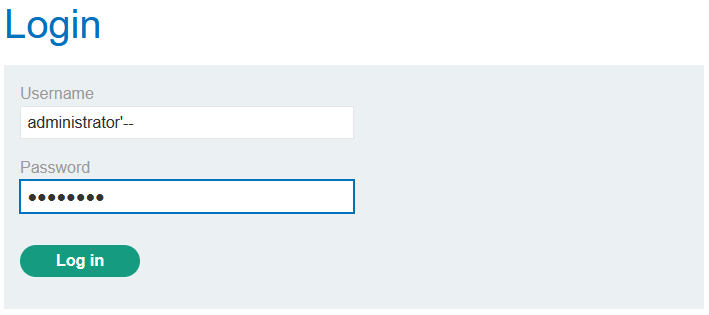
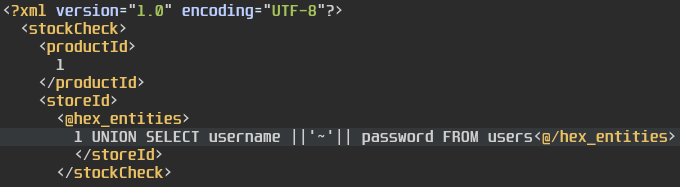
1. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data:
   * Exploited the vulnerability by injecting **‘ OR 1=1—** within theURL of the web application after category parameter in order to view the unreleased products.
2. SQL injection vulnerability allowing login bypass:
   * Exploited the vulnerability by injecting **administrator’--** within the username input field in order to gain the administrator privileges of the web application.
3. SQL injection with filter bypass via XML encoding:
   * Exploited the vulnerability present in the stock check option by intercepting the request using BurpSuite’s Repeater integrated with an extension from the Bapp store named Hackverter to encode the request into in order to prevent detection and retrieve the credentials. Used the **||’~’||** to append the username with its password because the original query returned only one value so it would’ve thrown an error if we’ve used **,** instead.



1. SQL injection attack, querying the database type and version on Oracle:
   * Exploited the vulnerability by first determining how many columns does the original query returns and their respective datatype in order to modify the injection accordingly as dual is a temporary table in Oracle database so we used that in order to know the version of the deployed database.
2. SQL injection attack, listing the database contents on non-Oracle databases:
   * Most databases other than Oracle maintains a table with a set of views containing information about the database.
   * Exploited the vulnerability by first checking the number/datatype returned by the original query and then first queried the database by injecting the SQL query in the URL by information\_schema, then queried the table\_name from it and retrieved the confidential data from the database.



1. SQL injection attack, listing the database contents on Oracle:
   * Exploited the vulnerability by first checking the numbers/datatypes of the parameters returned by the original query and then injected a query in order to retrieve table\_name from the all\_tables database.
   * Then got the names of the column of the desired table and at the end we retrieved the data from the table found.



1. SQL injection UNION attack, determining the number of columns returned by the query:
   * Exploited the vulnerability by checking the number of columns in the original query by injecting a basic UNION based SQL query and as soon as the number of parameters injected became equal to the actual original query, it stops throwing any errors. Now to find the column’s datatype we need to replace the specific NULL with the desired datatypes.



1. SQL injection UNION attack, retrieving multiple values in a single column:
   * Exploited the vulnerability by getting data from two different columns and appending them to act like the data retrieved from one single column as the original query contained just one parameter.
2. Blind SQL injection with conditional response:
   * Exploited the vulnerability by injecting a conditional query within the Tracking ID field of the request using BurpSuite in order to know the length of the administrator’s password.
   * Then we’ll mount a brute force attack on the same in order to get the password character by character until we get the entire password by using BurpSuite’s intruder: brute force payload, grep match to identify the correct character.
3. Blind SQL injection with conditional errors:
   * Exploited the vulnerability by first making sure that there exists a table named ‘users’ by appending query into the Tracking ID.
   * Then we went to look if there exists a user named administrator to confirm by appending query into the Tracking ID.
   * Then we tried to get the password’s length for the administrator account by appending query into the Tracking ID.
   * At the end we mounted a brute force attack on the administrator’s password character by character to retrieve the entire password.
4. Visible error-based SQL injection:
   * Exploited the vulnerability by injecting SQL query within the Tracking ID field in order to know if there exists any table named ‘users’.
   * Then we went to look if there exists any account named ‘administrator’ but our error message was truncated so we removed the Tracking ID completely.
   * Then we injected a query which went in to inquire about the administrator’s password in the Tracking ID request header.
5. Blind SQL injection with time delays and information retrieval:
   * We exploited the vulnerability by injecting some specific SQL query to make the server sleep for a given time if a given condition is true and so we first tried to get the password’s length.
   * Then we went to look if any account named administrator exists in the table user or not.
   * After which we mounted a brute force attack on the password character by character.
6. Blind SQL injection with out-of-band interaction: