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.

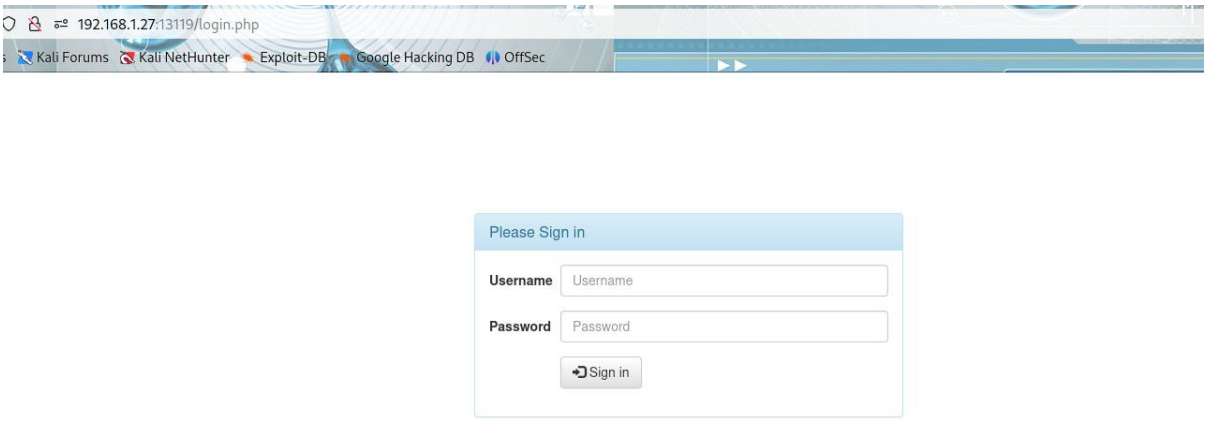
Vulnerability Detail

Vulnerability ID	V001
Title	SQL Injection
CWE ID	CWE-89
CVSS Score	
Technical Impact	· **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	· **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.

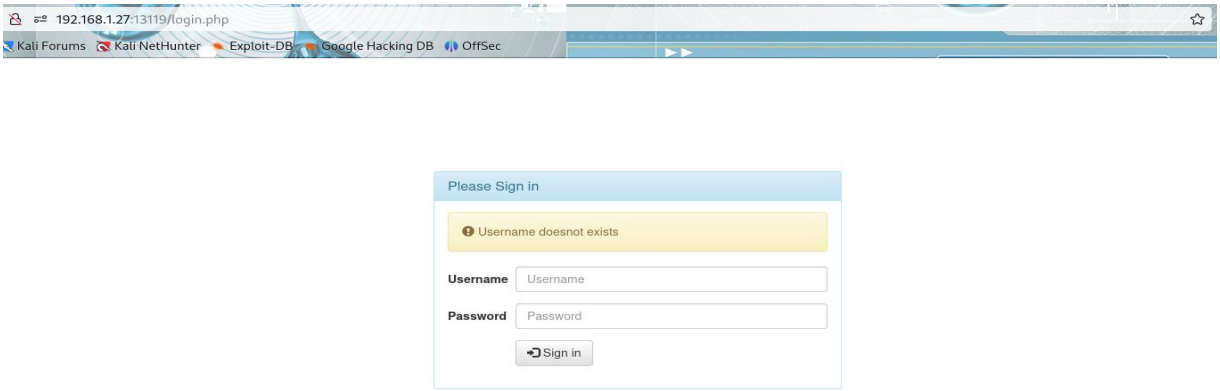
Evidence	
	1. Injection Point ➢ Login Parameter username **2. Malicious Payloads Used** (' OR 1=1 LIMIT 1 --) (' OR 1=1 LIMIT 0,1 --) (' OR 1=1 LIMIT 1,1 --) (' OR 1=1 LIMIT 2,1 --) **3. Observed Behavior / Server Response** ➢ Using each payload, the application bypassed authentication and logged in as different users. ➢ Evidence of different users being returned on each attempt confirms the database is being queried directly using injected input. **4. Backend SQL Interpretation (optional for technical clarity)** SELECT * FROM users WHERE username = '' OR 1=1 LIMIT 1,1; This bypasses filtering and forces the DB to return a specific row. **5. Impact** Unauthorized access to user accounts without valid credentials. Full enumeration of user data is possible with LIMIT-based SQLi.

