

# SQLI Vulnerability Detection Tool

Astha Sharma

University of New Orleans, USA

Email: asharna6@cs.uno.edu

**Abstract** - SQL injection is a security threat on web applications that are associated with a DBMS. This paper aims to detect and exploit SQL injection vulnerability on a webpage. Here, the response page of a user-fed URL is scrapped for finding the possibility of classic SQL injection. Prediction on vulnerability is made based on the SQL error match between expected and obtained output. The URL is considered to be either vulnerable or benign based on the output of the response page. Further, a third-party library, SQLMap, is used to dig deeper about the all possible SQL injections, the appended payload for test-cases and SQLI affected databases that need to be addressed. This information can be very valuable for basic penetration testing in large projects.

**Keywords** - SQL injection, SQLI detection tool, SQL injection detection

## I. INTRODUCTION

Structured Query Language (SQL) is a standard programming language that used to interact with relational databases. SQL statements can be used to modify the structure of databases and change the contents of databases [1].

According to the Open Web Application Security Project (OWASP), the SQL Injection (SQLI) Attack tops the list of all possible web application attacks [6]. This type of attack is encountered when data from the untrusted source is treated as part of a command or query by an interpreter. The malicious input from attackers can deceive the interpreter and make it execute unintended lines of commands and/ or access hidden data without any kind of authorization.

With growing technology, everything is digitizing. So, for a digital presence, there has to be a webpage/ website which is visible to the online audience. This leads to an increasing number of websites. And as most of the web applications today are using Relational Database Management System (RDBMS), the rate of SQL injection attacks is also rising at the same rate. This is because websites using DBMS are prone to SQL injections. Therefore, it is important for developers to adopt secure coding practices in keep the possible attacks at bay.

But by human nature, we tend to make mistakes and it is totally normal to assume that sometimes even the experts fail to protect or overlook some essential parts on their code, leading to some serious issues that need to be addressed as soon as possible. Also, it is found that defensive coding has not been successful in completely preventing SQLIAs [2]. Therefore, there is a necessity of a reliable detector that lists out the possible breakpoints for testing the security and fault tolerance of the web application. The tool developed in this course project is also intended to serve a part of it.

### A. What is SQL injection?

A SQL Injection attack is a kind of attack based on the user input which isn't validated. There are two major types of attacks, namely first-order and second-order attacks.

The first order is when the result of an attack is received immediately, either as a direct response from the application or some other response mechanism, say email. For example, passing Boolean-based attack "dave' or 1=1; --" for username field in a form can bypass authentication for the password field and can log you in to the system immediately after the execution.

Second-order attacks are the type of attacks when the injected data resides in the database, but the payload will not be immediately activated. The effects will not be shown unless some other functionalities in the program use the injected data without filtering.

For example, let's say there is an account with username = "John" in a website and we want to get logged in to the system as this user. However, we do not have the information about her password. What we can do is, make another user with the username "John --" and password, say "john123"[5]. Now login to the system and perform a change in the password to hacksuccess from john123. The executing sql operation looks like the following:

```
UPDATE users SET password='hacksuccess'
WHERE username='John'-- ' and password='john123'
```

But the query ends up as soon as it sees the double dashes (--). So, nothing beyond that point executes. And the query becomes something like this:

```
UPDATE users SET password='123' WHERE username='John'
```

The query, thus, updated the password for 'John' and not 'John' --'. So, it is now possible to access the user account unless the user explicitly changes the password again [5].

### B. Different types of SQL Injections:

There are many types, but the most common and basics ones are discussed below:

**Blind Injection:** Normally, when a website is vulnerable to SQL injections, it gives back information about the problem that leads to the SQL error when supplied with unusual data. But at times, the system interprets the input data without showing the error message. This makes it difficult for the attacker to interpret the point of vulnerability. However, it is not impossible. In this regard, the attacker can run some set of tests to see the results from the application in terms of true and false.

