

The Pennsylvania State University
The Graduate School
College of Information Sciences and Technology

**SECURE ACQUISITION OF DIGITAL EVIDENCE
FROM VMWARE ESXI HYPERVISORS**

A Thesis in
Information Sciences and Technology
by
Matthew Joseph Tentilucci

© 2015 Matthew Joseph Tentilucci

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2015

The thesis of Matthew Joseph Tentilucci was reviewed and approved* by the following:

Anna Squicciarini

Assistant Professor of Information Sciences and Technology

Thesis Advisor

Gerald Santoro

Senior Lecturer, Information Sciences and Technology

Assistant Professor of Communication Arts and Sciences

Dinghao Wu

Assistant Professor of Information Sciences and Technology

Carleen Maitland

Associate Professor of Information Sciences and Technology

Affiliate Professor, The School of International Affairs

Interim Associate Dean for Undergraduate and Graduate Studies

*Signatures are on file in the Graduate School

ABSTRACT

The use of computer virtualization technologies has rapidly grown since the early 2000's. Factors driving this growth include the ever-increasing utilization of cloud computing as well as benefits to consolidating physical hardware within a data center. In addition to the growth of virtualization technologies, computer security incidents are also increasing. However, researchers have drawn attention to the problem that many of the traditional computer forensics tools and investigation techniques cannot be used to gather and analyze digital evidence obtained from virtualization technologies or cloud computing resources.

To solve a part of this problem, this thesis proposes a new open source tool called ESXimager that securely acquires digital evidence from VMware ESXi hypervisors. The tool securely images selected virtual machine files running on VMware ESXi and ensures image integrity through the entire imaging process. Written in Perl and utilizing Tk, the tool makes use of an ESXi server's ability to execute shell commands. Bit-stream copies are created using the dd command, image integrity is verified using the MD5 and SHA1 hashing algorithms, and images are securely transferred to an external imaging machine with SFTP. With a secure image created, a forensics investigator can load the image into a separate computer forensics tool for analysis.

ESXimager's capabilities are validated in a small yet realistic test environment. The tool connects to an ESXi server, creates images of selected virtual machine files, calculates multiple hashes, and securely transfers images to a local imaging machine. In addition, the tool detects if the integrity of an image file is compromised.

With some additional development and testing in a larger environment, this could potentially become the go-to tool used to acquire images from VMware ESXi hypervisors.

Table of Contents

LIST OF FIGURES	vi
LIST OF TABLES	viii
ACKNOWLEDGEMENTS	ix
Chapter 1 Introduction	1
Overview of Computer Forensics	2
Chapter 2 Related Works	3
Chapter 3 Methodology	5
Environment	7
Hardware Specification	9
Chapter 4 Forensic Image Acquisition Tool – ESXimager	12
Graphical User Interface	12
Running the Program for the First Time	13
Creating a New Case	15
Connection to an ESXi Server	16
Selecting Files to be Acquired	16
Creating an Image with DD	19
Transferring Image with SFTP	20
Verifying Image Integrity	21
Additional Features	22
Directory Tree and File List	22
View File Information	23
Verify Integrity	25
Under the Hood	26
Finding Virtual Machines on the ESXi Server	26
Suspending Virtual Machines	27
Un-suspending Virtual Machines	29
Calculating MD5 and SHA1 Hashes	31
Logging	34
Chapter 5 Scenarios and Results	36

Scenarios.....	36
Scenario 1	36
Scenario 2	36
Scenario 3	37
Results	37
Scenario 1	37
Scenario 2	38
Scenario 3	40
Chapter 6 Limitations and Future Work	47
References	50
Appendix: A Source Code	53
ESXimager2.9A.pl	53

LIST OF FIGURES

Figure 1 – Logical topology of test environment	8
Figure 2 - Physical topology of test environment	9
Figure 3 - View of ESXimager program when first opened	13
Figure 4 - Information message, no configuration file found	14
Figure 5 - Window that allows user to create a configuration file	14
Figure 6 - Sample configuration file	14
Figure 7 - Create new case window	15
Figure 8 - Banner displaying the current open case	15
Figure 9 - Interface to enter user credentials to connect to an ESXi server	16
Figure 10 - Displays all virtual machines running on the connected ESXi server	17
Figure 11 - Names of virtual machines as seen in the vSphere client interface	17
Figure 12 - Program asking if the selected virtual machine(s) should be suspended before continuing	18
Figure 13 - Informational message box providing feedback to user	18
Figure 14 - User selects which file(s) to image	19
Figure 15 - Confirm user file selections before imaging	19
Figure 16 - Information message box displayed to use as first round of hashes are being calculated	20
Figure 17 - Information message box displayed while the DD command is being run ...	20
Figure 18 - SFTP transfer progress window	21
Figure 19 - Image verification, final calculation of MD5 and SHA1 hashes	21
Figure 20 - Error message informing user image is corrupt	22
Figure 21 - Imaging process complete	22
Figure 22 - Directory tree and file list window within ESXimager GUI	23
Figure 23 - Selected file in file-listing window	24
Figure 24 - File information menu bar item	24
Figure 25 - File information window	25
Figure 26 - Verify integrity menu bar item	25
Figure 27 - Verifying image integrity	26
Figure 28 - Code to execute find command and parse its output	26
Figure 29 - All local storage and external storage volumes are mounted in /vmfs/volumes/	27

Figure 30 - Equivalent output of the find command received by the program.....	27
Figure 31 - Code used to find a virtual machines world ID and then suspend the virtual machine	29
Figure 32 - Output of /sbin/vmdumper -l command as seen from the command line	29
Figure 33 - Perl code to restart a virtual machine once image acquisition is complete....	30
Figure 34 - Output of vim-cmd vmshvc/getallvms as seen on the command line.....	30
Figure 35 - Sample output of calculating MD5 and SHA1 hashes on an ESXi server	31
Figure 36 - Sample output of calculating MD5 and SHA1 hashes on a Linux desktop ...	31
Figure 37 - Sample output of calculating MD5 and SHA1 hashes on a machine running OS X	32
Figure 38 - Perl code used to calculate MD5 hashes on the local imaging computer. This code differentiates between machines running Linux or OS X	33
Figure 39 - Code snippet from checkOS subroutine showing how special Perl variable \$^O is evaluated.....	33
Figure 40 - Sample ESXimager log file output	35
Figure 41 - logIt Perl subroutine	35
Figure 42 - File information window showing hash history of specified file.....	38
Figure 43 - Error message shown when image is found to be corrupt	38
Figure 44 - iSCSI disk added to ESXi datastore.....	39
Figure 45 - .vmss file being acquired from iSCSI storage.....	39
Figure 46 - Datastore names are symbolic links to ESXi own naming convention	39
Figure 47 - Network traffic graph during image acquisition from iSCSI datastore	40
Figure 48 - ESXimager begins working on target 4 GB file	41
Figure 49 - ESXimager finishes imaging 4 GB file.....	41
Figure 50 - ESXi CPU performance graph for 4 GB file acquisition	42
Figure 51 - Linux imaging CPU utilization during 4 GB file acquisition	43
Figure 52 - Host machine disk queue length during 4 GB imaging test.....	44
Figure 53 - Host machine disk queue over 18-hour period	44
Figure 54 - ESXimager begins imaging 40 GB file.....	45
Figure 55 - ESXimager finishes imaging 40 GB file	45

LIST OF TABLES

Table 1 - Description of files that comprise a virtual machine.....	6
Table 2 – Host Machine hardware specifications	9
Table 3 - FreeNAS hardware specifications	10
Table 4 - CentOS 6.5 hardware specifications	10
Table 5 - VMware ESXi 5.5 hardware specifications	10
Table 6 - Puppy Linux hardware specifications	10
Table 7 - DSL hardware specifications.....	11
Table 8 - Windows Server 2008 R2 hardware specifications.....	11
Table 9 - Timeline breakdown during 40 GB file imaging	45

ACKNOWLEDGEMENTS

I would first like to thank my advisor, Anna Squicciarini, for your guidance on my thesis and all the advice you have given me over the past two years. In addition, thank you to Gerald Santoro and Dinghao Wu for being part of my committee and for your interest in and support of my project.

Thank you to my parents, Joe and Susan, for all your support through my entire academic career and help shaping me into the person I am today.

Lastly, thank you to my wife, Sarah, for all your love and encouragement over the past two years as I worked towards achieving this goal.

Chapter 1

Introduction

Within computing, virtualization has become an exceedingly popular technology that has revolutionized the operation of data centers all around the world. This virtualization technology allows a physical server to run multiple virtual servers that utilize a set of shared physical hardware. This pooling of computer resources allows companies to optimize their existing server infrastructure and reduce the number of physical servers required for business operations. In most cases this allows the company to save money on Information Technology (IT) related costs. The use of virtualization technologies is not limited to large enterprises; medium and even small enterprises make use of virtualization as well.

The concept of virtualization first appeared in the 1960's and 1970's in mainframe computing, but did not become popular in modern computing until the early 2000's. A company that led this resurgence is VMware. For five years running, Gartner has ranked VMware as a "leader" in its annual "Magic Quadrant for x86 Server Virtualization Infrastructure" [1]. With over 500,000 customers, presence in 100% of Fortune 100 companies, and revenues of \$5.21 billion in 2013, VMware is a massive supplier of virtualization software [2]. Virtualization has matured, becoming a popular and an almost necessary technology within IT operations. Gartner also estimates that "at least 70% of x86 server workloads are virtualized" [1].

While virtualization has positively benefited the overall IT community, a problem that continues to plague businesses and IT operations are computer security incidents and data breaches. In the past few years there have been numerous high profile security breaches at companies such as Target, TJX, Living Social, and Sony resulting in the disclosure of personal

information and intellectual property. In order to respond to these incidents, companies will utilize computer forensics experts to determine how the attack was executed, what information was stolen, and if the attackers still have access to the network.

There are numerous tools dedicated to conducting digital forensics that are both open source and proprietary. Despite the number of forensic tools available, a specific niche has yet to be filled that gives forensics investigators the ability to securely acquire digital forensic evidence of a virtual machine running on top of a virtualization platform.

