Artículo Científico / Scientific Paper

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
INGENIUS
Revista de Ciencia y Tecnología
https://doi.org/10.17163/ings.n25.2021.10
pISSN: 1390-650X / eISSN: 1390-860X

Análisis de vulnerabilidades con SQLMAP aplicada a entornos APEX 5

Vulnerability analysis with SQLMAP applied to APEX5 context

Esteban Crespo-Martínez[1,*]

Recibido: 14-09-2020, Revisado: 01-10-2020, aprobado tras revisión: 30-11-2020

Resumen

Las bases de datos son usualmente los principales objetivos de un ataque, específicamente por la información que en ella reside, ya que, de acuerdo con Druker, la información es poder. En este trabajo se realizan las pruebas de vulnerabilidad de la base de datos de un *software* ERP desarrollado en APEX 5. Para ello, se utilizan herramientas FOSS de prueba y análisis de vulnerabilidades de bases de datos, identificando que las sesiones que utiliza ERP basada en Oracle APEX son realizadas de manera aleatoria y que, además, son nuevamente generadas en determinados momentos. Se concluye que, con las pruebas aplicadas y las actualizaciones de SQLMAP a la fecha del experimento, no se ha conseguido vulnerar el *software* ERP con técnicas de inyección SQL.

Palabras clave: APEX, evaluación a sistemas de información, inyección SQL, protección de datos

Abstract

Databases are usually the main targets of an attack, specifically for the information that they store, since, according to Druker, information is power. In this work vulnerability tests are performed of the database of an ERP software developed in APEX 5. For this purpose, FOSS tools are used to test and analyze vulnerabilities of databases, identifying that sessions used by ERP based on Oracle APEX are carried out randomly, and besides are generated again at particular times. It is therefore concluded that, with the tests applied and the updates of SQLMAP to the date of the experiment, it has not been possible to vulnerate the ERP software with SQL injection techniques.

Keywords: APEX, Data protection, Information systems evaluation, SQL Injection.

[1,*]Universidad del Azuay, Ecuador. Autor para correspondencia ✉: ecrespo@uazuay.edu.ec.
https://orcid.org/0000-0002-3061-9045

Forma sugerida de citación: Crespo-Martínez (2021). «Análisis de vulnerabilidades con SQLMAP aplicada a entornos APEX 5». Ingenius. N.° 25, (enero-junio). pp. 104-113. doi: https://doi.org/10.17163/ings.n25.2021.10.

1. Introducción

Varios expertos en seguridad de la información coinciden con que los ataques informáticos cada vez son más recurrentes, y usualmente a los sistemas web, alterando o exponiendo la información personal [1], en especial a las aplicaciones web [2], debido a su complejidad, extensión, alta personalización y usualmente desarrollados por programadores con poca experiencia en seguridad [3].

No se puede negar que, en esta sociedad de la información y el conocimiento, las bases de datos contienen la mina de oro, esta se convierte en uno de los elementos estratégicos más importantes de la organización, pues de su análisis e interpretación en el momento oportuno se pueden proyectar estrategias para aventajar oportunidades y prever amenazas según el rol de la organización en la sociedad.

La protección de la información toma sentido cuando aparecen los primeros virus informáticos: alteraban o borraban la información del usuario, muchas veces con fines de demostrar la capacidad destructora y creativa de quien diseñaba el *malware* para acometerla. El contexto informático era más simple, no existía la intercomunicación entre organizaciones y los sistemas se limitaban a funcionar de manera centralizada [4]. Sin embargo, la explosión de oportunidades que se generan por la introducción de Internet hace que los atacantes vean de otra manera a los datos. Dañarlos o eliminarlos ya no tiene sentido, el robo, la copia o el secuestro son aspectos que se convierten en nuevos objetivos.

Ojagbule *et al.* [5] mencionan que, hoy en día, existen más de un billón de sitios web, y que muchos de ellos son desarrollados por gestores de contenido como *Drupal*, *Joomla* o *WordPress*, y que, según Mohammadi & Namadchian [6], contienen datos importantes.

Según Ojagbule *et al.* [5] y Kruegel *et al.* [6], al existir una gran cantidad de sitios, también existe una gran cantidad de bases de datos que están expuestas a vulnerabilidades y riesgos. De esto aparece una técnica conocida como inyección SQL (*Structured Query Language Injection Attack* por sus siglas en inglés SQLIA), que según Santin, Oliveira y Lago [7] citando a [8] y [9], es una técnica donde un atacante explora vulnerabilidades que permiten alterar los comandos SQL en una aplicación y que es conocida como una de las vulnerabilidades que generan mayor impacto en la organización.

Nofal y Amber [9] agregan que esta técnica usualmente no tiene patrones predecibles o específicos, lo que se convierte en un problema importante para investigadores y desarrolladores. Concluye Badaruddin [10] que la técnica de inyección SQL es el segundo error más común encontrado en los servidores web en Internet con alrededor del 44,11 %.

Con el propósito de descubrir fallas de seguridad en cuanto a vulnerabilidades de inyección SQL, en este trabajo se realiza una evaluación con SQLMAP a una base de datos Oracle que almacena la información del sistema UDA-ERP desarrollado en APEX 5 por la Universidad del Azuay. Este artículo se divide en las siguientes secciones: i) el estado del arte, donde se establecen algunos conceptos, así como también los trabajos relacionados; ii) la metodología aplicada para la obtención de los resultados, detallando las configuraciones realizadas en el equipo de laboratorio de pruebas; iii) los resultados obtenidos tras la ejecución de la herramienta; iv) la discusión sobre los resultados adquiridos y v) las conclusiones y trabajos futuros.