The blind injection is further divided as (i) Time based blind and (ii) Boolean-based blind

Example of time-based blind SQL injection in MySQL: `http://localhost/test.php?id=sleep(30)`

If this SQL statement is interpreted by the database, then the webpage will take 30 seconds to load. So, the application, if responds accordingly, is SQL injectable.

**Error Based:** This is the basic and most classic form of SQL injection. The first thing to do to check if a site is vulnerable or not is just by adding a single quote ('), or double quote (") or a backslash (\) to the end of the web URL and see if the response of the page. An example in MySQL is as follows: `http://localhost/test.php?id=""`

If error reporting is enabled and this request is vulnerable to SQL injection, then the following error will be produced:

*You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "" at line 5*

This tool is based on the analysis of errors in the different type of DBMS.

**Tautology Based Injection:** Example in MySQL: `http://localhost/test.php?username=' or 1=1 /*&password=1`

In this case, supplying a Tautology, or a statement that is always true provides a predictable result. In this case, the predictable result would be logging in the attacker as the first user in the database, which is commonly the administrator.

**Union Query:** It is another type of SQL injection technique that exploits the UNION sql operator that combines the result of two or more SELECT statements to get a single result [7].

Example:

```
http://www.abcdefabcdef.com/page=7' union select 1, schema_name,3,4,5,6 from information_schema — +
```

The query appended to the URL will show the database name to the attacker, provided that the URL is vulnerable.

### C. The Tool

All input points (sources for user input) in a web application like the URL, search bar, forms, file uploads, etc., can be vulnerable in terms of SQL injection. No user input can be trusted without validation. But if the web application is not validating the user input before processing it, this can result in different kind of threats against confidentiality, integrity, availability and more.

My goal with this tool was to first identify if the web application is susceptible to SQL injections and secondly, find out at which point is it vulnerable. So, I came up with this Python tool that checks if a webpage is approachable by attackers for exploitation.

### D. SQLMap

SQLMap further makes an analysis on the webpage based on its own working module. SQLMAP is one of the most popular and powerful open source tools for SQLI detection. It supports a wide range of database from Oracle, MySQL, to PostgreSQL, Ms-Sql etc. It looks for a good range of SQL injection attacks including Blind SQLs (Boolean-based, and time-based), Error based, Union query based, stack-based and out of bound. It is a great tool for huge projects with a bulky database where it can be difficult to find for vulnerability (unless mentioned explicitly). Figure 1, figure 2, and figure 3 shows the screenshots from the working tool.

```

----- SQL Injection using python -----
[*] Checking for Internet connection
[*] Internet Connection - Checked!
Enter a url to check for vulnerability. : http://www.agankala.com/book_detail.php?bookid=114

----- Checking if url is vulnerable -----


Select a level for error check (0, 1, 2): 2
[*] Connecting

[19:53:09] [*] http://www.agankala.com/book_detail.php?bookid=114
[V] Check1 MySQL found: Query Error:You have an error in your SQL syntax; check the manual that corresponds to your
for the right syntax to use near ''114''' at line 1
[V] Check2 MySQL found: Query Error:You have an error in your SQL syntax; check the manual that corresponds to your
for the right syntax to use near ''114''' at line 1
[V] Check3 MySQL found: Query Error:You have an error in your SQL syntax; check the manual that corresponds to your
for the right syntax to use near ''114''' at line 1
[V] Check4 MySQL found: None
[V] Check5 MySQL found: None
[V] Check6 MS SQL found: None
[V] Check7 MS SQL found: None
[V] Check8 Oracle found: None
[V] Check9 Oracle found: None
[V] Check10 Oracle found: None
[V] Check11 PostgreSQL found: None
[V] Check12 PostgreSQL found: None

Possible vulnerable URL!
[19:53:09] [*] http://www.agankala.com/book_detail.php?bookid=114