Step to Reproduce

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

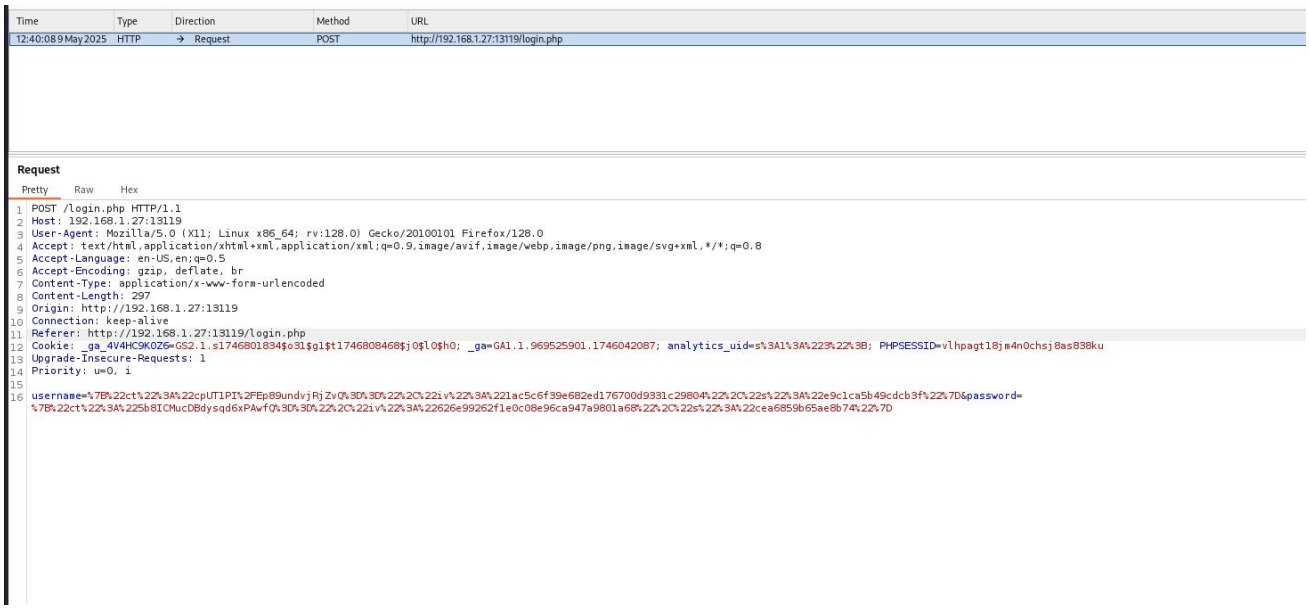


Step 2 : First We can try with the normal credentials like user/password and its couldn’t gave the access.

