Scientific Paper / Artículo Científico

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

Vulnerability analysis with SQLMAP applied to APEX5 context

Análisis de vulnerabilidades con SQLMAP aplicada a entornos APEX 5

Esteban Crespo-Martínez[1,*]

Received: 14-09-2020, Reviewed: 01-10-2020, Accepted after review: 30-11-2020

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.

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

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

Suggested citation: Crespo-Martínez (2021). «Vulnerability analysis with SQLMAP applied to APEX5 context». Ingenius. N.° 25, (january-june). pp. 104-113. doi: https://doi.org/10.17163/ings.n25.2021.10.

1. Introduction

Various experts in information security agree that cyber-attacks are increasingly recurrent, and usually target web systems, thus altering or putting on risk personal information [1], especially to web applications [2] , due to their complexity, extension, high personalization and because they are usually developed by programmers with little experience in security [3].

It cannot be denied that, in this society of information and knowledge, databases contain the gold mine, which becomes one of the most important strategic elements of the organization, because from its analysis and interpretation at the right time it is possible to project strategies to stay ahead of opportunities and foresee threats according to the role of the organization in the society.

The protection of the information started to make sense when the first computer viruses appeared: they altered or erased user information, often with the purpose of demonstrating the destructing and creative capacity of the designer of the malware used to undertake the attack. The computing context was simpler, there was no intercommunication between organizations and systems were limited to operate in a centralized manner [4]. However, as a result of the boom of opportunities generated by the appearance of the Internet, attackers now see data in a different manner. Damaging or eliminating them does not make sense, data theft, copy or seizure are aspects that become the new targets.

Ojagbule *et al.* [5] mention that, today, there are more than one billion websites, and that many of them are developed by content managers such as Drupal, Joomla or WordPress, and that, according to Mohammadi y Namadchian [6], they contain important data.

According to Ojagbule *et al.* [5] and Kruegel *et al.* [6], due to the existence of a large number of sites, there is also a large number of databases subject to vulnerabilities and risks. Thereof it appears a technique known as SQL injection (SQLIA, Structured Query Language Injection Attack), which according to Santin, Oliveira and Lago [7] citing [8] and [9], is a technique where an attacker explores vulnerabilities that enable altering the SQL commands in an application, which is known as one of the vulnerabilities that generate greater impact in the organization.

Nofal and Amber [9] add that this technique usually does not have predictable or specific patterns, which becomes an important problem for researchers and developers. Badaruddin [10] concludes that the SQL injection technique is the second most common error found in web servers in Internet, with around 44.11 %.

With the purpose of discovering security failures regarding SQL injection vulnerabilities, in this work it is carried out an evaluation using SQLMAP of an Oracle database that stores information of the UDA-ERP system developed by the Universidad del Azuay on APEX 5. This paper is divided in the following sections: i) state of the art, where some concepts are established, as well as related works; ii) the methodology applied for obtaining the results, detailing the configurations made in the laboratory test equipment; iii) the results obtained after executing the tool; iv) the discussion about the results obtained and v) the conclusions and future works.

1.1. The SQL injection

According to OWASP, the SQL injection is one of the ten most dangerous and popular vulnerabilities that may appear in web environments [11], which in general are difficult to protect due to their high personalization, complexity, scale [3], technology and development by programmers with little experience in security [3] [12], causing serious damages to the businesses of the victims [13]. In addition to this, Setiawan and Setiyadi [14] state that, in a computer networks context, any existing data in a computer connected to another computer becomes insecure.

Authors Santin, Oliveira and Lago [7] citing [15] state that there are no solutions that guarantee or resolve all vulnerabilities which occur at the hardware and software levels, statement also supported by Setiawan [14]. They also add that, since many elements are not constantly updated, they are more prone to cyber-attacks. On the other hand, Kals *et al.* [12] state that there are multiple vulnerabilities to the security of web applications, as a result of generic problems of input validation. In addition, vulnerabilities may be kept secret or reported by manufacturers, either publicly or privately [16].

An SQL injection attack may be basically represented as indicated in Figure 1.

