OS X as a Forensic Platform

GIAC (GCIA) Gold Certification

Author: David M. Martin, dmartin@mastersprogram.sans.edu
Advisor: Stephen Northcutt
Accepted:

Abstract

The Apple Macintosh and its OS X operating system have seen increasing adoption by technical professionals, including digital forensic analysts. Forensic software support for OS X remains less mature than that of Windows or Linux. While many Linux forensic tools will work on OS X, instructions for how to configure the tool in OS X are often missing or confusing. OS X also lacks an integrated package management system for command line tools. Python, which serves as the basis for many open-source forensic tools, can be difficult to maintain and easy to misconfigure on OS X. Due to these challenges, many OS X users choose to run their forensic tools from Windows or Linux virtual machines. While this can be an effective and expedient solution, those users miss out on the much of the power of the Macintosh platform. This research will examine the process of configuring a native OS X forensic environment that includes many open-source forensic tools, including Bulk Extractor, Plaso, Rekall, Sleuthkit, Volatility, and Yara. This process includes choosing the correct hardware and software, configuring it properly, and overcoming some of the unique challenges of the OS X environment. A series of performance tests will help determine the optimal hardware and software configuration and examine the performance impact of virtualization options.

1. Introduction

The Apple Macintosh computer and its OS X operating system have become increasingly popular among technical professionals since its introduction in 2001 (StackOverflow, 2016). It has steadily been gaining acceptance among computer forensic analysts, who are now frequently seen utilizing the popular MacBook Pro laptops. Apple hardware has always been one of its greatest selling points, boasting sleek, well-engineered form factors and powerful, high quality components. Apple has also been an early adopter of cutting edge technologies such as Firewire, Thunderbolt and USB-C. While Macintosh hardware has been popular in the digital forensic community for some time, OS X is a relative latecomer to the world of forensic software.

Many popular forensic suites such as Encase, FTK, and XWays only support Windows, but others like Autopsy and BlackLight Forensics now offer OS X-native versions. Most popular Linux forensic utilities like Bulk Extractor, Plaso, Rekall, Sleuthkit, Volatility, and Yara will either run natively or can be compiled for OS X. While some of these tools provide binaries or specific installation instructions for OS X, many require the user to translate the dependencies and installation process. Rather than engaging in this time-consuming process, many analysts choose to dual-boot a Mac between OS X and Windows or even replace OS X entirely. Others use OS X as their primary operating system, but run their forensic tools in a Windows or Linux virtual machine. While this can be an effective and expedient solution, those users miss out on much of the power of the Macintosh platform.

The aim of this research is to provide a guide for those who wish to create a native OS X environment for computer intrusion forensics. This entails choosing the correct hardware and software, configuring it properly, and finding solutions to some of the unique challenges of adapting tools to the OS X environment. The forensic analysis process generally consists of three stages: evidence acquisition; investigation and analysis; and reporting results. Most forensic tools focus on the first two stages, as the process of reporting results mostly depends on the analyst's organization and writing skills and is not tool-dependent. An effective forensics platform should provide tools that will allow an investigator to complete each of these stages as quickly and efficiently as possible. Such a platform should be able to analyze all the kinds of evidence an investigator is likely to encounter, including log files, disk, and memory images.

This guide will detail the process of configuring OS X 10.12 “Sierra” as a forensic analysis platform. Beginning with the system setup and installation of software dependencies, it will demonstrate the necessary steps to install a baseline of open-source forensic tools. Along the way, it will highlight unique challenges presented by the OS X environment and methods to overcome them. The guide will also explore the selection of devices such as write blockers, drive docks and external drives that are used in forensic response and how best to use them with Apple hardware. Several options for and the performance implications of virtualization will be examined. It will also present the results of hardware and software performance testing and options to optimize the performance of the tested tools. By following the guide, a user will have built a fully-functional, native OS X forensic analysis system.

2. System Setup

The first step in creating an OS X forensic environment is to install the software necessary to conduct a forensic analysis. While many people will already have some of these tools installed on their system, this guide will demonstrate the process using a freshly installed copy of OS X. The same procedure should work equally well on a system that has been in use for some time, assuming there are no configuration problems that would prevent it. If a system has been installed for quite some time, particularly if several failed installations or package conflicts have occurred, it may be best to install a fresh copy of the OS. The system used for testing throughout the process was a 15-inch MacBook Pro (Mid 2015) with a 2.8 GHz Core I7 processer, a 500GB PCIe M2 SSD and 16GB of RAM.

2.1. Developer Tools and Dependencies

Many familiar UNIX utilities such as *dd*, *grep*, *awk*, *cut* and *sed* come pre-installed on OS X and require no further attention. Others, like the *make*, *GCC*, *Perl*, *git*, *strings*, and *libtool* utilities that are required to compile and build UNIX software are not.

The Apple XCode development environment and the XCode command line tools provide these capabilities and should be one of the first packages installed by any coder or forensic analyst. XCode can be downloaded from the App Store. Once XCode is installed, the command line tools can be installed with the command **xcode-select** –*install*. Another piece of software required by several other tools is the Java Development Kit (JDK). Users should download and install the latest OS X version from the Oracle website. Once these development tools are installed, the next key piece of software to install is a package management system.

2.2. Package Management

Perhaps the most frequently lamented feature missing from OS X is a package management system like Debian's Apt or RedHat's Yum. While software installed from the Apple App Store will work with little or no configuration and receive regular updates, manually installed packages and programs compiled from source enjoy no such benefits. None of the common open-source forensics tools are available through the App Store, so users must find other mechanisms to install and maintain them. One option is to download a precompiled OS X binary of a tool, if available, and if not, compile it from source. At times this is the only option, but this approach has a number of drawbacks. Plaso, for example, has a dizzying array of dependencies that must be installed before even the pre-packaged version will work. Updating and maintaining standalone tools can also be challenging, as it often requires manual effort and sometimes requires that dependencies be updated manually.

To simplify this process, several groups have developed third-party package management software for OS X, such as Fink, HomeBrew, and MacPorts. While Fink is a promising project that aims to port the Debian Apt and Dpkg utilities and their packages to OS X, it still lacks support for many of the required forensic packages (The Fink Project, 2012). Homebrew and MacPorts offer similar functionality and packages but take different approaches to providing them. Each has its own dedicated proponents and many forensic analysts will likely already have one of these programs installed on their Mac. Due to this fact, this guide will provide instructions for setting up available tools in either system. It is possible to install both HomeBrew and MacPorts on the same computer, but doing so can result in conflicts or unpredictable behavior and is not recommended.

2.2.1. MacPorts

MacPorts is descended from the Ports package manager for BSD, the Unix variant on which OS X is based. It builds its own Unix directory structure under */opt/local/*, rather than relying on utilities provided by the operating system (Duling, Maibaum, Barton, & Lang, 2014). While this results in some duplication, it does help avoid difficult-to-diagnose version conflicts and prevents its tools from interfering with their native counterparts. MacPorts' website states that it currently includes 21,866 packages in 84 categories, though that number includes multiple variants of some packages. As with most package managers, root or sudo privileges are required to install or remove packages. MacPorts is most easily installed by downloading and installing the correct installer package from the macports.org website. MacPorts can be run by issuing the **port** command, followed by the operation to be performed and the package(s) to which it applies. The official MacPorts repository offers different major versions of some packages, but not specific minor versions. As an example, users can choose between Python 2.7 and 3.4, but not to install Python 2.7.6 rather than 2.7.13. Installing a specific version of a package is somewhat complicated and requires checking out the correct version from the MacPorts SVN repository. Users can create their own MacPorts packages by creating a portfile, which is a simple TCL script that defines the properties of the package. Portfiles can be stored in a local repository or submitted for inclusion in the official repository. A complete set of MacPorts installation instructions for the packages covered by this guide is available in Appendix G.

2.2.2. HomeBrew

HomeBrew is a package management system designed by a development team of beer enthusiasts, as evidenced by its beer-related nomenclature and icons. Packages are referred "formulae" and are stored in "kegs" in the "cellar" (*/usr/local/Cellar*). Users may also 'tap' third-party repositories, called 'taps', which function much like Ubuntu PPA's. HomeBrew runs in user space, so root privileges are not required for package installation or removal. Formulae installed with HomeBrew that would usually require root, function without it. This feature can occasionally be problematic when invoking tools that require privilege elevation to perform tasks such as hardware access or capturing network traffic. Homebrew installs its packages in its own directory structure in its “cellar” and creates symbolic links in the */usr/local/bin* directory. Unlike MacPorts, it will rely on system provided dependencies, if available and those dependencies have not already been installed through HomeBrew (McQuaid & Labinskiy). For example, HomeBrew installed Python packages will use the built-in version of Python and will not install HomeBrew’s version of Python as a dependency.

HomeBrew is called with the **brew** command followed by the operation to perform and packages on which to perform the operation. Where multiple versions of a formula are available they will be designated by *<formula>@<version>,* i.e. *gcc@4.6*. The **brew** *info <formula>* will list useful information such as dependencies and optional build parameters and is advisable to use prior to installing a formula. If the info page indicates that there is a development version of a formula available, it can be selected with the *--devel* flag. Users can easily create their own HomeBrew formulae from source code tarballs using the **brew** *create* command without having to edit scripts or configuration files. At the time of writing, there were 3661 formulae available through HomeBrew (McQuaid & Labinskiy). It is a good idea to install the *libtool, autoconf* and *automake,* and *pkg-config* packages immediately after installing HomeBrew.

A unique feature of HomeBrew is its Cask extension which manages applications in the same way HomeBrew manages command line BSD utilities. Cask automates the installation process of downloading a *.dmg* image, extracting the application within and copying it to the apps folder. It also allows users to upgrade all Cask-installed apps with a single command. Users can also easily create casks for apps not already found in the cask repository. Casks are available for most commonly used OS X applications, including web browsers, text editors, virtualization platforms and more. Cask can be installed with the commands **brew** *tap caskroom/cask* and **brew** *install brew-cask*, after which the **brew cask** command will function similarly to **brew**. A complete set of HomeBrew installation instructions for the tools presented in this guide is included in Appendix H.

2.3. Python

The Python scripting language and its many 3rd party libraries provide the foundation for many open source forensic tools. Unfortunately, it can also one of the most temperamental pieces of software to install, upgrade and maintain in OS X. Python 3 has been available for several years, but several critical forensic libraries are not yet compatible, so forensic users must continue to use Python 2. It is possible to install both Python 2 and 3 under different aliases. OS X Sierra comes with Python 2.7.10 installed by default. While this version is over a year old and three versions behind the current 2.7.13 release, it is compatible with all required packages and using it is a viable option.

Users also have the option of installing an updated, external Python version, which is recommended by many Python experts. Many Python experts recommend installing a more current version, but users should be aware doing so will result in an additional version of Python installed on their system (Reitz & Schlusser, 2016). Users cannot and should not attempt to remove the built-in Python installation from OS X, as this is an integral part of the operating system. Users who choose to install an external version of Python can either download the official installer from Python or install Python through MacPorts or HomeBrew. Whichever option is chosen, configuring Python

Version	Location	SymLink	Package Location
Native	/System/Library/Frameworks/Python.framework	/usr/bin/python	/Library/Python2.7/site-packages
Official	/Library/Frameworks/Python.framework	/usr/local/bin/python	/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages
MacPorts	/opt/Library/Frameworks/Python.framework	/opt/local/bin/python	/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages
HomeBrew	/usr/local/Cellar/python/2.7/Frameworks/Python.framework/	/usr/local/bin/python	/usr/local/lib/python2.7/site-packages/

properly in OS X can be a challenging and easily botched process.

