Implementing SQL Injection Vulnerability Assessment of an E-commerce Web Application using Vega and Nikto Tools

Abraham E. Evwiekpaefe[1] and Isacha Habila[2]

[1,2]Department of Computer Science,

Nigerian Defence Academy,

Kaduna,

Nigeria.

Email: [1]aeevwiekpaefe@nda.edu.ng, [2]isachahabila@nda.edu.ng

ABSTRACT

This work employed the use of Vega and Nikto web application vulnerability assessment tools to assess Structured Query Language (SQL) injection vulnerability in an e-commerce web application. Nikto which runs on a Kali-linux was used to identify the server and to test for Secured Sockets Layer (SSL) and Web Application Firewall (WAF). To demonstrate how to exploit web applications using SQL injection commands, a Damn Vulnerable Web Application (DVWA) was used which was installed on a localhost server and the procedures were carried out. Vega tool was utilized to scan and identify SQL injection vulnerabilities. The result shows that out of 43 high-risk vulnerabilities, 32 pages are vulnerable to SQL injection. Among the medium category, vulnerabilities such as auto-complete password field and file upload path were found. The findings further showed that the web pages are vulnerable as a result of poor security coding convention of the web pages. Web applications need utmost security especially the e-commerce web applications. Therefore, this work recommended that web application developers should have the working knowledge of common web application vulnerabilities especially SQL injection as published by Open Web Application Security Project (OWASP) foundation and how to prevent them so that they can develop secured web applications that cannot be exploited easily.

Keywords: *Nikto, SQL Injection, Vega, Vulnerability Assessment, web, web application.*

African Journal of Computing & ICT Reference Format

Abraham E. Evwiekpaefe and Isacha Habila (2021), Implementing SQL Injection Vulnerability Assessment of an E-commerce Web Application using Vega and Nikto Tools,
Afr. J. Comp. & ICT, Vol. 14, No. 1, pp. 1 – 8.

1. INTRODUCTION

SQL injection is a technique that is used to exploit database-driven applications, in which malicious SQL statements are inserted into an entry field for execution. For example, to dump the database contents to the attacker. SQL injection is mostly known as an attack vector for web applications but can be used to attack any type of SQL database [1]. SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server. It was reported that the average web application received four attack campaigns per month, and retailers received twice as many attacks as other industries [2]. SQL Injection (SQLi) is a type of injection attack that allows malicious SQL commands to be executed. These commands are used to control a database server that is connected to a web application.

SQL Injection flaws can be used by attackers to break application security protections. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. SQL injection can also be used to add, modify, and delete records in the database. According to [3] the internet made it possible to access all kinds of information. The rise in the number of web applications gave the internet high usage in the late 1990s. Although these web applications existed before 1980s, but they became more complex by the turn of the century especially with the advent of mobile devices. In the study, it was found out that almost all the participants experienced a web application breach within only one year. It was suggested that as threat awareness increases, web security researchers should move towards implementing an action plan for securing web applications.

Insecure software is damaging our financial, healthcare, defense, energy, and other key infrastructure. As our software gets increasingly complicated, and networked, the difficulty of achieving application security increases rapidly. The fast pace of modern software development processes necessitates identifying and resolving the most prevalent risks quickly and accurately. The term "OWASP" means (Open Web Application Security Project). For developers and penetration testers, the OWASP Top 10 is a common awareness document. It reflects widespread agreement on the most serious security threats to web applications. This document should be adopted by companies, and they should begin the process of ensuring that their online applications minimize these risks. In the area of web application security, OWASP makes publications, tools, techniques, documentation, and technology available to the public [4]. The use of open-source components in software development has become prevalent.

Therefore, this work assessed SQL Injection vulnerability by utilizing a simulation package Damn Vulnerable Web Application (DVWA), which is a PHP/MySQL web application which runs on a localhost whose main goal is to provide a legal environment for security professionals to test their skills and tools with respect to web application security. It also helps web application developers to better understand the processes of securing web applications and aids learning web application security. Vega and Nikto tools were used to assess an online e-commerce web application and provided necessary information for proper remediation which will enhance web security.

2. LITERATURE REVIEW

According to [5] SQL Injection Attacks are considered as a massive threat because it injects into web applications and accesses a database connected to a particular web application. The data can be highly confidential and can be of a very high value such as financial transactions or list that may reveal a lot of secret information. An unauthorized access to such important database by a hacker can result in the threat to CIA i.e Confidentiality, Integrity and Authority of the database. In the work, UNION-based SQL injection commands were used to fetch data from a school website.

[6] provided a new testing approach for vulnerability assessment of web applications by analyzing and using a combined set of tools to address a wide range of security issues. They demonstrated vulnerability assessment tests of a web application (on a localhost) by using combination of W3AF and Nikto tools. By using the two, vulnerability testing coverages for web applications can be increased considering the OWASP Top 10 based threat modelling of web applications.

According to [7] Web applications are widely used in client/server communication model. It provides a platform for attackers to hack into the database; therefore, their security becomes a major issue. SQL Injection attacks are top ten threats in web application. Every three years, Open Web Application Security Project (OWASP) releases top ten lists of most dangerous security flaws in web applications with SQL Injection always taking the first position. An SQL injection attack compromises the interactive web-based applications, running database in the backend. The applications provide a form to accept user input and convert it into the SQL statement and send the same to the database. The attackers change the structure of SQL statement by manipulating user inputs. The paper contains the design approach of a parallel algorithm for the detection of SQL injection. The Algorithm uses the concept of Hot Query Bank (HQB) to cooperate with the existing SQLIA detectors (such as AMNESIA, SQLGuard, etc) and enhances the system performance. It simply keeps the information of previously verified queries in order to skip the verification process on the next appearance. The system performance has been observed by conducting a series of experiments on multi core processors. The experimental results have shown that parallel-SQLIA detector was 65% more efficient in terms of time complexity. There are many SQLIA detectors that were proposed to prevent the occurrence of SQL Injection attacks. These detectors are either used to detect the vulnerable sources of SQL Injections, or blocks the malicious user inputs.

