COMP40780

Digital Investigations Project
Digital Forensics Shell

Project draft

Andrea Barberio
Student #--------
6 March 2016

Introduction

This project plan aims to identify the key points of the design and implementation of a digital forensics shell. The provisional name is FRESH, to remind its nature of Forensic Shell, but it is subject to changes. Despite the name “shell”, due to its layered architecture it can be seen as a forensic framework with a shell interface on top.

License

FRESH will be released under an open source license, the *2-clause BSD*[1], in order to:

- **encourage collaboration** in the forensic community
- encourage **rapid features development and bug fixes**
- **facilitate the adoption** in commercial and non-commercial environments
- **guarantee the auditability** of the source code and its functionalities
- **avoid the risks of vendor lock-in**
- give something in **return to the Open Source community**, from which everyone benefits every day

Goals

The shell is meant to be a tool for forensic examiners to automate and simplify the extraction and the analysis of digital artifacts from binary dumps, such as a phone or a disk image. It is intended as a tool for experts, but the long term goal is to have a framework on top of which it will be possible to build simple user interfaces for more general and non-technical consumption.

The capabilities of this shell will include, but will not be limited to, the following:

- ability to **automate the extraction** of the most common forensic artifacts from digital disk images
- **simplify the manual examination**, in order to get better insight of known artifacts, and to simplify the reverse engineering of unknown artifacts;
- ability to **analyse**, automatically and semi-automatically, multiple types of **forensic artifacts**

These points are foundational for the forensic shell, and constitute the requirements for the first two milestones, M1 and M2 (see below). **M1 and M2 constitute the Minimum Viable Product (MVP from now on), and represent the goal for this project.**

Other goals include:

- ability to **create new plug-ins**, in order to extend the shell’s capabilities and offer high flexibility;

[1] 2-Clause BSD License, https://opensource.org/licenses/BSD-2-Clause

- ability to **create scripts** that exploit the shell's capabilities and potential, to offer a high degree of automation;
- ability to **run on multiple machines**, thanks to a horizontally scalable architecture;
- ability to **expose an API**, to enable integration with external systems;
- ability to **reuse components** in different software projects;
- ability to **plug different back-ends** for either data access (e.g. NFS, SMB, AWS S3) or data processing (e.g. GPGPU, Hadoop);
- ability to **create reports.**

This second set of goals constitute the second milestone, and is optional.

Rationale

The digital forensics world benefits from a large number of specialized softwares. However, the most prominent are commercial and/or closed-source. The available free and open software, instead, generally comprises collections of small tools, with a low integration degree if compared to the commercial counterparts.

The forensic shell, along with its modular framework and its ability to interface and integrate with heterogeneous systems, aims to fill the gaps between commercial and free/open forensic software, while maintaining a free and open source license and a distributed development model.

Comparison with other systems

There are several other projects with similar, yet distinct, goals. This chapter makes a brief comparison between FRESH and the known projects. Projects that are not open source are intentionally excluded from this comparison.

OCFA - Open Computer Forensics Architecture

OCFA stands for Open Computer Forensics Architecture[2]. It is a project developed by the Dutch National Police Agency, and is the one with the closest goals to FRESH. There are three main differences between OCFA and FRESH:

1. OCFA is written in C++: while this approach can promise higher performances, it also makes the evolution of a project more complex because of the higher language bar. FRESH aims at making changes simple, designs dynamic and easily modifiable, and to attract more collaboration thanks to a language with a lower entry bar. In order to address the performance concerns of a language such as Python, the project will be profiled regularly, and the performance-critical components will be rewritten in C or C++;
2. OCFA's development has been discontinued in 2012, and as such cannot guarantee anymore to be current on the modern investigations;

[2] OCFA, Open Computer Forensics Architecture, http://forensicswiki.org/wiki/Open_Computer_Forensics_Architecture

3. FRESH aims at offering a more powerful user interface than OCFA, with the belief that the adoption and therefore the usefulness and the future of this project depends equally on the quality of the backend and the frontend. In particular, FRESH will not offer a specific user interface except the shell, but it will make easy to develop user interfaces thanks to a set of APIs.

TSK - The Sleuth Kit

TSK[3] is a mature and appreciated toolkit for forensic analysis. However, compared to FRESH, its main focus is to offer a toolkit that runs on a single machine, with a GUI (Autopsy) focused on the investigation report.
FRESH, in comparison, aims at going beyond these limits, by offering a more generic framework which is also able to support specialized use cases. Examples are: known artifact inspection, unknown artifacts reversing, horizontally-scalable multi-node processing, integration with memory forensics frameworks, and much more.