Table 1: Python Installation Locations

Python installations have three primary components: the main framework directory, a symbolic link to the main Python executable and the *site-packages* directory (Python Software Foundation, 2016). The locations of these three components depending on how Python is installed and can create additional complexity when installing modules when multiple versions are present. Another potential issue with multiple Python installations is that scripts starting with *#!/usr/bin/python* rather than *#!/usr/bin/env python* may use the wrong Python interpreter and lack required dependencies.

The System Integrity Protection (SIP) feature introduced in OS X 10.11 "El Capitan" complicated the process of installing 3rd party modules for the native OS X Python. SIP prevents the user, even running as root, from accessing the */System/* directory where Python is installed. This is usually not a problem, as packages are installed in the root-writable */Library/* directory, but a few packages come pre-installed in the protected */System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/ python/* directory and cannot be modified. Attempting to install packages that depend on these packages, or that attempt to remove or modify them can result in difficult-to-diagnose errors (mfripp, 2016). It can also cause some package installations to fail irretrievably if they attempt to write to the documentation directory, which is within the protected System directory. A full list of these packages is included in Appendix B.

Users who wish to install an external Python version can download the current version of Python from the python.org website as an installer package. This will install an additional Python instance and set it as the default Python interpreter by giving it precedence in the $PATH order. This will also install the Python *setuptools* package and the PIP package manager required to install other packages. Unlike older versions, the Python 2.7.13 installer no longer shares the */Library/Python-2.7/site-packages* directory with OS X Python, which had tended to create package conflicts (Python Software Foundation, 2016). The MacPorts version of Python is provided by the *python27* package. Following installation, the MacPorts versions of Python must be set as default using the command **port** *select –set python python27*. Homebrew provides both Python and PIP in the *python* package. It should be noted that HomeBrew installs the Python executable symlink to */usr/local/bin/*, so it will conflict with the installer version. In any case, users should install only one external version of Python.

2.3.1. PIP

PIP is the officially recognized package manager for Python and allows users to install and maintain the many libraries required by forensic tools (Python Packaging Authority, 2016). While some Python packages are available through HomeBrew or MacPorts, most are not and those that are tend to be older and less frequently updated than their PIP equivalents. For the sake of consistency, it is best to install Python packages from PIP rather than MacPorts or HomeBrew. The official Python installer and HomeBrew both install PIP with Python by default. To install PIP using OS X-native Python, use the legacy **sudo easy_install** *pip* command. In MacPorts, PIP can be installed from the package *py27-pip* and, like Python, should be selected with the **port** *select –set pip py27-pip* command.

Once installed, PIP can install, remove and update Python packages, including itself. Unlike most package managers, it does not resolve dependencies, except those that are explicitly specified by a package creator. PIP can install from the online Python Package Index (PyPI), version control systems like Git or SVN, and local distribution files. PIP's default is to download a package from PyPI unless a specific URL or file path is specified. PIP can also install a specific version of a package as designated by the syntax **pip** *install <package_name>==<version_number>* (Python Packaging Authority, 2016). Using PIP to install packages under the native OS X and MacPorts versions of Python requires root permissions, while the installer and HomeBrew versions can (and generally should) be run as a non-privileged user. Another feature that Pip lacks is an upgrade-all option, but the command **pip** *list --outdated | cut -d' ' -f1 | xargs pip install –upgrade* provides the same functionality.

2.3.2. Virtualenv

Virtualenv is a Python tool that can create and manage multiple, self-contained Python environments, each with a different set of packages installed. Virtualenv works by installing a new version of Python, Pip and Setuptools into a separate directory for that environment (Reitz & Schlusser, 2016). When a virtual environment is active, any

Python scripts will use that environment's version of Python and Pip will install packages into that directory structure. Virtualenv can be particularly useful for managing tools with complex dependencies, some of which may need to be installed from source. It is also the best option to avoid SIP problems when using PIP with the native OS X Python. As the Plaso documentation notes, using Pip in the native OS X Python without Virtualenv is highly inadvisable and will damage the site-packages directory (Metz, Gudjonsson, & White, Plaso Wiki, 2016). Virtualenv is also helpful for managing tools that rely on conflicting versions of the same dependency.

Like most Python packages, Virtualenv and its companion utilities in VirtualenvWrapper can be installed using PIP. If using the native OS X version of Python, the *--ignore-installed* option must be given or the installation will fail due to the system-protected *six* package. After configuration options are added to the *.bash_profile* file (see Appendix G or H), virtual environments can be created with the **mkvirtualenv** <*environment_name*> command. This will also activate the environment, which can thereafter be activated with **workon** <*environment_name*>, deactivated with the **deactivate** command and removed with **rmvirtualenv** <*environment_name*> (Reitz & Schlusser, 2016).

2.4. Virtualization

Virtualization is a useful and popular technique for digital forensics. It provides the ability to run multiple operating systems, isolate potentially dangerous software for analysis and the ability to revert changes when things go wrong. While

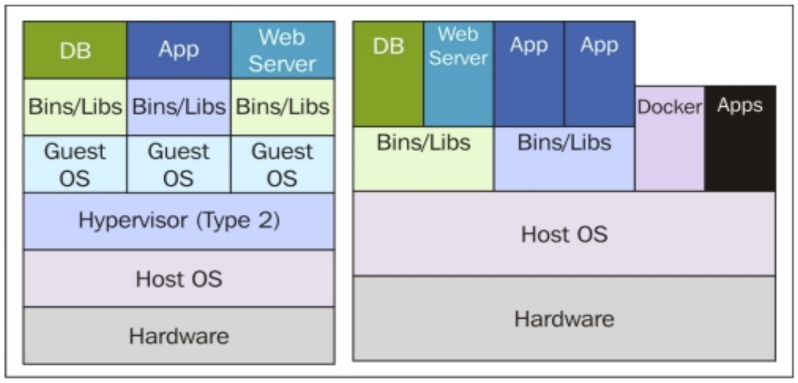


Figure 1: Virtualization vs. Containerization (Raj, Chelladhurai, & Singh, 2015)

virtualization provides many benefits, the tradeoff is performance and resource usage. A virtual machine can only be given a portion of the host operating system's resources and requires additional storage space for each virtual container. There are many types of

Author Name, email@address

virtualization available, but the two explored for use in an OS X forensic platform are containerization and desktop virtualization.

2.4.1. Containerization

Containerization or micro-virtualization is an application virtualization technique that creates an isolated environment for an application. Unlike traditional virtualization, which virtualizes hardware in which guest operating systems run, containerization virtualizes operating systems in which guest applications run. Provisioning lightweight containerized environments is quicker and those environments consume less system resources for virtualization overhead. The most popular containerization platform is Docker, which is available for multiple platforms, including Linux, Windows and OS X (Raj, Chelladhurai, & Singh, 2015). Docker can be downloaded and installed from its website or installed from HomeBrew Cask. When Docker is running, a whale icon will be visible in the OS X status bar. A Dockerized application and its environment are referred to as an image and a running instance of an image is referred to as a container.

Dockerfiles are simple configuration scripts that describe the environment to be built inside the Docker image. A dockerfile can import existing Docker images in much the same manner as Python imports libraries. Most import an existing operating system image, such as Ubuntu. Dockerfiles can also specify commands to be run, set environment variables and manipulate the filesystem inside the container. Each container contains an entrypoint that specifies the default program to be run when the image is activated. The entrypoint can be overridden with a command line parameter. For example, if the entrypoint into an Ubuntu image were bash, **docker** *run ubuntu* would start bash inside the container, while **docker** *run ubuntu top* would start Ubuntu's **top** utility instead. Containers can either be interactive, or run a single process within the container, then exit. Docker images contain a virtualized filesystem that can be mapped to interact with the host filesystem with the **docker** *run -v <host_path>: /<container_path>* option (Raj, Chelladhurai, & Singh, 2015). The host system directories Docker allows to be mapped to the container are controlled through the preferences menu in the Docker status bar app, but by default include */Users*, */Volumes*, and */tmp*.

The simplest way to use a Docker image is to find it on the Docker Store, which contains thousands of prebuilt Docker images, including many forensic tools. Once the correct image is located, it can be installed using the **docker** *pull <image_name>* command. Alternately, some tools will include a *dockerfile* in their distribution source code that can be used to create a Docker image. To do so, change to the directory containing the *dockerfile* and run the command **docker** *build -t <image_name> -f <dockerfile_name>.dockerfile*. Either method will add the image to the local Docker repository, from where it can be run with the command **docker** *run <image_name>*.

2.4.2. Desktop Virtualization

There are three main desktop virtualization platforms available for OS X: Parallels, VMWare Fusion, and Oracle VirtualBox. The free, open-source VirtualBox provides largely the same features as its commercial counterparts, though it tends to be somewhat less polished and user-friendly. A 2015 performance comparison showed that while VirtualBox lagged badly in graphics-intensive tasks, it performed similarly in most CPU-bound tasks, except multi-core integer operations. VirtualBox excelled at disk I/O, both within the Virtual Machine (VM) and between the host and guest operating systems, outperforming both commercial platforms. However, it performed poorly in transfers to and from attached USB3 devices (Tanous, 2015). There are a few excellent pre-built Linux-based forensic VM's available on the internet, including SANS' SIFT Workstation and Carnegie Mellon CERT's ADIA. These VM's provide a forensic environment with a wide selection of forensic tools configured and ready to run. The latest version of the SIFT Workstation available at the time of writing was used for all performance functionality comparisons (SANS Institute, 2014).

Attaching evidence items to Virtual Machines can be accomplished in several ways. All the virtualization platforms allow certain folders to be shared between the host and guest operating system. If working with evidence files, such as disk or memory images, this is the easiest and recommended option to get evidence into the VM. USB devices can also be passed directly into VM's though there is some performance overhead, particularly in VirtualBox. There is currently no support for directly attaching thunderbolt devices to VM's in any of the platforms, though Parallels can attach partitions from any physical device. Parallels provides the easiest to use interface for attaching non-USB physical devices like IDE, SATA and ThunderBolt hard drives, though the support is limited to individual partitions. Virtualbox allows access to raw devices using the **VBoxManage** *internalcommands createrawvmdk -filename "</path/to/file>.vmdk" -rawdisk </dev/disk#>* command. In this case, the vmdk file essentially serves as a symbolic link to the actual device and can be attached to the VirtualBox VM. VMWare Fusion provides similar functionality through the **/Applications/VMware\ Fusion.app/Contents/Library/vmware-rawdiskCreator** *create /dev/disk# fullDevice <full-path-to-.vmwarevm-file>/<filename>.vmdk ide* command. In order to attach the vmdk, however, the user must show the contents of the .vmwarevm package and directly edit the .vmx file inside to add the following lines:

- ide1:0.present = "TRUE"
- ide1:0.fileName = "<filename>.vdmk"
- ide1:0.redo = ""

If an *ide1:0* is already present in the file, the identifier must be changed to prevent a conflict.

2.4.3. Performance Comparison

The performance impact of virtualization varies somewhat in both scope and significance depending on the task being performed, the level of virtualization and the platform being used. A ten-second task that takes 10% longer in a VM will be nearly imperceptible, where the same level of overhead can be considerably more significant for a multi-hour process. To test the impact of virtualization on performance, several common forensic tools were run in each environment with the same parameters and timed using the GNU 'time' utility. A fully updated copy of the SIFT Workstation VM was used for the Parallels, VMWare and VirtualBox testing. The VM's and Docker were each given 4 cores and 8GB of RAM of the test system's 8 virtual (4 physical) cores and 16GB RAM. The first test was running the Volatility *imageinfo* plugin against an 8GB Windows Server 2008 memory image. The **htop** system monitoring tool indicated that this task was using only a single processor core. The second test ran the multi-threaded bulk extractor tool against the same memory image. The third ran a Yara scan, looking for the string "foo" in a large directory tree of source code (also multi-threaded). Finally, the Plaso *mft* (single-thread) and *winxp* (multithreaded) parsers, were run against the *xp-tdungan* hard drive image of SANS 408 fame.

