Personal Report: Data Leakage Detection Script (DLDS)

Project Title:

Data Leakage Detection Script (DLDS) – Scanning Local Systems for Sensitive Data

---

Overview

As part of my cybersecurity development projects, I created a Python-based tool called the Data Leakage Detection Script (DLDS). The goal of this script is to identify exposed sensitive information, such as API keys, passwords, and personally identifiable information (PII), in local file systems. While the script is built to run on local directories, it can be easily adapted to scan cloud storage environments (simulated or real).

This project simulates how security teams might automate internal audits or pre-commit scans to catch data leaks before pushing to public or shared repositories.

How It Works

The script uses regular expressions to search for known data leakage patterns, including:

- AWS access and secret keys
- Google API keys
- Generic API keys
- Passwords and email addresses
- Social Security Numbers (SSNs)

It recursively scans files in a target directory, logs findings to the console, and also writes a detailed log to a file named `dlds_log.txt`. I also implemented an optional feature to send alerts to a Discord webhook for real-time monitoring.

Errors and Debugging

While running the script initially, I encountered an argument parsing error. I had forgotten to include the required directory argument when executing the script, which triggered the following message:

makefile
CopyEdit

```
usage: DLDS.py [-h] directory
DLDS.py: error: the following arguments are required: directory
```

This was a simple but important mistake. I resolved it by adding the directory path explicitly when calling the script and also added better error handling in the code to guide users when no argument is supplied.

As part of documenting this process, I took photographic evidence of the error and the corrected command-line usage. This not only shows the iterative debugging process but also serves as proof of understanding and hands-on troubleshooting.

Improvements Made

After resolving the argument issue, I enhanced the script by adding:

- Automatic logging of findings to a text file for auditing purposes
- Timestamps in the log file to track when leaks were discovered
- Clearer console output
- Input validation to ensure that only valid directories are scanned

These changes make the tool more robust, especially for use in larger or more automated environments.

Reflections

This project helped me solidify my understanding of secure coding practices and pattern recognition for sensitive data. It also highlighted how small oversights—like a missing argument—can break functionality but also become learning opportunities.

Moving forward, I plan to expand this tool with features like:

- Scanning only specific file types
- Integration with Git history scanning
- Option to redact or quarantine flagged files

Photographic Evidence