre cuando una aplicación web expone mensajes de error generados por la base de datos. Estos errores pueden revelar detalles sobre la estructura de la base de datos o proporcionar pistas útiles para que los atacantes ajusten sus consultas maliciosas. Los atacantes

aprovechan estos errores para extraer información valiosa, como nombres de tablas, columnas, y otros detalles estructurales.

Características de la SQL Injection basada en errores

En una inyección SQL estándar, la información puede obtenerse a través de consultas exitosas, pero en una inyección basada en errores, el atacante induce a la base de datos a generar errores explícitos que son mostrados en la respuesta de la aplicación. Estos errores permiten al atacante identificar detalles de la base de datos y hacer inferencias sobre su estructura.

Al explotar los errores SQL no manejados correctamente, el atacante puede realizar consultas más complejas y usar las respuestas para extraer datos.

Riesgos y consecuencias

Cuando una vulnerabilidad de SQL Injection basada en errores es explotada, el atacante puede obtener rápidamente información crítica de la base de datos. Entre los riesgos más importantes se encuentran:

Exposición de datos confidenciales: Los errores pueden revelar nombres de tablas, nombres de columnas y, en algunos casos, los mismos datos que se almacenan.

Mapeo de la estructura de la base de datos: Con suficiente información obtenida a través de errores, el atacante puede comprender la estructura completa de la base de datos, lo que facilita ataques posteriores.

Explotación combinada: Una vez que el atacante ha obtenido suficienteinformación a través de errores, puede realizar otros tipos de inyecciones SQLpara manipular los datos, robar información o incluso comprometer la seguridaddel sistema.

Ejemplo de impacto

Considera una aplicación vulnerable que expone detalles de la base de datos cuando se produce un error. Si el atacante lanza una consulta mal formada, puede recibir una respuesta detallada que incluya el nombre de la tabla afectada. Al continuar utilizando consultas mal formadas deliberadamente, puede descubrir los nombres de columnas, los tipos de datos y otros detalles internos.

Herramientas para explotar SQL Injection basada en errores

SQLMap:

Una de las herramientas más utilizadas para explotar SQL Injections, que también incluye módulos para explotar inyecciones basadas en errores. SQLMapautomatiza la explotación de errores y puede detectar estructuras de bases de datos a partir de ellos.

Havij:

Havij es una herramienta fácil de usar que explota vulnerabilidades de SQL Injection y está diseñada para extraer información usando errores de la base de datos.

Burp Suite:

Aunque más general, Burp Suite permite realizar pruebas manuales de SQL Injection, incluidas las basadas en errores. Permite a los testers observar los mensajes de error detallados devueltos por la aplicación y ajustar sus ataques en consecuencia.

OWASP ZAP:

Similar a Burp Suite, ZAP es una herramienta gratuita que también puede utilizarse para detectar y explotar SQL Injections basadas en errores.

Cómo prevenir SQL Injection basada en errores

Para prevenir este tipo de ataques, se deben seguir algunas prácticas de seguridad clave:

No mostrar mensajes de error detallados: Asegúrate de que los errores generados por la base de datos no se muestren directamente al usuario. En su lugar, usa mensajes de error genéricos o redirige al usuario a una página de error controlada.

Validación de entradas: Implementa un estricto filtrado y validación de todas las entradas de usuario, asegurándote de que solo se acepten los datos esperados (por ejemplo, validar que los números sean números y las cadenas de texto sean válidas).

Uso de consultas preparadas: Las consultas parametrizadas o preparadas ayudan a evitar que el código SQL del atacante se interprete como parte de la consulta principal, evitando inyecciones.

Control de excepciones: Maneja las excepciones de forma segura y asegúrate de que cualquier error interno en la base de datos no sea visible para el atacante. Las aplicaciones deben devolver errores genéricos y registrar los errores detallados solo a nivel interno.

Principio de privilegios mínimos: Los usuarios y aplicaciones deben tener solo los permisos estrictamente necesarios para realizar sus tareas. Esto limita el impacto de una inyección SQL exitosa.

Uso de ORMs y bibliotecas seguras: Usar Object-Relational Mappers (ORMs) como Hibernate o Entity Framework puede ayudar a prevenir inyecciones al abstraer el acceso a la base de datos, aunque no debe considerarse una protección absoluta.