[8] used acunetix web application vulnerability assessment tool to assess 10 Ministries, Departments and Agencies (MDAs) websites. It was found out that there are vulnerabilities in all websites with different degrees of security risks. They analyzed the vulnerabilities based on OWASP top 10 in order to understand and identify the impact of these vulnerabilities on the web applications. They found out that majority of these vulnerabilities belong to informational risk groups which are classified as high, low, informational with the percentages 17.53%, 27.88%, 45.82% respectively.

According to [9] the web is now an important means of transacting business and without security, websites cannot thrive in today's complex computer ecosystem as there are new threats emerging as old ones are being tackled. Vulnerability assessment of websites is one of the means by which security can be improved on websites. The research work seeks to study and use vulnerability assessment as a tool to improve security by identifying vulnerabilities and proposing solutions to solve the security issues. Assessment was done on 5 web hosts belonging to different institutions in Ghana. Nmap, Nikto and Nessus were the tools used for the assessment, the assessment was carried out in four stages, and the first stage in the assessment was planning which involved activities and configurations performed before the actual assessment. The second stage was information gathering which involved obtaining information about the targets necessary to help identify vulnerabilities. This was followed by vulnerability scanning to identify vulnerabilities on the target hosts. The results indicated all the five hosts had security flaws which needed to be addressed.

[10] evaluated a case study for conducting security assessment on a dummy application (college institution website/ complaint logging website) for getting a basic understanding of how to do vulnerability assessment and penetration testing in a web application by using tools and without tools A number of tools such as SQLmap, Acunetix, BurpSuite, IronWasp, NMAP along with manual assessment were used on an application to identify vulnerabilities. It was found out that each tool/method has its own limitation. Some are specialized for acting upon a particular issue. The SQLmap is the best tool for detecting various types of injection attacks which was used to detect Time-Based SQL injection in the dummy application. He further proposed that, multiple tools are needed to find or verify different types of vulnerability issues alongside manual assessment.

[11] studied the opensource vulnerability scanners for vulnerabilities in web applications where they evaluated OWASP top 10 vulnerabilities with three vulnerability scanners w3af, Skipfish and OWASP Zed Attack Proxy on vulnerable applications. They scanned the targeted vulnerable web applications using the three scanners and compared their running time to identify the tool that worked efficiently with minimal running time. The study found out that OWASP ZAP outperformed the other scanners.

[12] proposed detection and prevention methods for SQL Injection attacks such as JDBC (Java Database Connectivity) checker which is an SQLI attack prevention and detection technique that used Java APIs and SAFELI (Static Analysis Framework for discovering SQL Injection) a static analysis framework to detect SQL injection vulnerabilities. The goal of SAFELI framework is to identify the SQL injection attacks during compile time not at run time. It uses white-box static analysis that involves reviewing the code in order to find out if there exists any possible defect in the code. SQL DOM (Document Object Model) is an SQLI prevention technique that uses manual defense coding practice to defeat SQLI attacks. These attacks are still significant threats to back-end DBMS (Data Base Management Systems). Web developers therefore need to be aware of these attacks so as to make their web applications secure. Their work also examined a tool called SQLmap used in Kali Linux operating system which is mostly used to assess the SQLI vulnerability of web applications. The use of prepared statements and bind parameter are highly encouraged at application level to further prevent SQL Injection.

[13] examined the reported vulnerabilities over the recent years, with SQL injection (SQLi) being one of the most prominent, especially in web applications. They presented a Deep learning model that was able to classify PHP slices as vulnerable or not vulnerable to SQL Injection. They proposed an intermediate language to represent the PHP slices which interpreted them as text and resorted to well-studied Natural Language Processing (NLP) techniques as slices can belong to any variant. The use of the model showed that it can discover SQLi which in turn helps programmers to write codes that are not vulnerable to various attacks.

[14] reported that web vulnerability assessment tools have become a necessity due to the increased security threats posed by hackers and other criminals who are on the lookout for confidential information. They proposed web assessment tool called SNEAKERZ that detected vulnerability and automatically analyzed exploitable web vulnerabilities and also proposed solutions for that vulnerability. The proposed solutions looked for security loopholes in web applications to prevent hackers from gaining unauthorized access to corporate information and data.

The extant review of literature shows that much work has been done on SQL injection vulnerability assessment using various tools such as SQLMap, Burpsuit, OWASP ZAP etc. However, not much work has been done within the Nigerian context to assess e-ccommerce web applications available today and none of the works used Vega tool for assessment.

3. METHODOLOGY

In this work, Vega and Nikto were used together to assess the vulnerabilities in an e-commerce web application. DVWA was also used to show how to use SQL injection commands to test for vulnerabilities in web applications on a localhost.
Automated tools can be used for vulnerability assessment of web applications. Many tools can be combined to assess a particular web application [15] . Each vulnerability tool can be used for a specific vulnerability assessment based on OWASP top 10 web application vulnerabilities. After the assessment, the results can be used to take actions to enhance a web application's security. This is typified in Figure 1.