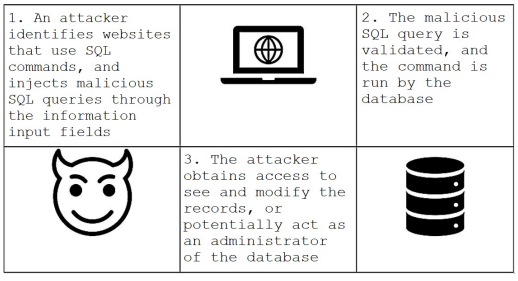

Figure 1. SQL injection process

Another way of representing the sequence of attacks is the one proposed by AVI Networks [17], which is presented in Figure 2.

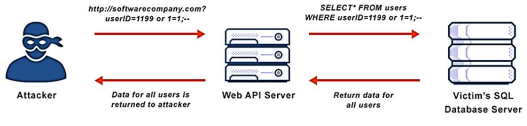

Figure 2. SQL Injection attack sequence. Source [17]

According to Charania and Vyas [18], the SQL injection attack techniques may be classified as follows:

i) Tautologies, a type of attack that uses conditional queries and inserts SQL tokens in them, demonstrating to be always true.

ii) Illegal or logically incorrect queries, where the attackers use the error messages of the databases to find vulnerabilities in the applications.

iii) Queries with UNION, where the attackers inject infected queries over secure queries using the UNION operator and, therefore, recover information from the database.

iv) Queries with support or Piggy-backed: the attackers attach delimiters such as ";" to the original query and run them simultaneously, with the first being legitimate and the remaining false, but returning valuable information.

v) Stored procedures, a subset of precompiled queries, depending on which they are there will be different forms of attack.

vi) Blind injection, in which the developers hide error messages that may be useful for attackers to plan and execute an SQLIA attack. In this situation the attacker finds a static page, where true and false questions are made using SQL commands until the objective is attained.

vii) Timed attacks, which enable the attacker to observe the time required to execute a query. The attacker generates a big query using if-else sentences and, in this way, measures the amount of time spent by the page to load and determine if the injected sentence is true.

viii) Alternative coding, where ASCII and Unicode coding enable to evade the filter which scans "special characters" [19].

An evaluation of vulnerabilities by SQL injection may be undertaken with the use of technological tools for such purpose. Novaski [20] suggests the use of FOSS tools, of which 14 are proposed to be used: Arachni, Beef, Htcap, IronWASP, Metasploit, Skipfish, SQLMap, Vega, W3af, Wapiti, Wfuzz, XSSer, Xenotix and ZAP.

From this work it adds that only the tools IronWASP, Vega, ZAP and SQLMap detected the vulnerability of SQL injection, while the reflected XSS vulnerability was only detected by the tools ZAP and Xenotix.

It is indicated in their work that it was only possible to conduct a complete intrusion test in the SQL injection vulnerability, and it was necessary to apply three different tools for carrying out such test: i) wapitigetcookie, to obtain the session identifier; ii) Htcap to obtain points of input; and SQLMap to detect and explore the vulnerability.

Among the related works there are the ones shown in Table 1. This work, as opposed to those cited works, focus on testing security aspects in an application developed in Oracle APEX 5.

It is clear that, despite the time elapsed from the first time that the SQL injection attack appeared two decades ago [21], both the injection and the evasion and mitigation techniques are numerous. Information technologies are increasingly common in our environment and have notably affected our lifestyle because every time that the use and reliability of computers and computer systems increase, the threat on sensitive data also increases.

SQL injection vulnerabilities in web applications are surprisingly vast and are definitely a big threat for the security of the personal data stored in the web [21].

In practice, Cetin *et al.* [22] demonstrate that a GitHub automatic analysis shows that 15.7 % of the 120412 Java source files published contain code vulnerable to SQL identifier injection attacks (SQL-IDIA), also pointing out that, after a manual revision, they proved that 18939 Java files identified during the automatic analysis are vulnerable to this type of attacks.

Puneet [23] classifies SQL injection in two types: i) classic SQL injection and ii) advanced SQL injection.