DFF - Digital Forensics Framework

DFF[4] is included in this comparison because its non-professional version is open source. This project however, limits the features of the free and open source version to a subset that is not suitable to real investigations[5], and therefore it is not considered an alternative to FRESH, for lack of a real free availability and for the presence of vendor lock-in risks.

Note on memory forensics frameworks

Frameworks such as Volatility or Rekall are focused on memory forensics, and therefore are not directly comparable with FRESH. One of the goals of FRESH, however, is to be able to interface itself to this type of frameworks.

Note on AFF4 - Advanced Forensic Framework 4

While AFF4's[6] name recits "framework", this is not really a framework, but a forensic file format. Therefore it is not possible to make a comparison between AFF4 and FRESH. However, one of the goals of FRESH is to be compatible with AFF4.

Design

As every modern complex software, the design is provisional, and intended to meet the goals of the MVP first. It is expected that the initial design will evolve over time, in order to adapt to the renewed goals and priorities.

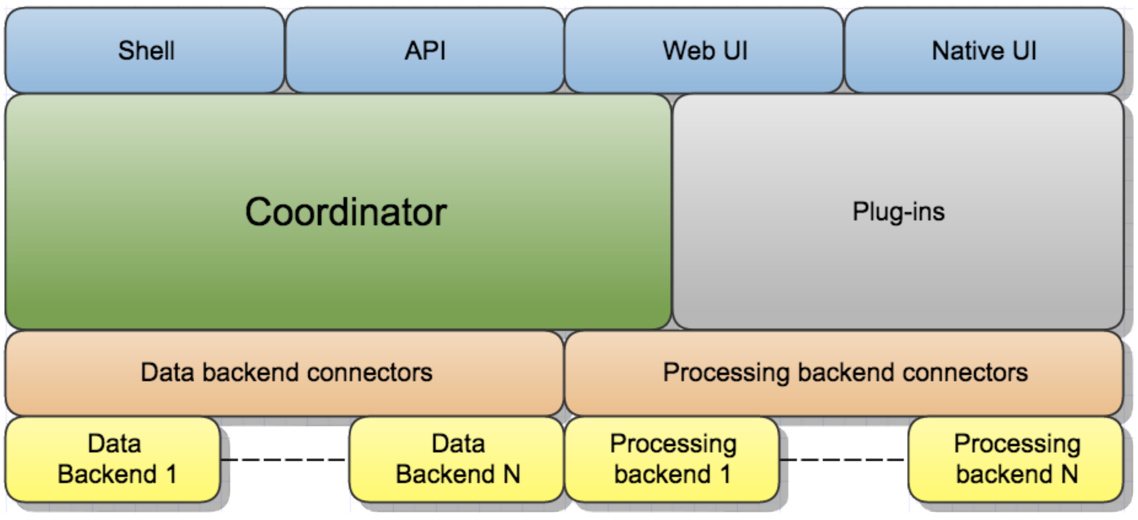
[3] TSK, The Sleuth Kit, http://www.forensicswiki.org/wiki/The_Sleuth_Kit
[4] DFF, Digital Forensics Framework, http://www.forensicswiki.org/wiki/Digital_Forensics_Framework
[5] DFF features, free vs. pro versions, http://www.arxsys.fr/features/
[6] AFF4, Advanced Forensic Framework 4, http://forensicswiki.org/wiki/AFF4

Language

The software will be developed with Python as its main language. The choice of Python depends on the following reasons:

- **facilitate the adoption and the collaboration**: Python is a widespread, appreciated and well-known programming language. Moreover it has a relatively low entry bar;
- **simplify the development**: Python excels at fast prototyping, yet it allows the development of complex architectures;
- **reuse existing components and frameworks**: Python has a vast collection of standard and third-party modules and framework, that extend and enhance its capabilities.

Architecture diagram



Components

The architecture of the forensic framework is inspired to the design of a modern OS.
On top of the system there is the front-end layer that includes the shell and the API, plus additionally any other UI (web or native).
At the core of the architecture the is the core layer, that includes the coordinator and the plug-in subsystems.
At the bottom of the architecture there is the backend layer, composed by the the data and processing sublayers, and the backend connector sublayer. The latter acts as a glue between the backend drivers and the core.