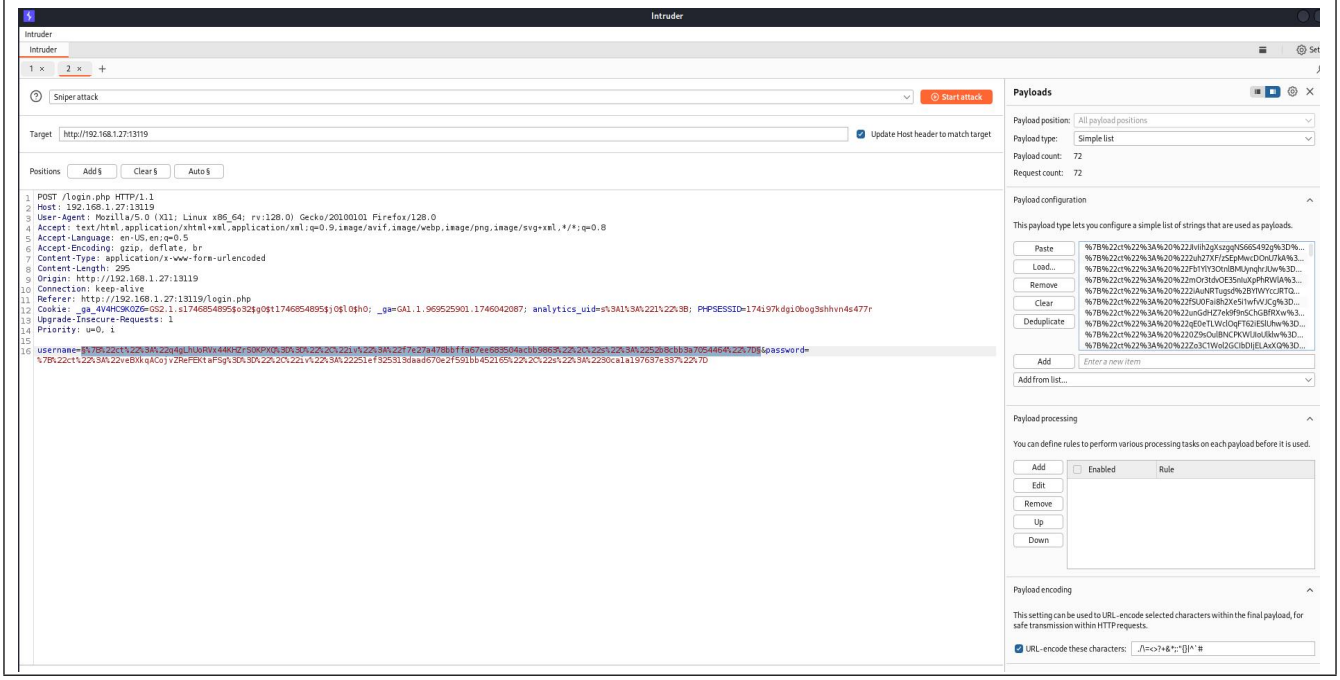


To be Continue …

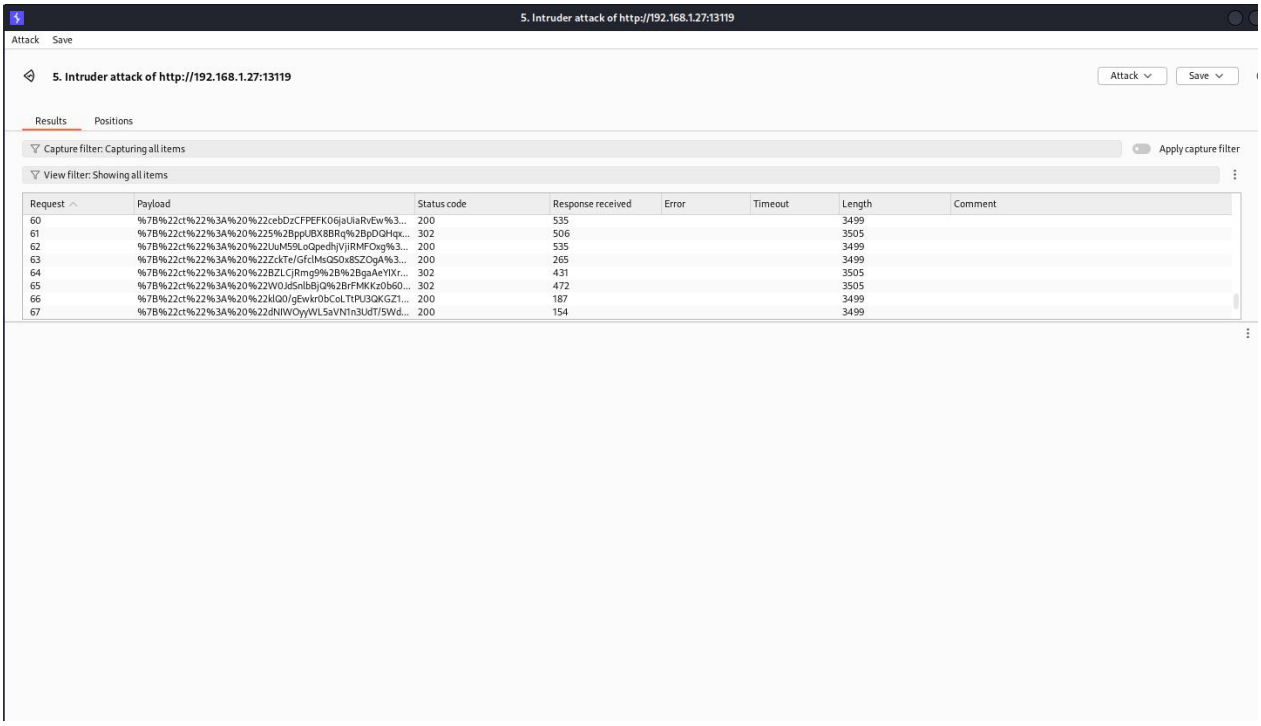
Step 3 : Here we intercept the request of the login page in Burp suit And we can see that Our credential converting into the AES formate encyrption.



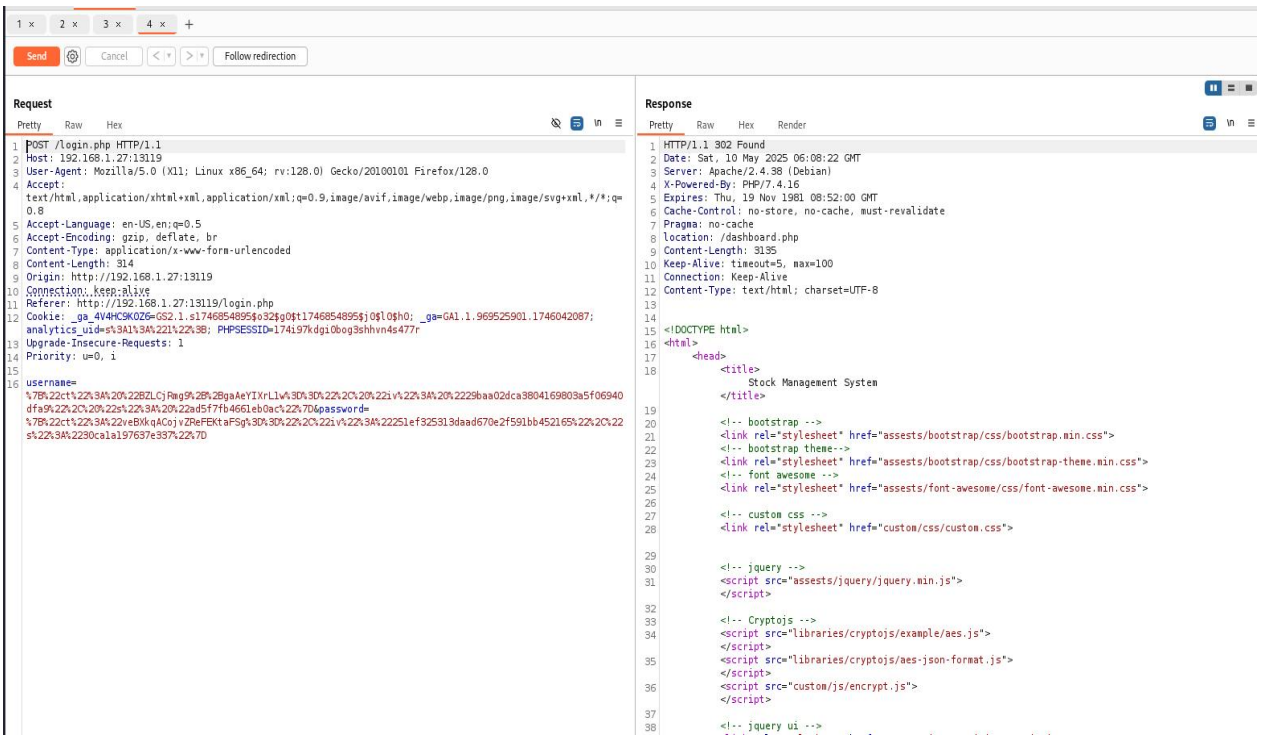
Step 4 : Here we take the sql payloads and the application accept the AES encrypted formate we convert all our payload to Encryption formate and then set to payload of intruder.



