Adaptation of PyFlag to Efficient Analysis of Seized Computer Data Storage

Aleksander Byrski Ph.D.
Assistant professor at the Department of Computer Science
AGH University of Science and Technology
Krakow, Poland
olekb@agh.edu.pl

Wojciech Stryjewski
AGH University of Science and Technology
Krakow, Poland
stryjew@student.agh.edu.pl

Bartłomiej Czechowicz
AGH University of Science and Technology
Krakow, Poland
czechow@student.agh.edu.pl

ABSTRACT

Based on existing software aimed at investigation support in the analysis of computer data storage seized during investigation (PyFlag), an extension is proposed involving the introduction of dedicated components for data identification and filtering. Hash codes for popular software contained in NIST/NSRL database are considered in order to avoid unwanted files while searching and to classify them into several categories. The extension allows for further analysis, e.g. using artificial intelligence methods. The considerations are illustrated by the overview of the system's design.

Keywords: digital forensics, exploration of seized data storage, disk analysis, PyFlag, system log analysis

1. INTRODUCTION

An analysis of seized computer data storage poses interesting challenges for a forensic expert. Files containing interesting data (e.g. captured network traffic, illegal multimedia) may be difficult to find, especially when dealing with (nowadays popular) large storage capacities. The data contained on the storage (hard disks, DVD, flash disks etc.), may be located in multiple nested directories of random structure, therefore being hard to analyze by a forensic expert equipped with simple tools for directory browsing only.

In order to help a forensic expert in finding interesting, unknown or wanted data files, while at the same time ignore files containing unimportant or already known data, a dedicated software should be designed. The possibilities of extracting high-level information (e.g. by the means data mining, social network data extracting) of certain data files (e.g. system or server logs) would be also welcome (see Cohen 2008). The system should process very large files, should be scalable (i.e. more available hardware should lead to faster processing) and should offer the possibility of further extending its features (e.g. by means of component-based architecture (see Szyperski 2002)).

At the same time, the software and the whole acquisition process should fulfill requirements posed by law (in order to retain the evidentiary value of the seized data). Such requirements may be referred, e.g. in (US DOJ 2004) for U.S. law. In Poland these requirements are usually fulfilled by using hardware blockers (for writable media such as hard disks) (see Goc and Moszczynski 2007; Fischer 2000). The software should also be easily adaptable to the particular examination requirements of the Police from any other country.

In the course of this paper, the extension PyFlag/FLUSH of a popular digital forensics tool PyFlag (see Cohen and Collett 2005) is presented. The software was adapted to the specific demands of forensic experts after consultation with Polish Police investigators. The presented system was designed according to the software component paradigm.

2. PYFLAG – UNIVERSAL FORENSIC DISK ANALYSIS SOFTWARE

Flag (Forensic and Log Analysis GUI[1]) is an advanced forensic tool for the analysis of log files and forensic investigation. PyFlag, Python-based implementation of Flag, was originally designed by the Australian Department of Defense and later released under the Gnu Public License. PyFlag has the following features (see Cohen 2008):

- network forensics: PyFlag can analyze network traffic captured in TCPDump format, it supports many popular network protocols (e.g. HTTP, MSN Chat, SMTP and POP and others),
- log analysis: PyFlag can analyze many log files (e.g. Apache Web Server, Microsoft IIS), user may define his own log analysis presets,
- disk forensics: PyFlag can analyze different file formats (e.g. windows event log, PCAP – Tcpdump format, PDF, HTML, zip),
- file carving: recently file carving capability (finding files based on their content in raw disk partitions when no file system is present) was also introduced into PyFlag,
- memory forensics – PyFlag can analyze captured RAM images from Windows XP SP2 (using Volatility Framework).

1 http://www.pyflag.net

User interface of PyFlag is web based, therefore it can be deployed on a central server and shared with a number of users at the same time. Data is loaded into cases which keeps information separated (see Cohen and Collett 2005).

In Fig. 1 an overview of the PyFlag structure is shown. IO Source is used to access the file system mounted, e.g. as a raw image (created by the use of Unix/Linux dd program), ISO image or file system directory. VFS stands for Virtual File System and is used to gather information about all files found in the file system using the hierarchical directory structure mirroring the original directory structure of the file system and containing so called i-nodes (string describing how to access an object, the concept similar to i-nodes from Unix/Linux file systems).