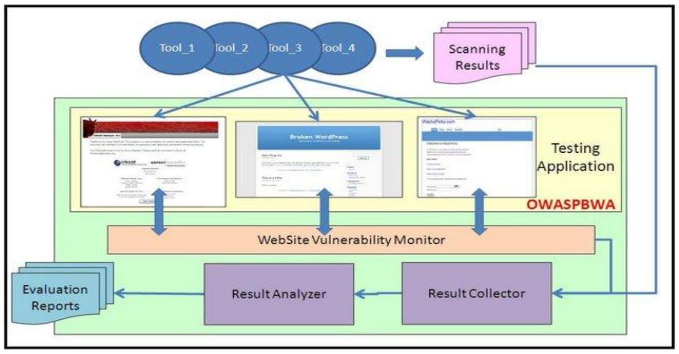


Figure 1. Automated SQL vulnerability Assessment tools procedure [15]

Vega Assessment Tool.

Vega is a free and open-source web security scanner and web security testing platform to test the security of web applications. Vega can help you find and validate SQL Injection, Cross-Site Scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities. It is written in Java, GUI based, and runs on Linux, OS X, and Windows. Vega can help find vulnerabilities such as: reflected cross-site scripting, stored cross-site scripting, blind SQL injection, remote file inclusion, shell injection, and others. Vega also probes for TLS / SSL security settings and identifies opportunities for improving the security of your TLS servers.
Vega includes an automated scanner for quick tests and an intercepting proxy for tactical inspection. The Vega scanner finds XSS (cross-site scripting), SQL injection, and other vulnerabilities. Vega can be extended using a powerful API in the language of the web: JavaScript. [16]

VEGA assessment is used to perform and find vulnerabilities in web applications. The new scan label button can be used to get an interface where you can enter a web url then hit the finish button as shown in Figure 2. After which it runs the scan and provides report about the scan.

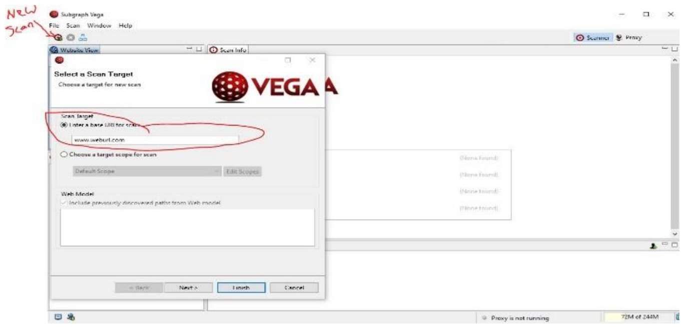


Figure 2 Showing Vega interface.

Nikto Assessment Tool in Kali Linux

Nikto is an automated web application assessment or penetration test tool in a Kali-Linux Operating system as shown in Figure 3. This command can be achieved by clicking on the Nikto tool and entering the following instructions one at a time:

1. Nikto -h <website url>
2. Nikto -h <website url> -ssl

The first command checks for server vulnerability which uncovers server misconfigurations. While the second command checks whether the web application is secured enough for communication over the internet. SSL which stands for Secure Sockets Layer, authenticates a websites identity and creates an encrypted link between a web server and a web browser.

Figure 3 Kali showing Nikto and other tools.

Crafted SQL Injection commands

This form of injection relies on the fact that SQL statements consist of both data used by the SQL statement and commands that control how the SQL statement is executed. The attacker sends SQL injection commands from a browser with the intention of getting valuable information stored in a database for exploitation as seen in Figure 4.

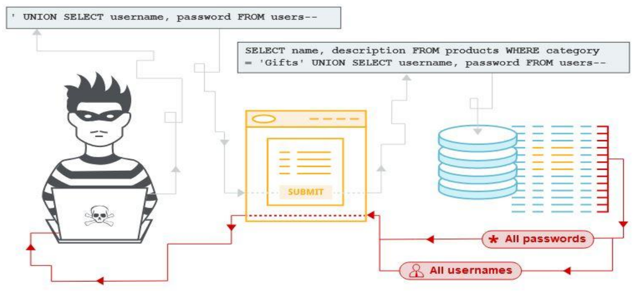
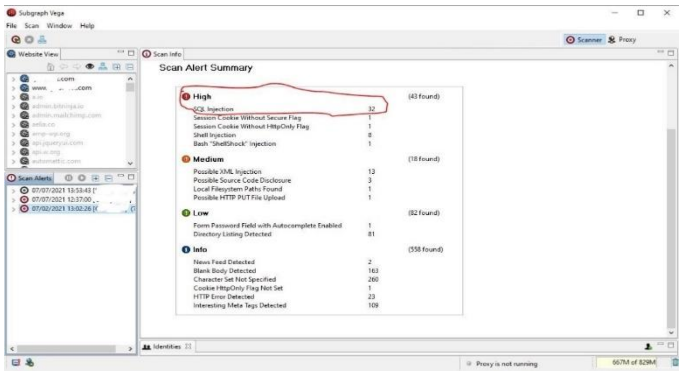

' UNION SELECT username, password FROM users--
SELECT name, description FROM products WHERE category = 'Gifts' UNION SELECT username, password FROM users--
SUBMIT
All passwords
All usernames

Figure 4 SQL Injection Architecture: using SQL injection commands [17]

4. RESULTS

Results from using Vega scanning tool

The e-commerce web application was scanned using vega assessment tool and 32 SQL injection vulnerabilities were found as shown in Figure 5.