1.1. La inyección SQL

Según OWASP, la inyección SQL es una de las diez vulnerabilidades más peligrosas y populares que pueden presentarse en entornos web [11], los cuales generalmente son difíciles de proteger debido a su alta personalización, complejidad, escala [3], tecnología y desarrollo por programadores con poca experiencia en seguridad [3, 12], causando serios daños a los negocios de las víctimas [13]. Sumado a esto, Setiawan y Setiyadi [14] sostienen que, en un contexto de redes de computadoras, cualquier dato existente en un computador conectado a otro se vuelve inseguro.

Los autores Santin, Oliveira y Lago [7] citando a [15] mencionan que no existen soluciones que garanticen o solucionen todas las vulnerabilidades, las cuales ocurren a nivel de hardware y *software*, expresión a la que se suma Setiawan [14]. Agregan que, como muchos elementos no son actualizados constantemente, son más propensos a ataques cibernéticos. Por otro lado, Kals *et al.* [12] sostienen que son múltiples las vulnerabilidades de seguridad de aplicaciones web, resultado de problemas genéricos de validación de entrada. Además, las vulnerabilidades pueden mantenerse secretas o reportadas por los fabricantes, sea de forma pública o privada [16].

De forma básica, un ataque de inyección SQL puede ser representado como se indica en la Figura 1.

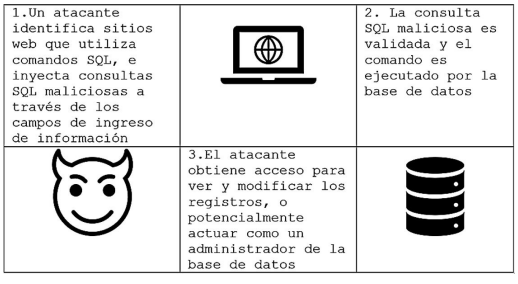

Figura 1. Proceso de la inyección SQL

Otra forma de representar la secuencia de ataques SQL es la que propone AVI Networks [17], la cual se presenta en la Figura 2.

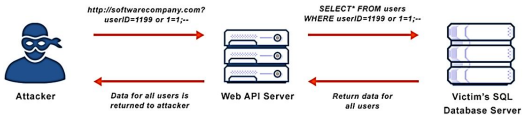

Figura 2. Secuencia de ataque SQL Injection [17]

De acuerdo con [18], las técnicas de ataque de inyección SQL pueden clasificarse de la siguiente manera:

i) Tautologías, un tipo de ataque que utiliza consultas condicionales e inserta tokens SQL en ellas, demostrando siempre ser cierto.

ii) Consultas ilegales o lógicamente incorrectas, donde los atacantes utilizan los mensajes de error de las bases de datos para encontrar vulnerabilidades en las aplicaciones.

iii) Consultas con UNION, donde los atacantes inyectan consultas infectadas sobre consultas seguras utilizando el operador UNION y, por lo tanto, recuperar información de la base de datos.

iv) Consultas con respaldo o *Piggy-backed*: los atacantes adjuntan delimitadores como ";" a la consulta original y las ejecutan simultáneamente, siendo la primera legítima y las demás falsas, pero retornando información valiosa.

v) Procedimientos almacenados (*Stored procedures*), un subconjunto de consultas precompiladas, dependiendo de cuales sean existirán diferentes formas de ataque.

vi) Inyección ciega o *Blind Injection*, en la que los desarrolladores ocultan mensajes de error que pueden ser útiles para los atacantes para planificar y ejecutar un ataque SQLIA. En esta situación el atacante se encuentra con una página estática, donde realiza preguntas verdaderas y falsas mediante el uso de comandos SQL hasta conseguir el objetivo.

vii) Ataques temporizados, que permiten al atacante observar el tiempo requerido para ejecutar una consulta. El atacante genera una consulta grande utilizando sentencias *if-else* y, de esta manera, medir la cantidad de tiempo que tarda la página en cargarse para determinar si la sentencia inyectada es verdadera.

viii) Codificaciones alternativas, donde codificaciones *ASCII* y *Unicode* permiten evadir el filtro que escanea "caracteres especiales" [19].

Una evaluación de vulnerabilidades por inyección SQL puede ser realizada con el uso de herramientas tecnológicas para tal propósito. Novaski [20] sugiere el uso de herramientas FOSS, de las cuales propone 14 para ser utilizadas: Arachni, Beef, Htcap, IronWASP, Metasploit, Skipfish, SQLMap, Vega, W3af, Wapiti, Wfuzz, XSSer, Xenotix y ZAP.

De este trabajo agrega que solo las herramientas IronWASP, Vega, ZAP y SQLMap detectaron la vulnerabilidad de inyección SQL, mientras que la vulnerabilidad XSS reflejada solo fue detectada por las herramientas ZAP y Xenotix.

Indica en su trabajo que solo fue posible realizar una prueba de intrusión completa en la vulnerabilidad de inyección SQL, y para realizarla fue necesario aplicar tres herramientas diferentes: i) wapiti-getcookie, para obtener el identificador de sesión; ii) Htcap para obtener puntos de entrada; y SQLMap para detectar y explotar la vulnerabilidad.