Table 1. Works related with the use of SQLMap

Id.	Autors	Topic	Year	Objective
1	Nofal D.E. Amer A. A. [9]	SQL Injection Attacks Detection and Prevention Based on Neuro-Fuzzy Technique	2020	Conduct a work to detect and prevent SQL injection attacks, applying a fuzzy logic inference system.
2	O. Ojagbule H. Wimmer R. Haddad [24]	Vulnerability Analysis of Content Management Systems to SQL Injection Using	2018	Compare the vulnerabilities of SQL injection in the three most widely used content managers, considering the Nikto and SQLMap tools for such purpose.
3	F. Santin J. A. Oliveira V. Lago Machado [7]	Uso da ferramenta SQLMap para detecção de vulnerabilidades de SQL Injection	2017	Focus on describing the main risks to which web applications are subject, related to the SQL injection. They use the SQLMap tool for such purpose.
4	Badaruddin Bin Halib, Edy Budiman, Hario Jati Setyadi [10]	Técnicas de pirateo de servidores web con SQLMap en Kali Linux	2017	Propose a technique for hacking web servers using SQLMap in Kali Linux.
5	S. D. Axinte [25]	SQL injection Testing in Web Applications Using SQLMap	2014	The author conducts an analytical analysis of the SQL injection technique, and presents methods, tools and prevention actions.
6	Barinas, Alarcón, Callejas [1]	Vulnerabilidad de ambientes virtuales de aprendizaje utilizando SQLMap, RIPS, W3AF y Nessus*	2014	Analyzes the security aspects of virtual learning environments, security and vulnerability analysis tools.
7	A. Tajpour, S. Ibrahim, M. Masrom [26]	SQL Injection Detection and Prevention Techniques	2015	Propose attack and mitigation techniques against SQL injection attacks, comparing various types of them.

1.1.1. Classic SQL injection

The basic injection techniques, suggested by [23] are summarized as follows:

a) Piggy Backed Queries

The intention of the attack is primarily the denial of service. The database receives multiple queries in which, during the execution, the normal query operates as in a normal case, while the second query adheres to the first to attain the attack. An example of this attack may be the following:

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

After the execution of the first query, the interpreter detects the semicolon ";" and executes the second query together with the first, eliminating all the data of the client "Francisco". This type of malicious data may be protected by first determining the correct SQL query by means of the appropriate validation or using appropriate detection techniques, as it is the static analysis, which does not need the supervision of the run time.

b) Stored procedure

The intention of attack is summarized as escape authentication and denial of service. Mistakenly, IT professionals think that the stored SQL procedures are a remedy for the SQL injection [17], since they are placed in front of the databases and the security characteristics are not directly applicable. The stored procedures do not use standard structured query language, they use their own script languages that do not have the same vulnerability as SQL, but keep other diverse vulnerabilities related with the scripting language. For example, it may be indicated the following:

```
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
```

Any malicious user may enter malicious data in the username and password fields. A simple entered command may destroy the whole database or cause a denial of service. In this way, [23] suggests that critical information is not collected in the stored procedures.

c) Union query

It is a type of attack that uses the union operator (U) while inserting the SQL query. The two SQL queries, normal and harmful, are joined together using this operator. The example shows how it is proceeded, visualizing that the second query is malicious and the following text (-) is not taken into account, since it is converted to a comment by the SQL Analyzer.

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

1.1.2. d) Alternative coding

With respect to this type of attack, the attacker changes the SQL injection pattern so that it is not detected by common detection and prevention techniques. In this method, the attacker uses hexadecimal, Unicode, octal and ASCII code representation in the SQL instruction, to avoid being detected due to the use of coded chains.