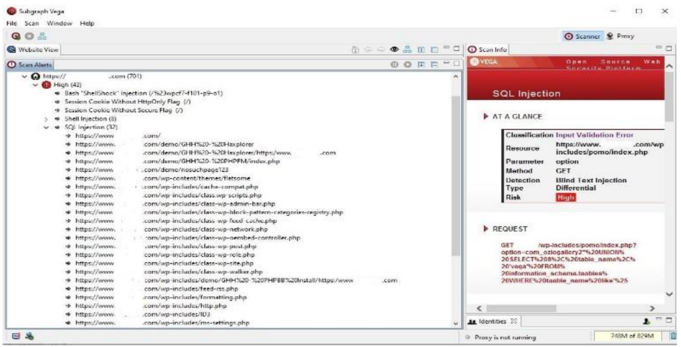
Scan Alert Summary

High		(43 found)
SQL Injection	32	
Session Cookie Without Secure Flag	1	
Session Cookie Without HttpOnly Flag	1	
Shell Injection	8	
Bash "ShellShock" Injection	1	
Medium		(18 found)
Possible XML Injection	13	
Possible Source Code Disclosure	3	
Local Filesystem Paths Found	1	
Possible HTTP PUT File Upload	1	
Low		(82 found)
Form Password Field with Autocomplete Enabled	1	
Directory Listing Detected	81	
Info		(558 found)
News Feed Detected	2	
Blank Body Detected	163	
Character Set Not Specified	260	
Cookie HttpOnly Flag Not Set	1	
HTTP Error Detected	23	
Interesting Meta Tags Detected	109	

```
07/07/2021 13:53:43
07/07/2021 12:37:00
07/02/2021 13:02:26
Proxy is not running    667M of 829M
```

Figure 5 Showing SQL injection vulnerability results in Vega.

Vega shows the various web pages that are vulnerable to SQL injection attacks. These web pages individually have a loophole that can be exploited. Vega treats SQL injection as "high risk" vulnerability because it has something to do with the application's database as shown in Figure 6.



```
https://          .com (701)
  High (43)
    Bash "ShellShock" Injection (/%23wpcf7-f101-p9-o1)
    Session Cookie Without HttpOnly Flag (/)
    Session Cookie Without Secure Flag (/)
    Shell Injection (8)
    SQL Injection (32)
      https://www        .com/
      https://www.       .com/demo/GHH%20-%20Haxplorer
      https://www.       .com/demo/GHH%20-%20Haxplorer/https:/www.          .com
      https://www        .com/demo/GHH%20-%20PHPFM/index.php
      https://www        .com/demo/nosuchpage123
      https://www.       .com/wp-content/themes/flatsome
      https://www.       .com/wp-includes/cache-compat.php
      https://www        .com/wp-includes/class.wp-scripts.php
      https://www        .com/wp-includes/class-wp-admin-bar.php
      https://www        .com/wp-includes/class-wp-block-pattern-categories-registry.php
      https://www        .com/wp-includes/class-wp-feed-cache.php
      https://www.       .com/wp-includes/class-wp-network.php
      https://www.       .com/wp-includes/class-wp-oembed-controller.php
      https://www.       .com/wp-includes/class-wp-post.php
      https://www        .com/wp-includes/class-wp-role.php
      https://www        .com/wp-includes/class-wp-site.php
      https://www        .com/wp-includes/class-wp-walker.php
      https://www        .com/wp-includes/demo/GHH%20-%20PHPBB%20Install/https:/www          .com
      https://www        .com/wp-includes/feed-rss.php
      https://www        .com/wp-includes/formatting.php
      https://www.       .com/wp-includes/http.php
      https://www.       .com/wp-includes/ID3
      https://www.       .com/wp-includes/ms-settings.php
```

SQL Injection

AT A GLANCE

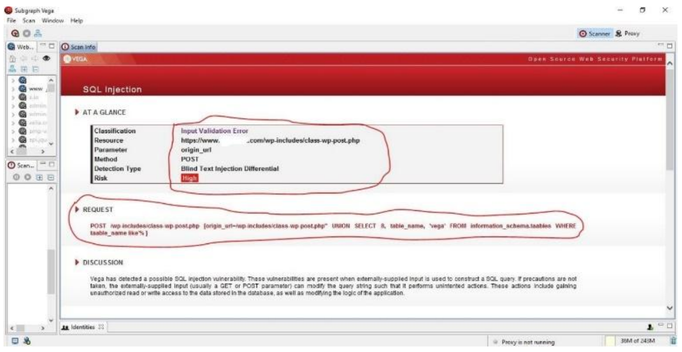
Classification	Input Validation Error
Resource	https://www. .com/wp includes/pomo/index.php
Parameter	option
Method	GET
Detection Type	Blind Text Injection Differential
Risk	High

REQUEST

```
GET        /wp-includes/pomo/index.php?
option=com_ozlogallery2"%20UNION%
20SELECT%208%2C%20table_name%2C%
20'vega'%20FROM%
20information_schema.taables%
20WHERE%20taable_name%20like'%25
```

Figure 6 Showing SQL injection vulnerability in each page in Vega

A particular web page can be vulnerable to SQL injection. Each page can be checked for vulnerability as shown in Figure 7.



SQL Injection

AT A GLANCE

Classification	Input Validation Error
Resource	https://www. .com/wp-includes/class-wp-post.php
Parameter	origin_url
Method	POST
Detection Type	Blind Text Injection Differential
Risk	High

REQUEST

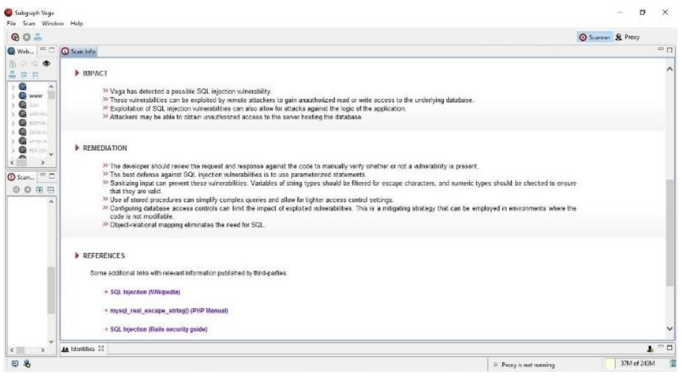
```
POST /wp-includes/class-wp-post.php [origin_url=/wp-includes/class-wp-post.php" UNION SELECT 8, table_name, 'vega' FROM information_schema.taables WHERE taable_name like'% ]
```

DISCUSSION

Vega has detected a possible SQL injection vulnerability. These vulnerabilities are present when externally-supplied input is used to construct a SQL query. If precautions are not taken, the externally-supplied input (usually a GET or POST parameter) can modify the query string such that it performs unintended actions. These actions include gaining unauthorized read or write access to the data stored in the database, as well as modifying the logic of the application.

