Apply filters to SQL queries

Project description

This project demonstrates a hypothetical situation where I work as a security analyst at a large organization with the task to investigate potential security incidents involving login attempts and employee machines. I was tasked with examining the organization's `log_in_attempts` and `employees` tables with the goal of identifying anomalous login activity and searching for employee computers that need specific security patch updates by department. Using SQL filters, I extracted and analyzed data to identify unusual login attempts and potential policy violations that may indicate compromised data or systems.

Retrieve after hours failed login attempts

All login attempts after business hours (after 18:00) need to be investigated to discern a potential security incident:

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE login_time > '18:00:00' AND success = FALSE;
+----------+----------+------------+------------+---------+-----------------+---------+
| event_id | username | login_date | login_time | country | ip_address      | success |
+----------+----------+------------+------------+---------+-----------------+---------+
|        2 | apatel   | 2022-05-10 | 20:27:27   | CAN     | 192.168.205.12  |       0 |
|       18 | pwashing | 2022-05-11 | 19:28:50   | US      | 192.168.66.142  |       0 |
|       20 | tshah    | 2022-05-12 | 18:56:36   | MEXICO  | 192.168.109.50  |       0 |
|       28 | aestrada | 2022-05-09 | 19:28:12   | MEXICO  | 192.168.27.57   |       0 |
|       34 | drosas   | 2022-05-11 | 21:02:04   | US      | 192.168.45.93   |       0 |
```

This SQL query selects records from the `log_in_attempts` table using the `SELECT *` where the asterisk `*` wildcard specifies that I want all columns from the table. I then applied filters with a specific set of conditions using the `WHERE` clause.

In this case, I filtered the results by selecting only the login attempt times past `18:00:00` denoted by `WHERE login_time > '18:00:00'` and under the condition that all login attempts failed indicated by `AND success = FALSE`, where `FALSE` means a failed login attempt. The `AND` operator is a logical operator that specifies that the two conditions must be `TRUE`.

As shown above, the resultant output of a query is always displayed after the end of the query (which is indicated by a semicolon `;`).

Note: The output always starts on a new line after the semicolon.

Retrieve login attempts on specific dates

I detected a suspicious login attempt on 2022-05-08. As a result, I needed to investigate what happened the day of the security event and the following day to identify any patterns or subsequent login attempts :

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE login_date = '2022-05-08' OR login_date = '2022-05-09';
+----------+----------+------------+------------+---------+----------------+---------+
| event_id | username | login_date | login_time | country | ip_address     | success |
+----------+----------+------------+------------+---------+----------------+---------+
|        1 | jrafael  | 2022-05-09 | 04:56:27   | CAN     | 192.168.243.140 |       1 |
|        3 | dkot     | 2022-05-09 | 06:47:41   | USA     | 192.168.151.162 |       1 |
|        4 | dkot     | 2022-05-08 | 02:00:39   | USA     | 192.168.178.71  |       0 |
|        8 | bisles   | 2022-05-08 | 01:30:17   | US      | 192.168.119.173 |       0 |
|       12 | dkot     | 2022-05-08 | 09:11:34   | USA     | 192.168.100.158 |       1 |
```

This query selects all columns from the `log_in_attempts` table and filters the login dates to on either `2022-05-08` or `2022-05-09` denoted by `WHERE login_date = '2022-05-08' OR login_date = '2022-05-09'`. The `OR` operator is a logical operator that specifies that at least one of the two conditions must be `TRUE` for a record to be included in the results.

By examining login attempts across these two consecutive days, I can look for related suspicious activity, such as repeated failed login attempts followed by a successful one, or access from unusual locations or IP addresses that might indicate a compromised account.

Retrieve login attempts outside of Mexico

After investigating this potential security incident further, I believe the suspicious activity may have originated outside of Mexico. As a result, I needed to focus my investigation on login attempts from countries other than Mexico:

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE NOT country LIKE 'MEX%';
+----------+----------+------------+------------+---------+----------------+---------+
| event_id | username | login_date | login_time | country | ip_address     | success |
+----------+----------+------------+------------+---------+----------------+---------+
|        1 | jrafael  | 2022-05-09 | 04:56:27   | CAN     | 192.168.243.140 |       1 |
|        2 | apatel   | 2022-05-10 | 20:27:27   | CAN     | 192.168.205.12  |       0 |
|        3 | dkot     | 2022-05-09 | 06:47:41   | USA     | 192.168.151.162 |       1 |
|        4 | dkot     | 2022-05-08 | 02:00:39   | USA     | 192.168.178.71  |       0 |
|        5 | jrafael  | 2022-05-11 | 03:05:59   | CANADA  | 192.168.86.232  |       0 |
```

This SQL query selects all columns from the `log_in_attempts` table and filters the results by all countries not in Mexico indicated by `WHERE NOT country LIKE 'MEX%'`. The `NOT` operator negates the condition following after it, excluding all records that match that condition.

The `LIKE` clause with the `%` wildcard is used for matching patterns. In this case, I specified that the `country` column can only contain records that start with `MEX` as some records under the `country` column contain `MEX` or `MEXICO` variations. The `%` after `MEX` means zero or more characters can exist after the `X` in `MEX`. By using `NOT` with this pattern, I excluded all login attempts from Mexico to focus on potential international security threats.

Retrieve employees in Marketing

I decided that it would be necessary to update the computers for specific employees in the Marketing department's East offices due to recently identified vulnerabilities in their software:

```
MariaDB [organization]> SELECT *
    -> FROM employees
    -> WHERE department = 'Marketing' AND office LIKE 'East%';
+-------------+--------------+----------+------------+----------+
| employee_id | device_id    | username | department | office   |
+-------------+--------------+----------+------------+----------+
|        1000 | a320b137c219 | elarson  | Marketing  | East-170 |
|        1052 | a192b174c940 | jdarosa  | Marketing  | East-195 |
|        1075 | x573y883z772 | fbautist | Marketing  | East-267 |
|        1088 | k8651965m233 | rgosh    | Marketing  | East-157 |
|        1103 | NULL         | randerss | Marketing  | East-460 |
|        1156 | a184b775c707 | dellery  | Marketing  | East-417 |
|        1163 | h679i515j339 | cwilliam | Marketing  | East-216 |
+-------------+--------------+----------+------------+----------+
7 rows in set (0.001 sec)
```

The above SQL query selects all columns from the `employees` table and filters the results to only employees who belong to the `Marketing` department `AND` work in `East` offices. This is denoted by `WHERE department = 'Marketing' and office LIKE 'East%'.` The `AND` operator ensures that both conditions must be `TRUE` for a record to be included in the results. The `LIKE 'East%'` specifically asks for records under the `office` column to start with the string `East` followed by zero or more characters. This condition allows the query to grab all East office locations, such as `East-170`, `East-195`, etc., in a single filter.

This targeted approach allows me to prioritize security updates for a specific employee group identified as having higher-risk systems.

Retrieve employees in Finance or Sales

I needed to prepare to make a separate security update to the machines for employees within the `Sales` and `Finance` departments. I first needed to retrieve information about all employees in these departments:

```
MariaDB [organization]> SELECT *
    -> FROM employees
    -> WHERE department = 'Sales' OR department = 'Finance';
+-------------+--------------+----------+------------+-------------+
| employee_id | device_id    | username | department | office      |
+-------------+--------------+----------+------------+-------------+
|        1003 | d394e816f943 | sgilmore | Finance    | South-153   |
|        1007 | h174i497j413 | wjaffrey | Finance    | North-406   |
|        1008 | i858j583k571 | abernard | Finance    | South-170   |
|        1009 | NULL         | lrodriqu | Sales      | South-134   |
|        1010 | k2421212m542 | jlansky  | Finance    | South-109   |
```

This query selects all columns from the `employees` table and filters the results to obtain employees under the `Sales` department `OR` the `Finance` department indicated by `WHERE department = 'Sales' OR department = 'Finance'`. Instead of using the `AND` operator, I used the `OR` operator to include employees from either department, not just those who somehow belonged to both departments simultaneously.

By identifying the employees within these departments, I could efficiently plan to apply critical security patches to their work computers that handle sensitive company and customer data.

Retrieve all employees not in IT

Lastly, I needed to make one more security update for employees that are not in the `Information Technology` department. These non-IT employees often have different security requirements and may need different guidance during updates. To carry out this update, I had to procure more information from these employees:

```
MariaDB [organization]> SELECT *
    -> FROM employees
    -> WHERE NOT department = 'Information Technology';
+-------------+--------------+----------+-----------------+-------------+
| employee_id | device_id    | username | department      | office      |
+-------------+--------------+----------+-----------------+-------------+
|        1000 | a320b137c219 | elarson  | Marketing       | East-170    |
|        1001 | b239c825d303 | bmoreno  | Marketing       | Central-276 |
|        1002 | c116d593e558 | tshah    | Human Resources | North-434   |
|        1003 | d394e816f943 | sgilmore | Finance         | South-153   |
|        1004 | e218f877g788 | eraab    | Human Resources | South-127   |
```

This SQL query selects all columns from the `employees` table and filters by employees who are `NOT` under the `Information Technology` department indicated by `WHERE NOT department = 'Information Technology'`, excluding all IT department employees from the results. As discussed previously, the `NOT` logical operator negates the condition specified after it.

This approach allowed me to identify all users who might need additional security training alongside their system updates, as non-IT staff typically have less technical background and may require more support with security practices.

Summary

To conclude, I investigated the organization's potential security risks and threats by using SQL queries and various filters. I examined the `log_in_attempts` and `employees` tables to review login activity by time of day, geographical location, success status, and departmental access.

Throughout my analysis, I explained the fundamentals of each SQL query using the `AND`, `OR`, and `NOT` logical operators. I also expanded to the `LIKE` operator and the percentage sign (`%`) wildcard to aid in identifying suspicious login patterns.

These SQL querying and filtering skills are essential for security analysts to conduct investigations. After applying these techniques, I was able to successfully identify unusual login attempts, separate potential security events by dates, isolate international login attempts, and plan for specific security updates for different departments.