```

Fig. 1. SQLI detection tool result for test URL



```

(1,2,11#stable)
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's res
all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damag
ogram

[*] starting at 19:39:58

[19:39:58] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (X11; U; Linux i686; en-US; AppleWebKit/534.16 (K
[19:39:58] [INFO] parsing multiple targets list from 'http://www.agankala.com/book_detail.php?bookid=114'
[19:39:58] [INFO] testing URL 'http://www.agankala.com/book_detail.php?bookid=114'
[19:39:58] [INFO] using 'http://www.agankala.com/book_detail.php?bookid=114' as the CSV results file in multiple t
[19:39:58] [INFO] testing connection to the target URL
[19:39:58] [INFO] checking if the target is protected by some kind of WAF/IPS
[19:39:58] [WARNING] turning off pre-connect mechanism because of connection reset(s)
[19:39:58] [CRITICAL] heuristics detected that the target is protected by some kind of WAF/IPS
do you want sqlmap to try to detect backend WAF/IPS? [Y/N] N
[19:39:58] [WARNING] dropping timeout to 10 seconds (i.e. '--timeout=10')
[19:39:58] [INFO] testing if the target URL content is stable
[19:39:58] [INFO] target URL content is stable
[19:39:58] [INFO] testing if GET parameter 'bookid' is dynamic
[19:39:58] [INFO] GET parameter 'bookid' appears to be dynamic
[19:39:58] [INFO] heuristic (basic) test shows that GET parameter 'bookid' might be injectable (possible DBMS: 'MySQL')
[19:39:58] [INFO] heuristic (XSS) test shows that GET parameter 'bookid' might be vulnerable to cross-site scripting (XSS) at
[19:39:58] [INFO] testing for SQL injection on GET parameter 'bookid'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]
[19:39:58] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'

```

Fig. 2. SQLMap heuristic result for test URL

```

GET parameter 'bookid' is vulnerable. Do you want to keep testing the ot
sqlmap identified the following injection point(s) with a total of 268 H
---
Parameter: bookid (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: bookid=114' AND 8805=8805 AND 'bpAy'='bpAy

Type: AND/OR time-based blind
Title: MySQL >= 5.0.12 AND time-based blind
Payload: bookid=114' AND SLEEP(5) AND 'AAIj'='AAIj

---
do you want to exploit this SQL injection? [Y/n] Y
[19:43:48] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.4.23, Apache 2.4.3
back-end DBMS: MySQL >= 5.0.12
[19:43:48] [INFO] fetching database names
[19:43:48] [INFO] fetching number of databases
[19:43:48] [INFO] retrieved:
[19:43:48] [CRITICAL] connection reset to the target URL. sqlmap is goin
[19:43:49] [ERROR] connection reset to the target URL, skipping to the n
[19:43:49] [INFO] you can find results of scanning in multiple targets m
018_0739pm.csv'

```

Fig. 3. SQLMap final result for test URL with test cases for SQL injection

## II. METHODOLOGY

As the goal was just to detect the possible vulnerable areas inside a website, I have used simple test cases that do not have a harmful effect (such as table drop or renaming a particular table) on the websites under test. If a website's database corresponds to a simple injection query, it is by default, vulnerable to more critical and dangerous

injections. Therefore, I just want to highlight the areas which require the focus of developers to come up with a more secure and protected system.

### A. Application Logic

The main logic behind the SQLI vulnerability detector tool developed in this project is fuzzing. Fuzzing, in simple terms, is playing with the system to see its response by supplying a different set of input. It gives us an idea about how the application is taking the input and processing it to give results. In addition to that, this logic is also used to check the error handling mechanism that is adopted by the application in case of unexpected inputs. Different applications may act differently when supplied with the same malicious input. For example, appending a single quote (') to the end of a URL can give various possible results - (i) the URL can be redirected to the main website, (ii) it can show you a 404 Page not found error, (iii) do not handle unexpected input so break the SQL query and show SQL error message in the response page, and so on.

The tool here uses the logic (iii) to identify if it is vulnerable to SQL injection. The webpage which is fragile breaks the SQL query and therefore it is susceptible to injection.