File System loader is used to populate VFS with i-nodes assigned to every possible data sources present in the file system. Scanners are modules operating on VFS i-nodes and gathering specific information about them, e.g. there may exist specific scanner used to find Apache Web Server logs or to process ZIP archive in order to extract the files contained there and include them into VFS. Database is used to store the outcome of scanning and may be used to browse information by the GUI.

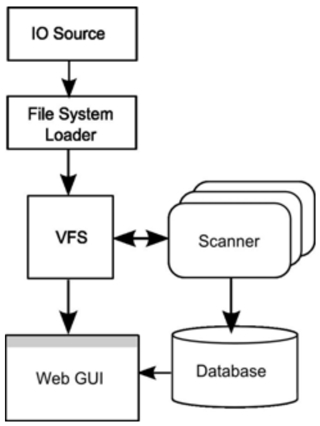

Figure 1: Overview of PyFlag structure (Cohen and Collett 2005)

It is easy to see that this structure is very easily extensible. For example, the core functionality of PyFlag and construction of VFS, allow searching, accessing and processing contents of the examined file system regardless of its nature. Moreover, different additional filters may be defined in order to search for different potentially interesting files such as personal communicator archives, web server logs, mailboxes etc.

The abstract structure of VFS and the possibility of adding new filters allow for further extension and introducing more sophisticated methods for data analysis (e.g. artificial-intelligence based such as data mining or expert systems).

3. NIST/NSRL DATABASE

National Software Reference Library (NSRL[2]) is a library where information about different types of popular software is collected. This library holds, among others, signatures of the files associated with certain software releases and additional meta data.

The project is sponsored by many organizations, e.g. U.S. Department of Justice's National Institute of Justice (NIJ) and National Institute of Standards and Technology (NIST). It is aimed at supporting the effective use of software technologies in the investigation of computer crimes. This library can be used by penal prosecution agencies, police or other forensic organizations.

Popular volumes containing operating systems incorporate thousands or even hundreds of thousands files, which poses a serious problem for the investigator who wants to find a certain file or all files of certain type. NSRL helps investigate large volumes of seized data , because it contains signatures of files that may be found in popular software releases, therefore the investigator can avoid localizing these files and focus on those unknown. In this way the NSRL database can be used as a basis for automatic filtering software able to recognize and classify the files present in the seized file system. The process of classification is based only on the the contents of the file (not the name, which may be deceptive) using popular hash codes MD5 and SHA1 (see Oppliger 2001).

The NSRL database provides the following data for the files held in popular software:

- SHA1 and MD5 hash codes,
- Name and size of file,
- CRC32 code,
- Product code,
- Operating system code.

PyFlag supports NIST/NSRL database import, however in order to use this information efficiently, there arises a need to modify the data contained in the database and to add additional meta data. These requirements will be covered later in this contribution.

4. PYFLAG/FLUSH FUNCTIONAL REQUIREMENTS

PyFlag was designed and implemented as a tool used mainly for discovering log files in the file system, displaying and manipulating them. However, the software

2 http://www.nsrl.nist.gov

has a great potential of possible adaptation to different needs of users, especially as the project is open-source and written in a very clear and simple-to-use language (Python) (Van Rossum and Drake Jr. 2006; Summerfield 2008). Moreover, some interesting concepts found in PyFlag can easily be reused and populated into more advanced environments, e.g. virtual file system allows creating a model of the file system which may be further manipulated by any algorithm, in particular embedded in one of PyFlag filters.

PyFlag has been chosen by the officers of Polish Police as a tool for exploring seized data storage, however, in order to improve and adapt it to their specific demands, several improvements had to be made.

After the analysis and consultations, the extension (PyFlag/FLUSH) fulfills the following demands:

- Localizing and pointing out different files of interest, e.g. communicator archives, mailboxes, multimedia files and others, based on the file content (not extension or naming convention).
- Correct handling of unwanted files (present in popular operating systems), such files should not be taken into consideration in analyses and reports.
- Correct handling of wanted files. In pursuit of illegal (mostly containing pedophilia) graphics, movies etc., the investigator would like to have the possibility of importing hashes describing common illegal files and include them into a report.
- Reporting occurrences of unknown files (not defined as known and not defined in the database of common files).
- The possibility of importing predefined hashes for known and unknown files.
- Integrate ready-to-use software tools for file carving (recovering deleted or cached files), compressed archive handling.