SQL INJECTION BASADA EN TIEMPO:

En la **inyección de SQL basada en tiempo**, el atacante modifica las consultas SQL para introducir funciones que retrasen la respuesta del servidor. Dependiendode cuánto tiempo tarde en responder la aplicación, el atacante puede inferir si las consultas maliciosas son correctas o no. Este tipo de ataque es útil cuando no hay retroalimentación directa de la base de datos (por ejemplo, en inyecciones ciegas).

CARACTERÍSTICAS DE LA SQL INJECTION BASADA EN TIEMPO:

La técnica se basa en el uso de funciones que permiten a la base de datos pausar la ejecución durante un período determinado. Al inyectar estas funciones, el atacante puede medir el tiempo de respuesta y, a partir de ello, deducir si la consulta fue ejecutada exitosamente.

Por ejemplo, una función comúnmente utilizada en ataques de SQL Injectionbasada en tiempo es SLEEP() en MySQL o WAITFOR DELAY en SQL Server,que puede forzar la base de datos a esperar antes de devolver una respuesta.

¿CÓMO FUNCIONA LA SQL INJECTION BASADA EN TIEMPO?

El atacante suele inyectar una condición booleana que evalúa cierto estado de la base de datos (por ejemplo, si una tabla o columna existe) y, en función de esa evaluación, induce una pausa en la respuesta:

Si la condición es verdadera, la base de datos esperará un período de tiempo determinado.

Si la condición es falsa, la base de datos devolverá la respuesta inmediatamente.

Cómo se explota la SQL Injection basada en tiempo

El atacante suele usar la técnica de inferencia booleana combinada con funciones de tiempo para extraer información, una letra o valor numérico a la vez. Esto se hace repitiendo muchas veces la inyección con pequeñas variaciones para ir deduciendo los datos.

PASOS TÍPICOS PARA UN ATAQUE:

Verificación de vulnerabilidad: El atacante prueba si la base de datos responde con una demora al usar funciones como SLEEP() o WAITFOR DELAY.

Extracción de datos: El atacante realiza consultas condicionales y mide el tiempo de respuesta para extraer información como nombres de tablas, columnas o datos dentro de las tablas.

Iteración: El proceso se repite múltiples veces, modificando las consultas inyectadas hasta que se hayan obtenido suficientes datos.

TÉCNICAS ADICIONALES EN SQL INJECTION BASADA EN TIEMPO:

Uso de funciones condicionales: Las funciones IF() en MySQL o CASE en SQL Server son clave para construir las condiciones que determinarán si la base de datos debe esperar.

Medición precisa del tiempo: Es fundamental que el atacante tenga una forma de medir con precisión el tiempo de respuesta de la aplicación para determinar si la condición fue verdadera o falsa.

RIESGOS Y CONSECUENCIAS:

Extracción de datos sensibles: Al igual que en otras formas de SQL Injection, el objetivo del atacante es obtener datos confidenciales de la base de datos. Aunque este tipo de ataque es más lento, es igual de efectivo.

Dificultad en la detección: Como este ataque no genera errores visibles y depende solo del tiempo de respuesta, es más difícil de detectar para los sistemas de monitoreo convencionales.

Impacto en el rendimiento: Un ataque exitoso puede sobrecargar el sistema si provoca muchas consultas que incluyen retrasos largos, lo que puede afectar la disponibilidad del servicio.

HERRAMIENTAS QUE PERMITEN EXPLOTAR SQL INJECTION BASADA EN TIEMPO:

SQLMap: Esta herramienta incluye módulos para detectar y explotar SQL Injections basadas en tiempo. SQLMap puede automatizar el proceso de enviar consultas y medir los tiempos de respuesta.

Havij: Havij es otra herramienta que permite realizar ataques de SQL Injection y tiene opciones para realizar inyecciones basadas en tiempo.

Burp Suite: Aunque no está específicamente orientada a SQL Injection, Burp Suite puede ser utilizada para realizar este tipo de ataques manualmente, observando los tiempos de respuesta de las consultas.