1.1.3. Advanced SQL injection

The advanced SQL injection techniques suggested by [23], are summarized as follows:

a) Deep Blind SQL Injection Attack

In a great number of web applications the visualization of mysql errors or another SQL is disabled. In this attack, the information is inferred by means of true/false questions. If the injection point is absolutely blind, then the only way to attack is through the use of the WAIT FOR DELAY or BENCHMARK [23] command.

b) Fast flux SQL Injection Attack

The objective of the attack is the extraction of data or the identity theft through phishing. A host that carries out phishing may be easily detected by tracking its IP address or through the identification of its domain name. However, according to [23] and [27], the protection systems of many web hostings may suspend the service due to the massive traffic generated, thus cancelling out the criminal purposes. In this manner, to avoid this problem, attackers apply technique of the Fast Flux, which is a DNS technique to hide the phishing and malware distribution sites behind a network of constant change.

1.1.4. c) Compounded SQL injection attacks

It is a mix of two or more attack techniques, generating an effect greater than the indicated with the previously described techniques. Compounded SQL injection, as it is known in the dark world, derives from the mix of the SQL attack and other web applications attacks as, for example, the SQL injection attack + the distributed denial of service (DDoS) attacks. Based on what is exposed by [23] citing [28], the code to perform this type of attack would be:

```
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-
```

Another way of combining an attack is mixing an SQL injection with insufficient authentication. This attack is exploitable when the parameters of security have not been initialized where the application fails when identifying the location of the user, the service or the application. This enables the attacker to access to privileged information without verifying the identity of the user.

In this manner, this type of attack is relatively simpler than with any other type of attack [23], where the first step is locating a website that has insufficient authentication.

2. Materials and methods

The purpose of the security analysis was to evaluate the security of an application developed in Oracle APEX, platform which es being developed in the UDA ERP software of the Universidad del Azuay. With this premise, it was configured a laboratory considering the materials and methods which are described in the following.

For the tests, it was considered the KALI suite and it was utilized the SQLMap tool, which is based in Free and Open Source, developed under a GNU GPLv2 license by Miroslav Stampar and Bernardo Damele, considering that, according to Charania and Vyas [18], SQLMap supports, among others, the Oracle database, which is the one used by the UDA ERP software.

It also states that SQL support six injection techniques: Boolean-based blind, time-based blind, error-based, UNION query-based, out-of-band and stacked queries.

The previously mentioned techniques take part in the testing parameters which are included in SQLMap, which are automatically applied. Using BurpSuite, the capturing of session cookies is adjusted, applying the configuration of a local proxy (127.0.0.1:8080) with the purpose of capturing the POST requests, which will be further used with SQLMap. Before executing the tests, the dependencies and packages of the suite were updated.

Acknowledging that SQLMap is a tool that enables exploring database servers, for its use it is important to point to the URL address that contains the SQL script. The structure "sqlmap -u URL –[parameters]" is the common sentence, where the -dbs parameter will enable obtaining the database. After detecting the vulnerability, it should be used the -D parameter and the name of the database which will be analyzed. If the result obtained is effective, the -tables parameter will enable recovering all the tables from the specified database.

With the purpose of identifying the vulnerabilities, three tests were conducted, in each of which the analysis was increased and aspects such as level of aggressiveness in the tests, use of cookies of established sessions and evasion to identification systems were varied.

The first test consisted in listing the databases; in the second, it was increased the degree of aggressiveness and the number of tests to obtain information from the databases; and in the third attack it was pretended to use a random agent evading the proxies with the unique purpose of capturing a session cookie, element which is used as the base for automatic tests, in which a valid session of an active user is simulated.

3. Results

3.1. First test

The evaluation of vulnerabilities of the database by executing the command root@kali: /sqlmap-dev# python3 sqlmap.py -u –dbs, gave the result expressed in Table 2, considering that the attack generated its own cookie for evaluation: ('USUARIO=ORA_WWV-cUs...UNNyk6flfB'). The tests starts at 13:03:52 on 2020-03-04 and ends at 13:04:22 /2020-03-04/

Table 2. Results of the first test

Test	Description
Heuristic analysis	The heuristic analysis detected that the target is protected by some type of WAF/IPS.
URL Content	The 'p' parameter of the GET method does not seem to be dynamic. The basic heuristic tests conducted indicate that the 'p' parameter is not injectable.
SQL injection	It is not vulnerable.
Test 'AND boolean-based blind-WHERE or HAVING clause'	Reflective values found and filtered out. It means that there are "reflective" values within the response that contains (parts of) the useful load. This is significantly bad in some cases, especially in Boolean injections.