As expected, the native OS X versions of the tools were nearly always the fastest, particularly in the case of Yara, which was more than three times faster than its nearest competitor. The one exceptions were that running Plaso with only the single *mft* parser was 7-8% faster in two of the VM's and Bulk Extractor was 1% faster in VirtualBox. Performance was mostly similar between the virtualization platforms, though there were some notable outliers, such as Volatility in VMWare and Yara in VMWare and VirtualBox. Docker performed comparably to the traditional virtualization platforms though it was usually one of the slower performers. The full test results are included in Appendix F.

3. Evidence Acquisition

The first step in the forensic process is to acquire a reliable copy of the evidence to be analyzed. This evidence can take several different forms, as can the acquisition process. In some cases, particularly in the case of memory captures, the evidence must be acquired directly on the subject system. In other cases, such as hard drives from desktop computers, the analyst's computer may be used to acquire the evidence. In order to preserve the integrity of the evidence, an analyst must be able to make an exact copy of the original evidence and be able to prove that it is identical. They must also ensure that they do not alter the original evidence in the process of acquisition and analysis. The standard procedure for acquiring disk-based evidence is to connect the drive using a write-blocking device, then make an exact copy using imaging software. A hash "fingerprint" of the disk image should be computed at the time of acquisition so that the integrity of the image can be confirmed. Any subsequent analysis should be performed on the image, rather than the original evidence.

The simplest format for disk images is the raw or "dd" format, which is simply a bit-by-bit copy of the original. Any metadata like hash values, date/time information, and notes must be stored in an external file. There are also several dedicated formats for forensic images, including SMART, Expert Witness (EWF) and the Advanced Forensic Format (AFF). Newer versions of several formats such as the EWF2/Ex01format are beginning to gain acceptance. Others, like the under-development AFF4 format promise significant improvements over previous formats, but are not yet widely supported by open source tools. (Schatz, 2016) Currently, the most common of these is the EWF format, also known by its extension of E01. It was first introduced by the EnCase forensic suite and is now considered the de facto standard for forensic images. Due to its prevalence and nearly universal support in forensic tools, this was the default image file format used for all testing.

3.1. Hardware

The most common and reliable way to image a hard disk is to attach the evidence drive to the analysis system through a hardware write blocker, then write the image to an external hard drive. Modern hard drives are quite large and obtaining a forensic image can require a significant amount of time, so imaging performance is a major concern. One unique feature of Macintosh computers is their early adoption of I/O technologies like FireWire, Thunderbolt and USB C. These interfaces have always been on the cutting edge of performance, but support among write blockers and external drive docks has lagged. The generation of Mac Pro desktops and MacBook Pro laptops available at the time or writing supported only two types of ports: USB 3.0 and Thunderbolt 2. The most recent generation of MacBooks has abandoned both types of ports in favor of a single port that supports both the USB-C and Thunderbolt 3 standards. Unfortunately, none of these new MacBooks were available for testing.

Interface	Speed (MB/s)
USB 2.0	60
USB 3.0	525
USB 3.1 (USB-C)	1250
FireWire 400	49
FireWire 800	98
ThunderBolt	1250
ThunderBolt 2	2500
ThunderBolt 3	5000
SATA	150
SATA 2	300
SATA 3	600
PCI Express 3.0 x4 (M2)	3938

Table 2: Maximum Interface Speeds (Wikipedia, 2016)

Table 2 lists the theoretical maximum speed of various interfaces, though real-life performance will vary based on a number of factors. For example, while ThunderBolt 3 is capable of transfers up to 5 GB/s, there are currently no flash storage devices capable of that high a transfer rate. Additionally, the overhead imposed by communications protocols will reduce the real throughput achievable through a given interface. In any data transfer, the actual speed will be determined by the slowest interface in the transfer. Baseline transfer speeds for each tested device were measured using the BlackMagic Disk Speed Test tool available in the OS X App Store. To test imaging speed, the 16GB *xp-tdungan* image was restored to a source drive and imaged to a file on the destination drive using the **dd** utility with a 1MB block size.

3.1.1. Write Blockers

While OS X allows drives to be mounted in read-only mode, hardware write blockers are more reliable, as they prevent data from being written to a drive regardless of how it is mounted. These devices connect an IDE or Serial ATA hard drive to the analysis computer through one of several interfaces. Most write blockers support USB, FireWire and eSATA connections, though the speed of these interfaces varies depending on the age of the device. The performance of two commonly used write blockers was tested during this research. The WiebeTech Forensic Ultradock v5 is the current model distributed with the SANS FOR408 course and supports USB 2.0/3.0, FireWire 400/800 and eSATA interfaces. An older Tableau Digital T35 that was tested has the same ports available, except that its USB interface only supports USB 2.0. With only USB 3 and Thunderbolt 2 ports available on the MacBook Pro, testing the other write blocker ports required adapters. Apple makes FireWire 800 to ThunderBolt adapters, while several aftermarket adapters are available to convert eSATA to either ThunderBolt or USB 3.

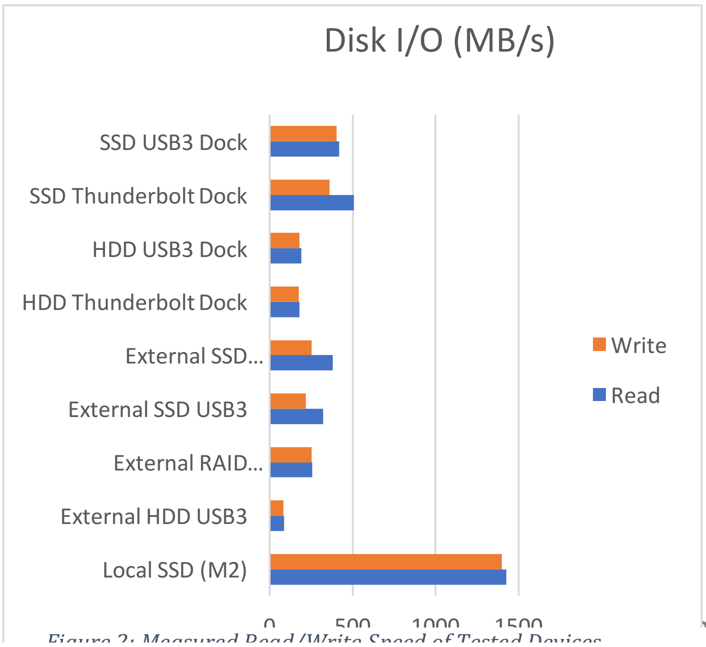


Figure 2: Measured Read/Write Speed of Tested Devices

There were no appreciable performance differences between the two write blockers when connected through USB 2.0 or FireWire

800, interfaces. The eSATA interface on the Tableau, however, was limited to approximately 125MB/s, indicating that it likely uses the older SATA 1.5GB/s interface. It was not used in further testing as this was significantly slower than either source drives tested. The tests quickly illuminated the limitations of spinning hard drives, particularly the slower 5400 rpm drives, which were limited to approximately 100 MB/s. The Western Digital Blue WD10EZEX (7200rpm/64MB cache) drives used for testing had a maximum transfer rate of approximately 190MB/s read and 180MB/s write. By comparison, an older Western Digital Green WD10EADS (5400rpm/32MB Cache) drive was limited to approximately 100MB/s read and write. The Samsung 850 EVO SATA III SSD's used for testing demonstrated a 500MB/s read speed and 400MB/s write speed in tests. Overall, the tests indicated that both write blockers were limited to approximately 225MB/s, even when reading from the faster SSD, but during actual imaging, their speed was closer to 200MB/s. There were minor variations between tests, but the USB 3 and eSATA (with both adapters) interfaces performed nearly identically with a throughput of approximately 200 MB/s. The USB2 and FireWire 800 interfaces were predictably limited to approximately 40 and 80 MB/s, respectively and should not ordinarily be used, as better options are available.

3.1.2. Destination Media

The second part of the imaging equation is the destination drive to which the image file is written. Relatively small images can be stored on the internal SSD if enough space is available. This approach makes use of the extremely high transfer speed of the internal PCIe M2 SSD used in the MacBook. The downside to this approach is that the data is being written to the same drive housing the operating system and applications, which will be competing for I/O bandwidth. A more common approach, that is often necessary when imaging the 500GB – 2 TB hard drives commonly encountered, is to image to an external drive. In the field, it can be convenient to use a portable hard drive as an imaging destination. These drives are typically self-contained, reasonably durable and powered from either the USB or ThunderBolt cable, eliminating the need for additional power adapters. Three different external hard drives were tested: a 2TB Western Digital USB3 HDD and two ruggedized LaCie ThunderBolt drives, one SSD and one portable HDD RAID0. The other common approach, which is the standard in situations where the destination drive must be checked into evidence for long-term storage, is to connect an internal SATA hard drive to an external dock. A HighPoint RocketStor Thunderbolt drive dock and a StarTech USB3 drive dock were tested with both SSD and HDD destination drives.

The external USB hard drive was easily the slowest of the lot, with its transfer speed limited to the neighborhood of 85 MB/s, likely due to its use of a 5400 RPM HDD. The ThunderBolt HDD RAID was somewhat faster with transfer rates around 135 MB/s. The external SSD boasted read speeds of 380 MB/s and write speeds of 250 MB/s. Speeds between the two hard drive docks were identical for the HDD at 180 MB/s and similar for the SSD, between 210 and 220 MB/s. While the MacBook's internal M2 SSD boasted raw speeds of nearly 650MB/s read and 400MB/s write. However, its speed during imaging tests fluctuated wildly from less than 10MB/s to over 200MB/s and was eliminated from testing due to its inconsistent performance. The other drive combinations were consistently limited to the slowest of the two devices, though most combinations were at or near the write blockers' 200MB/s limit. The full test results are contained in Appendix B.

3.2. Software

The third piece of the imaging process is the software used to create and later mount a forensic image. While the image creation process is the same regardless of the filesystem on the device being images, the ability to mount images is dependent on file system support. In many cases, the same tools or suites of tools are used both for collecting and mounting images and libraries for the necessary filesystems must be installed first. OS X has full read-write support for its native HFS/HFS+ filesystems, as well as FAT, FAT32, and exFAT. It also provides Read-only support for the Windows native NTFS, but does not support the ext2/3/4 or xfs filesystems used in common Debian and RedHat-based Linux distributions. The OSXFuse compatibility framework provides kernel drivers and user-land interfaces that allow developers to add support for other filesystems (Fleischer & Larsson, 2016).

OSXFuse can be installed from MacPorts with the package name *osxfuse*, though it lacks the legacy MacFuse compatibility layer required by some filesystems like ext2/3/4. MacPorts users requiring this feature must install OSXFuse from the installer on the GitHub page. Homebrew users can install from the *osxfuse* Cask, which does install the MacFuse compatibility layer. Users who require support for the ext2/3/4 filesystems will need to download and install the *fuse-ext2* driver from the project's GitHub page. The driver and its dependencies must be compiled using the instructions there, as no package installer is yet available on GitHub. There are, indeed, several ext filesystem drivers for Fuse and it is easy to become confused about which to install. The older Ext2fuse and Ext4fuse packages available in both HomeBrew and MacPorts are not actively being maintained and neither works with OS X 10.11 and above. Similarly, Fuse-xfs is not available through either package manager, but does provide an OS X installer on the Fuse-xfs SourceForge page. It is currently in Alpha and provides read-only support for xfs volumes. The Fuse Ntfs-3g driver provides read-write support for NFTS, but its performance is limited due to fact it can only access block devices and not raw devices. It is available under the names *ntfs-3g* for MacPorts and *homebrew/fuse/ntfs-3g* for HomeBrew. Users who need frequent read-write access to NTFS or ext filesystems are probably best off investing in a commercial solution such as those offered by Tuxera or Paragon. Unfortunately, there is no commercial driver for XFS at this time.

