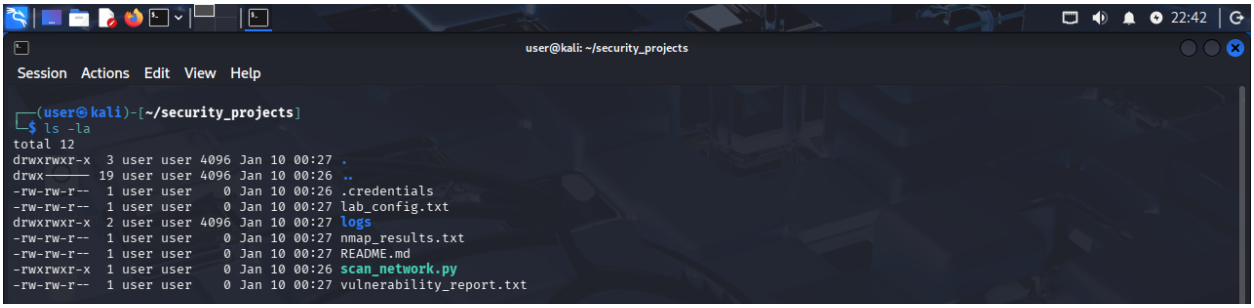
Hardening File Permissions on My Linux System by Maksym Kovalchuk

Project Description

I recently set up a home lab environment running Ubuntu to practice cybersecurity skills and host a few personal projects. After the initial setup, I realized I had been running commands with sudo carelessly and hadn't paid attention to file permissions. Some scripts and configuration files were readable by anyone on the system, which isn't great practice—even on a personal machine. I decided to audit the permissions in my home directory and fix anything that didn't follow basic security principles.

Check File and Directory Details

The following code demonstrates how I used Linux commands to examine the existing permissions in my projects directory.



The first line shows the command I ran: ls -la ~/security_projects. The output displays all files in the directory, including hidden ones. I used the -la options to get a detailed listing that shows permissions, ownership, and file types. Looking at the results, I have a few Python scripts, a .credentials file, and a logs directory. Right away, I noticed the .credentials file had permissions that would let any user on my system read it.

Understanding the Permissions String

Each file has a 10-character permission string that breaks down like this:

1st character: File type

- d = directory
- - = regular file

2nd-4th characters: User (owner) permissions

- r = can read the file
- w = can write/modify the file
- x = can execute the file (for scripts/programs)
- - = permission not granted

5th-7th characters: Group permissions (same r, w, x structure)

8th-10th characters: Other (everyone else on the system) permissions

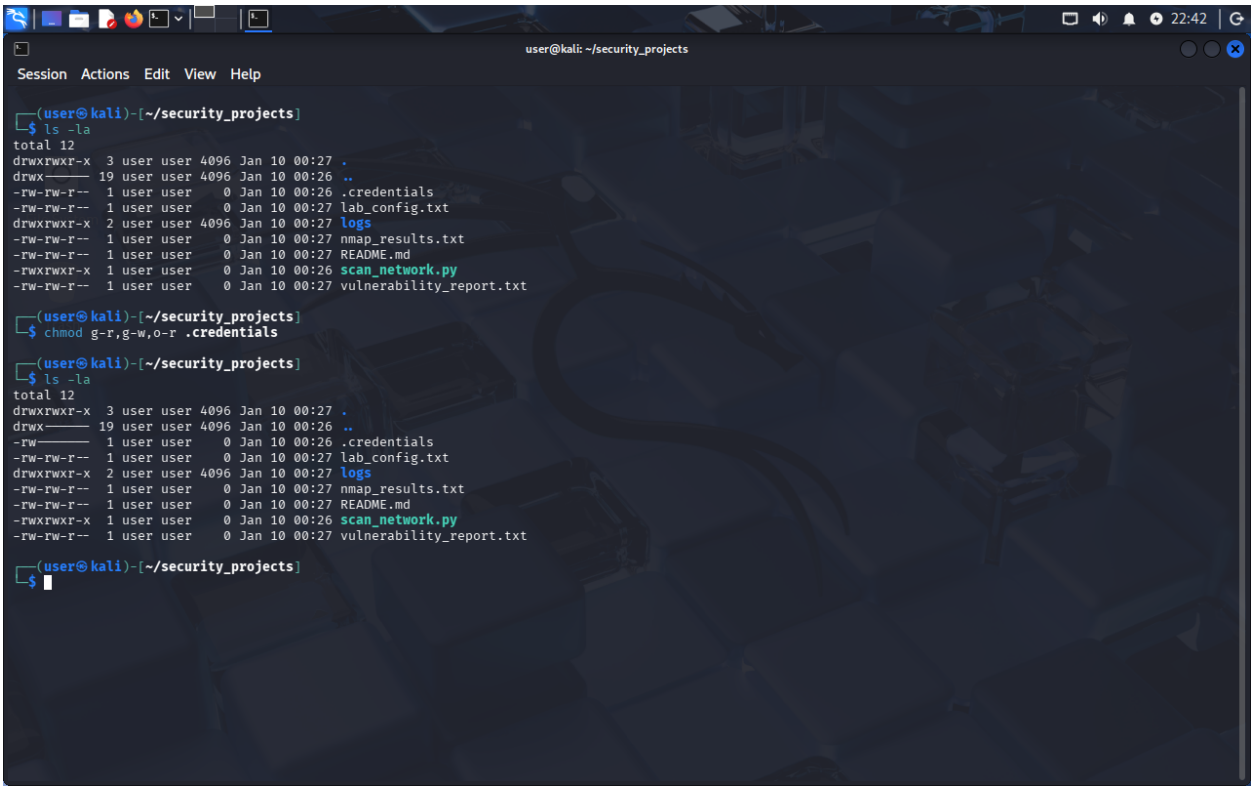
For example, my scan_network.py script showed -rwxr-xr-x:

