Mac OS X persistent evidences for forensics purposes[1]

Authors
Joaquin Moreno Garijo, MSc (Royal Holloway, 2014)
Lorenzo Cavallaro, ISG, Royal Holloway

Overview

Computer forensics is a discipline focused on obtaining evidences that provide a clue about how, when and who did an action over an asset. This is important in the event of attacks from intruders and malware. However, no open source forensics tools have been developed to extract the Mac OS X file persistent evidences, despite recent trend of malwares and attacks on the operating system and applications.

This article is about identifying the Mac OS X persistence evidences, and providing a technical explanation about how these evidences can be extracted. The implementation of the tool that was able to extract this information can be found on the Plaso project website (http://plaso.kiddaland.net/).

Introduction

Mac OS X represents 7.54% of the world desktop operation systems market share. Despite this market share, research in the field of computer forensics and malware was not developed. It was believed that Mac OS X was not the target from potential intruders or malware. However, in 2012 the Flashfake botnet infected around 600,000 Mac OS X computers. This trend continued in the following years with other Mac OS X malware focused on exploit Java and Office vulnerabilities.

Computer forensics: getting evidences about how, when and who did an action over an asset.

Due to this necessity to develop a Mac OS X forensics tool, many companies as well as governments and other researchers, have been researching and implementing solutions over OS X. For example, in the early 2013 the Defense Cyber Warfare Technology Center from South Korea published a Keychains extraction paper. However, much has not been made public. In the open source community, some tools have been developed in the context of memory, such as Volatility or Volafox and others related to live system evidence. However, no open source forensics tool has been developed to extract the Mac OS X file persistent evidences due to the

[1] This article is to be published online by Computer Weekly as part of the 2015 Royal Holloway info security thesis series. The full MSc thesis is published on the ISG's website.

lack of documentation, the binary format that most of the evidences are stored and the fact that most of these evidences are unique for Mac OS X.

> **Persistent evidence:** evidences that exist despite system shutdown.

Volatile evidences are the group of evidences that exists temporally in the asset and are lost when the system is halted. Persistent evidences, however, are evidences that still exist in a host despite system shutdown, in other words, the evidences that can be found in the file system. The majority of the persistent evidences can be ordered by time line, where auditors can reconstruct the actions during a specific time period. The article is concerned with identifying the Mac OS X persistence evidences and providing a technical explanation about how these evidences can be extracted. A tool was implemented that was able to extract this information. More details of the tool can be found in the full thesis and on the Plaso project website.

Mac OS X, the hybrid platform

In March 1999 Apple announced the new operation system Mac OS X Server 1.0, in March 24 of 2001 the first Mac OS X 10.0 was released. It is designed using three types of technology: one originate from Apple, one originated from NeXT and the third part was taken from open source technology.

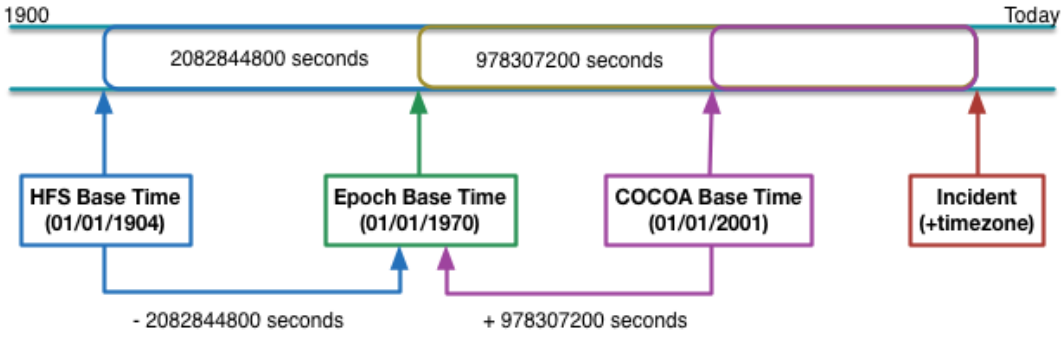
The Mac OS X kernel is called XNU which is based on I/O Kit, Mach and BSD, each with a different pedigree and written in a different language. The operation system is called Darwin which again is the combination of the XNU kernel, basic libraries and a group of system utilities. Darwin is entirely under Apple Public Source License, and the source code is public available. The upper layer over Mac OS X are the core services which have a restricted Apple license and the source code is not public available. These layers provide services and graphic interface and are backward compatible with the three different interfaces: Carbon (Classic or version 8), Cocoa (Rhapsody or version 9) and Aqua (Mac OS X, version 10).