Entre los trabajos relacionados están los que se encuentran en la Tabla 1. Este trabajo, a diferencia de los citados, hace énfasis en probar aspectos de seguridad en una aplicación desarrollada en Oracle APEX 5.

Está claro que, a pesar del tiempo transcurrido desde la primera vez que apareció el ataque de inyección SQL hace dos décadas [21], son numerosas las técnicas de inyección, así como las técnicas de evasión y mitigación. Las tecnologías de información están cada vez más frecuentes en nuestro entorno y han afectado notablemente el estilo de vida, pues cada vez que aumenta el uso y la confiabilidad en las computadoras y los sistemas informáticos, la amenaza a los datos sensibles también aumenta.

Las vulnerabilidades de inyección SQL en las aplicaciones web son sorprendentemente vastas y, definitivamente, son una gran amenaza para la seguridad de los datos personales que se almacenan en la web [21].

En la práctica, Cetin *et al.* [22] demuestran que un análisis automatizado de GitHub enseña que el 15,7 % de los 120 412 archivos fuente de Java publicados contienen código vulnerable a los ataques de inyección de identificador de SQL (SQL-IDIA) señalando, además que, tras una revisión manual, comprobaron que 18 939 archivos Java identificados durante el análisis automatizado son vulnerables a este tipo de ataques.

Puneet [23] clasifica a la inyección SQL en dos tipos: i) la inyección SQL clásica y ii) la inyección SQL avanzada.

Tabla 1. Trabajos relacionados con el uso de SQLMap

Id.	Autores	Tema	Año	Objetivo
1	Nofal D.E. Amer A. A. [9]	SQL Injection Attacks Detection and Prevention Based on Neuro-Fuzzy Technique	2020	Realizan un trabajo para detectar y prevenir ataques de inyección SQL, aplicando un sistema de inferencia de lógica difusa.
2	O. Ojagbule H. Wimmer R. Haddad [24]	Vulnerability Analysis of Content Management Systems to SQL Injection Using	2018	Comparan las vulnerabilidades de inyección SQL en tres gestores de contenido más utilizados, considerando las herramientas Nikto y SQLMap para tal propósito.
3	F. Santin J. A. Oliveira V. Lago Machado [7]	Uso da ferramenta SQLMap para detecção de vulnerabilidades de SQL Injection	2017	Se enfoca a exponer los principales riesgos a los que se someten las aplicaciones web relacionadas con la inyección SQL. Hacen uso de la herramienta SQLMap para tal propósito.
4	Badaruddin Bin Halib, Edy Budiman, Hario Jati Setyadi [10]	Técnicas de pirateo de servidores web con SQLMap en Kali Linux	2017	Proponen una técnica de hacking a servidores web mediante el uso de SQLMap en Kali Linux.
5	S. D. Axinte [25]	SQL injection Testing in Web Applications Using SQLMap	2014	La autora realiza un análisis analítico de la técnica de inyección SQL, y presenta métodos, herramientas y acciones de prevención.
6	Barinas, Alarcón, Callejas [1]	Vulnerabilidad de ambientes virtuales de aprendizaje utilizando SQLMap, RIPS, W3AF y Nessus*	2014	Trabajo en el que analizan los aspectos de seguridad de ambientes virtuales de aprendizaje, herramientas de seguridad y análisis de vulnerabilidades.
7	A. Tajpour, S. Ibrahim, M. Masrom [26]	SQL Injection Detection and Prevention Techniques	2015	Proponen técnicas de ataque y mitigación frente a ataques de inyección SQL, comparando varios tipos de ellos.

1.1.1. Inyección SQL clásica

Las técnicas de inyección básica, sugeridas por [23] se resumen en las siguientes:

a) Piggy Backed Queries

La intención del ataque es primordialmente la denegación de servicio. La base de datos recibe múltiples consultas, en las que, durante la ejecución, la consulta normal funciona como en un caso normal, mientras que la segunda consulta se adhiere a la primera para conseguir el ataque. Un ejemplo de este ataque puede ser el siguiente:

```
select cliente from cuentas where
login_id = "admin" AND pass = ‘123’;
DELETE FROM accounts WHERE
ClienteNombre = ’Francisco’;
```

Posterior a la ejecución de la primera consulta, el intérprete detecta el punto y coma ";" y ejecuta la segunda consulta junto con la primera, eliminando todos los datos del cliente "Francisco". Este tipo de datos maliciosos pueden protegerse determinando en primer lugar la consulta SQL correcta mediante la validación adecuada o utilizando adecuadas técnicas de detección, como es el análisis estático, que no necesita la supervisión del tiempo de ejecución.

b) *Stored procedure*

La intención de ataque se resume en autenticación de escape y en denegación de servicio. Erradamente, los profesionales de TI piensan que los procedimientos almacenados de SQL son un remedio para la inyección de SQL [17], ya que se los coloca el frente de las bases de datos y las características de seguridad no son directamente aplicables. Los procedimientos almacenados no usan el lenguaje de consulta estructurado estándar, usa sus propios lenguajes de script que no tienen la misma vulnerabilidad que SQL, pero que mantienen otras diversas vulnerabilidades relacionadas con el lenguaje de scripting. Como ejemplo, se puede indicar lo siguiente:

```
CREATE PROCEDURE Info_usuario @usuario
varchar2 @password varchar2 @idcliente
int AS BEGIN EXEC('Select info_cliente
from tabla_cliente where username='
"+@usuario " ' and pass = '
"+@password " ' GO
```