- First - means it's a regular file
- rwx means I (the owner) can read, write, and execute it
- r-x means the group can read and execute it
- r-x means everyone else can also read and execute it

That's unnecessarily open for a script that runs network scans on my lab environment.

Securing Credential Files

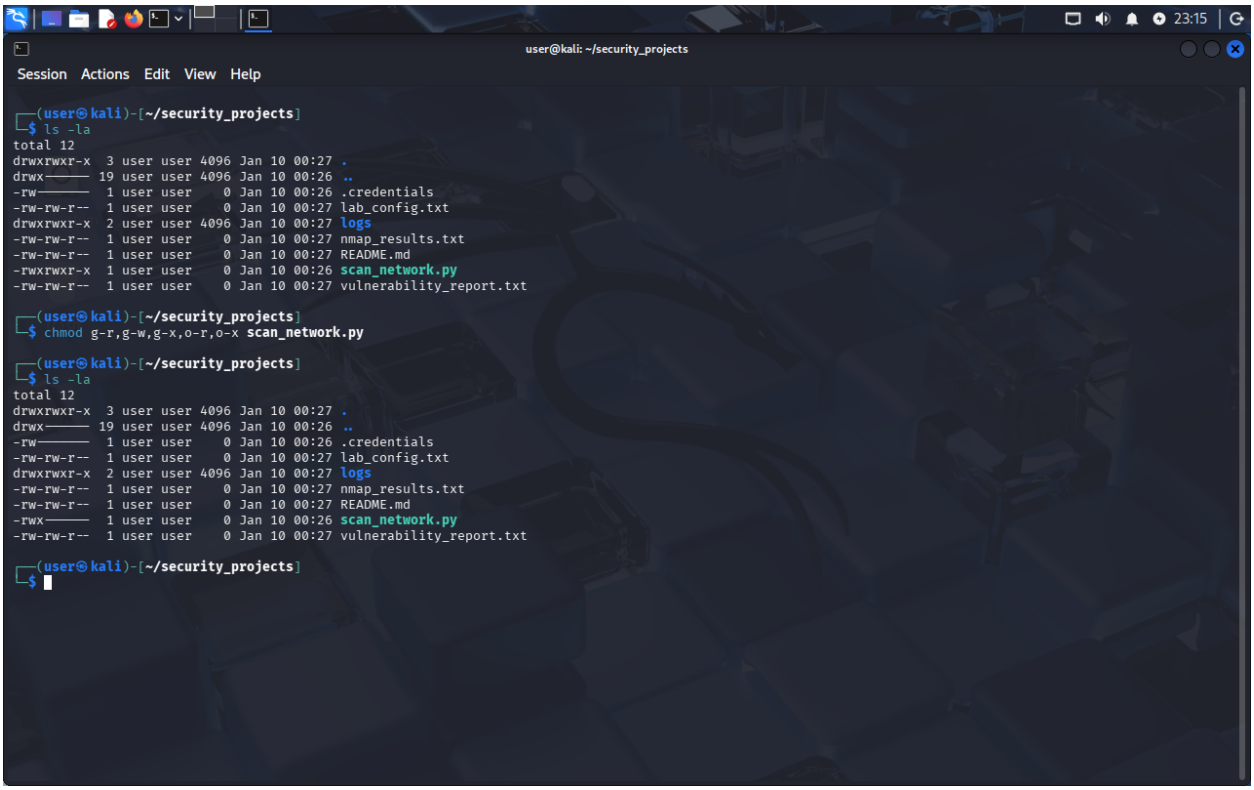
My .credentials file stores API keys for various security tools I use in my lab. Looking at the initial ls -la output, it had -rw-rw-r-- permissions, meaning my group could read and write it, and everyone else on the system could read it.