Due to this hybrid nature of Mac OS X and its backwards compatibility, its timestamps and persistent evidences are stored in a myriad of different formats, making the extraction of evidence a complex job.

Timestamp

One of the most important values in forensic evidences is the time when the action that creates the evidences was done. This information, called timestamp, can be obtained from the file system time or from a specific field contained by the evidence. In Mac OS X, timestamps are stored using four different representations depending on the source of the data and the layer where the data comes from:

- HFS: This is saved as a 4-byte hexadecimal value representing the number of seconds since 1st January, 1904.
- Epoch: This represents the number of seconds since 1st January, 1970.
- Cocoa: This is used by some user applications and property list attributes and represents the number of seconds since 1st January, 2001.
- Plaintext: some evidences store their timestamp using a clear plaintext where the information is human readable. The most common way to store their timestamps is using the BSD plaintext format: Month Day HH:MM:SS.



Types of evidences

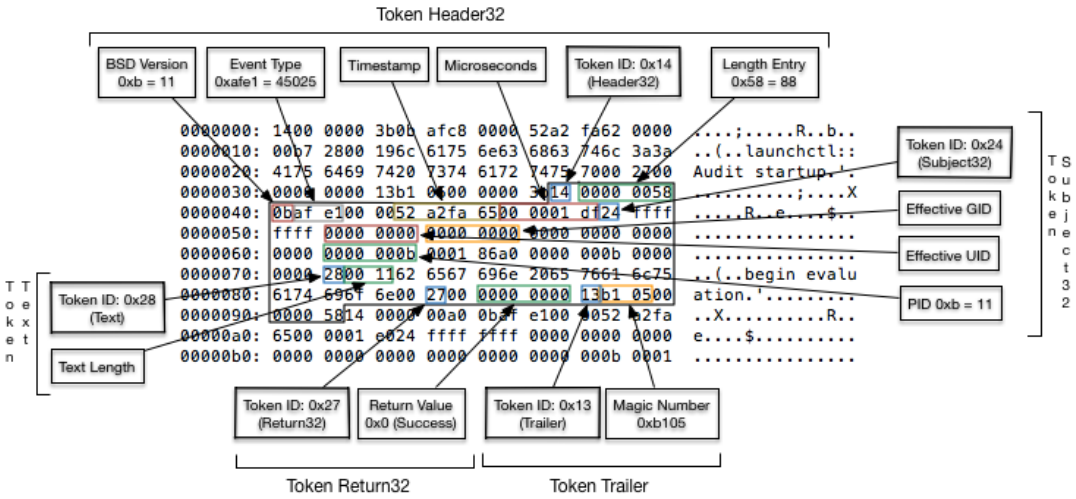
Mac OS X evidences are stored in plaintext, XML, SQLite or different binary formats. Each binary evidence has it owns structure or structures. Moreover, the information in the binary evidences can be stored using little endian or big endian representation. The most relevant forensics evidences are as follow:

Most relevant evidences:
- Basic Security Module (BSM)
- Apple System Log (ASL)
- UTMPX
- Property lists
- Keychain
- Printer daemon CUPS
- Document versions

Basic Security Module (BSM):
BSM is a kernel-based auditing mechanism. Information is stored using different files in the directory "/private/var/audit/". Each file can have one or more entries. An entry is a binary structure that stores a kernel event and is composed by group of tokens.

A token is a binary C structure that represents a specific piece of data such as arguments of the program or returned value. Each token has an initial 8-bit unsigned integer field that identify the type of the token follow by the structure of the token. The token structure depends on the ID. A token contains information such as timestamp (in Epoch format) and event identification (for example, the BSM ID 45015 is "creation of a new group"). Tokens are stored using a big endian representation.