Cualquier usuario malicioso puede ingresar datos maliciosos en los campos del nombre de usuario y *password*. Un simple comando ingresado podría destruir toda la base de datos o provocar una denegación del servicio. De esta forma, [23] sugiere que no se acopie información crítica en los procedimientos almacenados.

c) *Union query*

Es un tipo de ataque que utiliza el operador unión (U) mientras inserta la consulta SQL. Las dos consultas SQL, la normal y la nociva, son unidas mediante este operador. El ejemplo muestra cómo se puede proceder, visualizando que la segunda consulta es maliciosa y no se tiene en cuenta el siguiente texto (-), ya que se convierte en comentario para el Analizador de SQL.

```
select * from cuentas where id='212'
UNION select * from factura where
usuario='admin'-' and password='pass'
```

1.1.2. d) Codificación alternativa

Con respecto a este tipo de ataque, el atacante cambia el patrón de la inyección SQL para que no sea detectado por las técnicas comunes de detección y prevención. En este método, el atacante utiliza la representación de código hexadecimal, Unicode, octal y ASCII en la instrucción SQL, evitando ser detectados debido al uso de cadenas codificadas.

1.1.3. Inyección SQL avanzada

Las técnicas de inyección SQL avanzada, sugeridas por [23] se resumen en las siguientes:

a) Inyección SQL a ciegas o *Deep Blind SQL Injection Attack*

Una gran cantidad de aplicaciones web se deshabilitan la visualización de errores MYSQL u otro SQL. En este ataque, la información se infiere mediante preguntas de verdadero/falso. Si el punto de inyección es absolutamente ciego, entonces la única forma de atacar es mediante el uso del comando WAIT FOR DELAY o BENCHMARK [23].

b) *Fast flux SQL Injection Attack*

El objetivo del ataque es la extracción de datos o la suplantación de identidad mediante *phishing*. Un host que realiza *phishing* puede ser detectado fácilmente mediante el seguimiento de su dirección IP o la identificación de su nombre de dominio. Sin embargo, concordando con [23] y [27], los sistemas de protección de muchos *hostings web* podrían suspender el servicio debido al masivo tráfico que se genera, anulando los propósitos criminales. De esta manera, para evitar este inconveniente, los atacantes aplican la técnica del *Fast Flux*, que es una técnica DNS para ocultar los sitios de distribución de *Phishing* y *malware* detrás de una red de cambio constante.

1.1.4. c) Ataques de inyección SQL compuestos

Se trata de una mezcla de dos o más técnicas de ataque, generando un efecto mayor a los indicados con las técnicas anteriormente descritas. *Compounded* SQL *Injection*, como es conocido en el mundo oscuro, es derivada de la mezcla del ataque SQL y otros ataques de aplicaciones web, como, por ejemplo, el ataque de inyección SQL + ataques de denegación de servicio distribuidos (DDoS). Sobre la base de lo que expone [23] citando a [28], el código para hacer este tipo de ataque sería:

```
http://exploitable-web.com/link.php?id=1'
union select 1,2,tab1,4 from
```

```
(select decode(encode(convert
(compress(post)
using latin1),
des_encrypt
(concat(post,post,post,post),8)),
des_encrypt(sha1(concat(post,post,
post,post)),9))
as tab1 from table_1)a-
```

Otra forma de combinar un ataque es el de mezclar una inyección SQL con la autenticación insuficiente. Este ataque es explotable cuando los parámetros de seguridad no han sido inicializados donde la aplicación falla al identificar la ubicación del usuario, el servicio o la aplicación. Esto permite al atacante acceder a información privilegiada sin la verificación de la identidad del usuario.

De esta manera, este tipo de ataque es relativamente más sencillo que con cualquier otro tipo de ataque [23], donde el primer paso es ubicar un sitio web que tenga autenticación insuficiente.

2. Materiales y métodos

El propósito del análisis de seguridad fue el de evaluar la seguridad de una aplicación desarrollada en Oracle APEX, plataforma en la que está desarrollado el *software* UDA ERP de la Universidad del Azuay. Con esta premisa se configuró un laboratorio considerando los materiales y métodos que se exponen a continuación.

Para las pruebas se consideró la suite KALI y se utilizó la herramienta SQLMAP, herramienta basada en Free and Open Source, desarrollada sobre una licencia GNU GPLv2 por Miroslav Stampar y Bernardo Damele, considerando que, según [18], SQLMAP soporta, entre otras, la base de datos Oracle, que es la que utiliza el *software* UDA ERP.

Acota que SQL soporta seis técnicas de inyección: a ciegas basada en booleana (boolean-based blind), a ciegas temporalizado (time-based blind), basada en error (error-based), basada en consultas UNION (UNION query-based), fuera de banda (out-of-band) y consultas apiladas (stack querys).

Las técnicas anteriormente mencionadas forman parte de los parámetros de testeo que están incluidas en SQLMAP, las cuales se aplican automáticamente. Utilizando BurpSuite, se ajusta la captura de *cookies* de sesión, aplicando la configuración de un proxy local (127.0.0.1:8080) con el propósito de capturar las peticiones POST, que posteriormente serán utilizadas con SQLMAP. Previo a la ejecución de las pruebas se actualizaron las dependencias y paquetes de la suite.