OWASP ZAP: Similar a Burp Suite, OWASP ZAP es una herramienta gratuita para pruebas de seguridad web que permite detectar vulnerabilidades de SQL Injection, incluida la basada en tiempo.

CÓMO PREVENIR SQL INJECTION BASADA EN TIEMPO:

Consultas preparadas (parametrizadas): Usar consultas preparadas es una de las defensas más efectivas contra cualquier tipo de inyección SQL, incluidas las basadas en tiempo. Las consultas preparadas garantizan que las entradas del usuario no puedan ser tratadas como código SQL ejecutable.

Validación de entrada: Es crucial validar y sanear todas las entradas del usuario antes de usarlas en las consultas SQL. Las entradas que contienen caracteres

peligrosos como comillas simples (') deben ser filtradas o escapadas adecuadamente.

Deshabilitar mensajes de error detallados: Aunque este ataque no se basa en errores visibles, siempre es una buena práctica evitar que los mensajes de error detallados lleguen al usuario. En lugar de mostrar mensajes detallados, las aplicaciones deben manejar las excepciones internamente y mostrar mensajes genéricos al usuario.

Uso de ORM: Los frameworks ORM (Object-Relational Mapping) como Hibernate o Entity Framework pueden reducir el riesgo de SQL Injection al abstraer las consultas SQL directas, aunque no deben considerarse una protección absoluta.

Implementación de un WAF (Web Application Firewall): Un WAF bien configurado puede detectar patrones de comportamiento sospechoso, como inyecciones SQL, y bloquear las solicitudes maliciosas.

Monitoreo de tiempos de respuesta: Implementar sistemas de monitoreo quealerten sobre anomalías en los tiempos de respuesta puede ayudar a identificar este tipo de ataques.

SQL INJECTION EN PROCEDIMIENTOS ALMACENADOS:

Los **procedimientos almacenados** son rutinas SQL almacenadas y ejecutadas en el servidor de base de datos. Si los procedimientos almacenados no validan correctamente las entradas de los usuarios, pueden ser vulnerables a inyección de SQL. En estos casos, los atacantes pueden manipular las entradas que llegan al procedimiento almacenado para ejecutar comandos SQL arbitrarios.

¿CÓMO OCURRE LA INYECCIÓN SQL EN PROCEDIMIENTOS ALMACENADOS?

El ataque de inyección SQL sucede cuando los valores de entrada no son debidamente validados o sanitizados, permitiendo que un atacante inserte código SQL malicioso dentro de los parámetros del procedimiento almacenado. Esto puede darle acceso no autorizado a la base de datos, permitiéndole manipularla, extraer datos sensibles o realizar otras acciones maliciosas.

CONSECUENCIAS:

Robo de datos: El atacante puede obtener información confidencial como nombres de usuarios, contraseñas, números de tarjetas de crédito, etc.

Modificación de datos: Se puede alterar o eliminar información almacenada en la base de datos.

Toma de control del sistema: En algunos casos, el atacante puede ejecutar comandos que le otorguen un control más amplio sobre el servidor.

Daño a la integridad de la base de datos: Al inyectar comandos maliciosos, los datos en las tablas pueden corromperse o eliminarse.

Usar parámetros con tipos de datos definidos:

Asegúrate de que los parámetros de los procedimientos almacenados sean tratados como parámetros, en lugar de concatenar las entradas del usuario directamente en la consulta.

Validar y sanitizar entradas del usuario: Siempre valida los datos que recibes para asegurarte de que cumplen con los formatos esperados. Los caracteres especiales y las secuencias de escape deben ser neutralizados.

Usar procedimientos almacenados parametrizados: En vez de construir dinámicamente las consultas SQL, usa parámetros de entrada predefinidos. Esto evita que los valores del usuario se interpreten como código SQL.

Manejar los permisos adecuadamente: Asegúrate de que los procedimientos almacenados y las cuentas de usuario tengan los permisos mínimos necesarios para llevar a cabo las tareas que se les asignan.

Deshabilitar funcionalidades peligrosas: En algunos casos, es mejor evitar el uso de funciones peligrosas como EXEC() o consultas dinámicas en procedimientos almacenados.

Herramientas y enfoques para proteger los procedimientos almacenados

ORM (Object-Relational Mapping): Las herramientas ORM como Entity Framework (para C#), Hibernate (para Java), entre otras, usan consultas parametrizadas por defecto, lo que ayuda a prevenir inyecciones SQL.

Reglas de cortafuegos de bases de datos (WAF): Configurar firewalls a nivel de aplicación o base de datos puede detectar y bloquear patrones de inyección SQL.

SQL INJECTION EN ORM:

Los **ORMs (Mapeo Objeto-Relacional)** son herramientas que permiten a los desarrolladores interactuar con bases de datos utilizando código orientado a objetos. Aunque los ORM pueden mitigar algunos riesgos de SQL Injection al usar consultas preparadas o API seguras, no están exentos de vulnerabilidades si se usan de forma incorrecta. Los ataques pueden ocurrir si se permiten consultas dinámicas construidas a partir de entradas no controladas.

¿QUÉ ES UN ORM?

Un ORM (Mapeo Objeto-Relacional) es una herramienta que facilita la interacción entre una aplicación y una base de datos al mapear tablas y registros de la base de datos a objetos en el lenguaje de programación utilizado por la aplicación. Esto

permite que los desarrolladores trabajen con objetos en lugar de escribir consultas SQL manualmente.

Algunos ORM comunes incluyen:

Hibernate (Java)

Entity Framework (C#/.NET)

SQLAlchemy (Python)

Doctrine (PHP)

Active Record (Ruby on Rails)

¿CÓMO OCURRE LA INYECCIÓN SQL EN UN ORM?

A pesar de que los ORM generalmente generan consultas SQL seguras utilizando consultas parametrizadas, existen escenarios en los que el mal uso del ORM puede dejar la aplicación vulnerable a inyecciones SQL. Esto ocurre principalmente cuando los desarrolladores utilizan métodos de ORM para construir consultas dinámicas o concatenar manualmente cadenas de SQL.

HERRAMIENTAS Y TÉCNICAS ADICIONALES PARA PROTEGERTE CONTRA INYECCIONES SQL EN ORM:

ORM de código abierto con auditorías de seguridad: Algunos ORM tienen auditorías de seguridad regulares y actualizaciones para prevenir ataques de inyección. Mantén el ORM actualizado.

ORM con opciones de sanitización integradas: Algunos ORM incluyen características integradas para la sanitización de entradas, lo que ayuda a mitigar el riesgo.

Análisis de seguridad y pruebas: Utiliza herramientas de análisis estático o dinámico para identificar posibles puntos de vulnerabilidad en el código que utiliza ORM.

HERRAMIENTAS PARA DETECTAR Y PREVENIR SQL INJECTION:

Detectar:

SQLMap: Herramienta automatizada para detectar y explotar vulnerabilidades de inyección SQL.

Burp Suite: Proporciona un escáner web que puede identificar inyecciones SQL y otros problemas de seguridad en aplicaciones.

Acunetix: Escáner de seguridad que detecta inyecciones SQL, entre otras vulnerabilidades.

Prevenir:

Validación y sanitización de entradas: Validar todas las entradas para asegurar que no contengan SQL malicioso.

Consultas preparadas (Prepared Statements): Usar sentencias SQL preparadas que separan los datos de los comandos SQL.

ORMs seguros: Implementar buenas prácticas con ORMs, evitando concatenar consultas dinámicas.

WAF (Web Application Firewalls): Los firewalls de aplicaciones web como ModSecurity pueden detectar patrones de ataques de inyección SQL.

Principio de privilegios mínimos: Asegurarse de que las cuentas de usuario de la base de datos tengan los privilegios más bajos posibles necesarios para su función.

HERRAMIENTAS DE ANÁLISIS ESTÁTICO (STATIC APPLICATION SECURITY TESTING, SAST):

Estas herramientas examinan el código fuente de una aplicación para identificar posibles vulnerabilidades, incluidas las inyecciones SQL. Buscan patrones peligrosos, como la concatenación directa de variables de usuario en consultas SQL.

EJEMPLOS DE HERRAMIENTAS DE ANÁLISIS ESTÁTICO:

SonarQube:

Realiza análisis de seguridad del código y busca vulnerabilidades, como inyecciones SQL, en una variedad de lenguajes de programación.