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

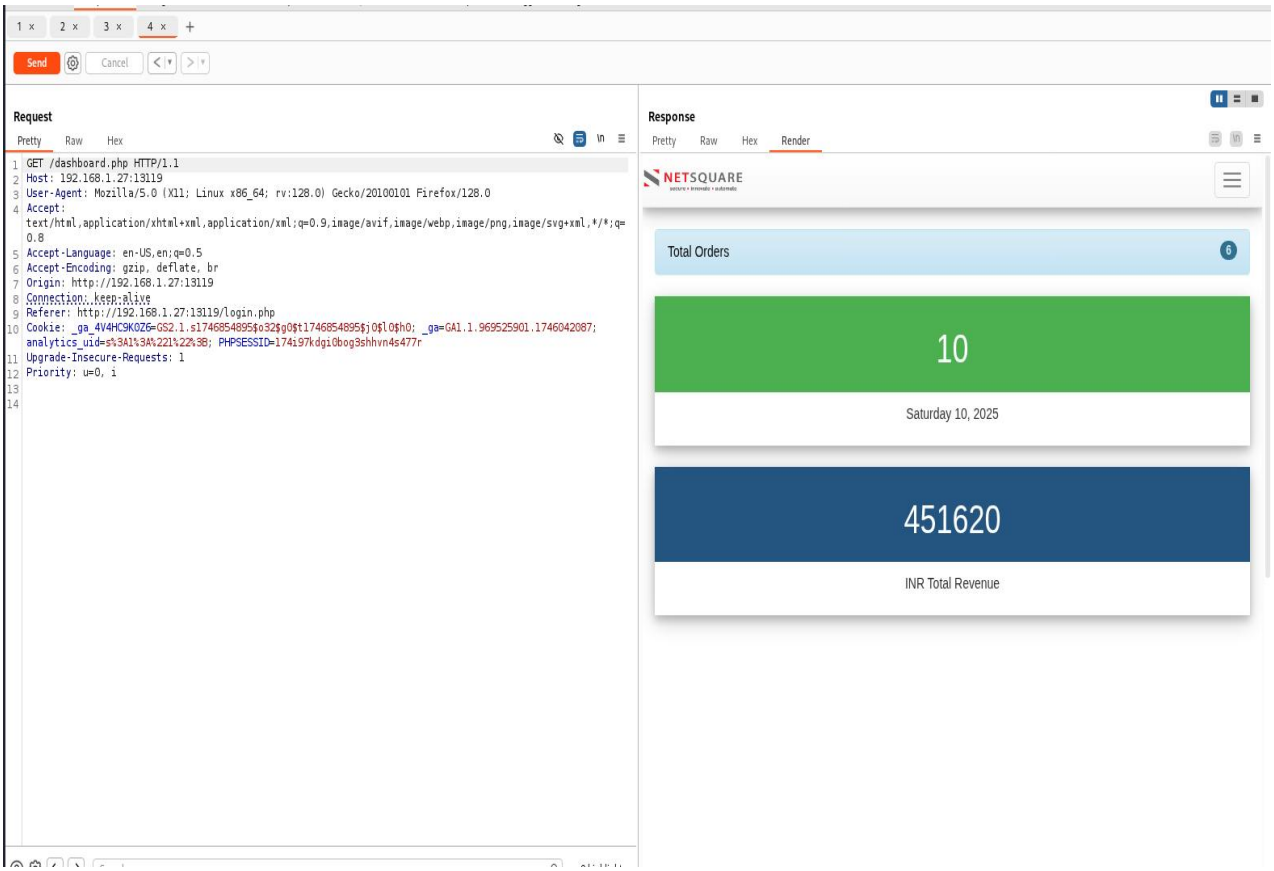


Step 6 : In the previous Image you can see that we received the 302 status code in response for more information send this request to repeater and bellow image you can see it tells to follow redirection.