3.2. Second test

In the second test it was executed the command with options: python3 sqlmap-dev/sqlmap.py -u "http://172.16.1.87:8080/ords/f?p=502" –level=5 –risk=3 -dbs -a –tamper=between. The test starts at 12:30 on 2020-03-05 and ends at 15:18 on 2020-03-05.

With the parameters chosen the intensity of the attack is increased, as well as the level and the number of tests. The results are expressed in Table 3.

Table 3. Results of the second test

Test	Description
Heuristic analysis	The heuristic analysis detected that the target is protected by some type of WAF/IPS.
URL Content	The 'p' parameter of the GET method does not seem to be dynamic. The basic heuristic tests conducted indicate that the 'p' parameter is not injectable.
SQL injection	It is not vulnerable.
Test 'AND boolean-based blind – WHERE or HAVING clause'	Reflective values found and filtered out. It means that there are "reflective" values within the response that contains (parts of) the useful load. This is significantly bad in some cases, especially in Boolean injections.
Test UNION con consulta NULL y Método heurístico con parámetro 'User-Agent'	The 'p' parameter of the GET method does not seem to be dynamic. The basic heuristic tests conducted indicate that the 'p' parameter is not injectable. The User-Agent parameter is not injectable. The 'Referer' parameter does not seem to be dynamic. In the heuristic analysis the 'Referer' parameter does not seem to be injectable. The HOST parameter does not seem to be dynamic. In the heuristic analysis the 'Host' parameter does not seem to be injectable.

3.3. Third test

It is captured the cookie ORA_WWV-W7Hhdq_v8DH8Oli2Fp4IsyM and it is proceeded to use it while the application is active.

It is executed the command python3 sqlmap-dev/sqlmap.py -u "http://172.16.1.87:8080/ords/f?p=502:1:1347964822807:::::" –tables –cookie=ORA_WWV-W7Hhdq_v8DH8Oli2Fp4IsyMR –random-agent –ignore-proxy –level 5, including a random agent and ignoring the proxies, because the latter is only used with the purpose of capturing cookies, as well as increasing the level of analysis to the maximum. The module of identification of tables is added. The analysis starts on 2020-03-16 at 12:21:44. Table 4 reflects the results obtained.

4. Discussion

The attacks common to computer systems occur by viruses, worms and human adversaries [3]. The detection of an attack by SQL injection may occur when it is usual to check log verifications, access registers, intrusion detection, among others [7], [15], to which it is added the application of the principle of defense in depth applying tools such as IDS or WAFs [3]. The continuous evaluations to the applications that are developed and their certifications, before passing to production environments, become another fundamental practice, aspect which is usually omitted in the organizations due to the attempt to publish as soon as possible.

Table 4. Resultados de la tercera prueba

Test	Description
Connection with the URL	The connection requests to redirect to a new URL generated randomly. This is not accepted since a connection already established is being used.
Heuristic analysis	The WAF/IPS is evaded.
URL Content	The 'p' parameter of the GET method does not seem to be dynamic. The basic heuristic tests conducted indicate that the 'p' parameter is not injectable.
SQL injection	It is not vulnerable.
Test 'MySQL Boolean-based blind – Parameter replace (MAKE_SET)'	Reflective values found and filtered out. It means that there are "reflective" values within the response that contains (parts of) the useful load. This is significantly bad in some cases, especially in Boolean injections.

The SQL injection is not a technique which is applied by pressing a button. Knowledge about SQL language is required, Clarke [8] adds that the use of tools to carry out this type of attacks is important, because they enable to automate the attack. The combination of tools enables obtaining more precise results. In agreement with Satin *et al.* [7] and Ojagbule *et al.* [5], the application of the SQLMap tool for the analysis was chosen due to its popularity, availability and the access to its diverse distributions, verifying that tools based on FOSS enable getting results as interesting as using paying instruments.

It has been evidenced that in the exploits database (https://www.exploit-db.com), the last mechanism to violate a system made in APEX was released on April 16th, 2009.

