**System Software and Tools Used**

1. **Development Tools:**
   * **Visual Studio**: IDE for compiling and debugging the C++ code.
   * **Python Interpreter**: To run the Python backend script (virusChecker.py).
2. **Programming Languages:**
   * **C++**: For developing the graphical user interface (GUI) using Windows API.
   * **Python**: For backend malware detection and analysis.
3. **Libraries:**
   * **Windows API**: For GUI development and system interactions in C++.
   * **pefile**: For parsing Portable Executable (PE) file formats in Python.
   * **joblib**: For loading the pre-trained machine learning model.
   * **logging**: For maintaining logs of malware detection activities.
   * **os** and **sys**: For system path handling and command-line interactions.
   * **math**: For entropy calculations in malware detection.
   * **pickle**: For loading serialized feature files.
4. **Machine Learning Tools:**
   * Pre-trained malware detection model (model.pkl), used with Python libraries.
5. **Operating System:**
   * **Windows OS**: Required for running the desktop application and its dependencies.
6. **Dependencies:**
   * Required Python packages (installable via pip):
     + pefile, joblib, etc.

**Documentation: File Scan Dashboard Application**

This documentation explains the purpose and functionality of various parts of the File Scan Dashboard application code. The program is a Windows desktop application for scanning files and directories using a Python script.

**Global Variables**

* **szWindowClass**: Holds the class name for the application window.
* **szTitle**: Holds the title of the application window.
* **hInst**: Stores the instance handle of the application.
* **PYTHON\_SCRIPT**: Specifies the path to the Python script that performs the file scanning.

**Key Functions**

**1. runPythonScriptWithOutput**

Executes a Python script with a given command and captures its output.

* **Parameters**: A command string containing the script path and arguments.
* **Returns**: A string containing the script's output or an error message if execution fails.

**2. openFileDialog**

Displays a file selection dialog and returns the selected file path.

* **Parameters**: The parent window handle (hwnd).
* **Returns**: A string containing the path of the selected file or an empty string if no file is selected.

**3. openFolderDialog**

Displays a folder selection dialog and returns the selected folder path.

* **Parameters**: The parent window handle (hwnd).
* **Returns**: A string containing the path of the selected folder or an empty string if no folder is selected.

**Main Application Entry Point**

**WinMain**

The entry point for the application:

* Registers the window class.
* Creates the main application window.
* Enters the message loop to handle user interactions.

**Message Handler**

**WndProc**

Handles messages sent to the application window:

* **WM\_CREATE**: Initializes the UI elements, such as buttons and labels, and displays the splash screen.
* **WM\_TIMER**: Transitions from the splash screen to the main UI after a delay.
* **WM\_COMMAND**: Handles button click events:
  + **SCAN (5)**: Opens a file dialog, runs the Python script on the selected file, and displays the output.
  + **SCAN ALL (6)**: Opens a folder dialog, runs the Python script on the selected folder, and displays the output.
  + **EXIT (7)**: Closes the application.
  + **UPDATE VERSION (8)**: Displays a placeholder message for updating the version.
* **WM\_DESTROY**: Cleans up resources and exits the application.

**UI Elements**

**Buttons:**

* **SCAN**: Allows the user to select a file and run a scan.
* **SCAN ALL**: Allows the user to select a folder and run a scan.
* **EXIT**: Exits the application.
* **UPDATE VERSION**: Placeholder for version update functionality.

**Labels:**

* **Splash Screen Labels**:
  + "ProtectWare": Displayed as the main title.
  + "Version 1.0": Displayed as the subtitle.
* **Output Display**: Shows the results of the scan or other messages.

**Styling**

* Fonts: Uses the "Segoe UI" font with bold styling for better readability.
* Background: A solid blue color to enhance the visual appeal.

**Workflow**

1. The application starts with a splash screen displaying the title and version.
2. After 3 seconds, the main UI is displayed.
3. Users can choose to:
   * Scan a file or folder.
   * Exit the application.
   * Update the application (placeholder).
4. Scan results or messages are displayed in the output area.

**Notes**

* **Python Script Path**: Ensure the path to the Python script (virusChecker.py) is correct and accessible.
* **Error Handling**: The program reports errors if the Python script fails or returns a non-zero exit code.

**Documentation: Malware Folder and File Scanner**

**Overview**

This Python script scans files or folders for potential malware using a pre-trained machine learning model. It extracts features from Portable Executable (PE) files, evaluates them, and classifies each file as either "Legitimate" or "Malware". The results are logged with timestamps for tracking purposes.

**Script Workflow**

1. **Input Validation**:
   * Accepts a single argument: the path to a file or folder.
   * Validates the path.
2. **File or Folder Processing**:
   * If the path is a file, it processes the file directly.
   * If the path is a folder, it recursively processes all files in the folder.
3. **Feature Extraction**:
   * Extracts characteristics from PE files.
   * Computes entropy, resources, version information, and other PE header details.
4. **Classification**:
   * Loads a pre-trained machine learning model.
   * Predicts if the file is legitimate or malware based on extracted features.
5. **Logging Results**:
   * Logs the file path, classification result (Legitimate/Malware), and timestamp to scan\_log.txt.

**Key Functions**

**1. setup\_logging()**

Initializes the logging mechanism to write results to scan\_log.txt in the format:

Timestamp, File Path, Status

**2. get\_entropy(data)**

Computes the Shannon entropy of the provided data, a measure of randomness useful for identifying malicious content.

**3. get\_resources(pe)**

Extracts information about resources embedded in a PE file, including their entropy and size.

**4. get\_version\_info(pe)**

Retrieves version information from the PE file, such as file version and product version.

**5. extract\_info(fpath)**

Extracts a wide range of PE header features, including section entropy, imports, exports, and other key details.

**6. checkFile(file)**

* Loads the pre-trained model and feature mappings.
* Extracts features from the file.
* Predicts whether the file is legitimate or malware.

**7. log\_result(file\_path, status)**

Writes the result of the scan to the log file. Example entry:

2025-01-21 14:35:00, C:\Path\To\File.exe, MALWARE

**8. process\_file(file\_path)**

Processes a single file:

* Extracts features.
* Classifies the file as "Legitimate" or "Malware".
* Logs the result.

**9. process\_folder(folder\_path)**

Recursively processes all files in the specified folder, handling errors gracefully and logging them.

**10. main()**

The entry point of the script. It:

* Validates command-line arguments.
* Determines if the input is a file or folder.
* Invokes the appropriate processing function.

**Log File Details**

The script creates/updates a log file named scan\_log.txt in the working directory. Each entry includes:

* **Timestamp**: Date and time of the scan.
* **File Path**: Full path of the scanned file.
* **Status**: Result of the scan ("Safe" or "Malicious").

**Usage**

**Command-Line Execution**

Run the script using:

python <script\_name>.py <path\_to\_file\_or\_folder>

**Examples**

1. **Scanning a Single File**:
2. python malware\_check.py sample.exe
3. **Scanning a Folder**:
4. python malware\_check.py C:\Users\User\Downloads

**Error Handling**

* Logs any errors encountered during file processing to the log file with details.
* Prints the error to the console for user awareness.

**Notes**

1. Ensure the model file (model.pkl) and feature mappings (features.pkl) are correctly placed in the specified paths.
2. The script is designed for Windows PE files and may not work with non-PE file types.
3. Modify paths in the script (C:\Users\adith\source\repos\ProtectWare) if your setup differs.