Proporciona recomendaciones para mejorar la seguridad y corregir problemas de código.

Fortify Static Code Analyzer:

Detecta inyecciones SQL al revisar el código fuente y proporciona informes detallados sobre las posibles vulnerabilidades.

Ofrece recomendaciones para mitigar las vulnerabilidades detectadas.

Checkmarx:

Escanea el código fuente en busca de vulnerabilidades de seguridad, incluidas las inyecciones SQL.

Soporta múltiples lenguajes y es altamente personalizable.

Brakeman:

Especializado para aplicaciones desarrolladas en **Ruby on Rails**.

Detecta vulnerabilidades específicas de Rails, como inyecciones SQL en consultas Active Record.

2. Herramientas de análisis dinámico (Dynamic Application Security Testing, DAST)

Estas herramientas simulan ataques de inyección SQL en una aplicación en funcionamiento. No requieren acceso al código fuente, sino que se centran en cómo la aplicación responde a diversas entradas.

Ejemplos de herramientas de análisis dinámico:

OWASP ZAP (Zed Attack Proxy):

Es una herramienta gratuita y de código abierto que permite detectar vulnerabilidades en aplicaciones web, incluidas inyecciones SQL.

Funciona como un proxy para interceptar solicitudes HTTP y simular ataques a la base de datos.

Tiene módulos específicos para probar inyecciones SQL.

Burp Suite:

Es una de las herramientas más populares para pruebas de seguridad de aplicaciones web.

Permite realizar pruebas manuales y automáticas, como inyecciones SQL, simulando solicitudes maliciosas para identificar vulnerabilidades.

La versión profesional incluye módulos avanzados para pruebas automatizadas y generación de informes.

SQLMap:

Es una herramienta de código abierto diseñada específicamente para identificar y explotar vulnerabilidades de inyección SQL en bases de datos.

Automatiza la detección de diferentes tipos de inyección SQL y ofrece opciones avanzadas para explotar las vulnerabilidades, como extracción de datos, modificación de bases de datos, etc.

Acunetix:

Un escáner de seguridad de aplicaciones web que busca una amplia gama de vulnerabilidades, incluidas las inyecciones SQL.

Detecta y reporta inyecciones SQL en formularios, parámetros URL, cookies y otras entradas de usuario.

3. Web Application Firewalls (WAF)

Los **firewalls de aplicaciones web (WAF)** protegen las aplicaciones web al filtrar y monitorear el tráfico HTTP para detectar y bloquear inyecciones SQL antes de que lleguen a la base de datos. Los WAF se colocan entre el usuario y la aplicación web, inspeccionando las solicitudes en tiempo real.

Ejemplos de WAF:

ModSecurity:

Un WAF de código abierto que puede detectar y bloquear inyecciones SQL y otras amenazas.

Funciona mediante reglas configurables que permiten identificar patrones de ataque en las solicitudes HTTP.

Puede integrarse con servidores web como Apache y Nginx.

AWS WAF:

El firewall de aplicaciones web de Amazon Web Services, que protege las aplicaciones alojadas en AWS de amenazas como inyecciones SQL.

Utiliza reglas preconfiguradas y personalizables para identificar y bloquear ataques SQL Injection.

Cloudflare WAF:

Proporciona protección en la nube para aplicaciones web y ayuda a mitigar inyecciones SQL al analizar el tráfico en tiempo real.

Ofrece reglas automáticas para bloquear solicitudes maliciosas basadas en patrones comunes de ataques SQL.

Imperva WAF:

Un WAF avanzado que protege contra inyecciones SQL y otros ataques.

Analiza las consultas SQL y las solicitudes HTTP para bloquear las que contienen código malicioso antes de que lleguen a la base de datos.

4. Bases de datos con medidas de seguridad avanzadas

Algunas bases de datos incluyen características que ayudan a mitigar los ataques de inyección SQL mediante la implementación de consultas parametrizadas, validación de entradas y restricciones de permisos.

Ejemplos:

Microsoft SQL Server:

Incluye mecanismos de seguridad como consultas parametrizadas mediante sp_executesql, lo que ayuda a prevenir inyecciones SQL.