Figure 7: Showing SQL injection vulnerabilities results in every webpage in Vega

Vega shows the impact the vulnerability can have on a web application and remediation. The remediation suggests vital information on what to be done in order to fix the problem as shown in Figure 8.



IMPACT

- Vega has detected a possible SQL injection vulnerability.
- These vulnerabilities can be exploited by remote attackers to gain unauthorized read or write access to the underlying database.
- Exploitation of SQL injection vulnerabilities can also allow for attacks against the logic of the application.
- Attackers may be able to obtain unauthorized access to the server hosting the database.

REMEDIATION

- The developer should review the request and response against the code to manually verify whether or not a vulnerability is present.
- The best defense against SQL injection vulnerabilities is to use parameterized statements.
- Sanitizing input can prevent these vulnerabilities. Variables of string types should be filtered for escape characters, and numeric types should be checked to ensure that they are valid.
- Use of stored procedures can simplify complex queries and allow for tighter access control settings.
- Configuring database access controls can limit the impact of exploited vulnerabilities. This is a mitigating strategy that can be employed in environments where the code is not modifiable.
- Object-relational mapping eliminates the need for SQL.

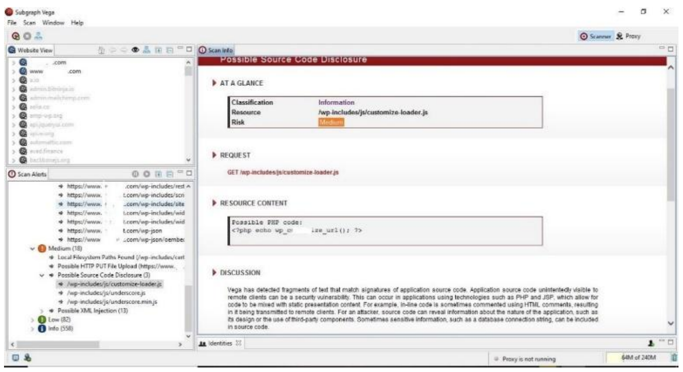
REFERENCES

Some additional links with relevant information published by third-parties:

- SQL Injection (Wikipedia)
- mysql_real_escape_string() (PHP Manual)
- SQL Injection (Rails security guide)

Figure 8 Showing SQL injection vulnerability impact and remediation page in Vega

Veg shows that there is a possible source code disclosure and the content is shown.
This can give the hacker an opportunity to hover around the web application as shown in Figure 9.



```
Medium (18)
  Local Filesystem Paths Found (/wp-includes/cert
  Possible HTTP PUT File Upload (https://www.
  Possible Source Code Disclosure (3)
    /wp-includes/js/customize-loader.js
    /wp-includes/js/underscore.js
    /wp-includes/js/underscore.min.js
  Possible XML Injection (13)
Low (82)
Info (558)
```

Possible Source Code Disclosure

AT A GLANCE

Classification	Information
Resource	/wp-includes/js/customize-loader.js
Risk	Medium

REQUEST

```
GET /wp-includes/js/customize-loader.js
```

RESOURCE CONTENT

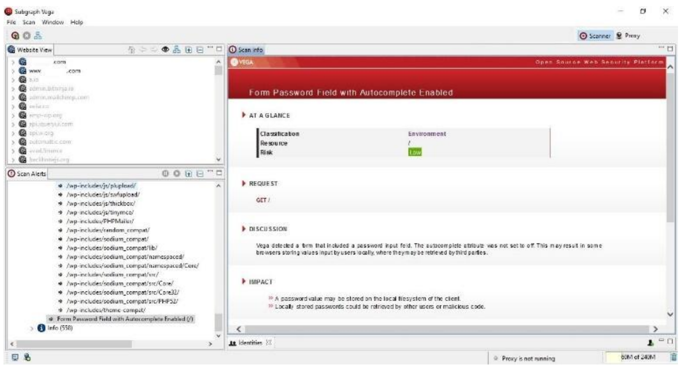
```
Possible PHP code:
<?php echo wp_c      ize_url(); ?>
```

DISCUSSION

Vega has detected fragments of text that match signatures of application source code. Application source code unintentedly visible to remote clients can be a security vulnerability. This can occur in applications using technologies such as PHP and JSP, which allow for code to be mixed with static presentation content. For example, in-line code is sometimes commented using HTML comments, resulting in it being transmitted to remote clients. For an attacker, source code can reveal information about the nature of the application, such as its design or the use of third-party components. Sometimes sensitive information, such as a database connection string, can be included in source code.

Figure 9 Showing Possible Source-code Disclosure in Vega

Vega shows that there is a form password field with autocomplete enabled which can be very helpful for exploitation of the system by the attacker as shown in Figure 10.