Reconociendo que SQLMAP es una herramienta que permite explorar servidores de bases de datos, para su uso es importante apuntar a la dirección URL que contiene un script SQL. La estructura "sqlmap -u URL –[parámetros]" es la sentencia común, donde el parámetro -dbs permitirá obtener la base de datos. Posterior a la detección de la vulnerabilidad, se debe usar el parámetro -D y el nombre de la base de datos que será analizada. Si el resultado obtenido es efectivo, el parámetro -tables permitirá recuperar todas las tablas de la base de datos especificada.

Con el propósito de identificar las vulnerabilidades, se realizaron tres pruebas, en cada una se incrementó el análisis y se variaron aspectos como el nivel de agresividad en las pruebas, el uso de cookies de sesiones establecidas y la evasión a sistemas de identificación como elaborarlo.

La primera prueba consistió en listar las bases de datos; en la segunda, se aumentó el grado de agresividad y cantidad de pruebas para obtener información de las bases de datos; y en el tercer ataque se pretendió usar un agente aleatorio evadiendo los *proxys* con el único propósito de capturar una cookie de sesión, elemento que es usado como base para las pruebas automatizadas, en las que se simula una sesión válida de un usuario activo.

3. Resultados

3.1. Primera prueba

La evaluación de vulnerabilidades de la base de datos mediante la ejecución del comando root@kali: /sqlmap-dev# python3 sqlmap.py -u –dbs, dio como resultado lo expresado en la Tabla 2, considerando que el ataque generó su propia cookie para evaluación: ('USUARIO=ORA_WWV-cUs...UNNyk6flfB'). La prueba inicia a las 13:03:52 del 2020-03-04 y termina a las 13:04:22 /2020-03-04/

Tabla 2. Resultados de la primera prueba

Test	Descripción
Análisis heurístico	El análisis heurístico detectó que el objetivo es protegido por algún tipo de WAF/IPS.
Contenido URL	El parámetro 'p' del método GET no parece ser dinámico. Las pruebas heurísticas básicas realizadas indican que el parámetro 'p' no son inyectables.
Inyección SQL	No es vulnerable.
Test 'AND boolean-based blind-WHERE or HAVING clause'	Valores reflectivos fueron encontrados y filtrados. Reflective value(s) found and filtering out. Significa que existen valores "reflexivos" dentro de la respuesta que contiene (partes de) la carga útil. Esto es notablemente malo en algunos casos, especialmente, en inyecciones booleanas.

3.2. Segunda prueba

En la segunda prueba se ejecutó el comando con las opciones: python3 sqlmap-dev/sqlmap.py -

u "http://172.16.1.87:8080/ords/f?p=502" –level=5 –risk=3 -dbs -a –tamper=between. La prueba inicia a las 12:30 del 2020-03-05 y culmina a las 15:18 del 2020-03-05.

Con los parámetros seleccionados se aumenta la intensidad del ataque, así como el nivel y cantidad de pruebas. Los resultados se expresan en la Tabla 3.

Tabla 3. Resultados de la segunda prueba

Test	Descripción
Análisis heurístico	El análisis heurístico detectó que el objetivo es protegido por algún tipo de WAF/IPS.
Contenido URL	El parámetro 'p' del método GET no parece ser dinámico. Las pruebas heurísticas básicas realizadas indican que el parámetro 'p' no son inyectables.
Inyección SQL	No es vulnerable.
Test 'AND boolean-based blind – WHERE or HAVING clause'	Valores reflectivos fueron encontrados y filtrados. Reflective value(s) found and filtering out. Significa que existen valores "reflexivos" dentro de la respuesta que contiene (partes de) la carga útil. Esto es notablemente malo en algunos casos, especialmente en inyecciones booleanas.
Test UNION con consulta NULL y Método heurístico con parámetro 'User-Agent'	El parámetro 'p' del método GET no parece ser dinámico. Las pruebas heurísticas básicas realizadas indican que el parámetro 'p' no son inyectables. Parámetro User-Agent no es inyectable. Parámetro 'Referer' no parece ser dinámico. En el análisis heurístico el parámetro 'Referer' parece no ser inyectable. El parámetro HOST no parece ser dinámico. En el análisis heurístico el parámetro 'Host' parece no ser inyectable.

3.3. Tercera prueba

Se captura la cookie ORA_WWV-W7Hhdq_v8DH8Oli 2Fp4IsyM y se procede a utilizarla mientras la aplicación está activa. Se ejecuta el comando python3 sqlmap-dev/sqlmap.py -u "http://172.16.1.87:8080/ords/f?p=502:1:13479648228 07:::::" –tables –cookie=ORA_WWV-W7Hhdq_v8DH 8Oli2Fp4IsyMR –random-agent –ignore-proxy –level 5, incluyendo un agente aleatorio e ignorando los proxys, pues este último se lo utiliza únicamente con el propósito de capturar cookies, así como también aumentando el nivel de análisis al máximo. Se agrega el módulo de identificación de tablas. El análisis inicia el 2020-03-16 a las 12:21:44. La Tabla 4 refleja los resultados obtenidos.

4. Discusión

Los ataques comunes a sistemas de computación se dan por virus, gusanos y adversarios humanos [3]. La detección de un ataque por inyección SQL puede darse cuando se acostumbra a revisar verificaciones de logs, registros de acceso, detecciones de intrusos entre otras [7], [15], a las que se agrega la aplicación del principio de defensa en profundidad aplicando herramientas como IDS o WAFs [3]. Las evaluaciones continuas a las aplicaciones que se desarrollan y sus certificaciones, previo el paso a entornos en producción, se convierten en otra práctica fundamental, aspecto que usualmente se omite en las organizaciones por el intento de publicar lo antes posible.