Coordinator

At the core of the architecture there is the coordinator (or kernel). Like an OS kernel, it is meant to:

1. be a service provider to the frontend layer (shell and API, but any UI could be developed)
2. provide coordination capabilities among plugins
3. provide job creation, deletion, lifecycle management and monitoring
4. provide an interface to the lower layers, the data and processing backends, through the backend connectors

The coordinator is inspired to the core subsystems of an OS kernel, namely the scheduler, the memory management and the task management.

Shell

The shell is the primary interface between the user and the framework. Its role is to instruct the coordinator to run the specified actions, to handle the errors and to show appropriately the return values to the user.

The shell component is entirely inspired to the shell of an operating system, from which takes its name.

Implementation details: the implementation of the shell in the MVP will be an extension of the IPython shell, and will share IPython's syntax, which is a superset of Python's shell syntax.

Plug-ins

The plugins are at the core of the architecture, and represent the larger population in terms of number of components and lines of code. The plugins will be a collection of independent components that will be able to process, analyse, represent, encode and manipulate arbitrary data.
For example, a plugin may be able to parse the content of an EML file; or may be able to scan an ISO image for further analysis; or may be able to export the results in a format suitable for a report.

The plug-ins subsystem is inspired to the kernel drivers subsystem.

Backend connectors and backends

The backend layer is aimed at offering pluggable data and processing backends. The components of the backend layers are the backend connectors and the backend drivers.

The purpose of the data backends is offer a mean to read the images to analyse and to store the output files (e.g. extracted files, reports, exported data).

The processing backends are useful instead for plugging different processing schedulers. An example could be a processing backend that is able to divide the workload across multiple networked hosts.

The backend connectors subsystem is inspired to the VFS of UNIX-like kernels.
The backend drivers subsystem is inspired to the filesystem subsystem of UNIX-like kernels.

API and User Interfaces

The shell is not the only front-end for the forensic framework. While not included in the MVP, it is possible to implement multiple front-ends, such as web interfaces, native interfaces and REST APIs, for a high degree of usability and integration.

Milestones

M1 - April 8th

- shell: open image file, start analysis, print uninterpreted results per plug-in
- framework: initial plugin interface; ability to start a plugin, and return the results
- plug-ins: MBR, DOS partitions, zip, keyword search
- backend connector: initial interface definition and basic implementation
- backends: local file reader

M2 - May 6th

- shell: print interpreted results through plugins’ reporting interface; print insights of the supported artifacts
- framework: stable plugin interface; ability to monitor, terminate, pause and resume a plugin
- plug-ins: email (plain, no mime, no EML), tar, gzip, jpeg, bmp, pdf, binary pattern search
- backend connector: stable implementation, no major interface changes
- backends: HTTP/HTTPS file reader

M3 - June 3rd

- shell: beautified text output
- framework: ability to monitor resources usage globally and per-plugin; implementation of a task queue to handle parallel tasks or to enqueue recursive tasks; ability to collect task results
- plug-ins: FAT32 undeletion, basic EML
- backends: ability to export selected extracted files, ability to save JSON reports; authenticated HTTP and HTTPS

M4 - July 8th

This is a review milestone, where the features will be discussed, assessed and stabilized if necessary. There may be further addition or changes to the project depending on the outcome of the previous milestones.

Wishlist

- AFF4 file format backend, and more: http://www.forensicswiki.org/wiki/Category:Forensics_File_Formats
- all the main compression, image, and document file formats
- encryption detection (containers and entropy analysis)
- support for encryption keys for decryption
- filesystem plug-ins for the most important file systems
- undeletion plug-ins for the most important file systems
- raw disk analysis
- slack space analysis
- Ability to read (and write where possible) forensic file formats, such as AFF4 and EWF
- rekall/volatility plug-ins
- network backends to analyse the live or non-live (pcap, pcap-ng) network traffic
- acceptable performances with large amounts of data

Implementation notes

The framework will be developed mainly using Python 3 on a modern Linux system. An eye to portability will be kept in order to run on other UNIX-like operating systems, but this is not guaranteed for the MVP.

In particular:

- The connector will be written mainly in Python 3, and may contain C or C++ parts for the most performance-sensitive parts. Any C/C++ components will be interfaced to the Python core via Python's C-API.
- The plug-ins and the backend drivers will be written mainly in Python 3, and may contain non-Python parts in order to interface them to existing third-party systems or tools.
- The shell, the API and the backend connectors will be written in Python 3.