5. DESIGN AND IMPLEMENTATION OF PYFLAG/FLUSH

The key feature of the PyFlag/FLUSH is based on classifying the files that may be found by the system into four categories:

- Wanted Files - all files found in the data storage whose SHA1 or MD5 hash-code were given by the user; they usually contain common illegal material present in many seized computers.
- Known Files - all files which SHA1 or MD5 hash-code may be found in extended NIST/NSRL database.
- Unknown Files - all files which were neither classified as Wanted or

NIST/NSRL.

- Suspected Files - all files which were selected by the investigating officer as those potentially containing illegal material - they must be examined by a forensic expert.

The following functions were added to PyFlag during the implementation of PyFlag/FLUSH so as to meet the functional requirements identified during the interview with Police investigating officers:

- Load Wanted Files - loading into hash database selected files, which are to be found by the user in the data storage. Such files usually contain illegal content (e.g. pedophilia). For each of the selected files the system computes SHA1 and MD5 hash codes which are added into database. If the system finds one of these files during scan, it will be added to the category Wanted Files. After finishing the scan, the user will get the appropriate report containing wanted files.
- Load Wanted Hashes –analogous to Load Wanted Files, loading hashes written in simple CSV-like text file, containing SHA1 or MD5 hashes (both of them may be given at the same time).
- Load Known Database – loading hashes from NIST/NSRL format. The hashes are used for classification of the files found during the examination of the seized data storage.
- Extend Known Database - loading into hash database selected files that should not be found by the system, i.e. they should be treated exactly as the files contained in NIST/NSRL database.
- Generate (HTML, CSV) Report – generating report in HTML or CSV format containing a list of found files for each category.
- Find Known Files – looking for the already known files in the system (their hashes are contained in NIST/NSRL database).
- Find Wanted Files - looking for the files classified as Wanted Files.
- Find Suspected Files - looking for the files that are unknown (neither contained in NIST/NSRL database, nor imported by the user). Actually these are all the files without corresponding hash in the database.
- Find Log Files – looking for HTTP server log files: Apache and IIS (based on the file's content, not the name), more localizing filters are planned (e.g. communicator archives).
- Export Suspected files - copying selected files into chosen directory, the directory structure of the files is preserved and additional report (HTML, CSV) is generated.

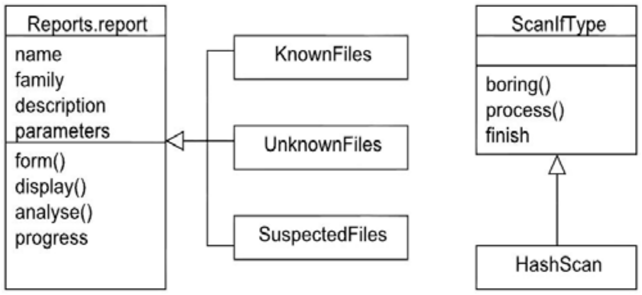

Figure 2: UML Class diagram for PyFlag/FLUSH

Taking advantage of the extension possibilities of PyFlag, the authors managed to enclose our functional requirements into a group of classes that may be added to the basic PyFlag environment. Due to Python programming language nature, no compilation is needed. Therefore, in order to implement the functions that operate on VFS, and to analyze the contents of storage at the same time using the information contained in NIST/NSRL database, all that had to be done was to inherit from PyFlag form classes (Reports.report). This class has the following methods:

- form(): displays the main dialog window for the report,
- analyse(): processes the informations provided by the dialog window defined in form(),
- progress(): informs about the progress of analysis; these two methods are implemented usually for time-costly applications,
- display(): displays the information contained in database.

In this way the classes for all mentioned functions (e.g. for finding UnknownFiles, KnownFiles, SuspectedFiles etc.) were implemented. All these classes are bound with the graphical user interface of PyFlag and are responsible for displaying the results in the PyFlag forms.

The relations among the classes was presented in Fig. 2 in the form of UML Class Diagram (Booch et al 1998).