Tabla 4. Resultados de la tercera prueba

Test	Descripción
Conexión con el URL	La conexión pide una redirección a una nueva URL generada de manera aleatoria. No se acepta la misma ya que se está utilizando una conexión ya establecida.
Análisis heurístico	Se evade el WAF/IPS.
Contenido URL	El parámetro 'p' del método GET no parece ser dinámico. Las pruebas heurísticas básicas realizadas indican que el parámetro 'p' no son inyectables.
Inyección SQL	No es vulnerable.
Test 'MySQL Boolean-based blind – Parameter replace (MAKE_SET)'	Valores reflectivos fueron encontrados y filtrados. Reflective value(s) found and filtering out. Significa que existen valores "reflexivos" dentro de la respuesta que contiene (partes de) la carga útil. Esto es notablemente malo en algunos casos, especialmente en inyecciones booleanas.

La inyección SQL no es una técnica que se aplica con el pulsar de un botón. Se requieren conocimientos sobre el lenguaje SQL. Agrega Clarke [8] que el uso de herramientas para realizar este tipo de acometidas es importante, pues permiten automatizar el ataque. La combinación de herramientas permite obtener resultados más precisos. Concordando con Santin *et al.* [7] y Ojagbule *et al.* [5], la aplicación de la herramienta SQLMAP para el análisis fue seleccionada por la popularidad, la disponibilidad y el acceso a sus diversas distribuciones, comprobando que herramientas basadas en FOSS permiten lograr resultados tan interesantes como el usar herramientas de pago.

Se ha evidenciado que en la base de datos de exploits (https://www.exploit-db.com), el último mecanismo para vulnerar un sistema hecho en APEX fue liberado el 16 de abril del 2009.

A diferencia del trabajo realizado por Clarke (2009) en el que se utiliza una aplicación vulnerable a propósito (en inglés damn vulnerableweb site), desarrollada en PHP con motor de datos MySQL cuyo objetivo es proporcionar una plataforma de pruebas a profesionales que requieran probar sus niveles de destrezas y conocimientos.

El resultado de la prueba 'AND boolean-based blind – WHERE or HAVING clause' se puede ejemplificar en una página estática, excepto con una pequeña parte donde refleja el valor del parámetro probado (el mismo donde el SQLMap realiza la inyección). En

caso de que dicho contenido "reflexivo" no se detecte y neutralice, existe una posibilidad considerable de que la respuesta aparezca como un cambio debido a las cargas útiles de inyección SQL utilizadas (por ejemplo, AND 2 > 3). Por lo tanto, surge el riesgo de detección de falsos positivos (o falsos negativos en algunos casos).

La herramienta, en la primera ejecución, culmina con el argumento de que todos los parámetros evaluados no parecen inyectables, sugiere aumentar el nivel y el riesgo si se desea realizar más pruebas. En sospecha de algún tipo de mecanismo de protección involucrado (ej. WAF), se podría usar la opción '–tamper' (e.g. '–tamper=space2comment') y/o cambiarlo por '–random-agent'. El ajuste en la prueba 3 con la configuración del parámetro NIVEL=5 fue necesario para que SQLMAP realice la prueba de vulnerabilidad de *cookies*.

La detección y prevención, según [21] se convierte en una tarea difícil si no se comprende adecuadamente el concepto de este tipo de ataques. Realizar evaluaciones binarias, tal como lo proponen [29] podría considerarse como una alternativa de mitigación, ya que se trata de un método extremadamente automatizado que detecta y bloquea ataques de inyección SQL en aplicaciones web.

Frente a ataques de inyección SQL a ciegas, la técnica más popular es AMNESIA (de las siglas en inglés Analysis and Monitoring for Neutralizing SQL-injection Attacks) [30], que es una herramienta aplicable únicamente para proteger las aplicaciones basadas en JAVA y que utilizan monitoreo en tiempo de ejecución [21]. Esta herramienta utiliza algoritmos de aprendizaje de máquina para proveer mecanismos de prevención y detección de amenazas Blind SQL Injection.

Otro mecanismo de prevención es el algoritmo de identificación de patrones, propuesto por Aho-Corasick [31], algoritmo que tiene dos fases: i) una fase estática y; ii) una fase dinámica.

En la fase estática, según [32], las consultas SQL generadas por el usuario son comparados con una lista de patrones que contienen una muestra de los patrones de ataque más conocidos. Si la sentencia SQL concuerda exactamente con uno de los patrones dados en la lista de patrones estáticos, significa que se está intentando un ataque SQL.

Otra alternativa que propone [31] citando a [33] es la SQLRand, donde la idea básica es generar sentencias SQL con el uso de comandos aleatorios en el que la plantilla de consulta dentro de la aplicación pueda ser aleatorizada. En esta forma, los comandos SQL que son inyectados por usuarios maliciosos no son codificados, por cuanto el proxy no reconoce los comandos inyectados haciendo que el ataque no se lleve a cabo.

Adicionalmente, [31] mencionando a [34], indica que existe un método adicional conocido como el Método de consulta por Token (Query Tokenization Method), en el que se genera un token de la consulta original y de la consulta con inyección. Luego, los tokens resultantes de este proceso se almacenan en un arreglo. La longitud de arreglos obtenidos de la consulta original y de la consulta con inyección son comparadas, si existe una coincidencia entonces no existe un intento de inyección SQL, caso contrario, se trata de un ataque.