BSM Mac OS X Entry:

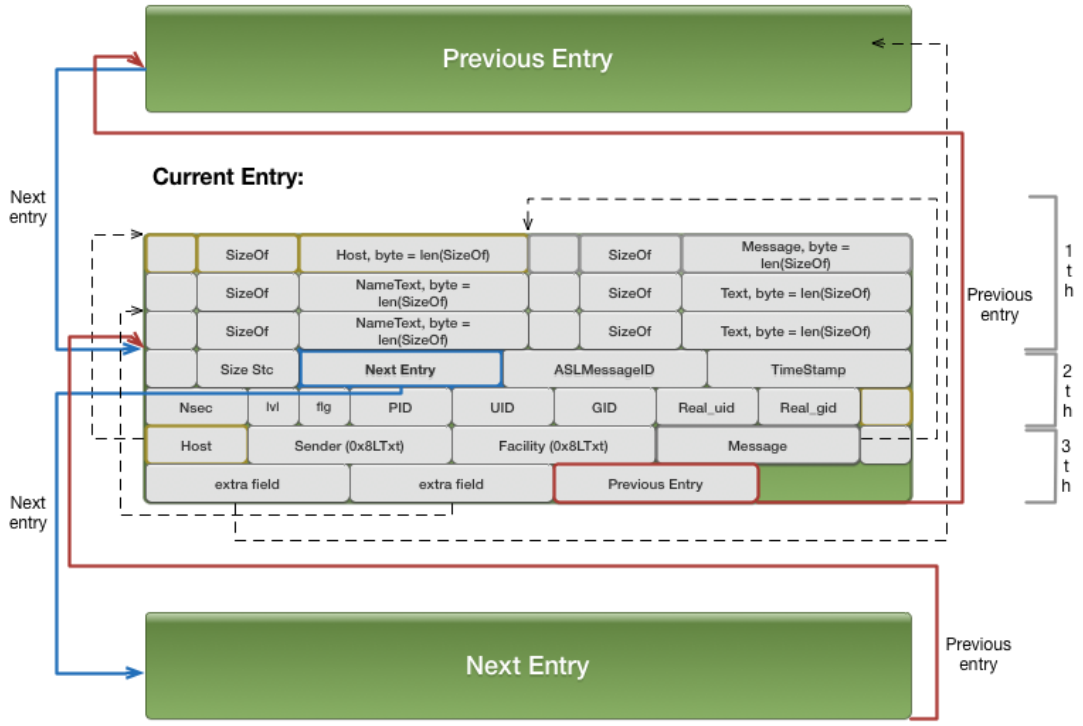


Apple System Log (ASL):

Apple System Log (ASL) is a daemon that manages and stores log information provided by the applications. The information is stored using different criteria such as the application name, the process identification and the user. This information can also be correlated with the priority level of the information. These files are mainly stored in the directory "/private/var/log/asl".

ASL binary file stores the information in big endian and the timestamp is stored using Epoch format. ASL binary format is stored as a doubly linked list of entries. Each ASL binary file has one header and one or more entries. The header has two important fields indicating the start entry and the last entry. The entries are stored consecutively in the file. Each entry has also two pointers: a pointer to the previous entry and another pointer to the next entry.

Each entry is divided in three sectors. The first sector is the entry's heap where the dynamic information is stored. The second part is the structure of the entry where the static values such as timestamp and ASL message identification are stored. The end part of the entry contains the host name, sender, facility, the information of the message, and depending of the message, and extra fields.



UTMPX:
Traditionally, Unix systems stored the register of the user logins, active user and last logins using the binary files UTMP, WTMP and Lastlogin. Since Mac OS X 10.5 these logs have been delegated to the Apple System. Nevertheless, and for backwards compatibilities, the UTMP still exists in a new version called UTMPX. The file is stored in "/private/var/run/utmpx".

The UTMPX file is a binary file that contains a group of entries. Each entry has different fields that represent active and inactive sessions since the system was booted, providing information such as when these accounts were logged in, the username and the type of terminal was used. In contrast to ASL or BSM, UTMPX stores the information in little endian instead of big endian.