También ofrece auditoría avanzada y reglas de encriptación de datos.

PostgreSQL:

Soporta consultas preparadas y parametrizadas de manera nativa, ayudando a prevenir ataques SQL Injection.

Incluye funciones avanzadas de auditoría y control de acceso que limitan el daño potencial de un ataque.

MySQL:

También soporta consultas preparadas mediante las API de MySQLi y PDO en PHP, lo que impide la ejecución de inyecciones SQL si se implementan correctamente.

5. Herramientas de escaneo de seguridad web completas

Estas herramientas no solo detectan inyecciones SQL, sino que también identifican otras vulnerabilidades comunes en aplicaciones web, como cross-site scripting (XSS), problemas de autenticación, configuraciones incorrectas, etc.

EJEMPLOS:

Netsparker:

Una solución de seguridad web que automatiza la detección de inyecciones SQL y otras vulnerabilidades críticas.

Ofrece integración con entornos de desarrollo continuo (CI/CD) para facilitar la detección temprana en el ciclo de vida del software.

Qualys Web Application Scanner:

Escanea aplicaciones web para detectar inyecciones SQL, XSS y otras vulnerabilidades de seguridad.

Proporciona informes detallados y recomendaciones para corregir las vulnerabilidades.

6. Inyección SQL en frameworks de desarrollo seguros

Algunos frameworks y bibliotecas incluyen mecanismos de protección incorporados para prevenir inyecciones SQL.

Django (Python):

Utiliza consultas SQL parametrizadas de forma predeterminada, evitando la inyección de SQL.

Ruby on Rails:

Al usar Active Record, Rails parametriza automáticamente las consultas, protegiendo contra la inyección de SQL.

Spring (Java):

El framework Spring fomenta el uso de ORM como Hibernate, que protege contra inyecciones SQL mediante consultas parametrizadas.

PRUEBAS MANUALES Y AUDITORÍAS

Además de las herramientas automatizadas, las pruebas manuales y auditorías de seguridad también son cruciales. Los **pentesters** (testers de penetración) y auditores de seguridad pueden realizar pruebas exhaustivas y simulaciones de ataques para descubrir vulnerabilidades de inyección SQL que las herramientas automatizadas podrían pasar por alto.

CONCLUSION:

En esta práctica, hemos explorado y comprendido uno de los ataques más comunes y peligrosos en aplicaciones web: la inyección SQL**.** A través de la manipulación de entradas no validadas, los atacantes pueden ejecutar comandos SQL maliciosos que comprometen la seguridad de la base de datos, permitiéndoles obtener, modificar o eliminar información sensible. Durante la práctica, se demostró la importancia de validar y sanear correctamente las entradas de usuario para evitar este tipo de vulnerabilidades. Además, se enfatizó el uso de consultas parametrizadas y procedimientos almacenados como una medida clave de prevención, ya que estas técnicas ayudan a evitar que los datos proporcionados por los usuarios sean tratados como comandos SQL ejecutables. También se revisaron diversas herramientas y enfoques para detectar y mitigar inyecciones SQL, como el análisis estático y dinámico de aplicaciones, así como los Web Application Firewalls (WAF)**.** El aprendizaje obtenido destaca la importancia de combinar buenas prácticas de desarrollo con el uso deherramientas de seguridad para minimizar el riesgo de ataques. Finalmente, la práctica subraya la necesidad de implementar mecanismos de defensa proactivos en el ciclo de desarrollo, haciendo que la seguridad sea una prioridad desde el inicio de la creación de la aplicación. Esto es fundamental para garantizar la integridad y confidencialidad de los datos en aplicaciones web y sistemas basadosen bases de datos.

BIBLIOGRAFIAS:

¿Qué es la inyección de SQL? | Explicación y protección (avast.com)

Blind SQL Injection | OWASP Foundation

Inyección SQL: Definición y ejemplos reales. - Backtrack Academy

¿Qué es una inyección SQL (SQLi) y cómo prevenirla? | EasyDMARC

SQL Injection, Procedimientos Almacenados, ASP, ADO y SQL Server (guillesql.es)