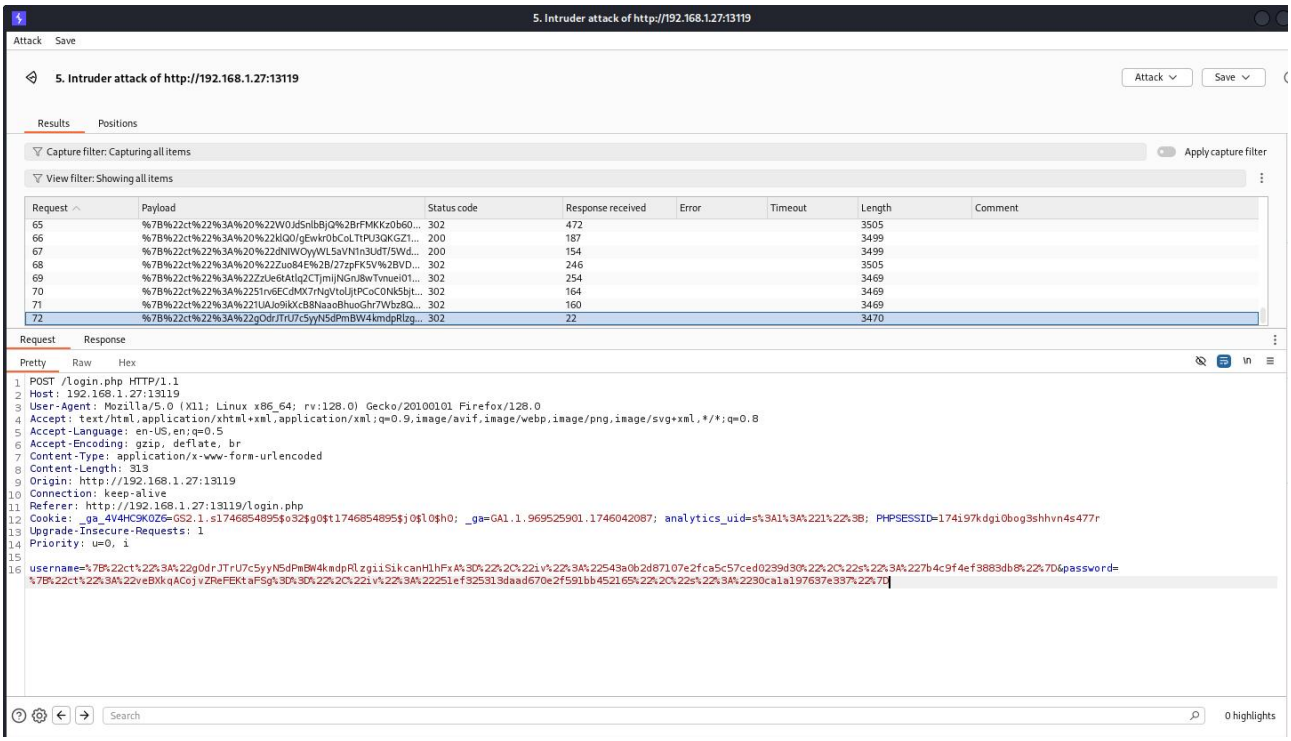


Step 7 :And you can see that the we just bypassed the login page. With the Sql payload of (‘ OR 1=1 LIMIT 1 --) We also tried the different payload but that gave 200 ok status code but no access of the App this payload hit corrrect it limit to the first raw and it retrive the username and Hit the login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