Overview of Computer Forensics

Nelson et. al. defines that “computer forensics involves obtaining and analyzing digital information for use as evidence in civil, criminal, or administrative cases.” Obtaining digital evidence usually involves creating a bit-stream copy, commonly referred to as an image, of the original target device [3]. The bit-stream copy will create an exact copy of the digital information stored on the original device. To validate the images integrity, hashing algorithms are used. Utilizing the content of a file as input, a hashing algorithm calculates a value unique to that file. If any piece of the file is changed, added, or removed the hash algorithm will calculate a different value. This is useful when verifying that the image file is an exact copy of the original device in which case the two will have the same hash value. Common hashing algorithms used to verify image integrity include Message Digest (MD5) and Secure Hash Algorithm (SHA1).

Once an image is created and its integrity verified, the image is analyzed using one of the many open source or proprietary computer forensics tools. Popular tools include EnCase, Access Data’s Forensic Toolkit (FTK), and The Sleuth Kit (TSK) [4][5][6].

Chapter 2

Related Works

In a way, the growing use of cloud computing and increasing use of cloud technologies is driving the growth of virtualization technologies like VMware. With cloud computing on the rise, there is still a need to be able to conduct digital forensics investigations on virtual machines or appliances that exist in the cloud [7]. A number of researchers have drawn attention to the fact that not much research has been done in the area of cloud forensics and question if traditional forensics tools and methods can be used to conduct forensics on the cloud [7][8][9]. Delport et. al. focused on methods of isolating a cloud instance targeted for investigation in order to preserve potential evidence, much like a physical crime scene. Work was done to determine if existing digital forensics tools and acquisition methods could work to perform cloud forensics [9][10]. Urias et. al. determined that many tools are not designed to deal with the complex and fluid structure of virtualization technologies utilized in cloud environments such as the pooling of CPU, memory, and storage resources that could potentially be spread across many different physical sets of hardware. Atkison and Cruz explained what tools could be used to acquire and analyze digital forensic images from virtual machines but pointed out new tools need to be created to fill the specific need of conducting digital forensics on virtual infrastructure. Martini and Choo developed a six-step process to collect digital evidence from a cloud platform, utilizing VMware vCloud as a case study. In addition, a proof-of-concept program was created that made use of vClouds REST (Representational State Transfer) API (Application Programming Interface) to acquire digital forensics information following their proposed process [8].

Research has also been done to evaluate using a virtual environment to conduct forensics analysis. After acquiring a digital forensics image from a suspect machines hard drive, it is converted into a virtual machine allowing an investigator to boot the machine and perform digital forensics without affecting the original evidence [10].

Work specifically related to analyzing a virtual machine has been done as well [11]. Hirwani securely acquired the virtual hard disk file and corresponding snapshots from a VMware virtual machine. After acquiring these digital forensics images, they were analyzed by a program developed by Hirwani that compared the snapshot files to determine what files had been created, deleted, or modified.

Chapter 3

Methodology

With the ever-increasing use of virtualization technologies, the majority being VMware products, and the swell in computer security breaches, there is a need to analyze these systems. When a virtual machine is involved in a security incident, a forensics investigator will need to create an exact copy of the virtual machine in order to conduct a proper investigation. Even if the virtual machine is providing a mission critical service, the virtual machine can be quickly suspended to preserve its state while an image is created or a snapshot taken.

In order to create an image of a virtual machine, a forensics investigator will need to manually enter a series of commands and transfer the image from the virtualization server to another storage medium. This process and sequence of commands to create a forensically sound image from a VMware ESXi hypervisor was detailed in a SANS blog article in 2010 [12]. SANS is a private company that offers cyber security related training and certifications. However, manually completing this task can lead to mistakes or human errors. In order for digital evidence to be admissible in court, carefully documented steps must be taken to ensure digital evidence is not altered. While manually entering commands is an acceptable approach, utilizing a program to create a digital forensic image is more reliable and can provide more detailed information about the imaging process. Therefore, the goal of this project was to create a free, open source piece of software to securely create digital forensic images from VMware ESXi hypervisors. The tool created was named ESXimager.

VMware ESXi hypervisors allow multiple virtual machines to run on top of a single set of physical hardware. By utilizing technologies like VMware ESXi, companies can consolidate

their physical hardware to better utilize their computing resources, saving on power, space, management costs and more.

With companies utilizing virtual machines run on top of a platform like VMware ESXi, most traditional digital forensics methods no longer work. A popular digital forensic method is to acquire an image or a bit-by-bit copy of a computer's hard drive in order to conduct a digital forensics investigation. However, if technology like VMware ESXi is in place, a forensics investigator cannot necessarily pull the hard drives out of the ESXi server and create a forensic image. The actual files for the virtual machine may be located on a Storage Area Network (SAN) or Network Attached Storage (NAS) device or it may spread across multiple physical disks within the ESXi server's local storage.

Virtual machines that run on a VMware ESXi server have their hardware (bios, memory, disk) virtualized. These virtual pieces of hardware are stored as a set of files on the ESXi server. The table below lists all the components of a VMware virtual machine [13].

File Extension	Purpose
.nvram	Stores virtual machines BIOS information
.vmdk	Contains contents of virtual machines virtual hard drive
.vmem	Backup of virtual machines memory
.vmsd	Metadata about snapshots
.vmsn	Snapshot state file
.vmss	Stores virtual machines suspended state
.vmx	Contains important virtual machine configuration data

Table 1 - Description of files that comprise a virtual machine

This method of storing a virtual machine's hardware as a set of files makes it possible to acquire a bit-by-bit copy of the virtual machine's hard drive.

The program created for this project is not only able to securely acquire a bit-by-bit copy of the virtual machines virtual hard drive file but also any other file that makes up a virtual

machine. The program creates a bit-by-bit copy utilizing the Linux `dd` command and the output file created is in a raw format, commonly denoted as such by having the `.dd` file extension. With a secure bit level copy of the virtual machines file(s) now acquired, a forensics investigator can take the raw `.dd` file and open it in another digital forensics tool, such as EnCase or Sleuth Kit, to analyze it [4][6].

In order to test the tools ability to securely create these digital forensics images, a test environment was created and a number of experiments were performed to verify that the program is able to execute the functions mentioned above.

Environment

In order to simulate the use of the ESXi forensics imager software, a test environment was created to represent a small business's use of an ESXi server and a scaled down environment of a medium to large enterprise. The test environment includes a VMware ESXi 5.5 server running two Linux virtual machines and one Windows virtual machine, a FreeNAS server used for iSCSI network storage, and a CentOS 5.5 machine used for running the ESXimager software. Logically, all of the machines are connected to the same Ethernet network, see Figure 1.

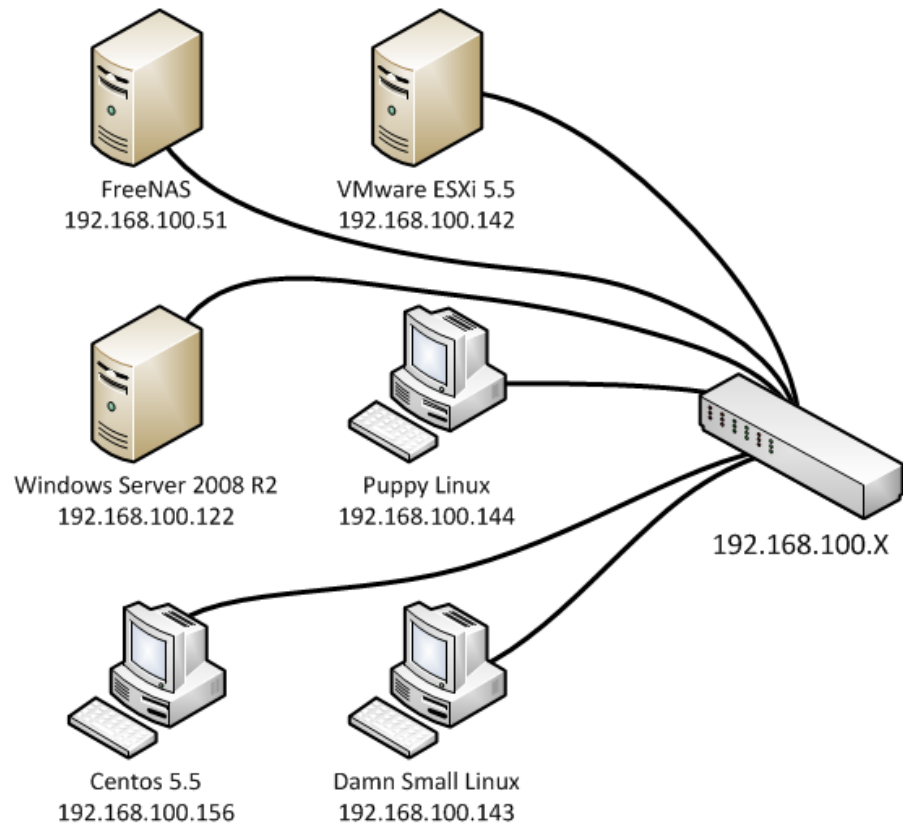


Figure 1 – Logical topology of test environment

However, the physical topology of the network is much more complex. Only two physical computers are present in the test environment, a desktop computer and the FreeNAS file server. The desktop computer hosts the other five computers in the environment in the form of virtual machines. As seen in Figure 2, the desktop computer is running VMware Workstation 9 and has two virtual machines, CentOS 6.5 and VMware ESXi 5.5. Then within the VMware ESXi 5.5 server, three more virtual machines are being run, Puppy Linux, Damn Small Linux, and Windows Server 2008 R2. These five virtual machines share the desktop computer's Ethernet adapter, which allows each virtual machine to logically appear on the same network.

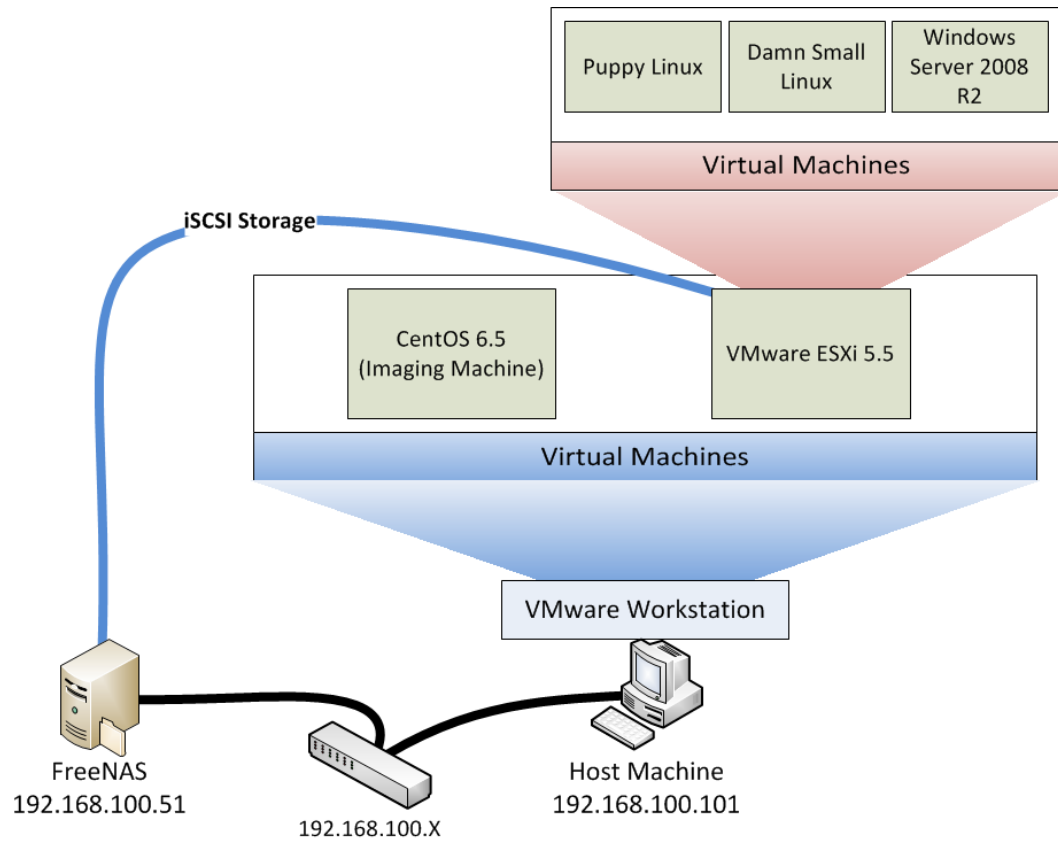


Figure 2 - Physical topology of test environment

Hardware Specification

Each physical and virtual machine has specific hardware resources available for use.

Name	Host Machine
Operating System	Windows 7
CPU	Hexa-Core 3.3 GHz
RAM	16 GB
Storage	1 TB (2 x 500 GB RAID0)

Table 2 – Host Machine hardware specifications

Name	FreeNAS
Operating System	FreeBSD
CPU	Quad-Core 3.4 GHz
RAM	8 GB
Storage	1 TB

Table 3 - FreeNAS hardware specifications

Name	CentOS 6.5
Operating System	CentOS 6.5
CPU	2 vCPU 3.3 GHz
RAM	2 GB
Storage	100 GB

Table 4 - CentOS 6.5 hardware specifications

Name	VMware ESXi 5.5
Operating System	VMware ESXi 5.5
CPU	2 vCPU 3.3 GHz
RAM	4 GB
Storage	140 GB Local Storage, 100 GB iSCSI storage

Table 5 - VMware ESXi 5.5 hardware specifications

Name	Puppy Linux
Operating System	Puppy Linux
CPU	1 vCPU 3.3 GHz
RAM	1 GB
Storage	4 GB

Table 6 - Puppy Linux hardware specifications

Name	DSL
Operating System	Damn Small Linux
CPU	1 vCPU 3.3 GHz
RAM	1 GB
Storage	5 GB

Table 7 - DSL hardware specifications

Name	Windows Server 2k8 R2
Operating System	Windows Server 2008 R2
CPU	1 vCPU 3.3 GHz
RAM	2 GB
Storage	40 GB Local Storage, 40 GB iSCSI Storage

Table 8 - Windows Server 2008 R2 hardware specifications

Chapter 4

Forensic Image Acquisition Tool – ESXimager

To facilitate and simplify the acquisition of digital evidence from VMware ESXi servers, there was a need to streamline the process and remove the likelihood of human errors during evidence collection. A program, aptly named ESXimager, was created utilizing the Perl programming language. ESXimager gives a forensics investigator a simple yet secure way of acquiring digital evidence from VMware ESXi servers. This section is broken into three parts: first, how to acquire an image through the Graphical User Interface (GUI), second description of additional features available through the GUI, third highlights of specific pieces of code to show how the program works under the hood.

Graphical User Interface

Important to any program is the Graphical User Interface (GUI). While there are many programs that forgo the use of a GUI in favor of a command line driven approach, adding a GUI front-end makes the program more user friendly, allows for information to be easily and clearly displayed, and allows the user to easily interact with the program's many features. ESXimager utilizes the Tk widget toolkit to create the GUI elements for the program. The look and feel of the program when it is first run can be seen in Figure 3.



Figure 3 - View of ESXimager program when first opened

The ESXimager GUI has a number of different features; the purpose and functionality will be explained below. The following sub-sections are laid out and explained in the order a user would logically work through the program to acquire an image.

Running the Program for the First Time

In order for ESXimager to run properly, there are a number of parameters that must first be set before any digital forensic evidence can be acquired. When the program is first executed by the user, it looks for a configuration file which it expects to exist in the user's home directory, for example `/home/matt/ESXimager/ESXimager.cfg`. If the `ESXimager.cfg` file cannot be located in the users home directory, the program assumes this is the first time the user is executing the program. The program will display an information message to the user and then prompt them to another window to create a configuration file, see Figure 4 and Figure 5, respectively.



Figure 4 - Information message, no configuration file found

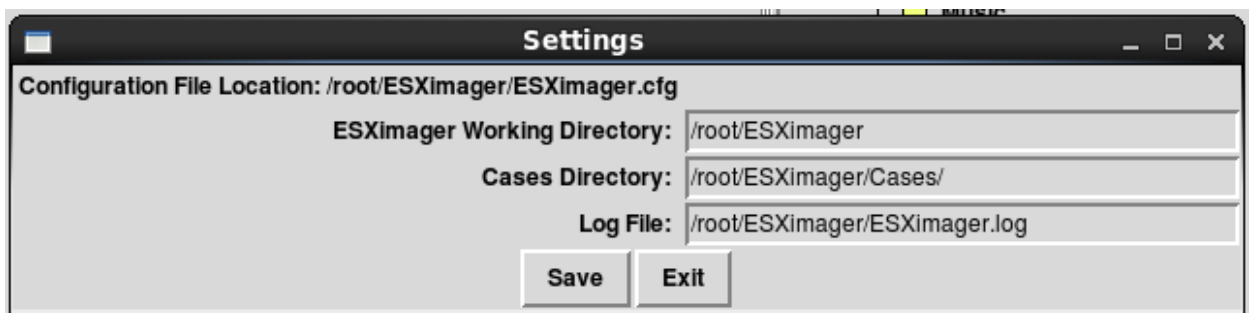


Figure 5 - Window that allows user to create a configuration file

The configuration file stores three important pieces of information, the location of the program's overall working directory, the location to store cases, and the location for the overall program log file. The ESXimager configuration file window pre-populates with default recommended values, but the user can manually specify these values. If left with the default values, the ESXimager working directory contains the `ESXimager.cfg` file, the cases directory, and the overall program log file. These values are saved to the configuration file and loaded each subsequent time the program is run. Because there are only three configuration parameters to keep track of between program executions, it was a design decision to store these parameters in a flat file format, a sample configuration file can be seen in Figure 6.

```
WorkingDir=/home/matt/ESXimager
CaseDir=/home/matt/ESXimager/Cases/
LogFile=/home/matt/ESXimager/ESXimager.log
```

Figure 6 - Sample configuration file

Creating a New Case

A common theme used by other computer forensics programs and within the computer forensics community is to organize different computer investigations by cases. ESXimager allows the user to create a new case in which any forensics images acquired will be stored. This allows different digital forensics investigations to be separated both physically in terms of where the images are stored and logically for ease of use purposes.

Before the user can create any forensic images, a case must be opened in order to determine where the forensic images will be stored. To create a new case, the user can use the menu bar to navigate to File → New Case after which a window will popup asking the user to specify a case name, see Figure 7. After specifying a case name, the program will create a new folder in the case directory specified in the `ESXimager.cfg` configuration file. To avoid any problems with the name of the case folder, any whitespace is removed from the case name. Using the example in Figure 7, a new case folder will be created in `/home/matt/ESXimager/Cases/` and the folder will be named `NewTestCase`. Once the new case has been created, the program sets this as the current open case and changes a banner at the top of the program window to reflect the name and location of the currently open case, see Figure 8.

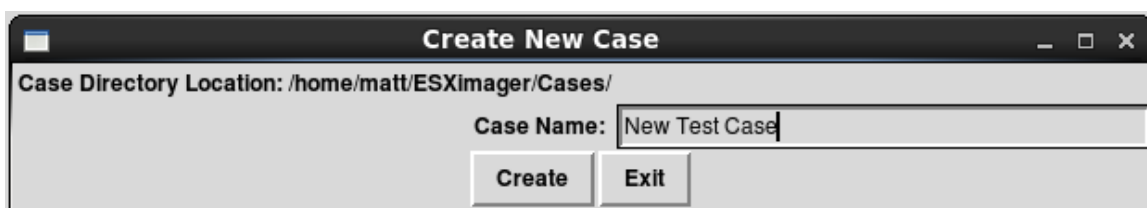


Figure 7 - Create new case window

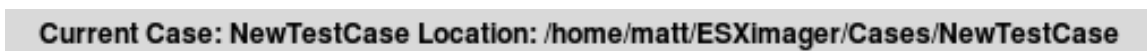


Figure 8 - Banner displaying the current open case

Connection to an ESXi Server

In order to acquire digital forensic images the user must connect to an ESXi server. Before the ESXimager program can connect to the ESXi server, SSH access must be enabled which is disabled by default. It is a potential security vulnerability to enable SSH access to an ESXi server so once the image acquisition process is complete, SSH should be disabled. The ESXimager program provides an easy interface that allows a user to specify the IP address, username, and password required to connect to the ESXi server, see Figure 9.



The image shows a graphical user interface for connecting to an ESXi server. It consists of four input fields and a button. The first field is labeled 'Server IP' and contains the text '192.168.100.142'. The second field is labeled 'Username' and contains the text 'root'. The third field is labeled 'Password' and contains a series of asterisks '*****'. The fourth element is a button labeled 'Connect'.

Server IP	192.168.100.142	Username	root	Password	*****	Connect
-----------	-----------------	----------	------	----------	-------	---------

Figure 9 - Interface to enter user credentials to connect to an ESXi server

Until a connection is made to an ESXi server, a large portion of the main program window remains blank and inactive. Once successfully connected, the main window automatically populates with any virtual machines found on the ESXi server.

Selecting Files to be Acquired

Once connected to an ESXi server, the main window will populate with all virtual machines found on the given ESXi server. From this point the program guides the user through a three-step process to select which files they want to acquire. First, the virtual machines running on the ESXi server the user has connected to are shown in the main program window, see Figure 10.

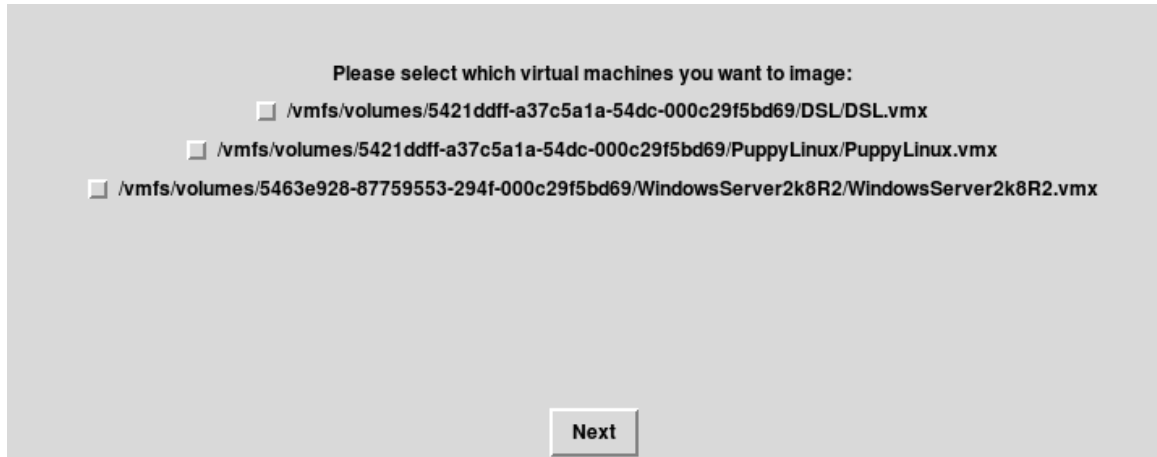


Figure 10 - Displays all virtual machines running on the connected ESXi server

The user is then able to select which virtual machine's files they want to view. For each virtual machine listed, the absolute path to that virtual machine's storage directory is displayed. Names displayed correspond to the virtual machines names shown within the VMware vSphere client interface. For example, in Figure 10 there is a virtual machine found that has a name of PuppyLinux, this name corresponds to the name of the virtual machine as seen in the VMware vSphere client interface, see Figure 11.

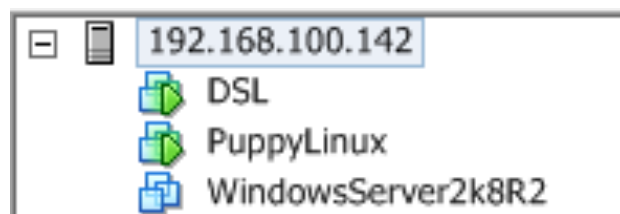


Figure 11 - Names of virtual machines as seen in the vSphere client interface

At this point the program determines if the virtual machine(s) selected by the user are currently running. Some portions of the virtual machine will be inaccessible and there may be some inconsistent results if the virtual machine is not first suspended before imaging. The user is asked if they wish to suspend the selected virtual machine before continuing and it is made clear that suspending the virtual machine is recommended before continuing onto the next step of the imaging process, see Figure 12. If the user does elect to suspend the virtual machine, a second

window will appear giving the user some feedback that the program is working on suspending the virtual machine, see Figure 13.

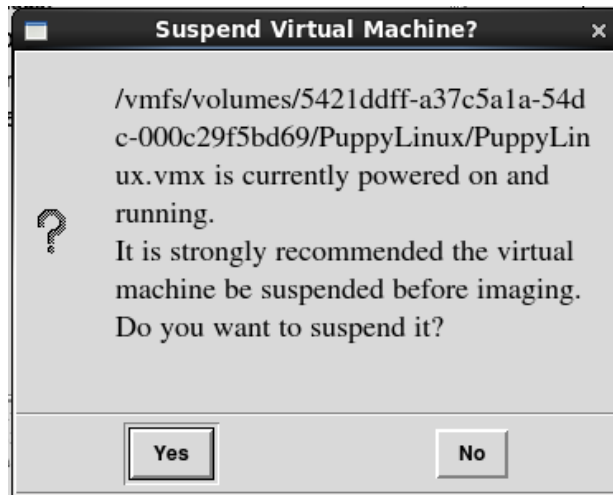


Figure 12 - Program asking if the selected virtual machine(s) should be suspended before continuing



Figure 13 - Informational message box providing feedback to user

Second, depending on which virtual machines the user selected from step one, the related files are shown and the user can select one or multiple files to image and acquire to their local machine, see Figure 14.



Figure 14 - User selects which file(s) to image

Last, the program confirms to the user which files they wish to image, see Figure 15.



Figure 15 - Confirm user file selections before imaging

Creating an Image with DD

After confirming the file selection to image, the ESXImager program now executes a number of instructions and commands that will securely image the selected files without further user intervention. It is at this point the first MD5 and SHA1 hashes are calculated for the current file being imaged. While the hashes are being calculated, the user will have no control of the

program, so an information message box is displayed informing the user what the program is currently doing, see Figure 16.

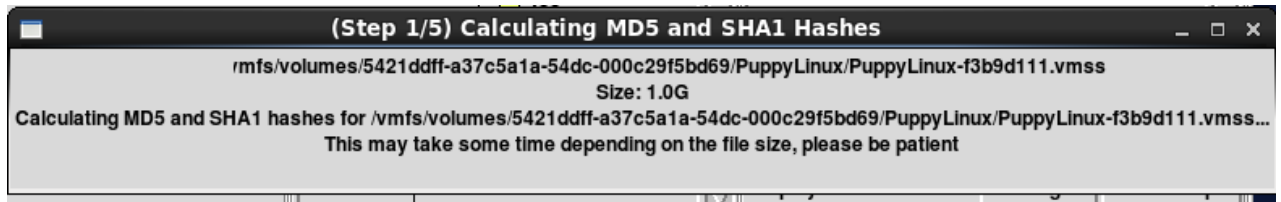


Figure 16 - Information message box displayed to use as first round of hashes are being calculated

After the first round of hashes is calculated, the program proceeds to create a bit-by-bit copy of the file using the Linux dd command. The image created by the dd command is initially stored in the same directory as the original file until it is copied and removed in later steps. The user has no control of the program at this stage, and an information window is displayed informing the user the current status of the program, see Figure 17.



Figure 17 - Information message box displayed while the DD command is being run

Finally, once the dd command is finished running, a second round of MD5 and SHA1 hashes are calculated. This time the image file created by dd is hashed in order to ensure the image does not differ from the original.

Transferring Image with SFTP

With the second round of hashes calculated, the image file is ready to be transferred from the ESXi server to the imaging machine where the ESXimager program is running. Secure File Transfer Protocol (SFTP) is used to move the image file off the ESXi server mainly because the image files are encrypted when being sent across the network. Also, the existence of the

Net::SFTP::Foreign Perl module simplified this process, providing a number of features that enabled the easy connection and transfer of files via SFTP. Like the previous steps of the imaging process, the SFTP transfer process does not require nor does it allow for any user interaction. Thanks to the features part of the Net::SFTP::Foreign Perl module, an informative transfer status window is displayed to the user that reflects the progress of the SFTP file transfer, see Figure 18.



Figure 18 - SFTP transfer progress window

Once the SFTP transfer is complete, a third set of MD5 and SHA1 hashes are calculated to ensure the image has not deviated from the original.

Verifying Image Integrity

The last step of the imaging process performed by the ESXImager program is a final integrity check of all the acquired image files. A fourth and final set of MD5 and SHA1 hashes of the image files now residing locally on the imaging machine are calculated and compared to the previous set of three MD5 and SHA1 hashes, see Figure 19. If any of the hashes calculated for a given image file differ from the previously calculated hashes, the program informs and warns the user that the image file is corrupt, see Figure 20. If no problems are found with the final image verification the program informs the user the imaging process is complete, see Figure 21.

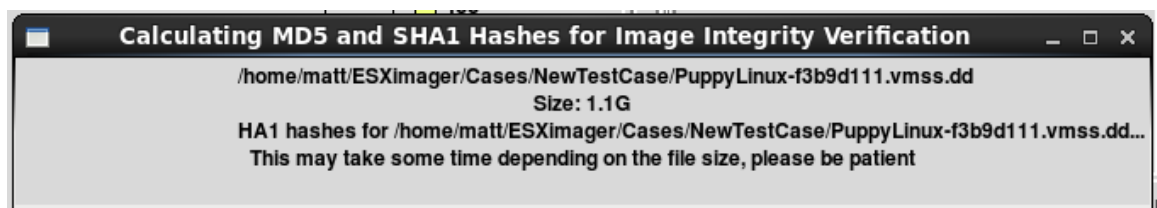


Figure 19 - Image verification, final calculation of MD5 and SHA1 hashes

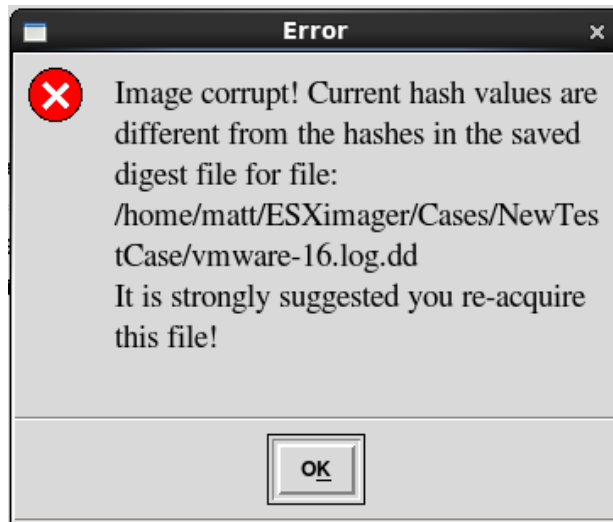


Figure 20 - Error message informing user image is corrupt



Figure 21 - Imaging process complete

Additional Features

In addition to the ESXimagers functionality to create an image as explained in the previous section, there are some additional features within the GUI that can be utilized outside the imaging process.

Directory Tree and File List

Within the main ESXimager program GUI, there are two additional windows that show a directory tree and directory file listing, see Figure 22. When the program opens, the directory tree expands to show all of the cases in the case directory. Once a case is opened or created, the file list window refreshes to show a file listing of the currently open case directory. As seen in Figure

22, the current case open exists in /home/matt/ESXimager/Cases/Scenario3P2.1, so the file list window shows a listing of that directory accordingly. If a user clicks on a directory in the directory tree window, the file list window will automatically update to show the files in the selected directory.

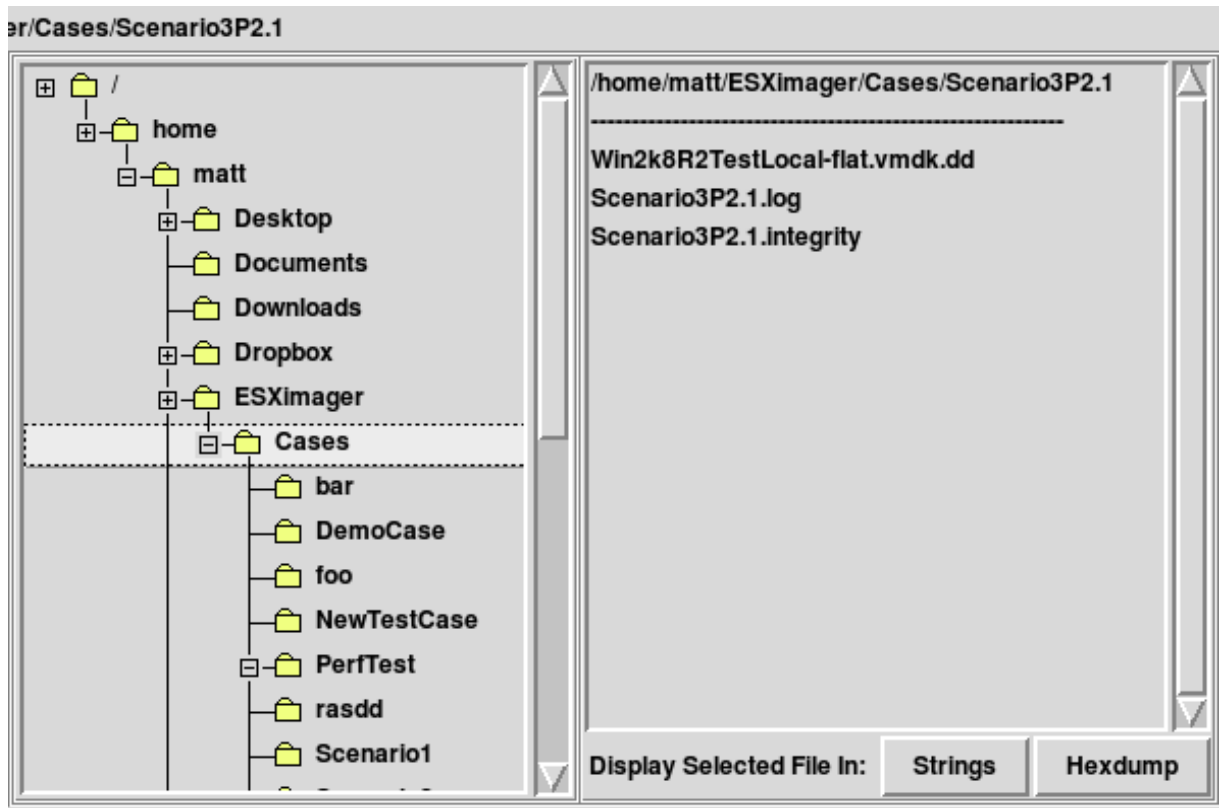


Figure 22 - Directory tree and file list window within ESXimager GUI

The user is not able to create or delete files from this interface, nor are they able to open a case by clicking on a directory in the directory tree. These windows give the user an easy way to see what images have been acquired to the currently open case, or view images acquired in previous cases.

View File Information

Through ESXimager's entire imaging process, four sets of hashes are calculated at various stages. These hash values are stored to ensure the integrity of the image files can be

verified. The program provides a way to allow a user to easily view all the hash values calculated for a specific file. Utilizing the file-listing window explained in the previous section, a user can select an image file, see Figure 23, then using the menu bars at the top navigate to View → File Information, see Figure 24.

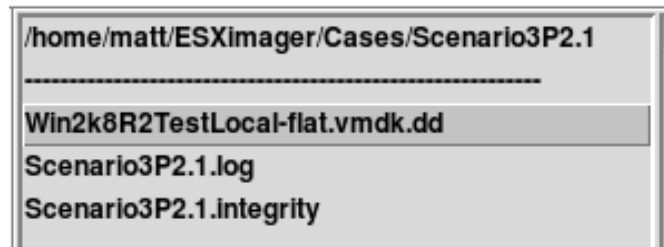


Figure 23 - Selected file in file-listing window

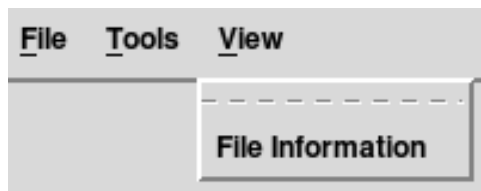


Figure 24 - File information menu bar item

With these two options selected, a new window appears showing some basic file information such as the name and size, but more importantly all of the hash values calculated for that file, see Figure 25. The hash value history includes a timestamp of when the value was calculated, what point during the imaging process the value was calculated, and the actual hash values. This allows a user to manually verify that the hash values have not changed and the integrity of the image is intact.

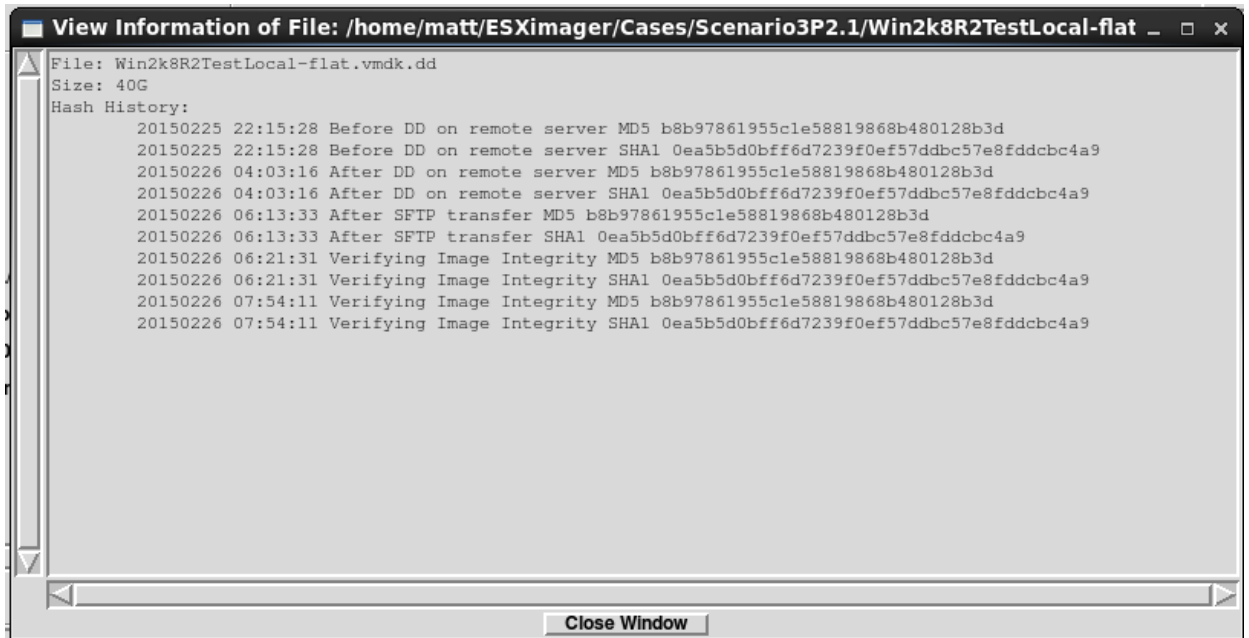


Figure 25 - File information window

Verify Integrity

While the ESXimager program calculates a number of hash values during the imaging process, the user may want to periodically check to ensure hash values of the images in a particular case have not changed. To re-validate the integrity of all the images within a case, the user can utilize the verify integrity tool. This can be done by opening a case and navigating to Tools → Verify Integrity, see Figure 26. When this option is selected, ESXimager will calculate both the MD5 and SHA1 hash values for each image file in an open cases directory and compare them to all previously calculated hash values for that image file, see Figure 27. If different hash values are calculated, ESXimager alerts the user to the problem. In addition, the new hash values are stored to be used for future comparisons, if necessary.

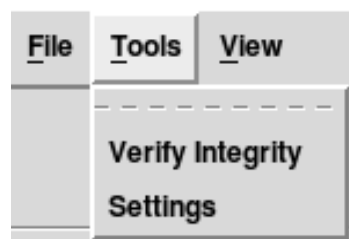


Figure 26 - Verify integrity menu bar item

```

20150226 07:45:48 [info] (Scenario3P2.1) Calculating md5 hash of local file this may take a while be patient...
20150226 07:49:12 [info] (Scenario3P2.1) Done calculating md5 hash of local file.
20150226 07:49:12 [info] (Scenario3P2.1) Calculating sha1 hash of local file this may take a while be patient...
20150226 07:54:11 [info] (Scenario3P2.1) Done calculating sha1 hash of local file.
20150226 07:54:11 [info] (Scenario3P2.1) Hashes of /home/matt/ESXimager/Cases/Scenario3P2.1/Win2k8R2TestLocal-flat.vmdk.dd:
MD5: b8b97861955c1e58819868b480128b3d
SHA1: 0ea5b5d0bfff6d7239f0ef57ddbc57e8fddcbc4a9
20150226 07:54:11 [info] (Scenario3P2.1) No integrity problems found for file: /home/matt/ESXimager/Cases/Scenario3P2.1/Win2k8R2TestLocal-flat.vmdk.dd
20150226 07:54:11 [info] (Scenario3P2.1) Writing to integrity file
20150226 07:54:11 [info] (Scenario3P2.1) Done performing image integrity verification

```

Figure 27 - Verifying image integrity

Under the Hood

Some unique portions of code had to be written to solve a number of interesting problems to fulfill all the goals of this program. In addition, implementing some of the more advanced features introduced some daunting coding challenges, which require further explanation.

Finding Virtual Machines on the ESXi Server

A very important part of the ESXimager program is its ability to locate all of the virtual machines stored on an ESXi server when successfully connected. The goal of this part of the program is to determine which virtual machines reside on this ESXi server and the absolute path to where those virtual machines are stored. To locate these virtual machines, the `find` command is used because of its ease of use and the quality of the information returned. After successfully connecting to the ESXi server, the program executes a `find` command on the ESXi server and then parses the output to obtain the name of each virtual machine and the absolute path to where they are stored, see Figure 28.

```

my $stdout = $ssh->capture("find $vmstore -name \"*.vmtx\"");
@vmxFound = split(/\s+/, $stdout);

```

Figure 28 - Code to execute find command and parse its output

The `$vmstore` variable is set to `/vmfs/volumes/` because any local storage or NAS storage will be mounted to a location in that directory, see Figure 29.

```

/etc/vmware/hostd # esxcli storage filesystem list
Mount Point                                     Volume Name
-----
/vmfs/volumes/5421ddff-a37c5a1a-54dc-000c29f5bd69  datastore1
/vmfs/volumes/5463e928-87759553-294f-000c29f5bd69  iSCSIDatastore
/vmfs/volumes/5421ddff-1c795409-5d5e-000c29f5bd69
/vmfs/volumes/698ed378-3180051e-cf52-ff610ea612d0
/vmfs/volumes/f723437c-32d4394f-baf9-d1e30aa134ca
/vmfs/volumes/5421ddf8-adcda020-9dd5-000c29f5bd69
/etc/vmware/hostd #

```

Figure 29 - All local storage and external storage volumes are mounted in /vmfs/volumes/

The `find` command is specifically looking for files in the `/vmfs/volumes/` directory that end with a `.vmx` extension. One `.vmx` file is required for each virtual machine as it contains important parameters and specific configuration information for a virtual machine. In addition, the `.vmx` file for a given virtual machine is stored in the same location as all the other virtual machine's files. Given the uniqueness of this file type and its storage location, using the `find` command not only determines what virtual machines are running on the ESXi server but also returns information on the virtual machine's storage location as well. The output captured by the `$stdout` variable is equivalent to what is seen in Figure 30.

```

/etc/vmware/hostd # find /vmfs/volumes/ -name *.vmx
/vmfs/volumes/5421ddff-a37c5a1a-54dc-000c29f5bd69/DSL/DSL.vmx
/vmfs/volumes/5421ddff-a37c5a1a-54dc-000c29f5bd69/PuppyLinux/PuppyLinux.vmx
/vmfs/volumes/5463e928-87759553-294f-000c29f5bd69/WindowsServer2k8R2/WindowsServer2k8R2.vmx

```

Figure 30 - Equivalent output of the `find` command received by the program

Suspending Virtual Machines

Suspending a virtual machine freezes the machine's current state. It is particularly important to first suspend a virtual machine before imaging one of its files because of the method the ESXImager program acquires an image. While it is possible to perform an image acquisition while the virtual machine is still running, some files will not be accessible and there is the possibility that inconsistencies can develop between the acquired image and the original. For these reasons, if the ESXImager program detects that a virtual machine to be imaged by the user

is currently running, it prompts the user to suspend the virtual machine. ESXimager needs one important piece of information in order to suspend a virtual machine and that is the virtual machines World ID Number. Each virtual machine running on an ESXi server has a unique World ID Number.

Within the code, the process of finding the World ID Number for a particular virtual machine involves running a command on the ESXi server then capturing and parsing the output to find the correct World ID. As seen in Figure 31, the code parses through the output of the `/sbin/vmdumper -l` command looking for a line in the output that matches the `$vmxpath` variable which contains the absolute path to the `.vmx` file, otherwise known as the configuration file for the virtual machine which is to be suspended. Once the World ID of the virtual machine to be suspended is found, the program executes the following command on the ESXi server to suspend the virtual machine, `/sbin/vmdumper $VMwidSplit[1] suspend_vm`. This is the same command used before to find the World ID but is now being passed two different arguments, the world ID, contained within the `$VMwidSplit[1]` variable, and `suspend_vm`.

```
#suspends a given VM, expects absolute path to vm's .vmx path
sub suspendVM
{
    my $vmxPath = $_[0];
    my $stdout = $ssh->capture('/sbin/vmdumper -l');
    my @lines = split(/\n/, $stdout);
    foreach(@lines)
    {
        my @lineParts = split(/\s+/, $_);
        foreach (@lineParts)
        {
            if ($_ =~ m/$vmxPath/)
            {
                my $VMwid = $lineParts[0];
                $VMwid =~ s/= /g;
            }
        }
    }
}
```

```

my @VMwidSplit = split(/\s+/, $VMwid);
my $stdout = $ssh->capture("/sbin/vmdumper
$VMwidSplit[1] suspend_vm") or warn "remote
command failed " . $ssh->error;
...

```

Figure 31 - Code used to find a virtual machines world ID and then suspend the virtual machine

The equivalent output the program would be parsing from the `/sbin/vmdumper -l` command can be seen in Figure 32. Also shown in the output in Figure 32 is the absolute path to the virtual machine's configuration file on the same line as the virtual machines World ID. Because the ESXimager program already knows the absolute path to the virtual machine's `.vmx` file, Perl can be used to parse the output and match a `.vmx` file to a world ID number.

```

/etc/vmware/hostd # /sbin/vmdumper -l
wid=42508      pid=-1   cfgFile="/vmfs/volumes/5421ddff-a37c5a1a-54dc-000c29f5bd69/DSL/DSL.vmx"  uuid="56 4d e9 93 70 dd
0d 96-da b7 9d d7 b6 92 0d b3" displayName="DSL"          vmxCartelID=42505
wid=1844872    pid=-1   cfgFile="/vmfs/volumes/5421ddff-a37c5a1a-54dc-000c29f5bd69/PuppyLinux/PuppyLinux.vmx"  uuid="5
6 4d 42 2d 0b 0d 86 f3-ea be b0 1a 74 c4 10 c5" displayName="PuppyLinux"          vmxCartelID=1844869
/etc/vmware/hostd #

```

Figure 32 - Output of `/sbin/vmdumper -l` command as seen from the command line

Un-suspending Virtual Machines

Another important feature offered by this program is the ability to un-freeze and power on a virtual machine after it was suspended. While it is important to keep a virtual machine in a suspended state while image acquisition is taking place, once the image acquisition process is complete the virtual machine should be automatically turned back on. Once the imaging process is complete, two commands are executed on the ESXi server, which can be seen in bold in the section of Perl code in Figure 33. The first command executed on the ESXi server and parsed by the Perl program is `vim-cmd vmsvc/getallvms | grep $vmxFile`. This command returns a list of all the virtual machines on the ESXi server, along with another identifier to a virtual machine, - the Virtual Machine ID (VMID). It is important to note that the VMID is different from the World ID used previously to suspend a virtual machine. Once the ESXimager program has parsed the output and obtained the VMID of the virtual machine to power back on, a second

command is executed on the ESXi server, `vim-cmd vmsvc/power.on $parts[0]`. Within this code, the `$parts[0]` variable contains the VMID.

```
#starts a given VM, expects absolute path to .vmx file in
question
sub startVM
{
    my $vmToRestart = $_[0];
    my $vmxFile = getFileName($vmToRestart);
    my $stdout = $ssh->capture("vim-cmd vmsvc/getallvms |grep
    $vmxFile");
    my @lines = split(/\n/, $stdout);
    my $count = @lines;
    #only one .vmx file was matched so we dont need to worry
    about starting the wrong VM
    if ($count == 1)
    {
        my @parts = split(/\s+/, $lines[0]);
        my $stdout = $ssh->capture("vim-cmd vmsvc/power.on
        $parts[0]") or warn "remote command failed " . $ssh-
        >error;
        ...
    }
}
```

Figure 33 - Perl code to restart a virtual machine once image acquisition is complete

Figure 34 contains an equivalent output of the `vim-cmd vmsvc/getallvms` the program would be parsing. The VMID can be seen in the first column and the corresponding .vmx file can be seen in the third column.

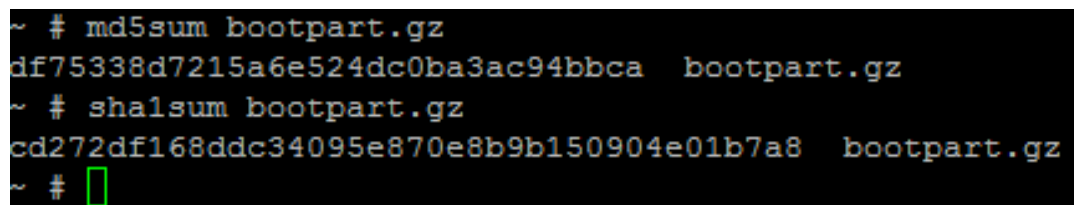
```
/etc/vmware/hostd # vim-cmd /vmsvc/getallvms
Vmid      Name      File      Guest OS
1         DSL      [datastore1] DSL/DSL.vmx      other26xLinuxGuest
2         PuppyLinux [datastore1] PuppyLinux/PuppyLinux.vmx  other26xLinuxGuest
3         WindowsServer2k8R2 [iSCSIDatastore] WindowsServer2k8R2/WindowsServer2k8R2.vmx windows7Server64Guest
/etc/vmware/hostd #
```

Figure 34 - Output of `vim-cmd vmsvc/getallvms` as seen on the command line

Calculating MD5 and SHA1 Hashes

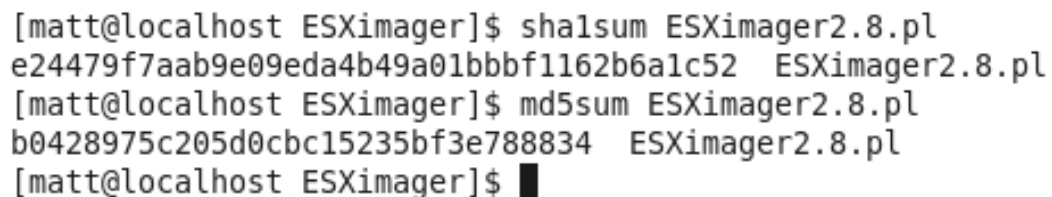
Calculating hashes of the created images is a very important step to ensure the image's integrity. The ESXimager program calculates four sets of MD5 and SHA1 hashes at different stages of the image acquisition process. Hashes are calculated for the original target file before a copy is created. Hashes are calculated again after a copy has been made with the `dd` command, again when the image is transferred to the imaging machine via SFTP, and the last set is calculated at the end of the image acquisition process.

Calculating hashes either on the ESXi server or locally on the imaging machine is relatively straightforward. Because the ESXi server runs a variation of Linux, the same command syntax is used to calculate MD5 and SHA1 hashes regardless if the commands are run on the ESXi server or locally on the imaging machine. In addition, the MD5 and SHA1 commands on an ESXi server or Linux machine have the same formatted output. The output of running the `sha1sum` and `md5sum` commands on an ESXi server and locally on a Linux desktop machine can be seen in Figure 35 and Figure 36 respectively.



```
~ # md5sum bootpart.gz
df75338d7215a6e524dc0ba3ac94bbca  bootpart.gz
~ # sha1sum bootpart.gz
cd272df168ddc34095e870e8b9b150904e01b7a8  bootpart.gz
~ #
```

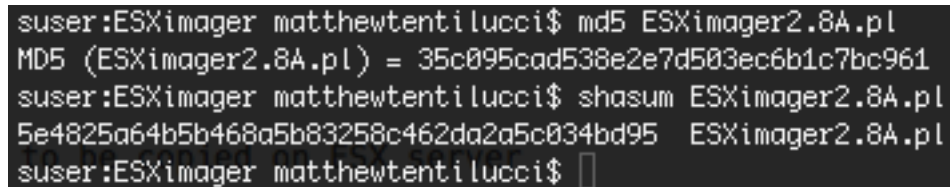
Figure 35 - Sample output of calculating MD5 and SHA1 hashes on an ESXi server



```
[matt@localhost ESXimager]$ sha1sum ESXimager2.8.pl
e24479f7aab9e09eda4b49a01bbbf1162b6a1c52  ESXimager2.8.pl
[matt@localhost ESXimager]$ md5sum ESXimager2.8.pl
b0428975c205d0cbc15235bf3e788834  ESXimager2.8.pl
[matt@localhost ESXimager]$
```

Figure 36 - Sample output of calculating MD5 and SHA1 hashes on a Linux desktop

The ESXimager program can also run on OS X. However, when calculating hashes on an OS X machine, the commands to calculate MD5 and SHA1 hashes are slightly different and produce differently formatted output. The command and output of calculating MD5 and SHA1 hashes on an OS X machine can be seen in Figure 37.

A terminal window with a dark background and light-colored text. The prompt is 'suser:ESXimager matthewtentilucci\$'. The first command is 'md5 ESXimager2.8A.pl', followed by its output: 'MD5 (ESXimager2.8A.pl) = 35c095cad538e2e7d503ec6b1c7bc961'. The second command is 'shasum ESXimager2.8A.pl', followed by its output: '5e4825a64b5b468a5b83258c462da2a5c034bd95 ESXimager2.8A.pl'. The prompt is shown again at the end.

```
suser:ESXimager matthewtentilucci$ md5 ESXimager2.8A.pl
MD5 (ESXimager2.8A.pl) = 35c095cad538e2e7d503ec6b1c7bc961
suser:ESXimager matthewtentilucci$ shasum ESXimager2.8A.pl
5e4825a64b5b468a5b83258c462da2a5c034bd95 ESXimager2.8A.pl
suser:ESXimager matthewtentilucci$
```

Figure 37 - Sample output of calculating MD5 and SHA1 hashes on a machine running OS X

A subroutine had to be added to account for the need to run different variations of the MD5 and SHA1 commands depending on the operating system. A Perl subroutine can be seen in Figure 38 that allows the ESXimager program to run a different MD5 or SHA1 calculation command depending on the type of operating system.

```
#Calculate MD5 hash of local file
sub calculateMD5HashLocal
{
    my $fileToHash = $_[0];
    my $operatingSystem = checkOS();

    logIt("[info] ($currentCaseName) Calculating md5 hash of
    local file this may take a while be patient...", 1, 1, 1);
    my $stdout;
    $stdout = `md5sum $fileToHash` if $operatingSystem == 1;
    $stdout = `md5 $fileToHash` if $operatingSystem == 2;
    my @split = split (/s+/, $stdout);

    #[$#split] gives you the last element of an array
    $stdout = $split[0] if $operatingSystem == 1;
    $stdout = $split[$#split] if $operatingSystem == 2;
}
```

```

        logIt("[info] ($currentCaseName) Done calculating md5 hash
of local file.", 1, 1, 1);
        chomp $stdout;
        return $stdout;
    }

```

Figure 38 - Perl code used to calculate MD5 hashes on the local imaging computer. This code differentiates between machines running Linux or OS X

One of the first steps in this subroutine is to define the variable `$operatingSystem` which holds the value returned by `checkOS`, another Perl subroutine. The `checkOS` subroutine makes use of a special Perl variable called `$^O` which holds the name of the operating system the Perl code is being executed on. A snippet of how the `checkOS` subroutine evaluates the `$^O` variable and returns the current operating system can be seen in Figure 39. The `$^O` variable's value is a string that will contain the word `linux` or `darwin` depending if the Perl code is being executed on a Linux or OS X operating system, respectively. Some string comparison is used to determine if the operating system is Linux or OS X and depending on the result, the `$osValue` is set to 1 or 2 and then returned.

```

my $OS = $^O;
my $osValue;
if($OS eq "linux")
{
    $osValue = 1;
}
#darwin aka osx
elsif($OS eq "darwin")
{
    $osValue = 2;
}
...
return $osValue;

```

Figure 39 - Code snippet from `checkOS` subroutine showing how special Perl variable `$^O` is evaluated

With a value representing the operating system the code is running on now returned by the checkOS subroutine, the program now executes the correct code depending on the operating system being used. Referring back to Figure 38, depending if the \$operatingSystem variable is either 1 or 2 the correct operating system specific MD5 command is run and the correct element of the @split array, containing the hash value, is obtained. The same process occurs when calculating a SHA1 hash on the local machine on which the ESXimager program is running.