3.2.1. Image Creation

The original and still widely used UNIX imaging utility is Dd, which is built into OS X and will make an identical, bit-by-bit copy of any addressable UNIX filesystem object and write it to a given destination. Since all UNIX variants, including OS X address disks as filesystem objects, Dd can be used to copy a drive or partition to another drive or an image file in another location. In OS X, disks are listed by the naming convention of */dev/diskXpY*, where *X* is the disk number and *Y* is the partition number, both starting from 0. The **diskutil** *list* command will display information about the attached disks and partitions. The **dd** command takes two parameters: an input file or device specified by *if*= and an output file or device specified by *of*=. Users may also designate several options, the most important of which is the block size, specified by *bs*=. As it directly accesses hardware devices, Dd must be run with root privileges. This also allows the utility to overwrite important data if the wrong output destination is specified, so great care should be taken when running this command. Once invoked, Dd does not show any indication it is running except a blinking cursor until it completes its copy. At that time, it will display the number of bytes transferred, the amount of time taken and transfer rate in bytes/second. To compute a hash for later verification at the same time, some command line gyrations are in order. The command **dd** *if=/dev/rdisk1 bs=1m conv=sync,noerror | tee evidence.001 | md5sum > evidence.md5* will send one copy of the drive to the image file and another to the Md5sum utility, which will compute the md5 hash and write it to a separate file when the image completes.

The original dd utility has since been improved for forensic use by Dcfldd and Dc3dd. Dcfldd was created by the U.S. Department of Defense Computer Forensics Lab as a fork of the original Dd utility. Dcfldd supports a variety of useful forensic features such as computing hashes on the fly, splitting output files and displaying a progress indicator while imaging. The downsides of Dcfldd are that it is based on an older version of Dd and has difficulty handling corrupted source drives (Forensics Wiki, 2015). Dc3dd is a patch to Dd released by the Department of Defense Cyber Crime Center (CD3). Like Dcfldd, Dc3dd supports piecewise hashing without an external utility, split output and progress indicators. Unlike Dcfldd, it is based on the most current version of Dd and updated whenever Dd is (Forensics Wiki, 2015). Both Dcfldd and Dc3dd are easily installed with no additional configuration from the HomeBrew or MacPorts packages of the same names.

Dd and its descendants can only create uncompressed, raw disk images. Creating EWF images requires either the open source Libewf library and its Ewfacquire utility or the older, closed source command line version of FTK Imager. Libewf is available from both package managers, and includes the Ewf-Tools suite, that includes *ewfacquire* as well as *ewfmount*, *ewfinfo* and *ewfverify*. Libewf is still under active development and while the latest stable version is 20140608, newer experimental versions are available on GitHub (Metz, libewf Wiki, 2016).

While it was expected that all three utilities would perform similarly, the first set of performance data indicated that all the utilities performed surprisingly poorly. Regardless of the interface, block size or even compression level, Dd, Dcfldd, Dc3dd, and Ewfacquire all transferred at approximately 32 MB/s, below even USB 2.0 speeds. A mention of a “raw disk” device in the manpage for the Hdiutil hard disk utility proved the key to this behavior (HDIUTIL(1), 2016). Raw disk devices are not displayed by Diskutil, Mount or other monitoring utilities, but can be addressed with the *rdiskXpY* notation. When an *rdisk* device was specified as the source, all the Dd variants and Libewf tools performed as expected.

The various imaging programs were tested using a variety of different options to determine the most efficient settings. The most important from a performance perspective was the block size, particularly with Dd. With its default setting of a 512-byte block, Dd could only image at 3.6MB/s, where at the optimal setting of a 1MB block, it ran at nearly 190MB/s. Dcfldd and Dc3dd default to larger block sizes, which results in faster speeds, but it is still worthwhile to set them to use the optimal 1MB block size, as can be seen in Appendix D. Ewfacquire was somewhat slower than the Dd variants, even when creating uncompressed E01 files, possibly due to the fact it computes the image’s md5 hash during creation. Ewfacquire performance was optimized with values of 8192 or more sectors at once (-b). Compression had a significant effect on Ewfacquire performance, with “fast compression” reducing the maximum imaging speed from 166MB/s to 67MB/s, while reducing the image size from 16GB to 7GB. Selecting “best compression” reduced the imaging speed to 16MB/s, while only reducing the image size by 100MB. These results are contained in Appendix E.

3.2.2. Image Mounting

Some forensic tools will work directly on an image file without the need to mount it, however many others require the image to be mounted on the examiner’s computer. It is always advisable to mount images in read-only mode to preserve the integrity of the evidence. Most forensic examiners are familiar with the classic **mount** *–t <filesystem> –o ro,loop image.001 /mnt/destination* syntax for mounting Dd images in Linux. In OS X, while the **mount** command is present, it lacks the *loop* and *offset* options and does not properly mount dd images. The exception to this rule is that dd images of single partitions containing filesystems supported by OSXFuse drivers can be mounted with the **mount** *–t <filesystem> image.001 /path/to/mountpoint* command. OS X users can mount images containing natively supported filesystems (HFS+/FAT/NTFS) with the **hdiutil** *attach –readonly -imagekey diskimage-class=CRawDiskImage /path/to/image.001* command. The *–readonly* flag should be omitted for ntfs volumes, as this will produce an error saying the volume is already read-only. Hdiutil will mount any partitions in the dd image containing readable filesystems under the standard OS X */Volumes/<Volume Name>* directory (HDIUTIL(1), 2016).

Mounting EWF images is a two-step process that involves first mounting the ewf container, then the filesystems contained on the partitions within. The most common way to mount an E01 image is with the *ewfmount* utility which should have already been installed as part of the *libewf* package (Metz, libewf Wiki, 2016). The syntax **ewfmount** *<image> <destination>* will mount the image in the specified directory as raw (dd) image file named "ewf1". This file can be treated in the same manner as any other dd image and mounted using *hdiutil* or *mount*, as appropriate.

4. Investigation and Analysis

4.1. Disk Forensics

One of the best known open-source forensic tools is The Sleuth Kit (TSK) suite of command line utilities. TSK can directly access and analyze forensic images and the filesystems within, even those the host operating system does not support. Its tools can be run individually, scripted or integrated into larger forensic tools. Many other common Unix forensic tools rely on TSK, so it should be installed even if the user does not plan to use TSK tools directly. It can be installed from the *sleuthkit* package in either package manager. The Python bindings for TSK should also be installed from the PIP *pytsk3* package. One of the most common TSK utilities is *mmls*, which is used to show partition information for a disk image file and is often necessary for disk mounting. Others include *ifind* and *icat* to locate and extract files, *fls* to list filenames and directories and *jls* to list the filesystem journal. A great more documentation is available on the TSK website (Carrier, The Sleuth Kit, 2017).

Another commonly used and valuable open-source forensic tool is Bulk Extractor, which is designed to parse image files looking for interesting strings and patterns. Unlike

TSK, it is filesystem agnostic and instead parses the image in 16MB pages, looking for a variety of patterns, including IP and email addresses, domain names and URL's. It exports the artifacts it finds to a series of "feature files" in the specified output directory (Garfinkel, 2015). It is installed from *bulk_extractor* package using either package manager. MacPorts installs *afflib* and *libewflib* as dependencies and enables EXIF, AFF, and EWF support by default. HomeBrew users should have *exiv2*, *afflib* and *libewf* installed and set the *--with-exiv2, --with-afflib,* and *--with-libewf* flags during installation. The Java-based BEViewer GUI is also included with the package and can be run from the command line with the *beview* command. It can be used to view the feature files output by Bulk Extractor, which can also be launched directly from the GUI. There is no Bulk Extractor Docker container available in the Docker Store, but several *dockerfiles* to build one are available on GitHub.

VirusTotal's Yara pattern matching utility is an indispensable tool for any forensic analyst who needs to look for malware. It uses rule files that define a set of binary and text strings and the conditions in which those strings appear, then searches for files that match the defined parameters. Yara does not have image file support, so the evidence image must be mounted before Yara is run against it (Alvarez, 2015). Yara can be installed from either package manager under the package name *yara*. Its Python bindings can be installed from the PIP *yara-python* package, but the *openssl* package must be installed prior to installing the Python bindings. HomeBrew users must also use the command **ln** *-s /usr/local/Cellar/openssl/<version>/include/openssl /usr/local/include/* to create a symbolic link or the *yara-python* install will fail. An updated Docker container for Yara is available from the Docker store and can be installed using **docker** *pull blacktop/yara*.

The only free, open-source forensic suite and one of only two suites available for OS X is Autopsy, which is based on the Sleuth Kit. There is no packaged release for OS X yet, but it can be installed from source. Several 3rd party plugins for Autopsy are written for Windows only, and as such, incompatible with OS X. Like many tools, there are no OS X-specific instructions provided, and several installation pitfalls were discovered that are not mentioned in the documentation. Autopsy requires the latest Java 8 JDK, which should be installed and the Apache Ant compiler, which can be installed from HomeBrew or MacPorts. The *JAVA_HOME* and *JDK_HOME* variables must be set to */Library/Java/JavaVirtualMachines/jdk<version>.jdk/Contents/Home/* in the shell prior to installation. Autopsy depends on the SleuthKit, which should have already been installed and HomeBrew's version of SleuthKit can be used to build Autopsy. Unfortunately, MacPorts' version of SleuthKit does not provide the necessary Java bindings for Autopsy, so they must downloaded be built from source. The latest TSK source code should be downloaded and decompressed, then the *TSK_HOME* variable should be set to the directory where it is decompressed. TSK should be set up using the standard **./configure** ... **make** ... **make install** procedure. The SleuthKit Java bindings should then be built, but the existing file *bindings/java/dist/Tsk_DataModel.jar* must first be deleted. The correct bindings may then be built by running **ant** *dist-PostgreSQL* from the bin*dings/java* subdirectory of the source code. Once these prerequisites are met, the Autopsy source code can be downloaded and decompressed into the directory of choice, keeping in mind that it will run from that directory. Autopsy can be built by running the command **ant** from its root directory after which executing **ant** *run* from the same location will launch the GUI (Carrier, Autopsy, 2016).

4.2. Timeline Analysis

The Plaso/Log2Timeline tool revolutionized the intrusion forensics field by extracting timestamps a multitude of forensic artifacts and organizing them into a single timeline. It is often one of the first steps in forensic analysis and can help to identify areas that need closer manual inspection and provide context about the origin and significance of artifacts. The two main Plaso utilities are *log2timeline.py*, which parses image files or directory structures for artifacts and outputs them to a binary *.plaso* file and psort.py that converts *.plaso* file to more human-readable formats (Metz, Gudjonsson, & White, Plaso Wiki, 2016). While this tool is extremely valuable, its many included parsers and staggering list of dependencies make it notoriously tricky to install. A current Docker image for Plaso is available from the Docker store and can be installed with the command **docker** *pull jbeley/plaso*.