In order to write the scanner for analyzing the contents of the files present in the data storage, the authors had to implement the interface ScanIfType having the following methods:

- boring(): used for checking whether the scanner needs the subsequent part of the analyzed data,

- process(): updates the state of the object computing the MD5 and SHA1 hashes for the processed file,
- finish(): uses the computed hashes for checking the file in NIST/NSRL database in order to classify it into classes already mentioned.

In this way HashScan scanner was implemented.

In Fig. 3, an overview of the structure of database implemented in PyFlag/FLUSH was presented using one of the most popular diagrams used in structural analysis - Entity Relationship Diagram. In order to quickly clarify the model presented in this figure, one should notice that there is 1:1 mandatory relationship between File and Inode tables, 1:1 optional relationship between File and SuspectedFiles, 1:N optional relationship between WantedHashes and WantedFiles and so on. For detailed description of the notation see e.g. Chen 2002.

There are three main groups of tables, the first one (in red region) comes directly from PyFlag and others were added during implementation of PyFlag/FLUSH. Tables displayed in green region were implemented in order to add the functionality demanded by the police investigators (the possibility of classifying the files found in the file system into distinct groups). Tables displayed in blue region are used to store the NIST/NSRL data and, moreover, they will be usable in the future for more sophisticated analysis (e.g. identifying of installed software, described later in this paper).

Detailed description of the tables is given below (see Fig. 3):

- File and Inode - legacy tables from PyFlag. They are used to store the information about the files and VFS i-nodes.
- KnownFiles, UnknownFiles, WantedFiles, WantedHashes - tables containing the hash codes and i-nodes of the files found during an examination performed in a certain case.
- hashdb_Hashes - contains unique values of hash codes (because files may be present in different software, which introduces multiple identical hash codes into the table nsrl_hashes).
- hashdb_Hashes was introduced in order to speed up the search process.
- nsrl_hashes - contains the informations about the the software to be found in the examined storage devices.
- nsrl_products, nsrl_products, nsrl_mfg - these tables are used to store the meta data concerning operational system, identification of software products and software manufacturers, and will be extensively used in the future.

