### **Project Description**

**In this project, I investigated and managed file permissions in a Linux environment for the organization’s research team. I checked existing file and directory permissions, interpreted the permission strings, and modified them to ensure proper authorization. By doing this, I ensured that only authorized users had the correct level of access while removing unauthorized write access to sensitive files.**

### **Check file and directory details**

**Command used:**

**ls -la**

**Explanation:  
 The ls -la command lists all files and directories, including hidden files, along with their permissions, owners, and groups. This provides a complete view of who has access to what in the file system.**

**Sample Output (example):**

**-rw-rw-r-- 1 researcher2 research 1200 Sep 28 10:15 research.txt**

**-rw-rw-rw- 1 researcher2 research 5000 Sep 28 10:15 data.txt**

**-rw-r--r-- 1 researcher2 research 3000 Sep 28 10:15 report.txt**

**-rw-rw-r-- 1 researcher2 research 1500 Sep 28 10:15 .project\_x.txt**

**drwxr-xr-x 2 researcher2 research 4096 Sep 28 10:15 drafts**

### **Describe the permissions string**

**Example chosen: -rw-rw-r--**

**Explanation:**

* **First character: - = regular file**
* **Next three characters (rw-): owner has read and write permission**
* **Next three characters (rw-): group has read and write permission**
* **Last three characters (r--): others have read-only permission**

**This means the file can be read and modified by the owner and group, but others can only read it.**

### **Change file permissions**

**Problem identified:  
 The file data.txt has permissions -rw-rw-rw-, which gives others write access. This is not allowed by organizational policy.**

**Command used to fix:**

**chmod o-w data.txt**

**Explanation:  
 This removes write access from “others” while keeping the correct permissions for the owner and group.**

**New Output:**

**-rw-rw-r-- 1 researcher2 research 5000 Sep 28 10:15 data.txt**

### **Change file permissions on a hidden file**

**Problem identified:  
 The hidden file .project\_x.txt currently has write access for both the owner and group. The requirement is that it should only be readable by the owner and group, and no write access should be allowed.**

**Command used to fix:**

**chmod 440 .project\_x.txt**

**Explanation:  
 This sets the file to read-only (r--) for both owner and group, and no permissions for others.**

**New Output:**

**-r--r----- 1 researcher2 research 1500 Sep 28 10:15 .project\_x.txt**

### **Change directory permissions**

**Problem identified:  
 The drafts directory should only be accessible by the user researcher2. Currently, it is open to others.**

**Command used to fix:**

**chmod 700 drafts**

**Explanation:  
 This grants read, write, and execute permissions only to the owner (researcher2), and removes all access from group and others.**

**New Output:**

**drwx------ 2 researcher2 research 4096 Sep 28 10:15 drafts**

### **Summary**

**In this project, I analyzed file and directory permissions in a Linux environment and updated them to align with organizational security policies. I removed unauthorized write access from files, ensured that hidden files had the correct read-only permissions, and restricted access to sensitive directories so that only the correct user could view and modify them. This activity demonstrates my ability to use Linux commands such as ls -la, chmod, and chown to manage authorization effectively in a secure system.**

**✅ This version includes:**

* **Project description**
* **Commands (ls -la, chmod)**
* **Permission string breakdown**
* **Fixes for files, hidden files, and directories**
* **Summary**

**All you need to do is paste this into the template and (if required) add screenshots from the lab.**

**PROJECT 2- APPLYING FILTERS TO SQL FILTERS**

# **Apply Filters to SQL Queries**

### **Project Description**

In this project, I worked as a security professional investigating potential login and employee security issues for a large organization. Using SQL queries, I applied filters such as AND, OR, NOT, and LIKE to identify suspicious login attempts and to retrieve employee machine data for system updates. These skills are important in cybersecurity investigations because they help analyze and respond to potential security threats efficiently.

## **Retrieve after-hours failed login attempts**

**Query:**

SELECT \*

FROM log\_in\_attempts

WHERE login\_time > '18:00'

AND success = 0;

**Explanation:** This query retrieves all failed login attempts (success = 0) that occurred after 6 PM (login\_time > '18:00'). This helps identify suspicious after-hours activity that could indicate unauthorized access attempts.

## **Retrieve login attempts on specific dates**

**Query:**

SELECT \*

FROM log\_in\_attempts

WHERE login\_date = '2022-05-08'

OR login\_date = '2022-05-09';

**Explanation:** This query retrieves all login attempts from May 8 and May 9, 2022. The OR operator ensures that results from both dates are included. This is useful for investigating a specific suspicious event that may have occurred across multiple days.

## **Retrieve login attempts outside of Mexico**

**Query:**

SELECT \*

FROM log\_in\_attempts

WHERE country NOT LIKE 'MEX%';

**Explanation:** This query returns all login attempts where the country is not Mexico. Since the database contains both MEX and MEXICO, the LIKE 'MEX%' pattern matches both, and using NOT LIKE excludes them. This allows the focus to remain on attempts from other countries.