A la lista de opciones de mitigación se suma la propuesta de [35] que consiste en un enfoque de validación de árbol gramatical, representado en una sentencia. Analizar gramaticalmente una sentencia requiere del conocimiento de la gramática del lenguaje en el que está escrita la sentencia. De tal manera que, cuando el atacante inyecte una consulta SQL maliciosa, entonces el árbol gramatical de la consulta original y la consulta con inyección no coinciden. En esta técnica, la sentencia en particular y la sentencia original son comparadas en tiempo de ejecución.

No está demás indicar que la codificación segura es crucial para el diseño de *software* y sistemas computacionales, aspecto que es omitido por los desarrolladores [12] debido en gran parte al desconocimiento de estándares de codificación segura, negligencia y pérdida de desempeño, a las que se suman las situaciones de usabilidad [36].

5. Conclusiones

Una de las características más notables de Oracle APEX es la de crear sesiones con *cookies* y enlaces URL al *software* con datos aleatorios. Las pruebas realizadas con la ejecución de las diferentes opciones de los comandos indicadas en esta técnica no permitieron acometer el *software* con la técnica de inyección SQL a una solución desarrollada en esta plataforma. Si bien una *cookie* puede ser capturada con Burp Suite, descifrar la misma toma un tiempo considerable. Sin embargo, en ese lapso, Oracle APEX dinámicamente ya generó una nueva *cookie* haciendo que el ataque por esta técnica sea básicamente imposible.

La contribución de este artículo ha sido el de evaluar diferentes técnicas de inyección SQL para enfatizar sobrescritura de código seguro, optimizar las etiquetas por omisión que se generan y así mejorar el nivel de seguridad de una aplicación desarrollada en Oracle APEX, sin olvidar que las pruebas han sido desarrolladas en un lapso temporal, sin eximir vulnerabilidades latentes que puedan presentarse en el día 0.

Referencias

[1] A. Barinas López, A. C. Alarcón Aldana, and M. Callejas Cuervo, "Vulnerabilidad de ambientes virtuales de aprendizaje utilizando SQLMAP, RIPS, W3AF y Nessus," *Ventana Informática*, no. 30,

pp. 247–260, 2014. [Online]. Available: https://doi.org/10.30554/ventanainform.30.276.2014

[2] S. Mohammadi and A. Namadchian, "Anomaly-based Web Attack Detection: The Application of Deep Neural Network Seq2Seq With Attention Mechanism," *The ISC International Journal of Information Security*, vol. 12, no. 1, pp. 44–54, 2020. [Online]. Available: http://doi.org/10.22042/ISECURE.2020.199009.479

[3] K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest, "Learning DFA representations of HTTP for protecting web applications," *Computer Networks*, vol. 51, no. 5, pp. 1239–1255, 2007, from Intrusion Detection to Self-Protection. [Online]. Available: https://doi.org/10.1016/j.comnet.2006.09.016

[4] B. Dwan, "The Computer Virus – From There to Here.: An Historical Perspective." *Computer Fraud & Security*, vol. 2000, no. 12, pp. 13–16, 2000. [Online]. Available: https://doi.org/10.1016/S1361-3723(00)12026-3

[5] O. Ojagbule, H. Wimmer, and R. J. Haddad, "Vulnerability Analysis of Content Management Systems to SQL Injection Using SQLMAP," in *SoutheastCon 2018*, 2018, pp. 1–7. [Online]. Available: https://doi.org/10.1109/SECON.2018.8479130

[6] C. Kruegel, G. Vigna, and W. Robertson, "A multi-model approach to the detection of web-based attacks," *Computer Networks*, vol. 48, no. 5, pp. 717–738, 2005, web Security. [Online]. Available: https://doi.org/10.1016/j.comnet.2005.01.009

[7] F. Santin, J. A. Oliveira de Figueiredo, and V. Lago Machado, "Uso da ferramenta sqlMap para detecção de vulnerabilidades de SQL Injection," in *Anais do EATI - Encontro Anual de Tecnologia da Informação*, 2017. [Online]. Available: https://bit.ly/340cKP6

[8] J. Clarke, *SQL Injection Attacks and Defense (Second Edition)*, second edition ed., J. Clarke, Ed. Boston: Syngress, 2012. [Online]. Available: https://doi.org/10.1016/B978-1-59-749963-7.00012-8

[9] D. E. Nofal and A. Amer, *SQL Injection Attacks Detection and Prevention Based on Neuro-Fuzzy Technique*. Springer, Cham, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-31129-2_66

[10] B. Bin Halib, E. Budiman, and H. Jati Setyadi, "Teknik Hacking Web Server Dengan SQLMAP Di Kali Linux," *Jurnal Rekayasa Teknologi Informasi*, vol. 1, no. 1, pp. 67–72, 2017. [Online]. Available: http://dx.doi.org/10.30872/jurti.v1i1.642

[11] OWASP. (2017) lobally recognized by developers as the first step towards more secure coding. [Online]. Available: https://bit.ly/2JTb9DF

[12] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "SecuBat: A Web Vulnerability Scanner," in *Proceedings of the 15th International Conference on World Wide Web*, ser. WWW '06. New York, NY, USA: Association for Computing Machinery, 2006, pp. 247–256. [Online]. Available: https://doi.org/10.1145/1135777.1135817

[13] J. Fonseca, M. Vieira, and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," in *13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)*, 2007, pp. 365–372. [Online]. Available: https://doi.org/10.1109/PRDC.2007.55