Form Password Field with Autocomplete Enabled

AT A GLANCE

Classification	Environment
Resource	/
Risk	Low

REQUEST

GET /

DISCUSSION

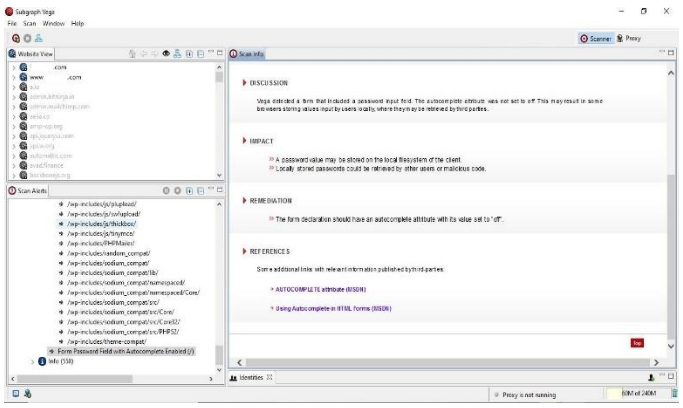
Vega detected a form that included a password input field. The autocomplete attribute was not set to off. This may result in some browsers storing values input by users locally, where they may be retrieved by third parties.

IMPACT

» A password value may be stored on the local filesystem of the client.
» Locally stored passwords could be retrieved by other users or malicious code.

Figure 10 Showing Form Password Field with Autocomplete Enabled in Vega

Vega shows the impact; that a password may be stored on the local file system of the client which could be easily retrieved by other users or malicious code as shown in Figure 11


DISCUSSION

Vega detected a form that included a password input field. The autocomplete attribute was not set to off. This may result in some browsers storing values input by users locally, where they may be retrieved by third parties.

IMPACT

» A password value may be stored on the local filesystem of the client.
» Locally stored passwords could be retrieved by other users or malicious code.

REMEDIATION

» The form declaration should have an autocomplete attribute with its value set to "off".

REFERENCES

Some additional links with relevant information published by third-parties.

* AUTOCOMPLETE attribute (MSDN)
* Using Autocomplete in HTML forms (MSDN)

Figure 11: Showing Impact and Remediation for Autocomplete Password field in Vega

Results from Nikto Scanning tool

The results obtained by running the command **#nikto -h <url>** which discloses the server type as Apache and some vital information regarding whether the website has some protection or not as shown in Figure 12.

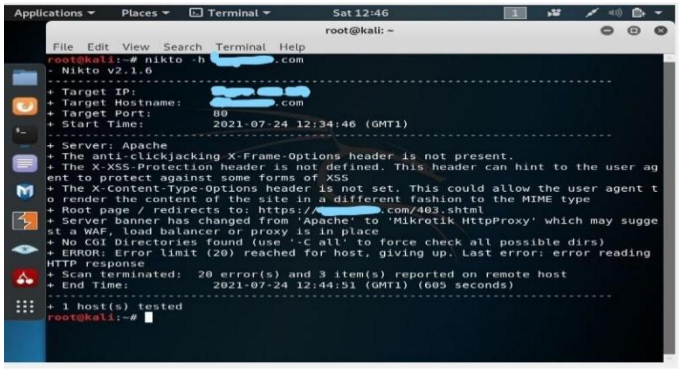

```
root@kali:~# nikto -h [redacted].com
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          [redacted]
+ Target Hostname:    [redacted].com
+ Target Port:        80
+ Start Time:         2021-07-24 12:34:46 (GMT1)
---------------------------------------------------------------------------
+ Server: Apache
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user ag
ent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent t
o render the content of the site in a different fashion to the MIME type
+ Root page / redirects to: https://[redacted].com/403.shtml
+ Server banner has changed from 'Apache' to 'Mikrotik HttpProxy' which may sugge
st a WAF, load balancer or proxy is in place
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ ERROR: Error limit (20) reached for host, giving up. Last error: error reading
HTTP response
+ Scan terminated:  20 error(s) and 3 item(s) reported on remote host
+ End Time:           2021-07-24 12:44:51 (GMT1) (605 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
root@kali:~#
```

Figure 12: Showing server report and possible vulnerability Nikto

The results obtained by running a command **#nikto -h** <url> **-ssl** shows that the site uses SSL for secured end to end communication on over the internet as shown in Figure 13.

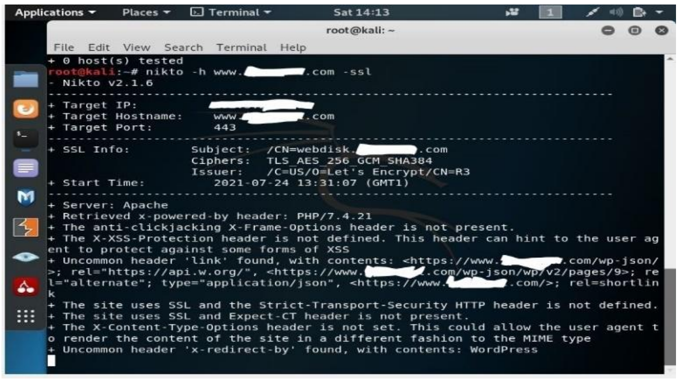
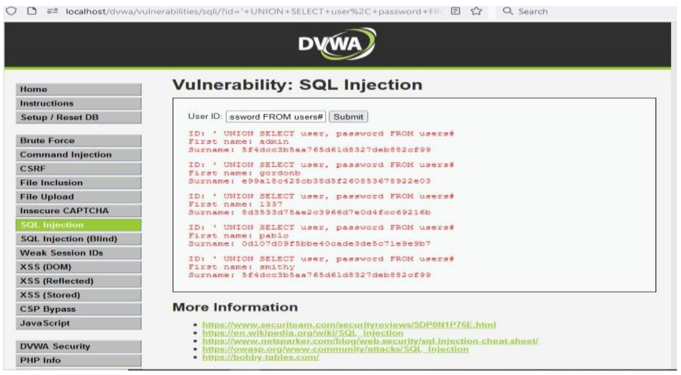

```
+ 0 host(s) tested
root@kali:~# nikto -h www.[redacted].com -ssl
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          [redacted]
+ Target Hostname:    www.[redacted].com
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /CN=webdisk.[redacted].com
                   Ciphers:  TLS_AES_256_GCM_SHA384
                   Issuer:   /C=US/O=Let's Encrypt/CN=R3
+ Start Time:         2021-07-24 13:31:07 (GMT1)
---------------------------------------------------------------------------
+ Server: Apache
+ Retrieved x-powered-by header: PHP/7.4.21
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user ag
ent to protect against some forms of XSS
+ Uncommon header 'link' found, with contents: <https://www.[redacted].com/wp-json/
>; rel="https://api.w.org/", <https://www.[redacted].com/wp-json/wp/v2/pages/9>; re
l="alternate"; type="application/json", <https://www.[redacted].com/>; rel=shortlin
k
+ The site uses SSL and the Strict-Transport-Security HTTP header is not defined.
+ The site uses SSL and Expect-CT header is not present.
+ The X-Content-Type-Options header is not set. This could allow the user agent t
o render the content of the site in a different fashion to the MIME type
+ Uncommon header 'x-redirect-by' found, with contents: WordPress
```