I used chmod g-r,g-w,o-r .credentials to remove read and write permissions from the group, and remove read permissions from others. After running the command, I verified the change with another ls -la, and the permissions now show -rw-------. Now only I can read and write the file. Even though I'm the only person who uses this machine regularly, it's good practice—especially if I ever give someone SSH access to help troubleshoot something.

Restricting Script Permissions

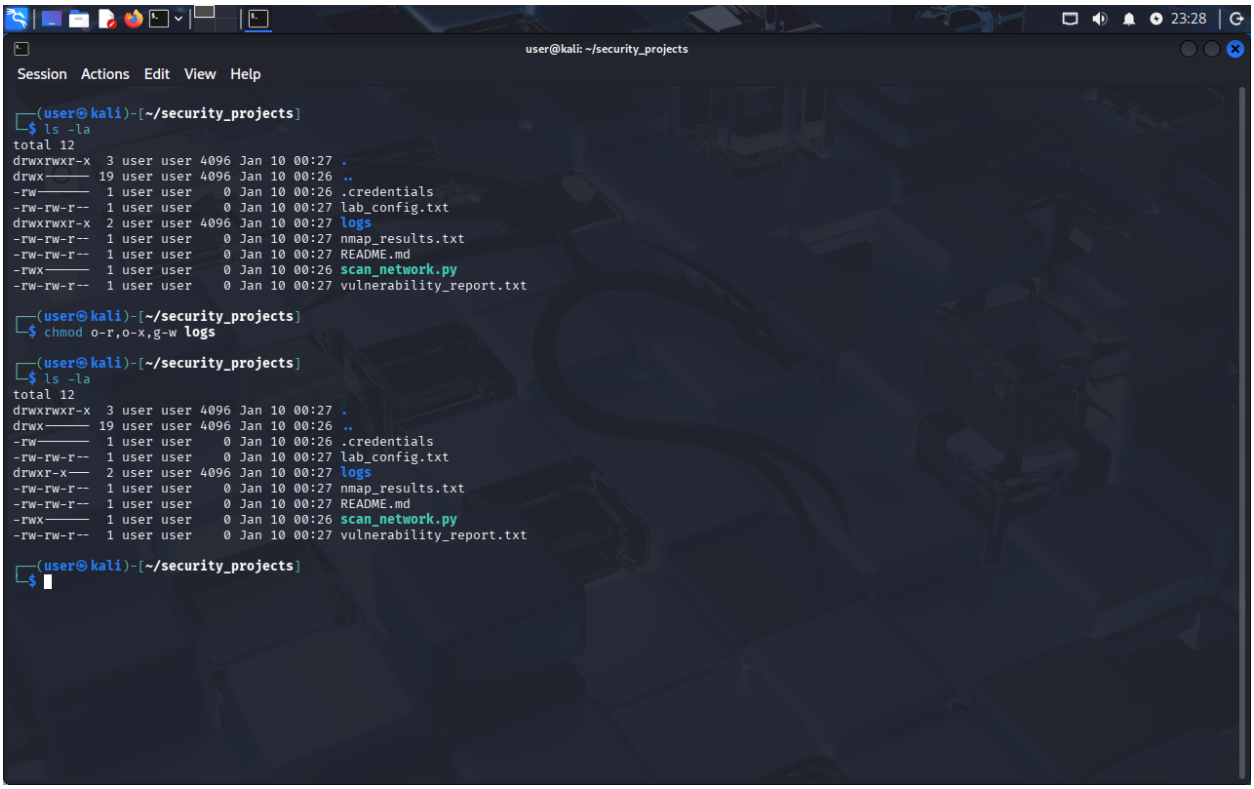
My scan_network.py script runs Nmap scans on my home lab. Looking at the initial output, it had -rwxrwxr-x permissions—my group had full read, write, and execute access, and others could read and execute it. There's no reason for other users to run this script, and they shouldn't even be able to read it since it contains IP addresses of my lab setup.



I ran chmod g-r,g-w,g-x,o-r,o-x scan_network.py to remove all group and other permissions. After verifying with another ls -la, the permissions now show -rwx------. Now only I can read, write, and execute the script. This follows the principle of least privilege—I'm the only one who needs to run these scans, so I'm the only one with access.