The recommended first step to install Plaso is to download the latest OS X release *.dmg* image from the Plaso Github page. Once mounted, the *install.sh* script should be run from the root directory of the mounted image. This script will install 54 Python dependencies and Plaso from OS X *.pkg* installers. The *.pkg* installers are designed to work with the native OS X Python and will install to its site-packages directory. Users using a different Python installation must install Plaso and its dependencies using a different method. Due to a version conflict with the untouchable system version of *pyparsing*, Plaso must be run using the *log2timeline.sh* and *psort.sh* scripts rather than their *.py* versions. This does cause one notable issue in that the *.sh* scripts cannot parse file paths containing spaces, even when quoted or escaped. Attempting to run *log2timeline* after the installer script completes will produce an error complaining that it is missing the *pefile* module. It is tempting to use pip to install the missing libraries, but the Plaso OS X troubleshooting page includes a dire warning against using PIP outside of a virtual environment. This warning is confirmed by the fact that installing *pefile* through pip will cause Plaso to no longer recognize *IPython*, which was installed by the script. Users can choose to install the missing dependencies from source or abandon the release installer altogether and install using pip inside a virtual environment.

Installing the dependencies inside of a virtual environment presents a different challenge, as the *.pkg* files will only install the packages in the system default site-packages directory. Instead, the dependencies and Plaso itself must be installed using PIP. After much trial and many errors, the package dependencies in the installer script were translated into PIP dependencies. These were combined with the *requirements.txt* file on the Plaso GitHub page to create a comprehensive *requirements.txt* file, which is contained in Appendix I. Once inside the virtual environment, simply issue the command **pip** *install –r requirements.txt* to install a working version of Plaso and its dependencies.

Once installed, Plaso can be run against an image file with the command **log2timeline.py** *<output_file.plaso> </path/to/image>*. If no parsers are specified with the *--parsers <parser1,parser2…>* flag, Plaso will choose the best meta-parser, such as *win7* or *winxp* based on the operating system detected. It is both more efficient and reliable to specify a list of parsers for the artifacts of interest, rather than rely on the autodetection, which may be incorrect. The selected meta-parser also may not include important artifacts, such as the Master File Table (MFT). A list of parsers can be displayed with the *--parsers list* command (Metz, Gudjonsson, & White, Plaso Wiki, 2016).

By default, *log2timeline* runs in "kitchen sink mode", which recursively parses the image, including volume shadow copies, and attempts to parse every file with the specified parsers. This approach can be highly inefficient, as demonstrated by the *mft* parser, which will parse every file in a filesystem, attempting to determine if it is the MFT. This file is only ever located in the root directory of NTFS volumes, so the parser will needlessly check hundreds of thousands of files. A better approach from a performance standpoint is to limit the search using the *-f <filter_file>* option. A filter file contains a list of paths to search and can make use of regular expressions and reserved words like *{systemroot}*. A filter file is included in Appendix J that will locate the majority of Windows artifacts, including the registry, event logs, MFT and prefetch files. The performance advantages of this approach are evident as a test running the *mft* and *winxp* parsers against the *xp-tdungan* image unfiltered took 54:35 and extracted 548,429 events. Processing the same image using the *winxp* and *mft* parsers using a filter file took 4:42 and found 647,857 events.

After *log2timeline* has created a *.plaso* file, the psort.py utility can be used to convert the binary file into multiple formats, including several varieties of CSV, JSON, XLSX and SQLite. Users with Elasticsearch or TimeSketch installed can output to those platforms as well. The problem with many of these formats is that they can be difficult to search due to the large number of artifacts often extracted by Plaso. Long-time *log2timeline* users will remember importing CSV files into the timeline color template that exceeded Excel's maximum number of lines. Psort allows users to limit the number of results by filtering on a time range or other criteria, as described in the Plaso documentation (Metz, Gudjonsson, & White, Plaso Wiki, 2016).

TimeSketch provides a convenient front end to view, sort and visualize *log2timeline* data. It requires the Elasticsearch and PostgreSQL databases be installed. HomeBrew provides both these packages, though users should install the *elasticsearch@2.4* package rather than the standard Elasticsearch 5.1, as it is not yet supported. MacPorts users can install *postgresql96*, but must download and install Elasticsearch from its website. After installing both programs, follow the installation steps in Appendix G. Note that the configuration files will be in different locations depending on how the packages were installed. HomeBrew users can start both as background services with the **brew** *services start <service>* command, while MacPorts users must start them manually. Once both databases are configured and running, run **pip** *install psycopg2 timesketch* to install TimeSketch and its Python dependencies. This should be installed into the same virtual environment used for Plaso. Once TimeSketch is installed, the command **tsctl** *runserver -h 0.0.0.0 -p 5000* will start the server on port 9000. The interface can be accessed through a web browser at http://localhost:5000. While the TimeSketch server is running, *psort.py* will have the option to write directly to the TimeSketch database with the *-o timesketch* flag. Users can also upload CSV or JSON files through the interface or set TimeSketch to automatically ingest and process *.plaso* files, though this requires additional configuration.

4.3. Memory Forensics

Memory forensics is a critical and growing discipline, particularly when intruders have covered their tracks on disk or used "diskless" malware that resides only in memory. There are currently two open-source memory analysis platforms in common use, Volatility and Rekall, and this guide will cover both. Rekall is a newer Python-based memory forensic framework that has been gaining in popularity. It should be installed with **pip** *install rekall* in its own virtual environment, which will also install all its dependencies. Once installed, its syntax is **rekal** *--plugin <plugin> -f <memory-image>*. Rekall includes a GUI frontend, which can be installed from the PIP *rekall-gui* package. The GUI can be launched with the **rekal** *webconsole --browser --worksheet </path/to/empty/dir>* command (Google, 2015).

Volatility is the more established of the two products and is available as either a Python package or a standalone executable for OS X. The standalone version can be downloaded from the Volatility Foundation website and will run with no additional configuration after it is unzipped. It is advisable to add the executable to a location in the path and change its name to an easier-to-type filename than its default *volatility_2.6_mac64_standalone*. It is also available from both package managers and PIP, though the version provided by PIP is over four years old and should be avoided. Once installed, Volatility can be executed with *(***vol.py**|**volatility_standalone***) <plugin> -f <memory_image>*. There are many plugins available for exploring various memory artifacts, but the first to run is invariably the *imageinfo* plugin that determines the memory profile, which is needed by many plugins (Volatility Foundation, 2016).

Volatility does not have include a GUI, however a third-party GUI called VolUtility is available on GitHub. As it depends on Volatility, it should be installed into the same virtual environment where the Python version of Volatility is installed. It also requires MongoDB to be installed and running. MongoDB is which is available from the HomeBrew package *mongodb*, but MacPorts users must install it manually. The VolUtility source code should be downloaded and unzipped to a directory of choice and pip should be used to install the requirements.txt file in the root directory of the package. Once the dependencies are installed, VolUtility can be started using a script in the VolUtility root with the syntax **manage.py** *runserver <ip>:<port>* and accessed through a web browser (kevthehermit, 2017).

5. Conclusion

Configuring open-source forensic tools in OS X can be a delicate and time-consuming process, but if done correctly, it can be a worthwhile one. Virtualization has its place in forensics, but for involved and time-consuming tasks, the performance advantages of running tools in the native operating system are significant. Given the power of the MacBook hardware and the OS X operating system, it seems to a waste to only use it as a VM host while conducting forensic analysis. As has been demonstrated herein, it is possible to possible to install a set of forensic tools natively in OS X that will handle nearly any forensic task. The hope is that this guide will provide a resource for forensic analysts who use OS X and eliminate much of the guesswork and frustration from the configuration process.

References

Akcan, A. (2017, January 17). *fuse-ext2*. Retrieved from GitHub: https://github.com/alperakcan/fuse-ext2

Alvarez, V. M. (2015). *Yara Documentation*. Retrieved from Read the Docs: http://yara.readthedocs.io/en/latest/gettingstarted.html

Berggren, J. (2016, December 19). *TimeSketch*. Retrieved from GitHub: https://github.com/google/timesketch/wiki

Carrier, B. (2016, March 15). *Autopsy*. Retrieved from GitHub: https://github.com/sleuthkit/autopsy

Carrier, B. (2017). *The Sleuth Kit*. Retrieved from http://wiki.sleuthkit.org/index.php?title=Help_Documents

Duling, M., Maibaum, M. A., Barton, W., & Lang, C. (2014). *MacPorts Guide.* Retrieved 12 16, 2016, from https://guide.macports.org

Fleischer, B., & Larsson, E. (2016, February 16). *OSXFuse*. Retrieved from GitHub: https://github.com/osxfuse/osxfuse/wiki

Forensics Wiki. (2015, March 14). *Dcfldd*. Retrieved from http://www.forensicswiki.org/wiki/Dcfldd

Garfinkel, S. L. (2015, March 23). *Bulk Extractor User Manual*. Retrieved from http://digitalcorpora.org/downloads/bulk_extractor/BEUsersManual.pdf

Google. (2015). *Rekall Manual*. Retrieved from http://www.rekall-forensic.com/docs/Manual/overview.html

Hardy, A. (2016, August 15). *fuse-xfs*. Retrieved from SourceForge: https://sourceforge.net/projects/fusexfs/

HDIUTIL(1). (2016, June 13). *BSD General Commands Manual*.

kevthehermit. (2017, January 2). *VolUtility Wiki*. Retrieved from GitHub: https://github.com/kevthehermit/VolUtility/wiki

McQuaid, M., & Labinskiy, N. (n.d.). *HomeBrew Documentation.* Retrieved December 16, 2016, from https://git.io/brew-docs

Metz, J. (2016, September 11). *libewf Wiki*. Retrieved from GitHub: https://github.com/libyal/libewf/wiki

Metz, J., Gudjonsson, K., & White, D. (2016). *Plaso Wiki.* Retrieved January 1, 2017, from GitHub: https://github.com/log2timeline/plaso/wiki

mfripp. (2016, April 25). *How to Use pip After the OS X El Capitan Upgrade*. Retrieved from StackExchange: http://apple.stackexchange.com/questions/209572

Python Packaging Authority. (2016). *Pip Docs.* Retrieved December 16, 2016, from https://pip.pypa.io/en/stable/

Python Software Foundation. (2016). *Python Packaging User Guide.* Retrieved December 16, 2016, from https://packaging.python.org/current/

Raj, P., Chelladhurai, J. S., & Singh, V. (2015, September 14). *Learning Docker.* Birmingham, UK: Packt. Retrieved January 1, 2017, from Jaxenter: https://jaxenter.com/containerization-vs-virtualization-docker-introduction-120562.html

Reitz, K., & Schlusser, T. (2016). *The Hitchhiker's Guide to Python.* Sebastopol, CA: O'Reilly Media.

SANS Institute. (2014). *SANS Investigative Forensics Toolkit Documentation*. Retrieved January 7, 2017, from SANS DFIR: http://sift.readthedocs.io/en/latest/

Schatz, B. (2016, October 26). AFF4: The New Standard in Forensic Imaging and Why You Should Care. *OSDFCon 2016.* Reston VA: Basis Technology.