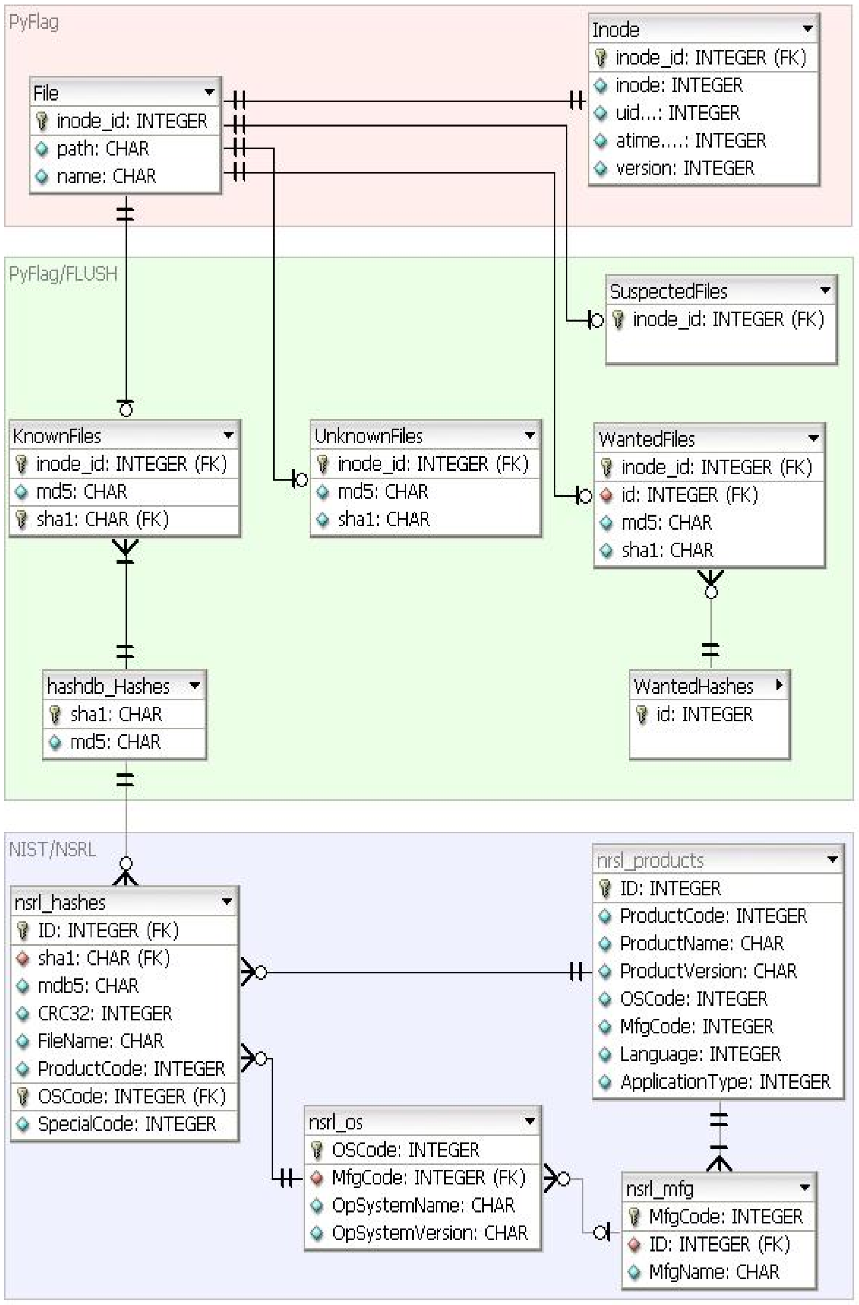

Figure 3: Overview of PyFlag/FLUSH database structure

6. INSTALLATION AND USING SCENARIOS OF PYFLAG/FLUSH

The installation process of PyFlag/FLUSH consists mainly in installing PyFlag and involves using standard Linux commands to acquire necessary packages and compile them.For details refer to (Van Rossum and Drake Jr. 2006). After successful installation of PyFlag, PyFlag/FLUSH files should be copied into the directory containing PyFlag plugins[3]. Now the latest version of NIST/NSRL database should be downloaded[4] and imported into PyFlag using the command LoadHashes, Load NSRL Hashes.

PyFlag/FLUSH may be used according to one of the following scenarios:

- Standalone installed - PyFlag/FLUSH may be installed in the fully configured Linux or Windows machine and then used as a means to investigate data storage connected with the use of a hardware blocker. According to the capabilities of PyFlag, an image of the file system may also be examined. This scenario should be suitable for individual forensic experts (also working on scene).
- Standalone virtual - Instead of installing PyFlag/FLUSH directly into the system, it may be installed in a virtual machine (e.g. VMWare based) (the authors recommend Ubuntu Linux as a distribution easy to install and handle, already tested with PyFlag/FLUSH). Now, the system may be run in most of the popular operating systems with the use of VMWare Player (distributed as freeware). Again, the examined data storage itself or its image may be connected to PyFlag. This scenario is practically the same as the previous one, at the same time having the advantages of virtualization (e.g. easy duplication of already installed and configured system, hardware independence).
- Server based - A dedicated server may be used to run PyFlag/FLUSH, and using the web-based interface many experts can use it simultaneously. However, in order to connect the examined data storage to the server, the most convenient way would be to share it across the network. For this configuration, the increased network load during the analysis should be considered.

In all the above mentioned cases, a need arises to provide PyFlag/FLUSH with enough disk space to hold NIST/NSRL database and to allow operation (e.g. counting hash-codes). During testing, the required data storage used for PyFlag/FLUSH operation was usually at least equal to the examined data. Considering this, the third solution (server based) might be optimal for performing the examination in a forensic laboratory (with dedicated server for disk storage analysis), especially when many experts want to share the

3 Usually located in /usr/local/lib/python/pyflag/Plugins

4 See http://www.nsrl.nist.gov/Downloads.htm

NIST/NSRL database.

The first or the second solution would be optimal for the installation on mobile computer or for selected forensic experts only.

7. FURTHER EXTENSIONS: DETECTION OF INSTALLED SOFTWARE

The analysis of seized data storages is usually aimed at collecting information about the user by localizing and accessing his archive files such as mailbox, communicators archives, history of browsed WWW pages, private documents or multimedia files. Testing of legality of installed software is also in the scope of the investigator's interests.