Protecting Log Files

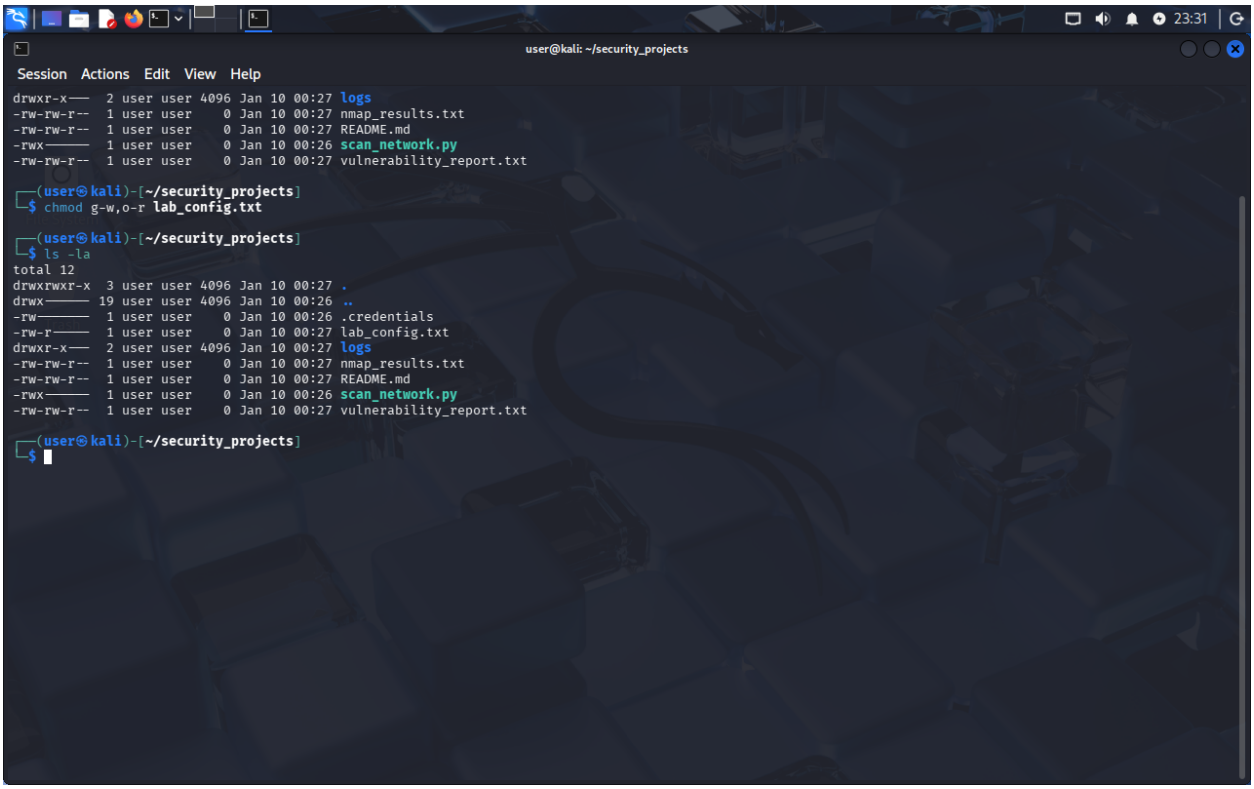
I have a logs directory where my security monitoring scripts dump output. These logs sometimes contain information about vulnerabilities I've found in my lab or network traffic patterns. Looking at the initial output, the directory had drwxrwxr-x permissions—my group could read, write, and execute, and others could read and execute.



I ran chmod o-r,o-x,g-w logs to remove read and execute permissions from others, and remove write permissions from the group. After verifying with ls -la, the permissions now show drwxr-x---. I have full access, my group can read and navigate the directory, and others are completely locked out. While I could have gone with drwx------ to lock it down completely, I kept group permissions in case I want to let a friend who's also interested in cybersecurity look at my setup.

Securing Configuration Files

I have a lab_config.txt file that documents my network setup, including device IPs, default passwords for test accounts, and notes about intentional vulnerabilities I've configured for practice. Looking at the initial listing, this file had -rw-rw-r--permissions, which meant my group had read and write access, and others on the system could read it.



I used chmod g-w,o-r lab_config.txt to remove write permissions from the group and read permissions from others. After running ls -la again, the permissions now show -rw-r-----. I can still read and write it, my group can read it if needed, but other users on the system are completely locked out.

Summary

I went through my security projects directory and systematically tightened file permissions to follow security best practices. Using ls -la, I identified which files had overly permissive settings that could expose sensitive information. Then I used the chmod command multiple times to apply appropriate restrictions based on the principle of least privilege.

I secured my .credentials file by removing all group and other permissions, ensuring only I can access API keys and sensitive credentials. My scan_network.py script now has permissions restricted to just the owner since there's no reason for anyone else to read or execute it.
The logs directory was locked down to prevent other users from accessing it while still allowing my group to read logs if needed for collaboration. Finally, I removed write access from the group and all access from others on lab_config.txt to protect my network documentation.

Even though this is my personal system, treating it like a production environment helps me build good security habits and demonstrates that I understand access control principles. This type of permission management is essential in real-world cybersecurity work where protecting sensitive data and limiting access are fundamental security practices.