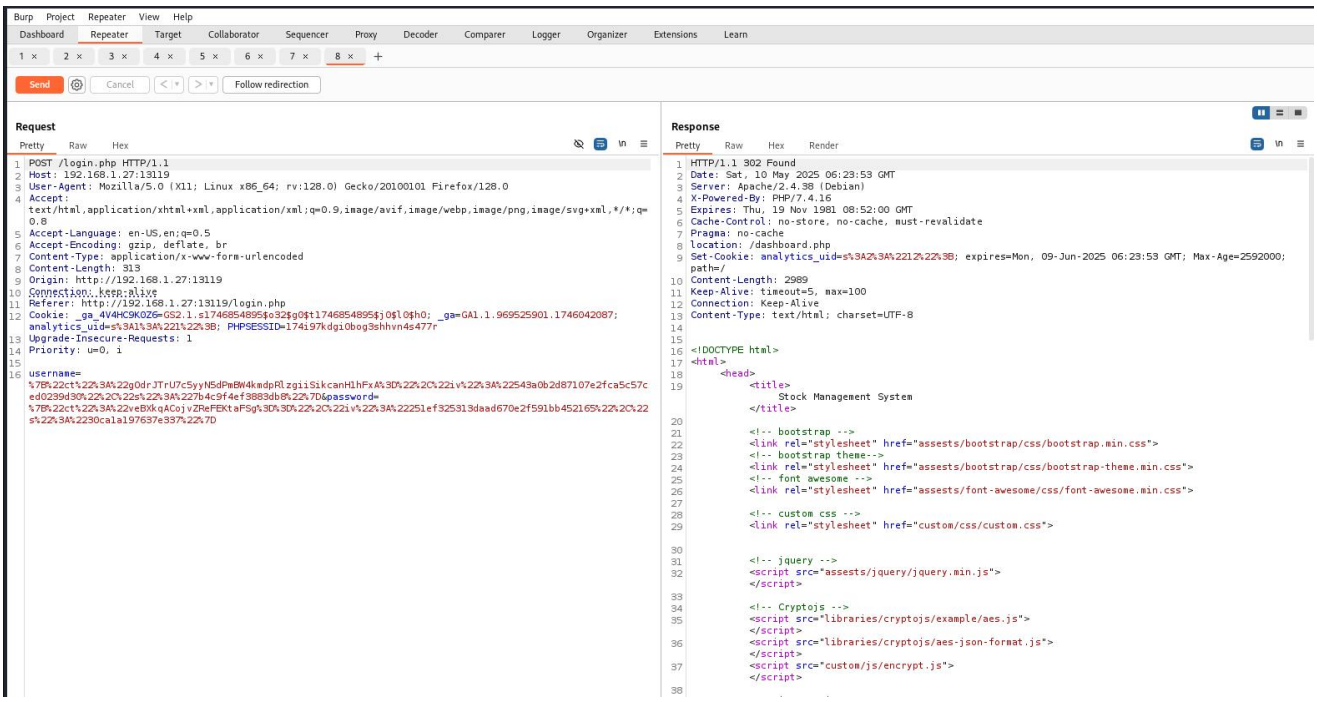
- SELECT * FROM users WHERE username = '' OR 1=1 LIMIT 1 -- '
- This is how it it manipulates the server's SQL query to bypass authentication.



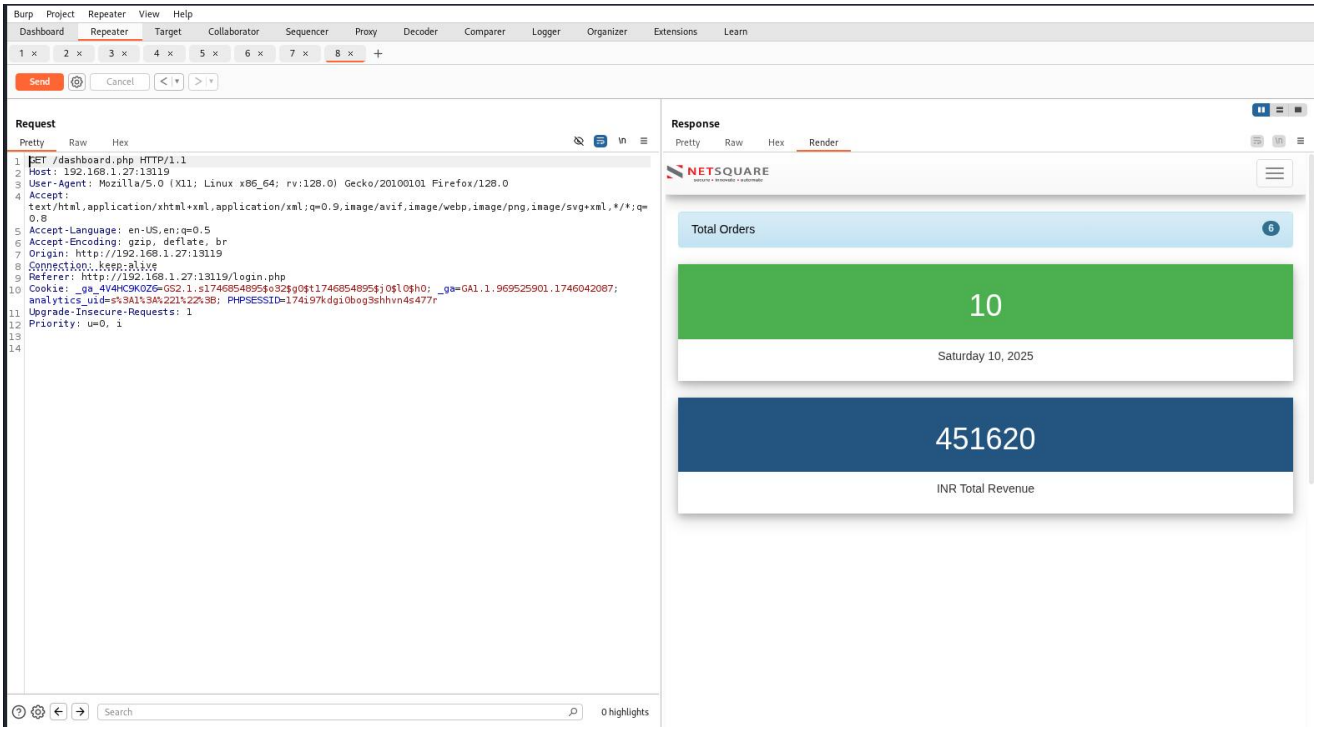
Step 8 : Here’s the another 302 response lets check that again to send repeter.