Figure 13 Showing SSL report in Nikto

SQL Injection vulnerability results in DVWA

DVWA which runs on a localhost was used to demonstrate the SQL injection by using the commands: ` UNION SELECT user and password FROM users# to fetch user's information from the database as shown in Figure 14.


localhost/dvwa/vulnerabilities/sqli/?id='+UNION+SELECT+user%2C+password+FR

Vulnerability: SQL Injection

User ID: ssword FROM users# Submit

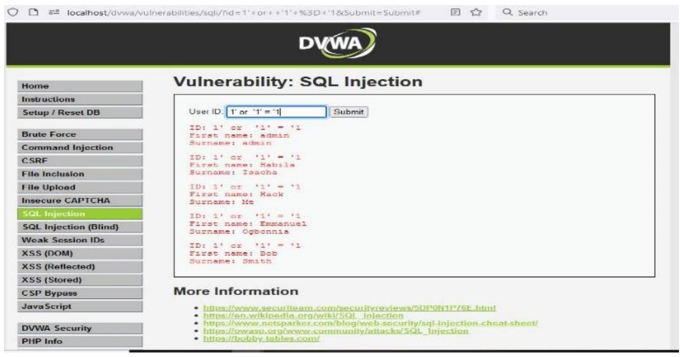
```
ID: ' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

More Information

* https://www.securiteam.com/securityreviews/5DP0N1P76E.html
* https://en.wikipedia.org/wiki/SQL_injection
* https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
* https://owasp.org/www-community/attacks/SQL_Injection
* https://bobby-tables.com/

Figure 14: showing SQL injection vulnerability results in DVWA

The results obtained in DVWA by entering the command "1` or `1` = `1" shows how powerful the command is and it has the capacity to retrieve information from any database server if not properly configured or coded as shown in Figure 15.



Vulnerability: SQL Injection