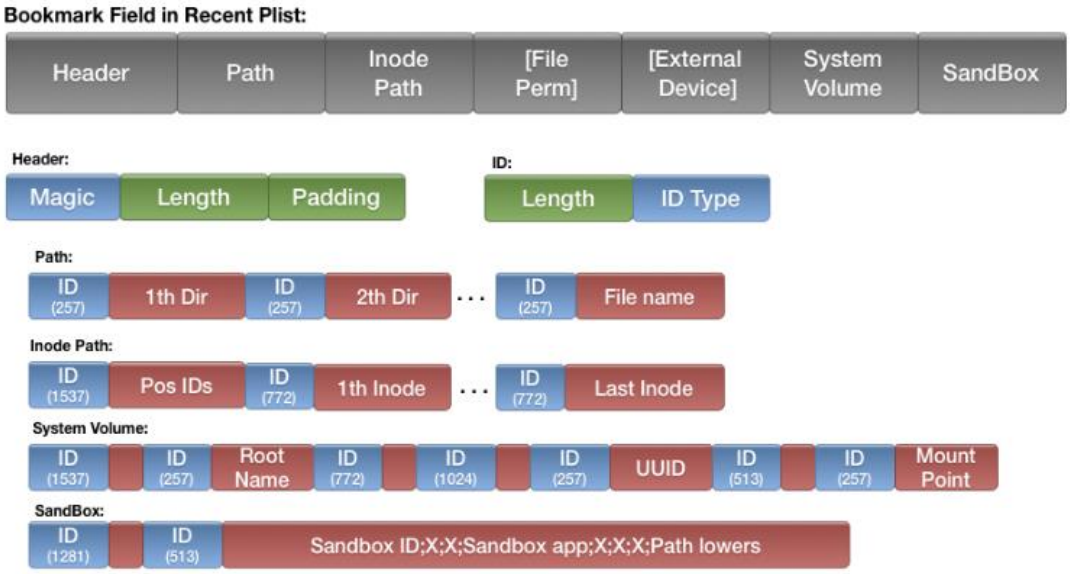
Property list:
Property List (Plist) files are used by Mac OS X and external applications to store configurations and program data. They contain useful information such as user accounts, recent files and applications, Apple Store accounts and firewall configuration. They can be stored in two different formats: XML and Plist binary format called BPlist.

The property list files contain items which are maps between item names and item values. The timestamp is usually stored as an integer in Cocoa timestamp representation.

The item value contains a Boolean, integer or string data, but some item values can contain a structure data. This structure can be another properly list or a unique binary structure. These unique binary structures are a challenge for forensics investigators because no documentation on how the information is

stored is available. This structure can be stored in little endian or big endian. Furthermore, some of these binary evidences contain timestamp in HFS, Cocoa or Epoch timestamp.

An example, in Mac OS X 10.5 Apple provided a new Launch Services Framework that contains the list of the recent opened files and applications using the LSSharedFileList library. The recent items are stored using a binary Plist file that contains a list of item. For each item the filename of the opened file and a Bookmark attribute is provided. This Bookmark attribute is a binary structure that contains the full path of the file, the file inode, UUID and the mount point of the root partition and some sandbox attributes. Also, if the file was opened from an external device, it contains the path to this external device.



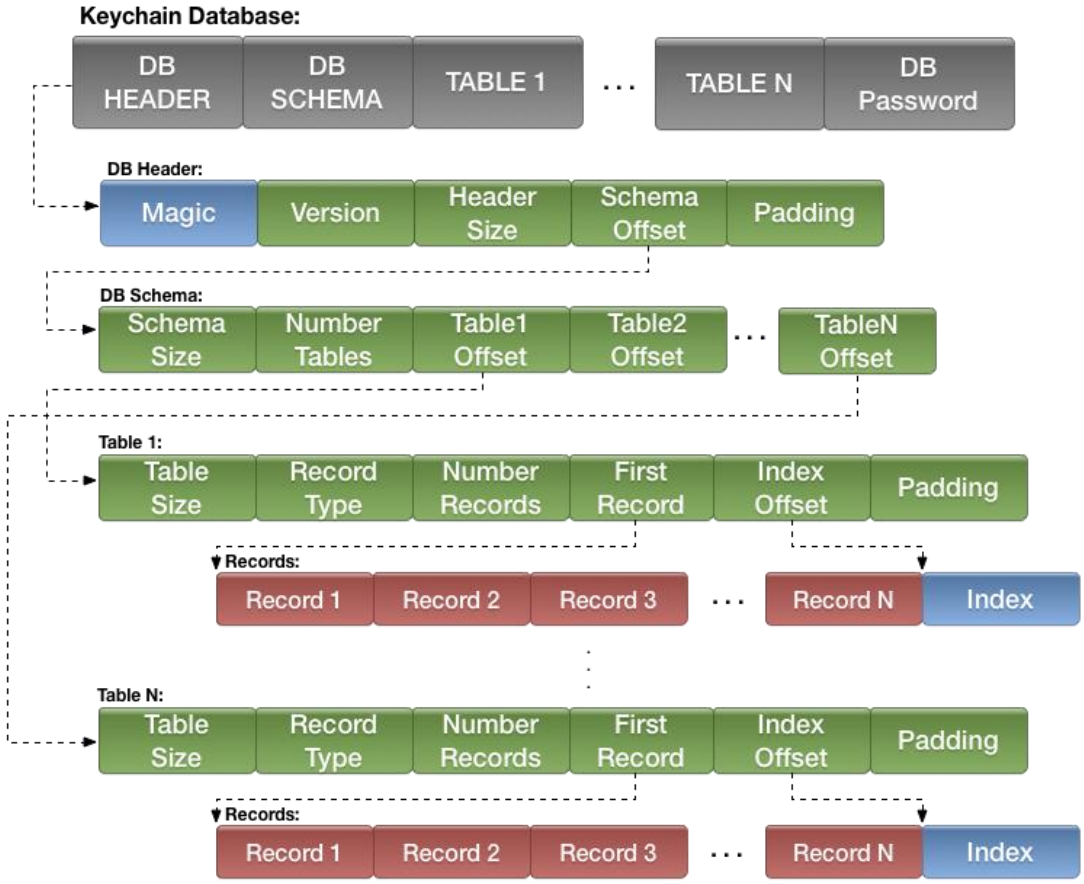
Keychain:

Keychain is a password protection mechanism for storing credentials. It stores saved credentials for applications, network connections, email accounts, web site, public/private key, certificates, etc., except for the system accounts. One keychain file is created for Mac OS X in "/Library/Keychains/System.keychain" and another for each user in "$home/Library/Keychains/login.keychain".

Keychain uses a 3-DES block cipher algorithm to encrypt the password and the secure notes but not all the other information. As a result, a forensics investigator who obtains the keychain file from the file system can learn much without the password – such as the URLs where the user stores the password, the username and when the password was last changed.

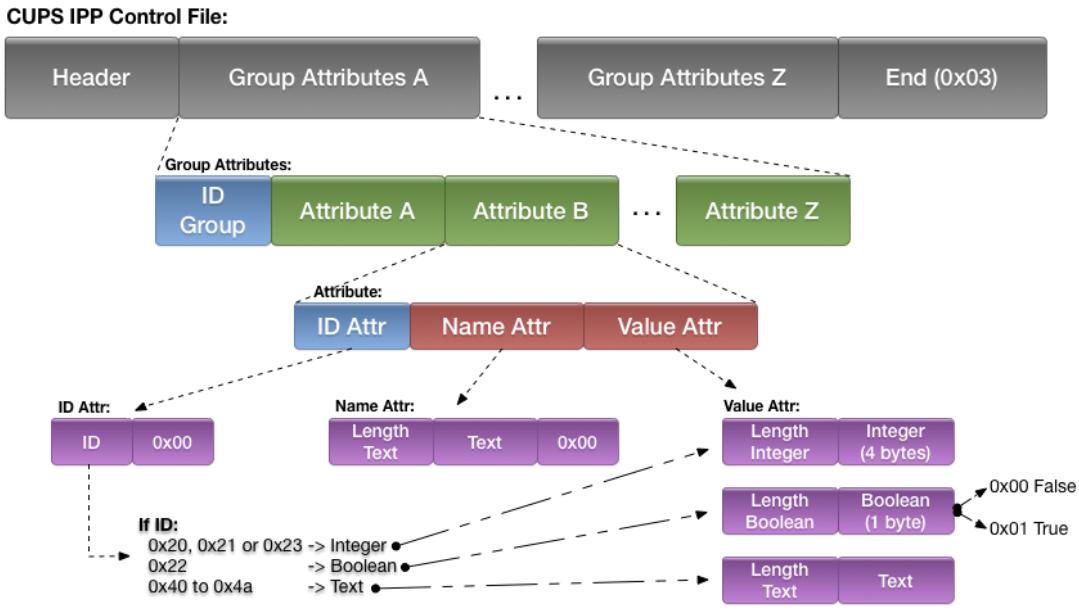
The keychain is a binary database stored in big endian format where the timestamps are stored in human readable form. The keychain uses the same schema design to store each database. Each database stores different types of information using a specific record schema. Each record in the same database has the same structure. The two most significant records schemes are the Record

Internet Password and Record Generic Password. Both schemes store the credentials for email accounts, wireless networks, URLs, applications and secure notes.