StackOverflow. (2016, December). *Developer Survey Results 2016*. Retrieved from StackOverflow: http://stackoverflow.com/research/developer-survey-2016#technology-desktop-operating-system

Tanous, J. (2015, September 4). *2015 VM Benchmarks: Parallels 11 vs. Fusion 8 vs. VirtualBox 5.* Retrieved from TekReview: https://www.tekrevue.com/2015-vm-benchmarks-parallels-11-vs-fusion-8

The Fink Project. (2012, November 11). *Fink - Documentation*. Retrieved December 16, 2016, from http://www.finkproject.org/doc/index.php?phpLang=en

Volatility Foundation. (2016, December 29). *Volatility Wiki*. Retrieved from GitHub: https://github.com/volatilityfoundation/volatility/wiki

Wikipedia. (2016, December 20). *List of Device Bit Rates.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/List_of_device_bit_rates

Appendix A – Package Support

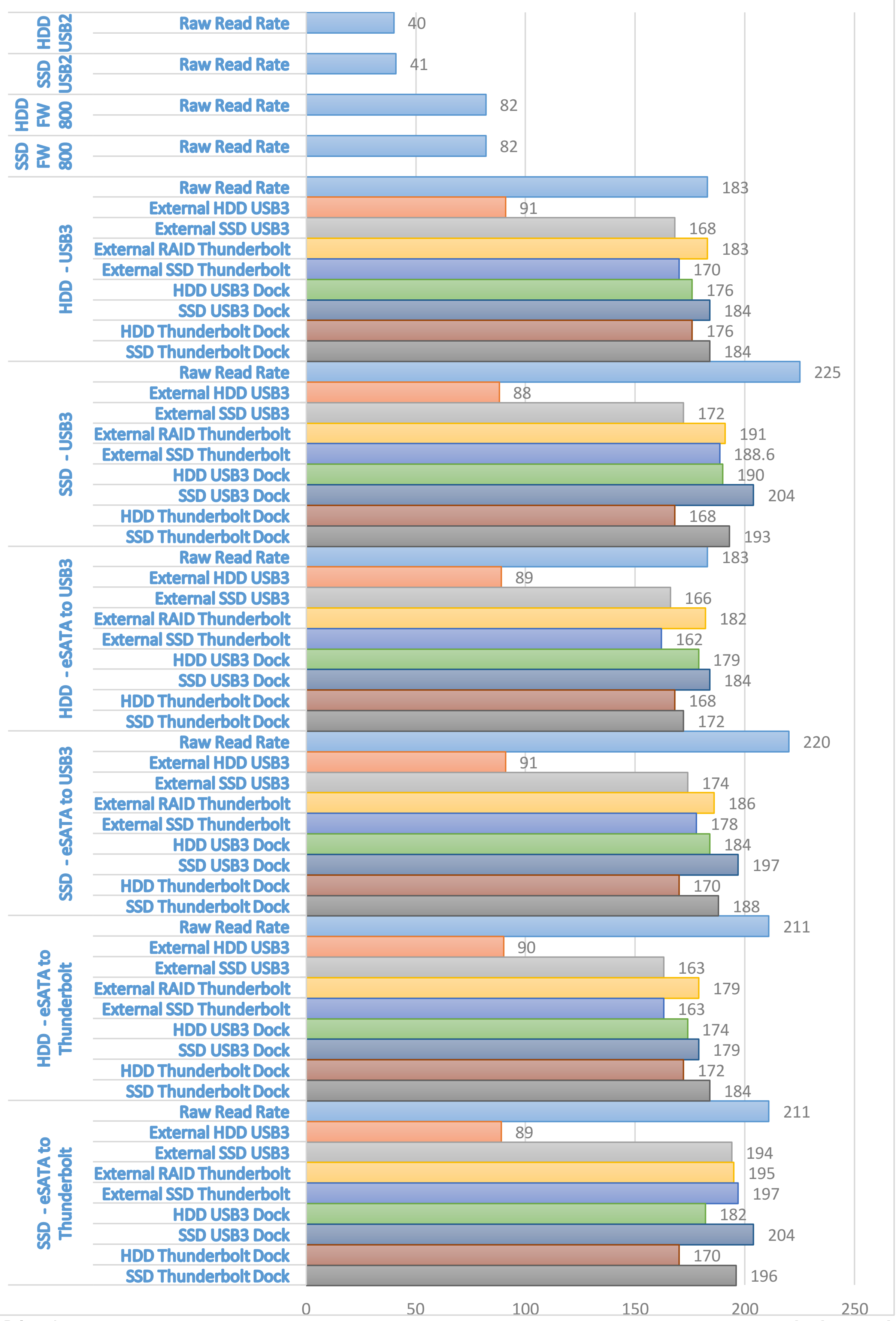
Package	Latest Version	Port Name	Port Version	Brew Name	Brew Version	PIP name	PIP version
Python	2.7.13	python27	2.7.13	python	2.7.13	N/A	N/A
pip	9.0.1	py27-pip	9.0.1	installed w/python	9.0.1	pip	9.0.1
virtualenv	15.1.0	py27-virtualenv	15.1.0	N/A	N/A	virtualenv	15.1.0
Virtualenv wrapper	4.7.2	py27-virtualenv wrapper	4.7.1	N/A	N/A	virtualenv wrapper	4.7.2
dcfldd	1.3.4-1	dcfldd	1.3.4-1	dcfldd	1.3.4-1	N/A	N/A
dc3dd	7.2.641	dc3dd	7.2.641	dc3dd	7.4.646	N/A	N/A
libewf	20140608	libewf	20140608	libewf	20140608	N/A	N/A
libewf Python Bindings	20160802	N/A	N/A	N/A	N/A	libewf-python	20160802
osxfuse	3.5.4	osxfuse	3.5.3	cask osxfuse	3.5.3	N/A	N/A
Fuse NTFS-3G	2016.2.22	ntfs-3g	2015.3.14	ntfs-3g	2016.2.22	N/A	N/A
Afflib	3.7.15	afflib	3.7.4	afflib	3.7.15	N/A	N/A
Sleuthkit	4.4.0	sleuthkit	4.3.1-5	sleuthkit	4.3.1	N/A	N/A
Sleuthkit Python Bindings	20161109	py27-tsk	20150111	N/A	N/A	pytsk3	20161109
Bulk Extractor	1.5.5	bulk_extractor	1.5.5_1	bulk_extractor	1.5.5_1	N/A	N/A
Yara	3.5.0	yara	3.4.0_1	yara	3.5.0	N/A	N/A
Yara Python Bindings	3.5.0	py27-yara	3.4.0_1	N/A	N/A	yara-python	3.5.0
Volatility	2.6	volatility	2.5	volatility	2.6	volatility	2.1
VolUtility	1.1	N/A	N/A	N/A	N/A	N/A	N/A
Rekall	1.6.0	rekall	1.0.2	N/A	N/A	rekall	1.6.0
Rekall GUI	1.5.0	N/A	N/A	N/A	N/A	rekall_gui	1.5.0.post4
Plaso	1.5.1	log2timeline	0.65_6	N/A	N/A	plaso	1.5.2
TimeSketch	2016.7	N/A	N/A	N/A	N/A	timesketch	2016.7
Elastic search	2.4.2	N/A	N/A	elasticsearch@2.4	2.4.2	N/A	N/A
PostgreSQL	9.6.1	Postgresql96	9.6.1	postgresql	9.6.1	N/A	N/A
MongoDB	3.4.1	N/A	N/A	Mongodb	3.4.1	N/A	N/A

Author Name, email@address

Appendix B – Python System Packages

- altgraph-0.10.2
- bdist_mpkg-0.5.0
- bonjour_py-0.3
- macholib-1.5.1
- matplotlib-1.3.1
- modulegraph-0.10.4
- numpy-1.8.0rc1
- py2app-0.7.3
- pyOpenSSL-0.13.1
- pyparsing-2.0.1
- python_dateutil-1.5
- pytz-2013.7
- scipy-0.13.0b1
- setuptools-18.5
- six-1.4.1
- xattr-0.6.4
- zope.interface-4.1.1

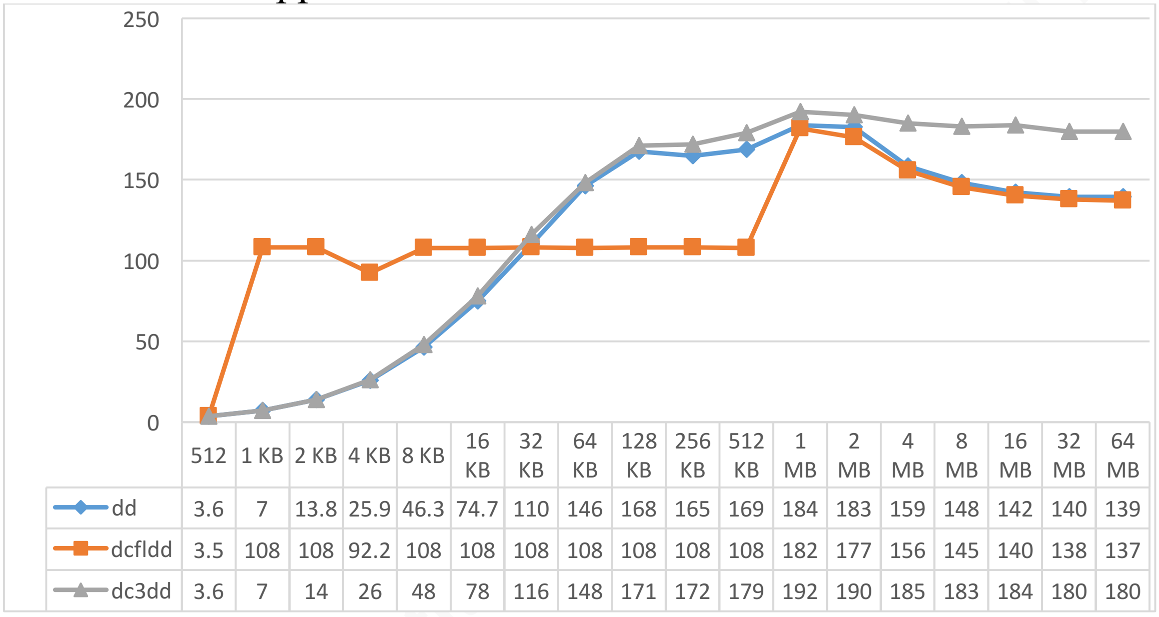
Appendix C – Imaging Performance



Destination / Source	Raw Read Rate (MB/s)	External HDD USB3	External SSD USB3	External HDD Thunderbolt	External SSD Thunderbolt	HDD USB3 Dock	SSD USB3 Dock	HDD Thunderbolt Dock	SSD Thunderbolt Dock
HDD - USB2	40	40	40	40	40	40	40	40	40
SSD - USB2	41	40	40	40	40	40	40	40	40
HDD - USB3	183	91	168	183	170	176	184	176	184
SSD - USB3	225	88	172	191	188	190	204	168	193
HDD - FW800 to Thunderbolt	82	82	82	82	82	82	82	82	82
SSD - FW800 to Thunderbolt	82	82	82	82	82	82	82	82	82
HDD - eSATA to USB3	183	89	166	182	162	179	184	168	172
SSD - eSATA to USB3	220	91	174	186	178	184	197	170	188
HDD - eSATA to Thunderbolt	183	90	163	179	163	174	179	172	184
SSD - eSATA to Thunderbolt	211	89	194	195	197	182	204	170	196

