**Research and Development Log: Handling E01 Forensic Images in Python**

**Introduction**

This document outlines the research and development activities undertaken to access and extract user-related information from a forensic disk image in Expert Witness Format (.E01) using Python. The objective of this work is to support the development of a digital forensics application. The process involved identifying suitable Python libraries, setting up a working environment, experimenting with various programmatic techniques, and overcoming practical challenges. Both successful and unsuccessful attempts are documented, in alignment with academic best practices for research transparency.

**Environment and Initial Setup**

The development environment was based on a Windows 10 system using Miniconda to manage a dedicated Python environment named forensics. The decision to use Miniconda was motivated by previous difficulties encountered when compiling legacy forensic libraries on Windows using standard Python environments and Visual Studio toolchains.

Python version 3.10 was selected for compatibility with the primary libraries of interest, and the following packages were targeted:

* pyewf (via pyewf-python)
* dfvfs (Digital Forensics Virtual File System)
* dfwinreg (for Windows Registry analysis)

**Challenges with Library Installation**

Initial efforts focused on installing pyewf-python, which is required to interface with .E01 images. Several sources for precompiled binaries, including GitHub repositories and the libyal project, were either unavailable (HTTP 404) or did not offer current builds for Windows. Installing from source was deemed impractical due to the lack of legacy dependencies on modern Windows systems.

Instead, the solution was to install dfvfs, which abstracts file system access and integrates pyewf internally. After resolving environment path issues and ensuring consistent shell usage, the installation was successful, and basic interaction with .E01 images became possible.

**File System Exploration**

The .E01 file provided by the university was confirmed to contain a Windows XP installation by inspecting its partition structure. Initial scripts failed due to incorrect assumptions about volume structures and path specification usage in dfvfs. After extensive debugging—including inspection of ScanContext attributes and path resolution failures—partition /p1 was successfully identified as containing the Windows XP file system.

Scripts were then refined to:

* Enumerate top-level directories
* Confirm the presence of user-related folders (e.g., Documents and Settings, Users)
* Filter out Linux-based partitions which were present in other volumes

**Listing User Profiles**

User profiles were initially searched under Documents and Settings, a common location for Windows XP systems. However, the folder was empty in the image. A successful user listing was eventually performed by switching to the Users directory, revealing profiles such as Wes Mantooth and Dracula.

This outcome confirmed the image’s unconventional structure, requiring flexible handling of multiple possible profile storage paths.

**Attempting Registry Parsing with dfwinreg**

Subsequent efforts focused on extracting user information from the Windows SAM hive. The initial approach was to use dfwinreg’s high-level WinRegistry class. However, the API did not provide access to a registry file reader as expected. Attempts to use internal properties such as \_registry\_file\_reader were unsuccessful due to NoneType errors.

The developer explored help() and introspection tools (dir()) to identify callable methods and attributes. This process clarified that the correct object for low-level registry parsing was WinRegistryFile.

**Solution: Copying the Registry Hive**

Due to limitations in reading registry files directly from virtual file systems, a new strategy was adopted: copying the SAM file from the .E01 image to the local file system. This was achieved using dfvfs to create a read stream from the partition and write the binary data to a local path. This method proved successful, providing a verified copy of the registry hive file that can be parsed with any external tool, including dfwinreg, python-registry, or regipy.

**Conclusion**

The development process thus far has involved significant troubleshooting, library evaluation, and reverse-engineering of documentation for forensic tools. Despite several false starts and unexpected image contents, the objectives of file system access, user profile enumeration, and registry hive extraction have been achieved.

Moving forward, the locally extracted registry hives will be parsed for user account metadata, and this functionality will be integrated into the larger application. This phase of work reflects both the complexity and the iterative nature of forensic software development in academic contexts.