The problem of legality testing usually arises while seizing hardware from the users suspected of any computer crimes such as illegal software exchange or pedophilia multimedia share.

The investigating officer can manually search in the file system for the installed popular commercial software and write the report on his own, however, several problems arise:

- the simplest solution would be to run the system and check the installed software, but there might be no possibility of getting an account with sufficient privileges or the hardware might be broken (even intentionally by the user),
- more complex solution would be to browse the mounted disk partition with standard set of search-tools (such as find, grep and other basic utilities of Unix-like systems), however, this process will be long, tedious and error-prone,
- the best solution would be to relay on a more complex file searching tool (e.g. PyFlag/FLUSH), extending it to capabilities of testing the found groups of files against accordingly modified NIST/NSRL database, which containins meta data describing versions of the software.

This possible extension of PyFlag/FLUSH should be based at least on some simple statistical analysis of the files found (and comparing them with the hash-codes contained in the NIST/NSRL database). More sophisticated methods will employ such methods of artificial intelligence as expert systems or fuzzy logic (see Russel and Norvig 1995).

Without such complex techniques it would be hard to determine the version of the software, because, e.g. a newer one might contain files from the older version. The proposed further extension of PyFlag/FLUSH is currently under development and will be presented in the future.

8. CONCLUSIONS

In the course of paper the extension of a well-known forensic utility PyFlag – PyFlag/FLUSH was presented. The extension was constructed according to the requirements analysis prepared with the help of Polish Police investigating officers.

The authors are aware, that many functional requirements described by us may be met by such popular forensic tools as EnCase[5], however we propose our solution as an open-source extension of PyFlag with emphasis on modularity and component-orientation, hoping that these features, although may not gather direct attention of forensic investigators, but should significantly reduce the amount of time spent for further adaptation and extending of the proposed system by the software developers (e.g. the ones hired for adaptation of the system for the specific needs of a certain research and investigation institute) .

The extension was prepared using the Python programming language in order to retain coherency with the original PyFlag. The presented architecture of PyFlag allows to extend its capabilities and we aimed at preparing certain software framework (according to the notions of software components) so that in the future, new extensions may be possible.

A more sophisticated analysis will be possible by introducing into the framework artificial-intelligence methods such as basic expert systems, allowing, e.g., the analysis of the contents of the seized storage.In the future, PyFlag/FLUSH will allow to identify the installed software (e.g. for checking its legality).

Although PyFlag/FLUSH responds mainly to the demands of Polish Police investigating officers, the nature of the software makes it ready for further improvements and adaptation to the particular user's needs, so that it may be adapted for other investigators.

REFERENCES

Booch, G., Rumbaugh, J., Jacobson, I. (1998). The Unified Modeling Language User Guide. Addison-Wesley.

Chen, P. (2002). Entity-relationship modeling: Historical events, future trends, and lessons learned. In: In: Software Pioneers: Contributions to Software Engineering. Springer, pp. 297-310.

Cohen, M. (2008). PyFlag an advanced network forensic framework. Digital Investigation 5 (1).20

Cohen, M., Collett, D. (2005) Python Forensic Log Analysis Gui (PyFlag). http://www.pyflag.net.

5 http://www.guidancesoftware.com

Fischer, G. (2000) Przestępstwa komputerowe i ochrona informacji. Aspekty prawno-kryminalistyczne (Computer crimes and information security. Law and criminalistic aspects, in Polish). Kantor Wydawniczy Zakamycze, Warsaw.

Goc, M., Moszczyński, J. (Eds.) (2007). Ślady kryminalistyczne (Criminalistic traces, in Polish). DIFIN Warsaw.

Oppliger, R. (2001). Contemporary Cryptography. Artech House.

Russell, S., Norvig, P., 1995. Arti_cial Intelligence: Modern Approach. Prentice Hall.

Summerfield, M. (2008). Programming in Python 3: A Complete Introduction to the Python Language. Addison-Wesley Professional.

Szyperski, C. (2002). Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

US Department of Justice (2004). Forensic examination of digital evidence: A guide for law enforcement. Tech. rep., U.S. Department of Justice, National Institute of Justice Special Report 199408.

Van Rossum, G., Drake Jr., F. (2006). The Python Language Reference Manual (version 2.5). Network Theory Ltd.