## **Retrieve employees in Marketing (East building)**

**Query:**

SELECT \*

FROM employees

WHERE department = 'Marketing'

AND office LIKE 'East%';

**Explanation:** This query identifies all employees in the Marketing department who are located in the East building. The LIKE 'East%' ensures that any office code starting with "East" (e.g., East-170, East-320) is included. This helps IT target security updates for the right department and location.

## **Retrieve employees in Finance or Sales**

**Query:**

SELECT \*

FROM employees

WHERE department = 'Finance'

OR department = 'Sales';

**Explanation:** This query returns all employees in the Finance or Sales departments. The OR operator ensures employees from either department are included. This allows IT to perform necessary updates for both groups at once.

## **Retrieve all employees not in IT**

**Query:**

SELECT \*

FROM employees

WHERE department <> 'Information Technology';

**Explanation:** This query retrieves all employees except those in the Information Technology department. The <> operator means "not equal to." Since IT already received updates, this query ensures only other departments are included.

### **Summary**

In this project, I applied SQL filtering techniques (AND, OR, NOT, LIKE) to investigate login activity and employee records. I identified failed login attempts after business hours, reviewed suspicious activity on specific dates, and excluded data from certain countries. I also filtered employee data by department and office location to support IT system updates. These queries demonstrate how SQL can be used in real-world cybersecurity investigations to extract relevant data and take appropriate security actions.

## **Algorithm for File Updates in Python**

### **Project Description**

In this project, I developed a Python algorithm that updates a file containing a list of IP addresses authorized to access restricted data within a healthcare company’s network. The algorithm ensures that employees who no longer require access are promptly removed based on a separate *remove list*. This project demonstrates my ability to work with file input/output (I/O), data parsing, and automation—skills essential in cybersecurity for managing network access and user permissions.

### **Open the File That Contains the Allow List**

import\_file = "allow\_list.txt"

with open(import\_file, "r") as file:

# The 'with' statement ensures the file closes automatically

ip\_addresses = file.read()

**Explanation:**

* import\_file = "allow\_list.txt" assigns the filename to a variable.
* open() is a **built-in Python function** used to open files.
* The "r" argument means the file is opened in **read mode**.
* The with keyword creates a **context manager** that automatically closes the file after use.
* file is the variable used to reference the open file while inside the with block.

### **Read the File Contents**

ip\_addresses = file.read()

**Explanation:**

* The .read() method reads the entire file and converts it into a **single string**.
* This string can then be manipulated using standard string methods.

### **Convert the String into a List**

ip\_addresses = ip\_addresses.split()

**Explanation:**

* The .split() method separates the string into **individual elements** (in this case, IP addresses) and stores them in a **list**.
* By default, .split() divides the string wherever there is whitespace or a newline.

### **Iterate Through the Remove List**

remove\_list = ["192.168.0.5", "10.0.0.7", "172.16.0.3"]

for element in remove\_list:

# Loop through each IP that should be removed

**Explanation:**

* remove\_list contains the IP addresses that should no longer have access.
* A for loop is used to **iterate** through each IP address in remove\_list.
* The variable element represents the **current IP address** being checked in each loop iteration.

### **Remove IP Addresses That Are on the Remove List**

for element in remove\_list:

if element in ip\_addresses:

ip\_addresses.remove(element)

**Explanation:**

* The if statement checks whether the IP address in the remove list exists in the allow list.
* The .remove() method deletes the first matching element from a list.
* This works efficiently here because there are **no duplicate IP addresses** in the allow list, so .remove() will only need to act once per IP.

### **Update the File with the Revised List of IP Addresses**

ip\_addresses = "\n".join(ip\_addresses)

with open(import\_file, "w") as file:

file.write(ip\_addresses)

**Explanation:**

* The "\n".join(ip\_addresses) statement combines all IP addresses into one string, placing each address on a **new line**.
* The "w" argument in open() opens the file in **write mode**, overwriting existing content.
* The .write() method writes the updated list of IP addresses back into the file.
* Using with open(...) again ensures that the file automatically closes after writing.

### **Summary**

This algorithm automates the process of updating an IP address allow list. It:

1. Opens a text file containing allowed IPs using the with and open() functions.
2. Reads and converts the file content into a list with .read() and .split().
3. Iterates through a remove list using a for loop.
4. Removes matching IPs from the allow list using the .remove() method.
5. Converts the list back into a formatted string and writes it to the file using .join() and .write().

By automating this task, the algorithm minimizes human error, ensures timely updates to access permissions, and demonstrates the power of Python for routine cybersecurity file management.

### **✅ What to Include in Your Submission**

Make sure your final portfolio file contains:

* The **Python code snippets** above (typed or screenshot from your lab).
* The **syntax explanations** for each section.
* A **project description** and **summary** (already included above).
* References to key Python tools: with, open(), .read(), .split(), .remove(), .join(), .write(), and the for loop.