Now, the logic is to match the error obtained from the response page to the generic error responses that are pushed by different types of DBMS in case of SQL failure. For this tool, I have considered MySQL, MySQL, Oracle, and PostgreSQL. The output is matched against multiple sequences of generic errors in standard DBMSs, mentioned as follows:

**MySQL:** 'check the manual that corresponds to your MySQL', 'SQL syntax', 'server version for the right syntax', 'expects parameter 1 to be'

**MySQL:** 'Unclosed quotation mark before the character string', 'An unhandled exception occurred during the execution', 'Please review the stack trace for more information'

**Oracle:** 'java.sql.SQLException: ORA-00933', 'SQLException.java.sql.SQLException', 'quoted string not properly terminated'

**PostgreSQL:** 'Query failed:', 'unterminated quoted string at or near'

The tool here tracks these errors and matches them with the http response of the fuzzed url.

### B. Web Scraping

Web Scraping is referred to the extraction of the content in the html, xml or any other web document. In order for the tool to compare the output with the

list of predefined generic errors, we need to perform a through web scraping of the response page. This was achieved by using a library called beautiful soup in Python 3.

### III. WORKFLOW

It is well known that the web pages that communicate directly with the database are susceptible to SQL Injections. Therefore, for this tool, we first fetch the web URL from the users as an input. This URL has to be in a proper order of protocol://hostname.domain/content\_page?key=value.

This can be interpreted in the given example: <http://www.dipintoguitars.com/category.php?id=1>

The tool will scan the response of the get request of the URL and lists out the number of matching errors as listed in default in the tool. Depending on the details needed, it also provides the description of errors (if any). The tool asks for a user input for the level information they need regarding the error. The tool has 3 levels of information generation for errors encountered.

**Level 0:** Boolean based. Defines if the provided URL is either vulnerable or not.

**Level 1:** Indicates the number for each error match.

**Level 2:** Prints the error as it appears on the website

The vulnerability is tested by passing a value (single quote, in this case) that will cause the website to function abnormally. The response of the page is then noted and matched with the set of possible error reports that may be generated in various types of DBMS to see if the website is vulnerable. It further asks for a user input to dig into the error list and find out the associated database with the errors using SQLMap.

The step by step workflow of the developed tool is as given below:

**Step 1:** Check for internet connectivity

**Step 2:** Fetch a URL through user input

**Step 3:** Fuzz the URL with unexpected input. Append a single quote character to the URL in this case.

**Step 4:** Verify the GET response against a randomly selected user agent.

**Step 5:** If the http response is one of the predefined SQL errors for different DBMS like MySQL, MySQL, Oracle, and PostgreSQL, then the URL is considered susceptible - prone to web based attacks.

**Step 6:** Store the user-fed URL in an external file and supply it as an input to a very powerful and sophisticated tool SQLMap to further check the depth

of vulnerability and SQL injection. The system again makes a final decision based on a more advanced check through SQLMap.

The different user agents involved for testing the URL are Mozilla, Linux, AppleWebKit, Chrome, Safari, Opera, etc.