Printer daemon CUPS:
Mac OS X uses CUPS as a printer daemon. One file for each jobs is stored in the directory "/private/var/spool/cups/". These files are called control files.

The control files provides information such as when the document was printed, the type of document, the name of the printer and the owner of the document. However, it does not provided the name, path and content of the original file. The control files are stored using a binary structure in big endian format and the timestamp values are stored using an integer value with Epoch representation.



Document version:
In Mac OS X 10.7, Apple released a functionality that permits comparison of different versions over the same file. Every time a user saves a file, Mac OS X creates a copy of this file. This feature is called "Document Versions" or "Versions".

Document Versions uses an SQLite database stored in the directory "/.DocumentRevisionsV100/db-V1". It records the relationship between the original file and the back up versions of the file. It provides information such as the timestamp when the copy version was done, the last time this file was opened and the original file.

Implementation using the forensics framework Plaso

Plaso is an open source forensics framework implemented in Python that provides a unique tool to parse computer forensics artefacts, log files and file system timestamp, generating a unique output where the timestamp from all the evidences are correlated in the same time base, making the daily job easily for computer forensics investigators. The core developers are Kristinn Gudjonsson and Joachim Metz.

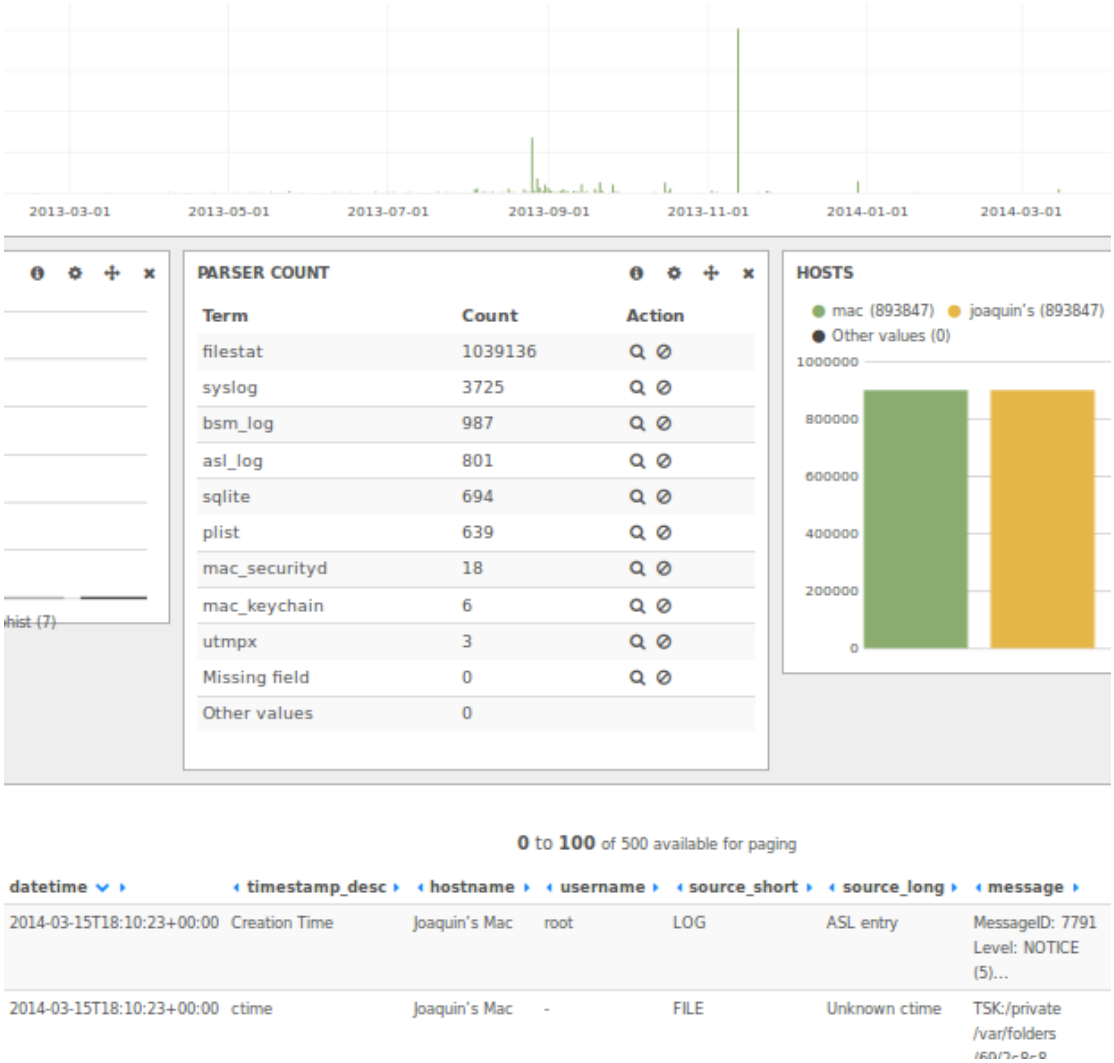
As it was remarked at the beginning of the document, no open source solution was able to extract Mac OS X evidences. However, during a forensics case the investigators might be required to explain how this information is stored and how it was extracted. For this reason, I implemented a group of parsers over Plaso which are able to extract all the identified Mac OS X evidences described in this article. The implementation over Plaso provides a solution that can be used by the forensics and incident response community.