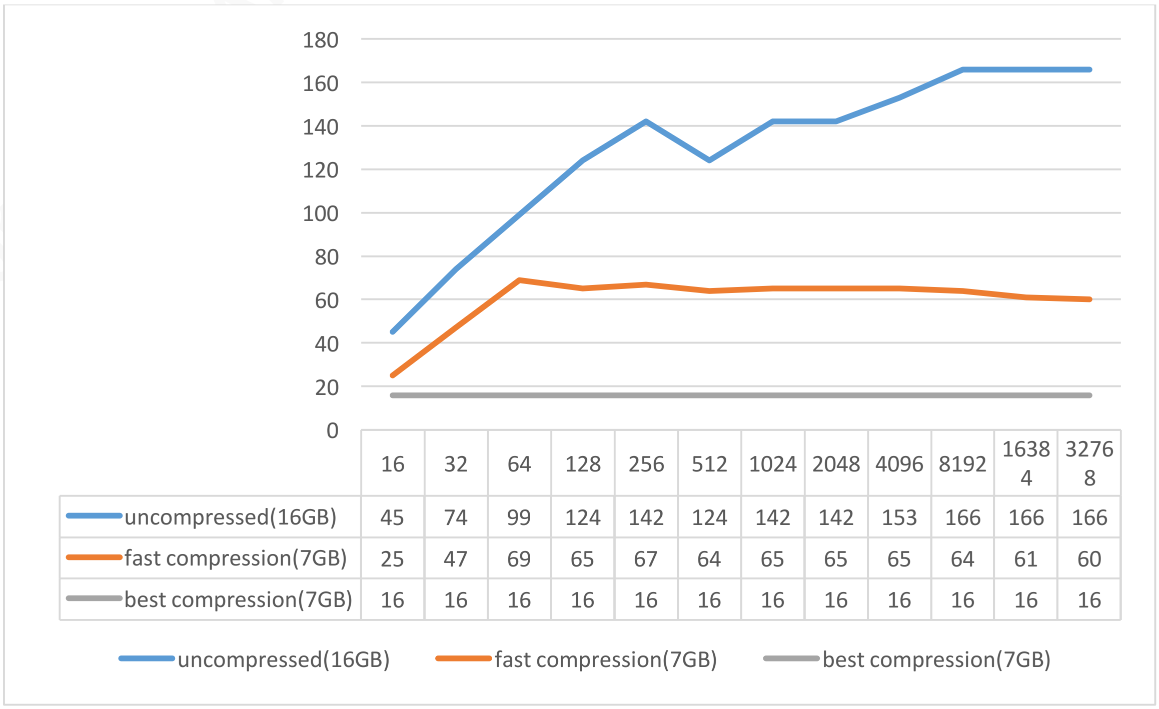
Author Name, email@address

Appendix D – DD Performance - Block Size



	512	1 KB	2 KB	4 KB	8 KB	16 KB	32 KB	64 KB	128 KB	256 KB	512 KB	1 MB	2 MB	4 MB	8 MB	16 MB	32 MB	64 MB
dd	3.6	7	13.8	25.9	46.3	74.7	110	146	168	165	169	184	183	159	148	142	140	139
dcfldd	3.5	108	108	92.2	108	108	108	108	108	108	108	182	177	156	145	140	138	137
dc3dd	3.6	7	14	26	48	78	116	148	171	172	179	192	190	185	183	184	180	180

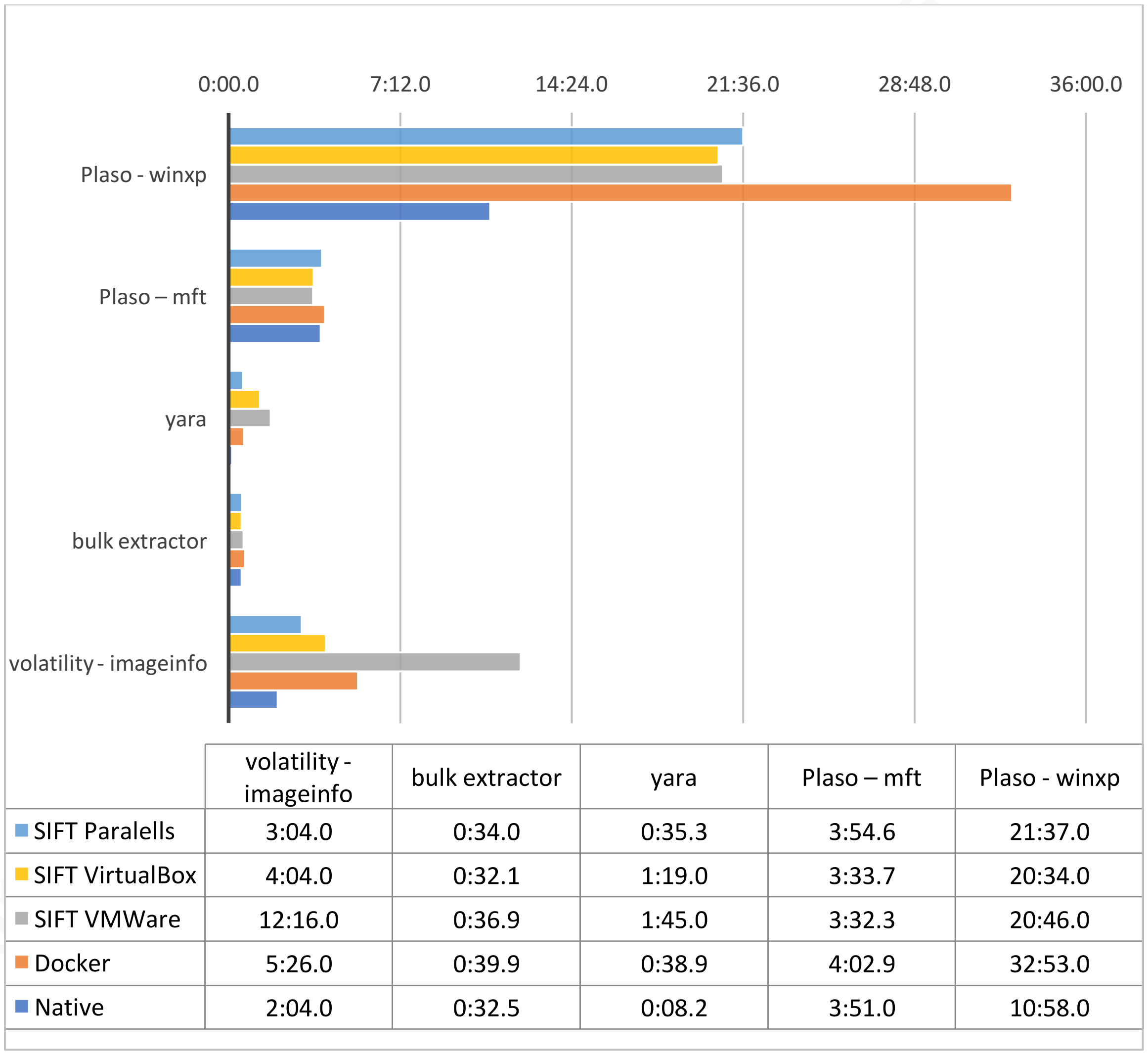
Appendix E – EWF Performance (Buffer Size and Compression)



	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
uncompressed(16GB)	45	74	99	124	142	124	142	142	153	166	166	166
fast compression(7GB)	25	47	69	65	67	64	65	65	65	64	61	60
best compression(7GB)	16	16	16	16	16	16	16	16	16	16	16	16

Author Name, email@address

Appendix F – Virtualization Performance



	volatility - imageinfo	bulk extractor	yara	Plaso – mft	Plaso - winxp
🟦 SIFT Paralells	3:04.0	0:34.0	0:35.3	3:54.6	21:37.0
🟨 SIFT VirtualBox	4:04.0	0:32.1	1:19.0	3:33.7	20:34.0
⬜ SIFT VMWare	12:16.0	0:36.9	1:45.0	3:32.3	20:46.0
🟧 Docker	5:26.0	0:39.9	0:38.9	4:02.9	32:53.0
🟦 Native	2:04.0	0:32.5	0:08.2	3:51.0	10:58.0

Appendix G – MacPorts Install Process

1. Install OS X
2. Install XCode
   a. Install from Apple AppStore
3. Install XCode
   a. **xcode-select** *--install*
4. Install Oracle JDK 8
   a. Download from http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
5. Install MacPorts
   a. Download from https://www.macports.org/install.php
   b. **sudo port** *selfupdate*
6. Install Dev Tools
   a. **sudo port** *install libtool autoconf automake pkg-config*
7. Install Python/PIP
   a. If using native OS X Python
      i. **sudo easy_install** *pip*
   b. If using installer Python
      i. Download from https://www.python.org/downloads/
   c. If using MacPorts Python
      i. **sudo port** *install python27 py27-pip py27-readline*
      ii. **sudo port** *select --set python python27*
      iii. **sudo port** *select --set pip pip27*
8. Install VirtualEnv/VirtualEnvWrapper
   a. If using native OS X Python:
      i. **pip** *install –ignore-installed virtualenv virtualenvwrapper*
      ii. Add to ~/.bash_profile:
      export WORKON_HOME=~/.virtualenvs
      source /usr/local/bin/virtualenvwrapper.sh
   b. Else, if using installer Python:
      i. **pip** *install virtualenv virtualenvwrapper*
      ii. Add to ~/.bash_profile:
      export WORKON_HOME=~/.virtualenvs
      source /Library/Frameworks/Python.framework/ Versions/2.7/bin/ virtualenvwrapper.sh
   c. Else, if using MacPorts Python:
      i. **sudo port** *install py27-virtualenv py27-virtualenvwrapper*
      ii. **sudo port** *select --set virtualenv py27-virtualenv*
      iii. Add to ~/.bash_profile:
      export PATH=/opt/local/bin:$PATH
      export WORKON_HOME=~/.virtualenvs
      source /opt/local/bin/virtualenvwrapper.sh-2.7
9. InstallOSXFuse
   a. **sudo port** *install osxfuse*

   b. OR Download installer from https://osxfuse.github.io/
10. Install dcfldd/dc3dd
    a. **sudo port** *install dcfldd dc3dd*
11. Install libewf/afflib
    a. **sudo port** *install libewf afflib*
    b. **sudo pip** *install libewf-python*
12. (Optional) Install Fuse NTFS-3G
    a. **sudo port** *install ntfs-3g*
13. (Optional) Install Fuse-EXT2
    a. Download from https://github.com/alperakcan/fuse-ext2/releases
    b. Follow directions on GitHub – must make several tools from source
14. (Optional) Install Fuse-XFS
    a. Download from https://sourceforge.net/projects/fusexfs/
    b. Install package
       i. Unrecognized developer – must allow to run from security menu
15. Install SleuthKit (TSK)
    a. **sudo port** *install sleuthkit*
    b. **sudo pip** *install pytsk3*
16. Install BulkExtractor
    a. **sudo port** *install bulk_extractor*
17. Install Yara
    a. **sudo port** *install yara*
    b. **sudo pip** *install yara-python*
18. (Optional) Install Autopsy
    a. **sudo port** *install apache-ant*
    b. edit ~/.bashrc and add the following lines:
       i. export JAVA_HOME='/Library/Java/JavaVirtualMachines/jdk<version>.jdk/Contents/Home/'
       ii. export JDK_HOME='/Library/Java/JavaVirtualMachines/jdk<version>.jdk/Contents/Home/'
    c. Download SleuthKit from: https://github.com/sleuthkit/sleuthkit/releases
    d. **cd** *~/Downloads*
    e. **tar** *–xzvf sleuthkit-<version>.tar.gz*
    f. **cd** *sleuthkit-<version>*
    g. **export** *TSK_HOME='~/Downloads/sleuthkit-<version>'*
    h. **./configure**
    i. **make**
    j. **make install**
    k. **cd** *bindings/java*
    l. **rm** *dist/Tsk_DataModel.jar*
    m. **ant** *dist-PostgreSQL*
    n. Download Autopsy from https://github.com/sleuthkit/autopsy/releases
    o. **mkdir** *~/Tools* (you can place Autopsy in whatever directory you want)
    p. **mv** *~/Downloads/autopsy-<version>.tar.gz ~/Tools*

Author Name, email@address

- q. **cd** *~/Tools*
- r. **tar** *-xzvf autopsy-<version>.tar.gz*
- s. **cd** autopsy-<version>
- **t. ant**
- u. **ant** *run*