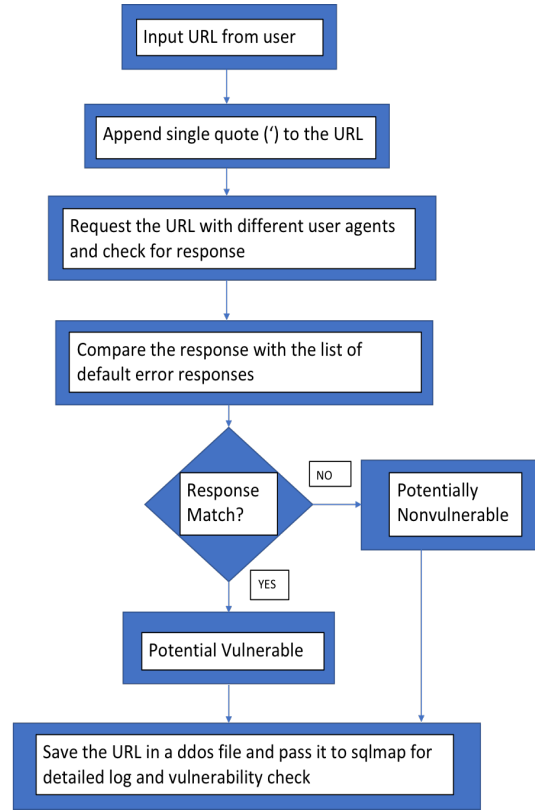


Fig. 4. Workflow

### IV. EVALUATION

In order to test the functionality of the tool, a set of webpages were listed from google search seeking for results on keyword 'php?id='. 10 pages were sorted for the test and supplied to the tool one at a time to evaluate their risk of being vulnerable. These URLs were also tested against SQLMap to see the difference in results and moreover to analyze the SQL injections in depth.

### V. FINDINGS AND RESULTS

Out of 10, 7 were found to be SQL injectable while all ten of them were vulnerable. The URLs that seemed suspicious were only considered for the study and passed through the tool and SQLMap to verify the results. The following table gives a brief description of how the results looked like.

TABLE I.  
RESULTS FROM SQLI DETECTION TOOL

URL	Check Level	SQL error match count	Possible Vulnerable
https://www.bradfordshoes.com/product.php?cat_id=5	2	3	Y
http://www.ankh.com/project/Show/ProductDetail.php?id=84	2	3	Y
http://www.km.co.ke/products.php?id=1	1	3	Y
http://www.jdcaravan.com/store.php?id=1	2	2	Y
https://www.f10products.co.za/index.php?id=5	1	2	Y
http://www.architecturalpapers.ch/index.php?id=10	2	3	Y
http://www.sarvodayayurved.com/add-to-cart.php?id=60	2	4	Y
http://www.dockguard.co.uk/page.php?id=18	2	2	Y
http://www.romaniawriters.ro/s.php?id=1	1	3	Y
https://www.solcavsko.info/index.php?id=46	2	3	Y

TABLE II.  
RESULTS FROM SQLMAP

URL	SQLMAP heuristics	SQLMAP conclusion	Boolean based	Time Based	Error Based	Union query
https://www.bradfordshoes.com/product.php?cat_id=5	might be injectable	doesn't seem injectable				
http://www.ankh.com/project/Show/ProductDetail.php?id=84	might be injectable	is injectable	Y	Y	N	N
http://www.km.co.ke/products.php?id=1	might be injectable	is injectable	Y	Y	N	N
http://www.jdcaravan.com/store.php?id=1	might be injectable	is injectable	Y	Y	N	N
https://www.f10products.co.za/index.php?id=5	might be injectable	is injectable	Y	Y	Y	Y
http://www.architecturalpapers.ch/index.php?id=10	might be injectable	doesn't seem injectable				
http://www.sarvodayayurved.com/add-to-cart.php?id=60	might be injectable	doesn't seem injectable				
http://www.dockguard.co.uk/page.php?id=18	might be injectable	is injectable	Y	Y	Y	Y
http://www.romaniawriters.ro/s.php?id=1	might be injectable	is injectable	Y	Y	Y	Y
https://www.solcavsko.info/index.php?id=46	might be injectable	is injectable	N	Y	N	N

The exact SQL injection that was possible in the given URLs were listed by the SQLMap. Also, the payloads to the URL that was used for the test cases were clearly mentioned. The results from the heuristics and the final conclusion based on the test cases were separately shown as in figure 1, figure 2 and figure 3.

## VI. THREATS TO VALIDITY

The results obtained from the tool had some threats to validity.

**Internal Validity:** 3 out of 10 test cases gave a false SQL injectable result. It gives a notion of a possible difference between a URL being vulnerable and it being actually injectable.

**External Validity:** As the selection of web URLs were intentionally made by exploiting the google dork for finding vulnerable web pages, we cannot generalize the result and conclude SQL injection to be a common problem in all existing websites.

## VII. PREVENTING SQL INJECTION

The most genuine way to prevent SQL injection attack is to validate and sanitize the user inputs. As it is suggested never to blindly trust the user input. It would rather be a wise decision to verify that the input is data and not code. Sanitization ensure the scanning and sorting of input with potential threats, before passing it into the system, while validation makes sure that the input is in a correct order and format as required by the system [8].