As opposed to the work conducted by Clarke (2009) in which an application deliberately vulnerable (damn vulnerable website) is used, developed in PHP with the MySQL data engine whose objective is providing a test platform to professionals that require testing their levels of skills and knowledge.

The result of the 'AND Boolean-based blind – WHERE or HAVING clause' test can be exemplified in a static page, except with a small part where it is reflected the value of the parameter tested (the same where the SQLMap carries out the injection). In case that such "reflective" content is not detected and neutralized, there is a considerable potential that the response appears as a change due to the useful loads of SQL injection utilized (for example, AND 2>3). Therefore, the risk of detecting false positives (or false negatives in some cases) arises.

In the first execution, the tool ends with the argument that all the parameters evaluated do not seem injectable, which suggests increasing the level and the risk if it is desired to conduct more tests.

If it is suspected of any type of protection mechanism involved (for example, WAF), it could be used the option '–tamper' (e.g. '–tamper=space2comment') and/or change it by '–random-agent'. The adjustment in test 3 with the configuration of the LEVEL=5 parameter was necessary for SQLMap to carry out the vulnerability test of cookies.

According to [21], the detection and prevention becomes a difficult task if the concept of this type of attacks is not appropriately understood. Carrying out binary evaluations, as proposed by [29], might be considered as a mitigation alternative, since it is an extremely automated method that detects and blocks SQL injection attacks in web applications.

When facing blind SQL injection attacks, the most popular technique is AMNESIA (Analysis and Monitoring for Neutralizing SQL-injection Attacks) [30], which is a tool only applicable to protect applications based on JAVA and which use monitoring in runtime [21]. This tool uses machine learning algorithms to provide prevention and detection mechanisms of blind SQL injection threats. Another prevention mechanism is the pattern identification algorithm, proposed by Aho-Corasick [31], which has two phases: i) a static phase and ii) a dynamic phase.

According to [32], in the static phase the SQL queries generated by the user are compared with a list of patterns that contain a sample of the best-known attack patterns. If the SQL sentence agrees exactly with one of the patterns given in the list of static patterns, it means that an SQL attack is being attempted.

Another alternative proposed by [31] citing [33] is the SQLRand, where the basic idea is to generate SQL sentences using random commands in which the query template within the application may be randomized. In this way, the SQL commands that are injected by malicious users are not coded, because the proxy does not recognize the commands injected, thus causing that the attack is not carried out.

In addition, [31] mentioning [34] indicates that there exists an additional method known as Query Tokenization Method, in which a token is generated of both the original query and the query with injection. Then, the tokens resulting from this process are stored in an array. The lengths of array obtained from the original query and from the query with injection are compared, and if there is a coincidence there is no attempt of SQL injection, otherwise, it is an attack.

The proposal [35] is added to the list of mitigation options, which consists in a grammar tree validation approach, represented in a sentence. To grammatically analyze a sentence requires knowledge of the grammar of the language in which the sentence is written. In such a way, when the attacker injects a malicious SQL query, then the grammar tree of the original query and of the query with injection do not coincide. In this technique, the sentence in particular and the original sentence are compared in runtime.

It is also important to indicate that secure coding is crucial for the design of software and computing systems, aspect which is omitted by developers [12] largely due to the lack of knowledge of secure coding standards, negligence and performance loss, to which it is added usability situations [36].

5. Conclusions

One of the most notable features of Oracle APEX is that of creating sessions with cookies and URL links to the software with random data. The tests conducted with the execution of the different options of the commands indicated in this technique, did not enable to undertake the software with the SQL injection technique to a solution developed in this platform. Although a cookie may be captured with Burp Suite, to decipher it takes a considerable time. However, during that time, Oracle APEX already dynamically generated a new cookie, making an attack through this technique basically impossible.

The contribution of this paper has been to evaluate different SQL injection techniques to emphasize on safe code overwriting, optimize the labels generated by default, thus improving the level of security of an application developed in Oracle APEX, without forgetting that tests have been developed in a time span, without exempting latent vulnerabilities that may appear on day 0.

References

[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