Plaso can work against a file system image or a partition image to provide all the evidences extracted from this image ordered in the same timestamp base. Furthermore, Plaso can work against a mounted partition or single evidence in a manner that can be used for a specific purpose or as an auxiliary tool during an incident. This can be done with only a few commands, making the extraction process easier and freeing up time for the investigator to work on the case itself.

As an example, the following command extracts all the file system, Mac OS X and application timestamp evidences storing the results in “output_file” in Pstorage format that contains all the extracted evidences:

```
$ log2timeline.py  output_file partition_image
```

The “output_file” can be exported in other different formats such as MySQL, SQLite, CSV or Elastic search. The following screenshot shows a Kibana interface using as an input a Plaso output that was executed over a Mac OS X file system. It shows the extracted evidences such as HFS file system, BSM, ASL and Keychains.



Furthermore, Plaso provides an auxiliary tool called Plasm that works over the output file. This tool uses a provided set of rules to identify possible threats in

the analysed evidences. As an additional, a set of Mac OS X rules were provided to identify the list of mounted USB devices, processes executed during boot time, artefacts that identify well known malware, executed applications using file system combination with BSM, etc.

Conclusion

The number of new security threats in Mac OS X has been increasing during the last few years, especially those related with malware and intruders. However, the development of tools and the amount of research conducted to deal with these incidents have been minimal. This article focuses on the persistent evidences generated or managed by Mac OS X.

This document identified where the persistent Mac OS X evidences are stored, the process or mechanism used to store the evidences, the kind of information that is stored and the manner in which this information can be extracted. It was observed that a large number of these evidences were either not well documented or not documented at all. Furthermore, due to the implementation design decisions and backwards compatibility of Mac OS X, the evidences are stored following many different criteria. The majority of these evidences are binary evidences unique to Mac OS X environment wherein a reversing process was required to be able to extract the information. The Plaso plugins that the author designed allows this to be achieved.

There are different approaches that could be adopted in the coming years. The main ones among these are the following three:

1. Persistent evidences improvement: For each new version of Mac OS X the evidences must be checked to be sure that the format has not changed. Also, each new version of Mac OS X provides new functionalities that might store new evidences that must be analysed. Additionally, Mac OS X might have more evidences that were not identified in this document that should be study.
2. Persistent and volatile evidences integration: unify both types of evidences and correlate the information.
3. Correlation and behaviour analysis: a behaviour analysis should be applied over the evidences obtaining the super timeline with all the evidences extracted, an automated tool must be able to analyse this output and recognise specific patterns.

Biographies

Joaquin Moreno Garijo is currently working for Context Information Security performing penetration tests and has been working as a security engineer since 2009. During this time, he has had significant experience working within a wide variety of business sectors. His qualifications include a BSc in Computer Science, a second BSc in Telecommunications Engineer at UV and a MSc in Information Security at RHUL. Additionally, he holds seven SANS GIAC certifications focusing

on incident response, forensics and intrusion analysis, network and web penetration testing, malware reverse engineering and systems hardening.

Lorenzo Cavallaro is a Senior Lecturer of Information Security in the Information Security Group (ISG) at Royal Holloway University of London. His research focuses largely on systems security. He has founded and is leading the recently-established Systems Security Research Lab (S2Lab) within the ISG, which focuses on devising novel techniques to protect systems from a broad range of threats, with the ultimate aim of building practical tools and providing security services to the community at large.