However, despite the different preventive measures, there are some essential techniques that when followed, helps prevent or mitigate SQL injection attacks. Here are a few of them:

**Trust none:** Presume user-submitted data to be evil and validate and sanitize everything.

**Avoid dynamic SQL whenever possible:** Instead, use prepared statements, parameterized queries or stored procedures.

**Update and patch:** Give immediate attention to each exploitable vulnerability in applications and databases. Apply patches and updates as soon as possible.

**Firewall:** Consider adding a Web Application Firewall to filter out malicious data.

**Use appropriate privileges:** Do not associate your database to an account with admin-level privileges unless there is some strong reason to do so. Using an account with limited access is far safer, and also limits the possible malicious activities.

**Let your secrets remain secret:** Adopt techniques for encrypting or hashing passwords and other confidential data including connection strings.

**Don't disclose more information:** Error messages can reveal so much information about your database architecture. Therefore, display minimal information so that a hacker knows nothing more than the fact that his actions resulted in an unhandled error.

**Adopt secure coding practices:** Make developers responsible for checking the code and for fixing possible security flaws in custom applications before the software is delivered.

## VIII. RELATED WORKS

There are a lot of other works related to the detection of SQL injection [3, 4]. The omnipresence of SQLI threats might have led security experts in coming up with a better solution to handle this issue properly. There is a range of projects from small to huge, from open source to paid versions, that deals with different levels of SQL injection attack. Dynamic tools are there to detect attacks while executing the audited program while the static analyzers scan the web applications for source code [3]. Some common examples of related tools are sqlmap, sqlninja, safe3 sql injector, SQLSus, etc.

## IX. CONCLUSION AND FUTURE WORKS

The popularity of web and web applications have significantly increased web-based attacks in past years [4]. In this paper, I have come up with an SQL injection detection tool that is used to find out if a given URL is vulnerable to SQL injection and at the same time a module of the SQLMap tool was used to exploit the areas of vulnerability - including the involved database. It also includes some effective measures of SQLI prevention. As a result of being on top of all the other web related vulnerabilities, there many small-to-large scale tools (both free and paid) that can be used to detect various types, and ranges of SQLI attacks. Each of the tools has different approaches to finding out the vulnerabilities,

from test cases, heuristics, to relational grammar. SQL injection is a vast topic and there are many different ways it can be analyzed, detected and prevented.

The current tool is only focused only one error-based SQL injection which can be further extended to see other types of injections like union query and/or blind (Boolean or time based) injections.

**Acknowledgment:** I would like to express my sincere gratitude towards Professor Dr. Minhaz F. Zibran for his help and guidance in the selection of the project base.

#### REFERENCES

- [1] C. Anley. Advanced SQL Injection in SQL Server Applications. Technical report, Next Generation Security Software, Ltd, 2002.
- [2] W. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In International Conference on Automated Software Engineering (ASE), pages 174–183, November 2005.
- [3] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In IEEE Symposium on Security and Privacy, 2006.
- [4] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic. SecuBat: A Web Vulnerability Scanner. In 15th International World Wide Web Conference (WWW), United Kingdom, May 2006.
- [5] H. Mahmood. Second Order SQL Injection Explained with Example. <https://haiderm.com/second-order-sql-injection-explained-with-example/>. July 2018.
- [6] OWASP. Top ten project. <http://www.owasp.org/>, May 2017.
- [7] S. V. Shanmughaneethi, Smt C. Emilin Shyni, S. Swamynathan. SBSQLID: Securing Web Applications with Service Based SQL Injection Detection. International Conference on Advances in Computing, Control, and Telecommunication Technologies, 2009.
- [8] P. Rubens. 10 Ways to Prevent or Mitigate SQL Injection Attacks. <http://www.enterprisenetworkingplanet.com/netsecur/article.php/3866756/10-Ways-to-Prevent-or-Mitigate-SQL-Injection-Attacks.htm>. Feb 2010.