```
ID: 1' or  '1' = '1
First name: admin
Surname: admin

ID: 1' or  '1' = '1
First name: Habila
Surname: Isacha

ID: 1' or  '1' = '1
First name: Hack
Surname: Me

ID: 1' or  '1' = '1
First name: Emmanuel
Surname: Ogbonnia

ID: 1' or  '1' = '1
First name: Bob
Surname: Smith
```

More Information

- https://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Figure 15: showing SQL injection vulnerability results in DVWA

5. DISCUSSION OF RESULTS

The results obtained showed that the web application scanned is vulnerable to SQL injection. 32 SQL injection vulnerabilities were found alongside other form of vulnerabilities. This shows that the web application under question can be exploited by a hacker. These injections are found by scanning a website index url and it fuzzes to the entire pages of the web applications. Each web page that is vulnerable is being reported. SQL injection vulnerabilities can be seen in the pages by clicking on each page. The tools used include the Nikto which runs on Kali Linux. Nikto reveals the type of server the web application is running on which is the first step to penetration testing. The e-commerce web application runs on Apache server as revealed by the Nikto. Nikto also revealed the state of the website if it is secured with SSL (secure socket layer) which provide secure communication on the internet. It was observed that the website used SSL which indeed provides secured communication between client-server devices. It is also observed that the web application was protected by using a web application firewall which helped filter data that goes in and out of the database. This step helped reveal if a web application's server can easily be shut down or not. This step is popularly known as information gathering.

After this, the Vega tool was used to scan for vulnerability in the web application to check if it is vulnerable to SQL injection. This tool is considered as one of the best tools for SQL injection vulnerability assessment. The results obtained shows a massive SQL injection vulnerability and other related vulnerabilities. Out of the 43 high vulnerabilities, 32 indicated vulnerability of SQL injection. This showed that the web application as highly exploitable. It is a database application and if a hacker can fetch the data from the database, then people's information is at stake. It was also observed that in each web page that could be exploitable, most are sensitive data. Other forms of vulnerabilities showed that there was auto-complete password, username field and file path. This is very serious as the attacker can monitor those using the web application and try to follow up to utilize the autocomplete field by inspecting element in a web page to harvest login credentials. Another issue is that if that path for file upload is not tackled, an attacker can upload a dangerous file to a folder capable of either corrupting people's information or stealing their information. E-commerce web applications require utmost security at all levels. Even though most of the web applications today use email verification for improved security and also use an encrypted password, if these passwords are accessed, they can easily be decrypted or deciphered. Therefore, the necessary steps should be adopted to resolve these vulnerabilities.

In order to demonstrate the use of SQL injection commands in web applications for exploitation, a DVWA is used. Using the commands, such as union, 'OR' 1"= "1" and UNION based on different levels of security which can be set to low, medium and high some SQL injection commands gave or returned the username and password. The DVWA is a good vulnerable web application to test and to learn SQL injection commands. It is primarily built for penetration testing and for simulation purposes to educate developers and cybersecurity practitioners on web application vulnerabilities. Using some commands, it is possible to fetch stored information from the database and to be displayed on the browser interface.

6. CONCLUSION

This work examined various ways to perform SQL injection vulnerability assessments, previous works and what needs to be done in order to prevent it. It showed the steps to take in order to perform vulnerability assessments of web applications. Web applications are prone to attacks if not properly scanned for vulnerabilities and fixed or if they are not well coded. A simulation was conducted using DVWA to show how SQL injection commands can be used to exploit a web application. Automated tools were used to scan for vulnerabilities in web applications; the tools include VEGA which is used for deep web application vulnerability assessment online and Nikto from Kali Linux operating system. The results obtained from the online were discussed and how it can be addressed. Many web applications are running online. These web applications can be exploited.

Series of attacks are being lunched on web applications to steal vital information belonging to individuals or organizations especially database web applications. Web applications needs to be regularly scanned to identify vulnerabilities and address them where necessary. It is very important because the number of hackers are on the rise. Therefore organizations, individual marketing web applications needs to be proactive about this so as to protect their data from being stolen. Therefore, it is recommended that for the purpose of building web applications, developers should be familiar with SQL injection commands and test them on their web applications on a localhost before hosting them online. Also, developers should be familiar with penetration testing and vulnerability assessment tools so that they can apply these tools periodically to check for vulnerabilities.

REFERENCES

[1] M. Stampar, "Data retrieval over DNS in SQL injection attacks". arXiv preprint arXiv:1303.3047, 2013. Retrieved on 31st July, 2021 from https://arxiv.org/abs/1303.3047.

[2] Imperva. "SQL Injection Attacks: So Old, but Still So Relevant", 2012. Retrieved on 30th July, 2021 from https://www.imperva.com/blog/sql-injection-attacks-so-old-but-still-so-relevant-heres-why-charts/

[3] RSI Security, "The Importance of Having a Web Application Vulnerability Management Plan", 2019. Retrieved on 30th July, 2021 from https://www.google.com/amp/s/blog.rsisecurity.com/the-importance-of-having-a-web-application-vulnerability-management-plan/amp/

[4] OWASP, "OWASP Top Ten", 2021. Retrieved on 30th July, 2021 from https://owasp.org/www-project-top-ten/

[5] D. H. Parekh, M. D. Dave, and R. Sridaran, Live Experiments depicting SQL Injection Attacks. International Journal of Advanced Networking & Applications, pp. 91-93, 2014.

[6] R. Vibhandik, and A. K. Bose, "Vulnerability assessment of web applications-a testing approach". In 2015 IEEE Fourth International Conference on e-Technologies and Networks for Development (ICeND), pp. 1-6, Sep2015.

[7] P. Kumar and C. P. Katti, "A Parallel-SQLIA Detector for Web Security. I.J. Information Engineering and Electronic Business", 2016. Published Online March 2016 in MECS (http://www.mecs-press.org/), Vol. 2, pp. 66-75, 2016.

[8] I. Idris, M. U. Majigi, S. Abdulhamid, M. Olalere, and S.I. Rambo, "Vulnerability assessment of some key Nigeria government websites. *International Journal of Digital Information and Wireless Communications*, Vol. 7 No 3, pp. 143-153, 2017.

[9] V. Appiah, M. Asante, I.K. Nti, and O. Nyarko-Boateng. "Survey of Websites and Web Application Security Threats Using Vulnerability Assessment", *Journal of Computer Science, Vol.* 15 No. 10, pp. 1341–1354, 2018.

[10] S. Thakre and S Bojewar. "Studying the Effectiveness of Various Tools in Detecting the Protecting Mechanisms Implemented in Web-Applications" In 2018 IEEE International Conference on Inventive Research in Computing Applications (ICIRCA), pp. 1316-1321, Jul2018.

[11] D. Sagar, S. Kukreja, J. Brahma, S. Tyagi and P. Jain. "Studying open source vulnerability scanners for vulnerabilities in web applications", *IIOAB Journal*, Vol. 9, No. 2, pp. 43-49, 2018.

[12] C. V. Gonzalez and G. Jung, "Database SQL injection security problem handling with examples", In 2019 IEEE International Conference on Computational Science and Computational Intelligence (CSCI), pp. 145-149, Dec2019.

[13] A. Fidalgo, I. Medeiros, P. Antunes, and N. Neves, "Towards a deep learning model for vulnerability detection on web application variants". In 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 465-476, 2020.

[14] M. Y. Darus, M.A. Omar, M.F. Mohamad, Z. Seman and N. Awang, "Web Vulnerability Assessment Tool for Content Management System", *International Journal of Advanced Trends in Computer Science and Engineering,* Vol *9, No* 1.3, pp. 440-444, 2020.

[15] Y.H. Tung, S.S. Tseng, J. F. Shih, and H. L. Shan, "A cost-effective approach to evaluating security vulnerability scanner. In 2013 15th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS), pp. 1-3, Sept2013.

[16] Subgraph, "Vega helps you find and fix cross-site scripting (XSS), SQL injection, and more", 2021. Retrieved on 30th July, 2021 from https://subgraph.com/vega/.

[17] PortSwigger, "SQL Injection", 2021. Retrieved on 31st July, 2021 from https://portswigger.net/web-security/sql-injection.