REPORT
SQL Query Analysis Incident

v1.0.2

Author:

Eldon Gabriel

July 9, 2025

TABLE OF CONTENTS

REVISION HISTORY

Version	Date	Author	Description of Changes
v1.0.0	02/11/2025	Eldon G.	Initial draft.
v1.0.1	02/22/2025	Eldon G.	Added cover, table of contents, and revision history
v1.0.1	06/12/2025	Eldon G.	Added hierarchical section numbering and security recommendations.
v1.0.2	07/12/2025	Eldon G.	Updated formatting of the report.

SECTION 1.0: SQL QUERY ANALYSIS INCIDENT

1.1 Project Description

As a security professional, it is my responsibility to safeguard the organization's systems, investigate potential security threats, and perform necessary updates to employee machines. The following steps demonstrate how I used SQL filters to query data and identify security-related issues, ensuring the system remains secure and up to date.

1.2 Retrieve After-Hours Failed Login Attempts

I discovered and investigated a potential security incident that occurred after business hours. I queried the `log_in_attempts` table to identify all failed login attempts that occurred after `18:00` (6:00 PM). The query filters for records where the `login_time` is greater than `18:00:00` and the `success` column has a value of `0`, indicating that the login attempt failed.

SQL Query:

```
MariaDB [organization]> clear
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE login_time > '18:00' 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 |
```

How it Works:

- The `WHERE` clause filters the `log_in_attempts` table.
- `login_time > '18:00:00'` identifies all login attempts made after 6:00 PM.
- `AND success = FALSE` ensures only failed login attempts are retrieved (where `success` is equal to `0`).

This query helps focus on login attempts that fall outside normal business hours, making it easier to detect unauthorized access attempts during off-hours.

1.3 Retrieve Login Attempts on Specific Dates

To investigate a suspicious event that occurred on `2022-05-09`, I reviewed all login attempts from both `2022-05-09` and `2022-05-08`. I created a SQL query that filters the `log_in_attempts` table to retrieve records where the `login_date` is either `2022-05-08` or `2022-05-09`. This helped me analyze login activity during the specified timeframe and identify potential security threats.

SQL Query:

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE login_date = '2022-05-09' OR login_date = '2022-05-08';SELECT *
+----------+----------+------------+------------+---------+-----------------+---------+
| 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 |
```

How it Works:

- The `WHERE` clause filters records from the log_in_attempts table.
- `login_date = '2022-05-08'` retrieves all login attempts on May 8, 2022.
- `OR login_date = '2022-05-09'` adds login attempts on May 9, 2022, to the results.
- This query ensures that all login activity during the two days is included in the investigation.

1.4 Retrieve Login Attempts Outside of Mexico

To investigate suspicious activity and exclude login attempts originating from Mexico, I created a SQL query that filters out records where the `country` column contains `MEX` or `MEXICO`. By using the `LIKE` keyword with `NOT`, I ensured that the query retrieved only login attempts from countries other than Mexico. This approach guarantees that all variations of the country value (`MEX` or `MEXICO`) are excluded from the results.

SQL Query:

```
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 |
```

How it Works:

- The `WHERE` clause specifies the filter criteria for the `country` column.
- `country LIKE 'MEX%'` matches all values that start with `MEX`, covering both `MEX` and `MEXICO`.
- `NOT` negates the condition, ensuring the query retrieves records where the `country` does **not** match `MEX` or `MEXICO`.

This query helps narrow down login attempts from other regions, focusing the investigation on non-Mexican activity.

1.5 Retrieve Employees in the Marketing Department

To support the team in performing security updates on specific employee machines, I queried the `employees` table to retrieve information on all employees in the Marketing department who are located in offices within the East building. I used the `AND` operator to combine two conditions: one that filters for employees in the Marketing department and another that uses the `LIKE` keyword to match office locations that start with `East`.

SQL Query:

```
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 |
```

How it Works:

- The `WHERE` clause specifies the conditions to filter the query results.
- `department = 'Marketing'` retrieves employees from the Marketing department.
- `AND office LIKE 'East%'` ensures that only employees located in offices starting with "East" (such as `East-170` and `East-320`) are included in the results.
- The `LIKE` keyword with `%` allows for a flexible match of all office values that begin with "`East`".

To assist with updates to employee computers, I queried the `employees` table to retrieve records for all employees in the Finance or Sales departments. The `OR` operator was used to combine two conditions, ensuring that the query returns employees from either department.

SQL Query:

```
MariaDB [organization]> SELECT *
    -> FROM employees
    -> WHERE department = 'Finance' OR department = 'Sales';
+-------------+-------------+----------+------------+-------------+
| 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   |
```

How it Works:

- The `WHERE` clause filters the `employees` table.
- `department = 'Finance'` retrieves employees in the Finance department.
- `OR department = 'Sales'` adds employees in the Sales department to the results.
- Using the `OR` operator ensures that employees from both departments are included in the query output.

This query provides a complete list of employees in the Finance and Sales departments, making it easy for the team to perform the necessary updates.

To assist with updating employee computers, I needed to identify all employees who are not in the Information Technology (IT) department. Using the `NOT` operator, I filtered the `employees` table to exclude records where the `department` is 'Information Technology'.

SQL Query:

```
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   |
```

How it Works:

- The `WHERE` clause filters the `employees` table.
- `NOT department = 'Information Technology'` ensures that employees from the IT department are excluded from the results.
- The `NOT` operator negates the condition, returning all employees except those in the IT department.

This query helps focus on employees in other departments, ensuring the update does not affect those already covered.

SECTION 2.0: CONCLUSION

2.1 Summary of Findings

Throughout this project, I leveraged SQL filters to identify specific data that assisted in investigating and resolving security incidents. By applying logical operators such as `AND`, `OR`, and `NOT`, I was able to narrow down large datasets and focus on the most relevant records. This approach allowed me to investigate suspicious activity, monitor login behavior, and support the security update process for employees in various departments, ultimately enhancing overall security measures.

2.2 Security Implications and Recommendations

The queries revealed several important patterns that carry security implications:

- **After-hours failed login attempts** may indicate brute-force attacks or credential misuse.
- **Logins from outside expected geographic regions** (e.g., countries other than Mexico) could suggest account compromise or unauthorized remote access.
- **Department-based user queries** help identify over-privileged accounts or unauthorized device access.

Based on these findings, I recommend the following:

- **Implement time-based access controls** to limit login attempts during non-business hours.
- **Deploy geolocation-aware access policies** to alert or block logins from high-risk or unexpected regions.
- **Enforce the principle of least privilege**, ensuring that users only have access based on their department and role.
- **Use multi-factor authentication (MFA)** across all departments to mitigate credential theft risks.
- **Regularly audit login activity** using SQL or SIEM tools to detect anomalies early.

Applying these recommendations will help the organization proactively mitigate threats and improve the integrity of its access control framework.