[14] E. B. Setiawan and A. Setiyadi, "Web vulnerability analysis and implementation," *IOP Conference Series: Materials Science and Engineering*, vol. 407, p. 012081, sep 2018. [Online]. Available: https://doi.org/10.1088%2F1757-899x%2F407%2F1%2F012081

[15] J. Atoum and A. Qaralleh, "A hybrid technique for SQL injection attacks detection and prevention," *International Journal of Database Management Systems (IJDMS*, vol. 6, no. 1, pp. 21–28, 2014. [Online]. Available: http://doi.org/10.5121/ijdms.2014.6102

[16] D. Herrmann and H. Pridöhl, *Basic Concepts and Models of Cybersecurity*, 2020, vol. 21. [Online]. Available: https://doi.org/10.1007/978-3-030-29053-5_2

[17] AVI Network. (2020) SQL Injection Attack. [Online]. Available: https://bit.ly/3mb96YF

[18] P. Ramasamy and S. Abburu, "SQL Injection Attack: Detection and Prevention," *International Journal of Engineering Science and Technology*, vol. 4, no. 4, pp. 1396–1401, 2016. [Online]. Available: https://bit.ly/3n7aSeV

[19] XS Code. (2020) XS:Code. [Online]. Available: https://bit.ly/37MYc6s

[20] D. Novski Neto, "Web (eternamente) revisitada : análise de vulnerabilidades web e de ferramentas de código aberto para exploração," 2019. [Online]. Available: https://bit.ly/37VrNui

[21] V. K. Gudipati, T. Venna, S. Subburaj, and O. Abuzaghleh, "Advanced automated SQL injection attacks and defensive mechanisms," in *2016 Annual Connecticut Conference on*

Industrial Electronics, Technology Automation (CT-IETA), 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/CT-IETA.2016.7868248

[22] C. Cetin, D. Goldgof, and J. Ligatti, "SQL-Identifier Injection Attacks," in *2019 IEEE Conference on Communications and Network Security (CNS)*, 2019, pp. 151–159. [Online]. Available: https://doi.org/10.1109/CNS.2019.8802743

[23] J. P. Singh, "Analysis of SQL Injection Detection Techniques," 2016. [Online]. Available: https://bit.ly/375XeDh

[24] O. Ojagbule, H. Wimmer, and R. J. Haddad, "Vulnerability Analysis of Content Management Systems to SQL Injection Using SQLMAP," in *SoutheastCon 2018*, 2018, pp. 1–7. [Online]. Available: https://doi.org/10.1109/SECON.2018.8479130

[25] A. Ciampa, C. A. Visaggio, and M. Di Penta, "A Heuristic-Based Approach for Detecting SQL-Injection Vulnerabilities in Web Applications," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, ser. SESS '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 43–49. [Online]. Available: https://doi.org/10.1145/1809100.1809107

[26] R. Alsahafi, "SQL Injection Detection and Prevention Techniques," *International Journal of Scientific & Technology Research*, vol. 8, no. 1, pp. 182–185, 2019. [Online]. Available: https://bit.ly/2W24Ksp

[27] L. Wichman, "Mass SQL injection for malware distribution," SANS Institute, Tech. Rep., 2011. [Online]. Available: https://bit.ly/2Ke3ks0

[28] JAVANICUS. (2016) Posts Related to Web-Pentest-SQL-Injection. [Online]. Available: https://bit.ly/2IEFUMc

[29] V. Sunkari and C. V. Guru rao, "Protect Web Applications against SQL Injection Attacks Using Binary Evaluation Approach," *International Journal of Innovations in Engineering and Technology (IJIET)*, pp. 484–490, 2016. [Online]. Available: https://bit.ly/377eVSR

[30] W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 174–183. [Online]. Available: https://doi.org/10.1145/1101908.1101935

[31] M. A. Prabakar, M. KarthiKeyan, and K. Marimuthu, "An efficient technique for preventing SQL injection attack using pattern matching algorithm," in *2013 IEEE International Conference ON Emerging Trends in Computing, Communication and Nanotechnology (ICECCN)*, 2013, pp. 503–506. [Online]. Available: https://doi.org/10.1109/ICE-CCN.2013.6528551

[32] G. Yiğit and M. Arnavutoğlu, "SQL Injection Attacks Detection & Prevention Techniques," *International Journal of Computer Theory and Engineering*, vol. 9, no. 5, pp. 351–356, 2017. [Online]. Available: https://bit.ly/3qKrEm5

[33] S. W. Boyd and A. D. Keromytis, "Boyd s.w., keromytis a.d." in *International Conference on Applied Cryptography and Network Security*, 2004, pp. 292–302. [Online]. Available: https://doi.org/10.1007/978-3-540-24852-1_21

[34] L. Ntagwabira and S. L. Kang, "Use of Query tokenization to detect and prevent SQL injection attacks," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 2, 2010, pp. 438–440. [Online]. Available: https://doi.org/10.1109/ICCSIT.2010.5565202

[35] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," in *Proceedings of the 5th International Workshop on Software Engineering and Middleware*, ser. SEM '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 106–113. [Online]. Available: https://doi.org/10.1145/1108473.1108496

[36] F. D. Nembhard, M. M. Carvalho, and T. C. Eskridge, "Towards the application of recommender systems to secure coding," *EURASIP Journal on Information Security*, vol. 2019, no. 1, p. 9, Jun. 2019. [Online]. Available: https://doi.org/10.1186/s13635-019-0092-4