19. Install Plaso (Log2Timeline)
    - a. **mkvirtualenv** *plaso* (will also activate the virtualenv)
    - b. Create a 'requirements.txt' file from Appendix I
    - c. **pip** *install -r requirements.txt*
20. Install TimeSketch
    - a. plaso virtualenv should be activated, if not:
        - i. **workon** *plaso*
    - b. Install Postgre SQL
        - i. **sudo port** *install postgresql96-server*
        - ii. **sudo port** *select --set postgresql postgresql96*
    - c. Configure and Launch PostgreSQL
        - i. **sudo mkdir** *-p /opt/local/var/db/postgresql96/defaultdb*
        - ii. **sudo chown** *postgres:postgres /opt/local/var/db/postgresql96/defaultdb*
        - iii. **cd** */opt/local*
        - iv. **sudo su** *postgres -c '/opt/local/lib/postgresql96/bin/initdb -D /opt/local/var/db/postgresql96/defaultdb'*
        - v. edit */opt/local/var/db/postgresql96/defaultdb/pg_hba.conf* to add:
          ```
          local   all          timesketch                    md5
          ```
        - vi. **sudo port** *load postgresql96-server*
        - vii. **sudo** *-u postgres createuser -d -P -R -S timesketch*
        - viii. **sudo** *-u postgres createdb -O timesketch timesketch*
    - d. Install Elasticsearch
        - i. Download latest Elasticsearch 2.X (doesn't yet work with 5.X) from: https://www.elastic.co/downloads/past-releases
        - ii. Copy to directory of choice
        - iii. **tar** *–xzvf elasticsearch-2.<version>.tar.gz*
    - e. Start Elasticsearch
        - **i. /<elasticsearch_directory>/bin/elasticsearch**
    - f. Install and Configure TimeSketch
        - i. **pip** *install psycopg2 timesketch*
        - ii. copy *.virtualenvs/plaso/share/timesketch/timesketch.conf* to */etc/* (must sudo)
        - iii. generate a key with **openssl** *rand -base64 32*
            1. paste key into */etc/timesketch.conf*
            2. add timesketch username and password to "SQLALCHEMY_DATABASE_URI" line (chosen in step c.vii)
            3. **tsctl** *add_user -u <username>*
    - g. Start TimeSketch
        - *i.* **tsctl** *runserver -h 0.0.0.0 -p 5000*

Author Name, email@address

21. Install Rekall and GUI
    a. **mkvirtualenv** *rekall*
    b. **pip** *install rekall recall-gui*
22. Install Volatility
    a. **sudo port** *install volatility*
    b. To build from source for latest version:
        i. Download latest Volatility source code from http://www.volatilityfoundation.org/releases
        ii. **mkvirtualenv** *volatility*
        iii. **pip** *install pycrypto distorm3*
        iv. **pip** *install ~/Downloads/volatility-<version.zip>*
23. Install VolUtility (should be in same virtualenv if Volatility built from source)
    a. Install MongoDB
        i. Download from: https://www.mongodb.com/download-center#community
    b. Copy to location of choice (/bin/ subdirectory should be in $PATH)
        i. **mkdir** *~/Tools/mongodb*
        ii. **cp** *-R -n mongodb-osx-x86_64-<version>/ ~/Tools/mongodb*
    c. Create data directory
        i. **mkdir** *-p /data/db*
        ii. **sudo chown** *–R <your_username> /data*
    d. Start MongoDB
        i. mongod
    e. Download VolUtility from: https://github.com/kevthehermit/VolUtility/releases
    f. Move to location of choise
        i. **mv** *Volutilility-<version>.tar.gz ~/Tools*
        ii. **cd** *~/Tools*
        iii. **tar** *–xzvf Volutilility-<version>.tar.gz*
        iv. **pip** *install –r Volutilility-<version>/requirements.txt*
    g. Run from *~/Tools/Volutility<version>*
        i. **./manage.py** *migrate*
        ii. **./manage.py** *runserver 0.0.0.0:8000*

Author Name, email@address

Appendix H – HomeBrew Install Process

2. Install OS X
3. Install XCode
    a. Install from Apple AppStore
4. Install XCode
    a. **xcode-select** *–install*
5. Install Oracle JDK 8
    a. Download from http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
6. Install HomeBrew
    a. **/usr/bin/ruby** *-e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"*
7. Install HomeBrew Cask
    a. **brew** *tap caskroom/cask*
8. Install Dev Tools
    a. **brew** *install libtool autoconf automake pkg-config*
9. Install Python/PIP
    a. If using native OS X Python
        i. **sudo easy_install** *pip*
    b. If using installer Python
        i. Download from https://www.python.org/downloads/
    c. **brew** *install python*
    d. (optional) **brew** *brew linkapps python*
10. Install VirtualEnv/VirtualEnvWrapper
    a. If using native OS X Python
        i. **pip** *install –ignore-installed virtualenv virtualenvwrapper*
        ii. Add to *~/.bash_profile*:
        export WORKON_HOME=~/.virtualenvs
        source /usr/local/bin/virtualenvwrapper.sh
    b. Else if using installer Python
        i. **pip** *install virtualenv virtualenvwrapper*
        ii. Add to *~/.bash_profile*:
        export WORKON_HOME=~/.virtualenvs
        source /Library/Frameworks/Python.framework/ Versions/2.7/bin/ virtualenvwrapper.sh
    c. Else if using HomeBrew Python:
        i. **pip** *install virtualenv virtualenvwrapper*
        ii. Add to *~/.bash_profile*:
        export WORKON_HOME=~/.virtualenvs
        source /usr/local/bin/virtualenvwrapper.sh
11. InstallOSXFuse
    a. **brew cask** *install osxfuse*

12. Install dcfldd/dc3dd
    a. **brew** *install dcfldd dc3dd*
13. Install libewf/afflib
    a. **brew** *install libewf*
    b. **brew** *install afflib –with-osxfuse*
    c. **pip** *install libewf-python*
14. (Optional) Install Fuse NTFS-3G
    a. **sudo brew** *install ntfs-3g*
15. (Optional) Install Fuse-EXT2
    a. Download from https://github.com/alperakcan/fuse-ext2/releases
    b. Follow directions on GitHub – must make several tools from source
16. (Optional) Install Fuse-XFS
    a. Download from https://sourceforge.net/projects/fusexfs/
    b. Install package
        i. Unrecognized developer – must allow to run from security menu
17. Install SleuthKit (TSK)
    a. **brew** *install sleuthkit --with-afflib --with-jni --with-libewf*
    b. **pip** *install pytsk3*
18. Install BulkExtractor
    a. **brew** *install bulk_extractor --with-afflib --with-exiv2 --with-libewf*
19. Install Yara
    a. **brew** *install yara*
    b. **pip** *install yara-python*
20. (Optional) Install Autopsy
    a. **brew** *install ant*
    b. edit *~/.bash_profile* and add the following lines:
        i. export JAVA_HOME=$(/usr/libexec/java_home)
        ii. export JAVA_HOME=$(/usr/libexec/java_home)
        iii. export TSK_HOME=/usr/local/Cellar/sleuthkit/<version>/
    c. **cd** */usr/local/Cellar/sleuthkit/<version>/bindings/java*
    d. **rm** *dist/Tsk_DataModel.jar*
    e. **ant** *dist-PostgreSQL*
    f. Download Autopsy from https://github.com/sleuthkit/autopsy/releases
    g. **mkdir** *~/Tools* (you can place Autopsy in whatever directory you want)
    h. **mv** *~/Downloads/autopsy-<version>.tar.gz ~/Tools*
    i. **cd** *~/Tools*
    j. **tar** *-xzvf autopsy-<version>.tar.gz*
    k. **cd** *autopsy-<version>*
    l. **ant**
    m. **ant** *run*
21. Install Plaso (Log2Timeline)
    a. **mkvirtualenv** *plaso* (will also activate the virtualenv)
    b. Create a '*requirements.txt*' file from Appendix I
    c. *pip install -r requirements.txt*
22. Install TimeSketch
    a. plaso virtualenv should be activated, if not:

Author Name, email@address

i. **workon** *plaso*

b. Install Postgre SQL
   i. **brew** *install postgresql*
c. Configure and Launch PostgreSQL
   i. edit */usr/local/var/postgres/pg_hba.conf* to add:

```
local   all           timesketch                          md5
```

   ii. **brew services** *start postgresql*
   iii. **createuser** *-d -P -R -S timesketch*
   iv. **createdb** *-O timesketch timesketch*
d. Install Elasticsearch
   i. **brew** *install elasticsearch@2.4*
e. Start Elasticsearch
   i. **brew** *services start elasticsearch@2.4*
f. Install and Configure TimeSketch
   i. **pip** *install psycopg2 timesketch*
   ii. **sudo cp** *.virtualenvs/plaso/share/timesketch/timesketch.conf /etc/*
   iii. generate a key
      1. **openssl** *rand -base64 32*
   iv. edit /etc/timesketch.conf
      1. paste key into "SECRET_KEY" line
      2. add timesketch username and password to "SQLALCHEMY_DATABASE_URI" line (chosen in step c.vii)
   v. **tsctl** *add_user -u <username>*
g. Start TimeSketch
   i. **tsctl** *runserver -h 0.0.0.0 -p 5000*

23. Install Rekall and GUI
    a. **mkvirtualenv** *rekall*
    b. **pip** *install rekall recall-gui*
24. Install Volatility
    a. **brew** *install volatility*
25. Install VolUtility
    a. Install MongoDB
       i. **brew** *install mongodb*
    b. Start MongoDB
       i. **brew** *services start mongodb*
    c. Download VolUtility from: https://github.com/kevthehermit/VolUtility/releases
    d. Move to location of choice
       i. **mv** *Volutilility-<version>.tar.gz ~/Tools*
       ii. **cd** *~/Tools*
       iii. **tar** *–xzvf Volutilility-<version>.tar.gz*
       iv. **pip** *install –r Volutilility-<version>/requirements.txt*
    e. Run from *~/Tools/Volutility<version>*
       i. **./manage.py** *migrate*
       ii. **./manage.py** *runserver 0.0.0.0:8000*

Appendix I – Plaso requirements.txt

```
artifacts >= 20150409
bencode
binplist >= 0.1.4
construct >= 2.5.2,<= 2.5.3
python-dateutil >= 1.5
dfdatetime >= 20160319
dfvfs >= 20160803
dfwinreg >= 20160320
dpkt >= 1.8
efilter >= 1-1.5
guppy >= 0.1.10
hachoir-core >= 1.3.3
hachoir-metadata >= 1.3.3
hachoir-parser >= 1.3.4
IPython >= 1.2.1
libbde-python >= 20140531
libesedb-python >= 20150409
libevt-python >= 20120410
libevtx-python >= 20141112
libewf-python >= 20131210
libexe-python >= 20160418
libfsntfs-python >= 20151130
libfvde-python >= 20160719
libfwnt-python >= 20160418
libfwsi-python >= 20150606
liblnk-python >= 20150830
libmsiecf-python >= 20150314
libolecf-python >= 20151223
libqcow-python >= 20131204
libregf-python >= 20150315
libscca-python >= 20161031
libsigscan-python >= 20150627
libsmdev-python >= 20140529
libsmraw-python >= 20140612
libvhdi-python >= 20131210
libvmdk-python >= 20140421
libvshadow-python >= 20160109
libvslvm-python >= 20160109
libwrc-python >= 20160419
mock
pbr >= 1.10.0
pefile >= 1.2.10-139
pip >= 7.0.0
```

```
psutil >= 1.2.1
pycrypto >= 2.6.0
pyparsing >= 2.0.3
pysqlite >=2.8.3
future >=0.15.2
pytest
pytsk3 >= 20160721
pytz
PyYAML >= 3.10
pyzmq >= 2.1.11
requests >= 2.2.1
six >= 1.1.0
XlsxWriter >= 0.9.3
yara-python >= 3.4.0
plaso
```

Appendix J – Plaso filterfile

```
/\$MFT
/\$LogFile
/\$Extend/$UsnJrnl
{systemroot}/system32/config/.+
{systemroot}/winevt/.+evtx
{systemroot}/prefetch/.+
/(Users|Documents And Settings)/.+/NTUSER.DAT
```