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

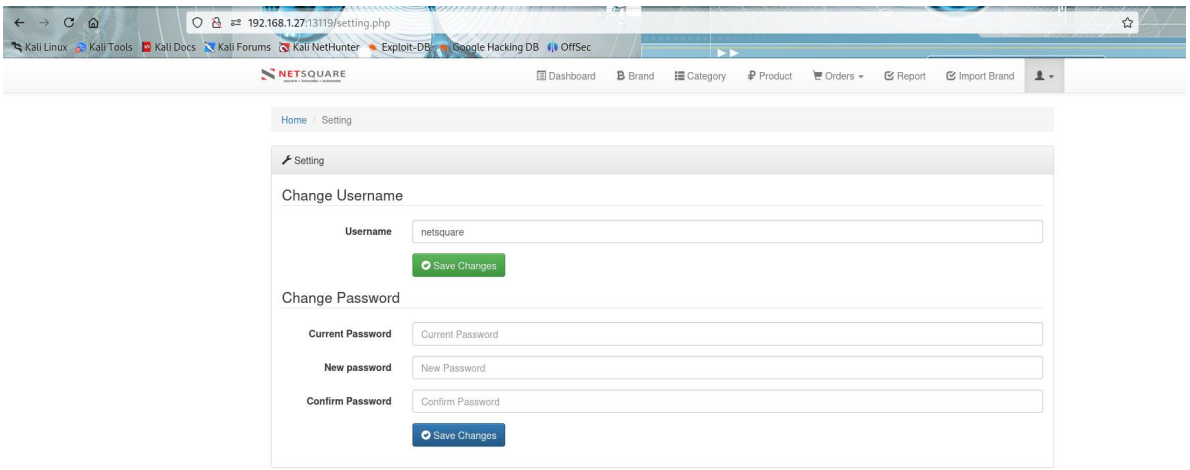


Step 10: Here's you saw that we gain the acess. This time the payload is (' OR 1=1 LIMIT 1,1 --)

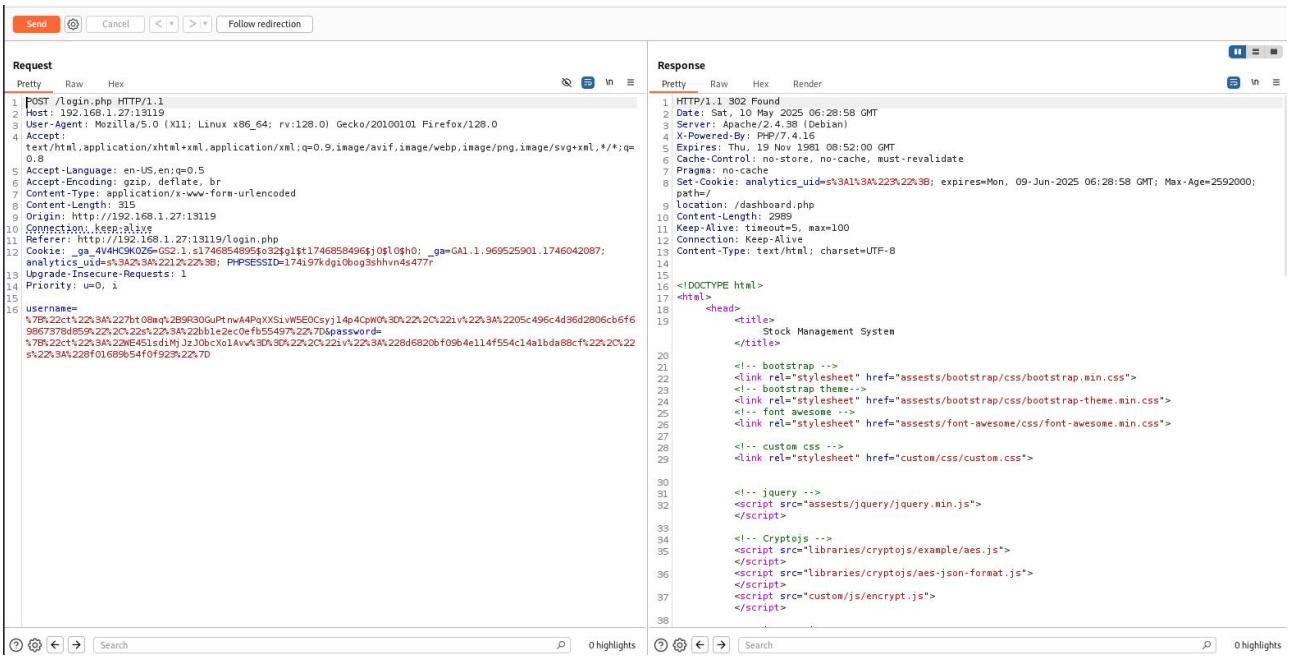


Step 11 : And we have the access of username netsquare.this how our payload works Here it retrive user1.

