Team 10.4

Members – Dev, Diti, Aditi and Abhishek

1. **SQL injection vulnerability in WHERE clause allowing retrieval of hidden data.**

   Query : - '+OR+1=1—

   Explanation: - In SQL (Structured Query Language), the '=' operator is used to compare values. When you enter '+OR+1=1—' as part of a query, it is attempting to exploit a vulnerability in the system by injecting a condition that always evaluates to true.

   Output:-

2. **SQL injection vulnerability allowing login bypass**

   Query : - administrator'--

   Explanation: - In SQL, the single quote (') is used to delimit string values. By appending '--' after the single quote, the intention is to comment out the remainder of the SQL statement. This technique is often used in SQL injection attacks to manipulate the behavior of the query.

   Output:-

3. **SQL injection UNION attack, determining the number of columns returned by the query**

Query : - '+UNION+SELECT+NULL,NULL--

Explanation: - In SQL, the UNION operator allows you to combine the result sets of two or more SELECT statements. It is often used to retrieve data from different tables or columns. However, when used maliciously, it can be used to extract sensitive information or perform unauthorized actions.

Output:-

4. **SQL injection UNION attack, finding a column containing text**

Query : - '+UNION+SELECT+NULL, 'iJhkab',NULL--

Explanation: - The attacker is selecting a NULL value for the first and third columns, while introducing the string 'iJhkab' as the value for the second column in the result set. This could be an attempt to inject arbitrary data into the result set or exploit a specific vulnerability related to the 'iJhkab' value.

Output:-

Lab: SQL injection UNION attack, finding a column containing text

PRACTITIONER LAB Solved

5. **SQL injection UNION attack, retrieving data from other tables**

Query : - '+UNION+SELECT+username,+password+FROM+users--

Explanation: - This attack attempts to retrieve the values from the "username" and "password" columns of the "users" table. By selecting these specific columns, the attacker may gain access to sensitive user information, such as usernames and passwords.

Output:-

Lab: SQL injection UNION attack, retrieving data from other tables

PRACTITIONER LAB Solved

6. **SQL injection UNION attack, retrieving multiple values in a single column**

Query:'+UNION+SELECT+NULL,username||'~'||password+FROM +users--

Explanation: - This attack attempts to retrieve the concatenated values of the "username" and "password" columns from the "users" table, using the "~" symbol as a separator. This technique is often employed to retrieve sensitive data, such as usernames and passwords, from a compromised database.

Output:-

7. **SQL injection with filter bypass via XML encoding**

Query : - <storeId>1 UNION SELECT NULL</storeId>

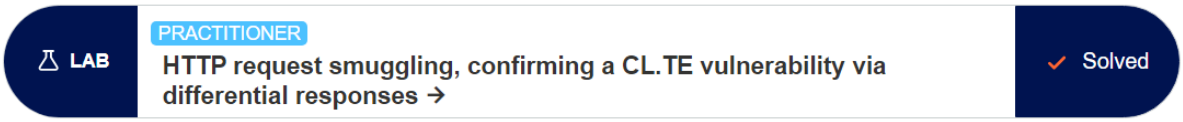
Explanation: - The attack seems to be injecting a malicious payload into the "<storeId>" XML element, attempting to exploit the application's handling of the data. However, the provided payload "1 UNION SELECT NULL" resembles a SQL injection attack rather than an XML injection.

Output:-

8. **HTTP request smuggling, confirming a CL.TE vulnerability via differential responses**

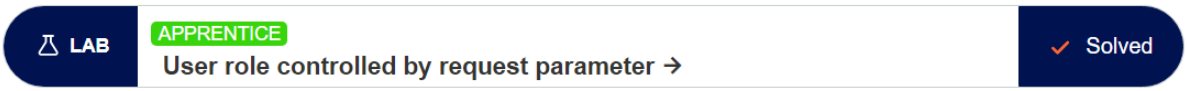
Explanation: HTTP request smuggling is a security vulnerability arising from discrepancies in how various components in a web application handle HTTP headers, specifically Content-Length (CL) and Transfer-Encoding (TE). While HTTP specifications deem it invalid for a request to include both headers, variations in header interpretation across components can lead to inconsistencies. Exploiting a CL.TE vulnerability involves crafting a malicious request with conflicting headers, passing it through components like a front-end proxy and backend server, and observing how each interprets the headers. The attacker then exploits the differences to manipulate processing, potentially allowing for unauthorized access or bypassing security controls. Confirming the vulnerability requires observing differential responses from these components. Responsible disclosure involves reporting the issue and implementing mitigation measures, such as configuring consistent header handling and updating

software. Ethical guidelines must be followed, ensuring activities are conducted with explicit permission to avoid legal consequences.



9. **User role controlled by request parameter**

Explanation: User role controlled by request parameters refers to a scenario in web applications where the privileges or permissions assigned to a user are determined based on values provided within the HTTP request parameters. Typically, a user's role dictates the actions they are authorized to perform within the application. In this context, the user role is dynamically set or modified by parameters included in the request, such as through URL parameters or form fields. This approach allows for flexibility in assigning different roles to users without requiring explicit changes in user profiles or settings. However, if not properly validated and secured, it poses a security risk, as an attacker might manipulate these parameters to gain unauthorized access or escalate privileges within the system. It is essential for developers to validate and sanitize user inputs, especially those influencing user roles, to prevent potential security vulnerabilities and unauthorized access.



10. **Password reset broken logic**

Explanation: Password reset broken logic refers to vulnerabilities or flaws in the design and implementation of the password reset functionality in a system. This security issue arises when there are inadequacies in the logic that governs the password reset process, potentially allowing attackers to manipulate or bypass the reset mechanism. Common issues include insufficient authentication checks, weak verification methods, or predictable tokens or links used in the reset process. For instance, if the reset link or token is easily guessable, an attacker might be able to gain unauthorized access to an account. Similarly, if the system lacks proper checks to ensure that the person initiating the password reset is indeed the legitimate account owner, it could lead to unauthorized access or account takeover. Ensuring a robust and secure password reset process is crucial for safeguarding user accounts, and developers should implement strong

authentication mechanisms and employ secure token generation to mitigate the risk of broken logic in password reset functionality.