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

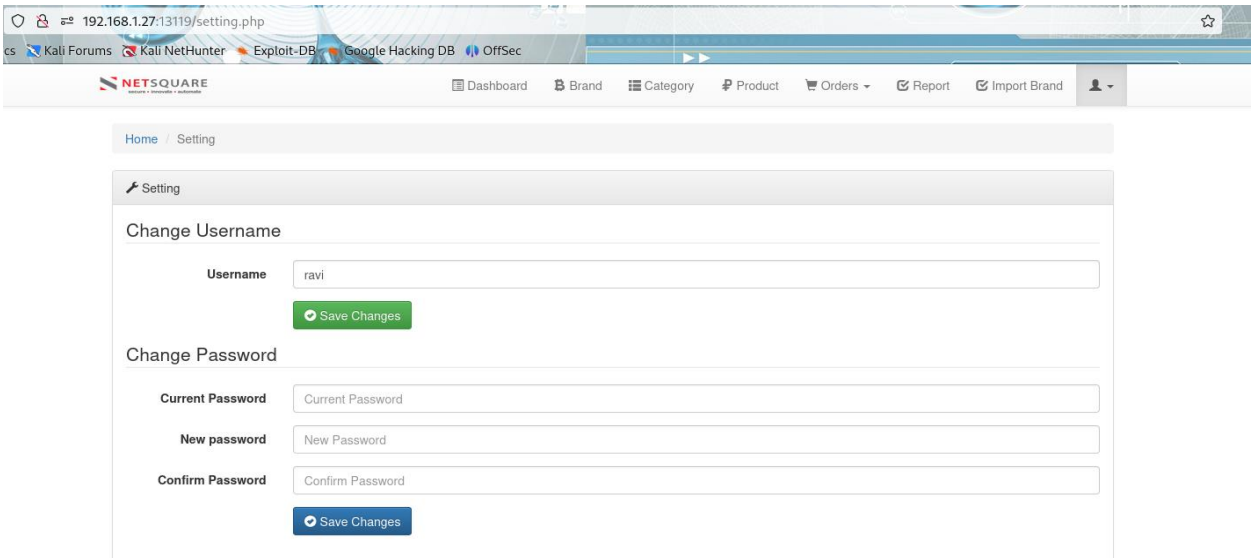


Step 12 : This is the another 302 request that we got during all payload check here we forward it to repeter and it show the follow redirection and click on it.

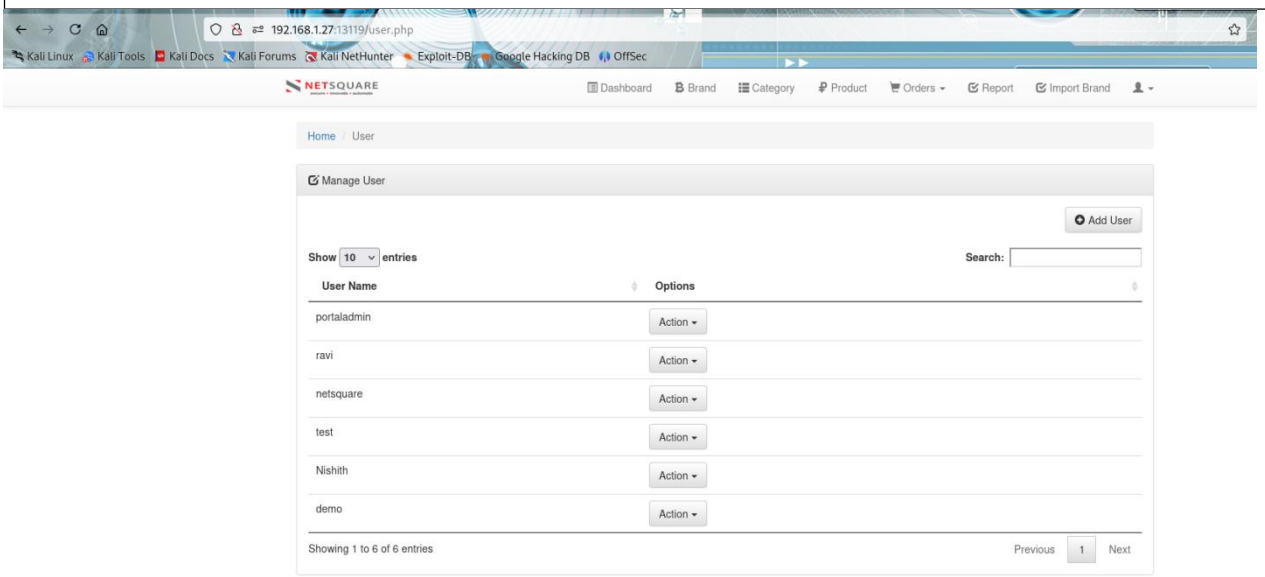


Step 13 :This time we are login with the another user name ravi.(' OR 1=1 LIMIT 2,1 --) its basically return user 2 and its ravi.This is How it works:

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



Step 14 : And you can also see the proof of sql that after that we can able to retrive 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