Logging

Logging is a very important part of the ESXimager program. Log files allow a forensics investigator to look back at all the steps the program executed during the imaging process and provide a type of paper trail. There are three log files maintained by the ESXimager program: an overall program log file, a debug log file, and a log file for the current open case. Each case has its own log file, which only includes log messages related to that particular case. The overall log file includes any messages written to a case log file, as well as some messages written while the program is initializing. The debug log file includes all the messages from the case and overall log files, in addition to more detailed information about what operations are being executed. The ESXimager program creates log entries in a specific format, Timestamp - Message type (debug, info, warning, error) – Where the message originated (main program or a specific case) – Message. A sample of one of the log files can be seen in Figure 40. To use the first log message in the figure below as an example to explain the convention above, it is an information message, the message came from a function related to acquiring an image so the case name is included (PerfTest), and the message is that the md5 hashing calculation has begun.

```

20150225 08:06:32 [info] (PerfTest) Calculating md5 hash of local file this may take a while be patient...
20150225 08:06:49 [info] (PerfTest) Done calculating md5 hash of local file.
20150225 08:06:49 [info] (PerfTest) MD5 calculation of file /home/matt/ESXimager/Cases/PerfTest/Win2k8R2TestLocal-adca79f5.vmsd took 17 seconds
20150225 08:06:51 [debug] (main) Checking operating system.
20150225 08:06:51 [debug] (main) Operating system is Linux.
20150225 08:06:51 [info] (PerfTest) Calculating sha1 hash of local file this may take a while be patient...
20150225 08:07:07 [info] (PerfTest) Done calculating sha1 hash of local file.
20150225 08:07:07 [info] (PerfTest) SHA1 calculation of file /home/matt/ESXimager/Cases/PerfTest/Win2k8R2TestLocal-adca79f5.vmsd took 16 seconds
20150225 08:07:07 [info] (PerfTest) Hashes of /home/matt/ESXimager/Cases/PerfTest/Win2k8R2TestLocal-adca79f5.vmsd:
MD5: 3d7683bb54e1d58ad9a2cedf5e526867
SHA1: d7b888125c85bacfda8f1c440c75b355e33b14b6

```

Figure 40 - Sample ESXimager log file output

Because there were multiple log files that had to be maintained, many including the same log messages, a simple Perl subroutine was created to write log messages to the appropriate files. The Perl subroutine was aptly named `logIt`, see Figure 41. The subroutine expects four arguments, the first being the message to print to the log files, then three numbers 0 or 1 representing which log files the message should be written to. The three log files are the overall program log file, the case log file, and the console log window in the ESXimager program. Regardless of what arguments are passed, the log message is written to the debug log file.

```

sub logIt
{
    my $lineToPrint = $_[0];
    my $programLogPrint = $_[1];
    my $caseLogPrint = $_[2];
    my $consoleLogPrint = $_[3];

    my $lineToPrint = getLoggingTime() . " " . $lineToPrint .
        "\n";
    print PROGRAMLOGFILE $lineToPrint if $programLogPrint == 1;
    print $currentCaseLog $lineToPrint if $caseLogPrint == 1;
    $consoleLog->insert('end', $lineToPrint) if
        $consoleLogPrint == 1;
    $consoleLog->see('end') if $consoleLogPrint == 1;
    print DEBUGLOGFILE $lineToPrint;
    return $lineToPrint;
}

```

Figure 41 - logIt Perl subroutine

Chapter 5

Scenarios and Results

Scenarios

In order to properly evaluate the performance and capabilities of the ESXimager program, three test scenarios were created. Each scenario highlights a specific feature or capability to show the ESXimager program can be used to conduct digital forensic image acquisitions in an actual information technology environment.

Scenario 1

Image integrity is of the upmost importance when conducting a digital forensics investigation. Once an image of the target hard drive or file is created, a forensics investigator must ensure the integrity of the image remains true to the original. Failure to ensure image integrity could result in false data being retrieved or digital evidence not being admissible in court.

This scenario highlights the ability of the ESXimager program to ensure image integrity through each step of the image acquisition process. ESXimager tracks the integrity of the image and if the integrity is compromised, the program can detect this change and notify the user accordingly.

Scenario 2

Large enterprises may have dozens of VMware ESXi servers that may utilize a Storage Area Network (SAN) or Network Attached Storage (NAS) device. A SAN could be connected to an ESXi server through a variety of different protocols, fiber channel, Internet Small Computer System Interface (iSCSI), or Network File System (NFS) [14].

In this scenario it is demonstrated that the ESXimager program can find and acquire virtual machines stored on a remote storage device. In this case, remote storage has been added to the ESXi server via iSCSI.

Scenario 3

Virtual machines and files of all different sizes can exist on an ESXi server. The size of a file to be acquired can have an impact on the amount of time it takes to acquire that file, but also on the ESXi servers CPU, memory, and disk resources.

The effect of the image acquisition process on the CPU performance, memory usage, and disk resources of the ESXi server and the time required to acquire images of varying sizes will be examined in this scenario.

Results

Scenario 1

In this first scenario, the image integrity verification functions of the program were tested. An ESXi server was connected to and the `.vmss` (virtual machine save state) file of the Damn Small Linux virtual machine was selected to be imaged. As the ESXimager program was working through the imaging process, some phony data was injected into the image utilizing the `echo >>` command to append additional text to the image file. Echoing even one character into the image file will cause a different hash value to be calculated alerting the digital forensics investigator that the image file differs from the original.

Data was echoed into the image file at two stages of the imaging process. First, as the initial copy was being created with the `dd` command on the ESXi server and second as the file was being transferred to the imaging machine via SFTP. The changes in the MD5 and SHA1 hash values calculated after each of these stages can be seen in Figure 42. Clearly, the calculated MD5

and SHA1 hash values change after the dd command and after the SFTP transfer, the two places where bogus data was appended to the image file before these hashes were calculated.

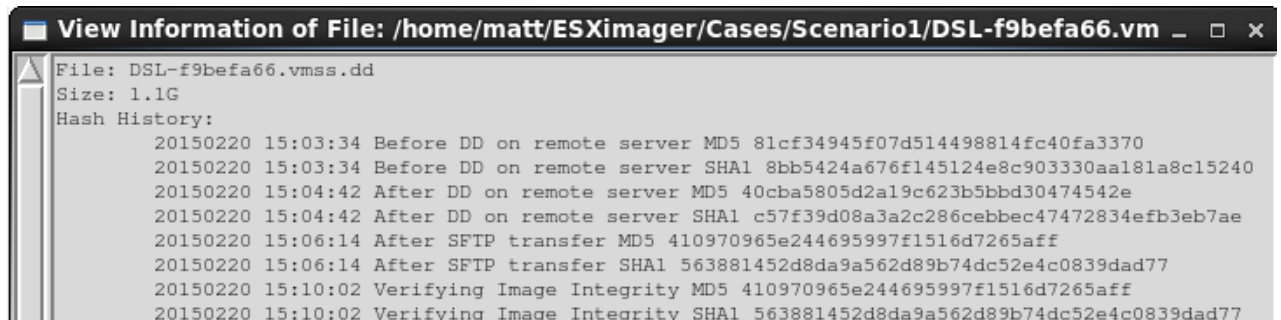


Figure 42 - File information window showing hash history of specified file

At the end of the imaging process, the ESXimager program compares all of the hash values to one another to ensure the image files integrity has not been corrupted during any point of the imaging process. If inconsistent hash values are found, such as when bogus data was appended to the image file in this scenario, the program warns the user, see Figure 43.

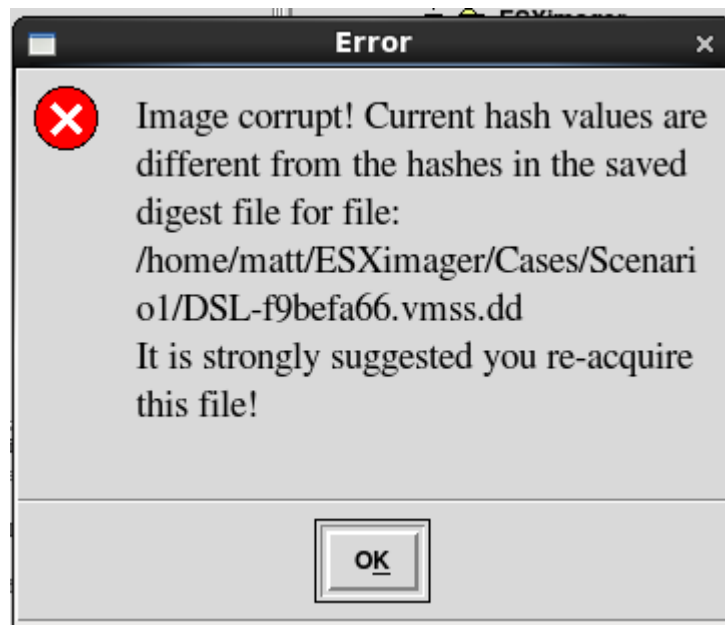


Figure 43 - Error message shown when image is found to be corrupt

Scenario 2

Scenario 2 demonstrates ESXimager's ability to find and acquire virtual machines that are not stored directly on the ESXi server's local storage. With the particular test environment used

for these experiments, remote storage was added to an ESXi server utilizing iSCSI. As shown in Figure 44, a FreeBSD iSCSI Disk with approximately 100 GB of storage has been added as a datastore on the ESXi server.

Datastores							
Identification	Device	Drive Type	Capacity	Free	Type	Last Update	Hardware Acceleration
datastore1	Local VMware, Disk (mpx.vmhba1:C0:T0:L0):3	Non-SSD	32.50 GB	17.19 GB	VMFS5	2/20/2015 2:54:26 PM	Not supported
datastore2	Local VMware, Disk (mpx.vmhba1:C0:T1:L0):1	Non-SSD	49.75 GB	48.80 GB	VMFS5	2/20/2015 2:54:26 PM	Not supported
iSCSIDatastore	FreeBSD iSCSI Disk (t10.FreeBSD_iSCSI_Disk____)	Non-SSD	99.75 GB	56.67 GB	VMFS5	2/20/2015 2:54:26 PM	Supported

Figure 44 - iSCSI disk added to ESXi datastore

A Windows Server 2008 R2 virtual machine was configured utilizing the iSCSI datastore for its storage. After connecting and selecting the virtual machine to image, the windows servers' .vmss (virtual machine save state) file was selected to be acquired, see Figure 45.

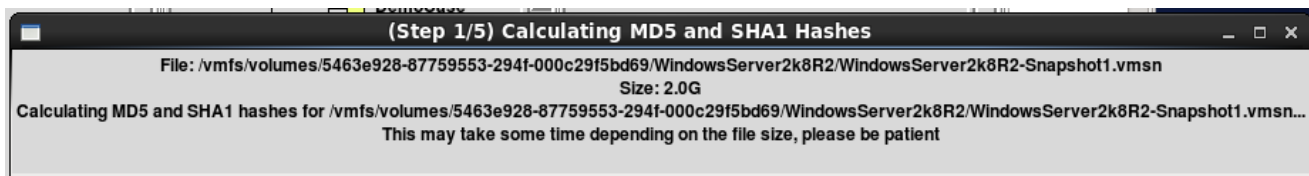


Figure 45 - .vmss file being acquired from iSCSI storage

On the back end, the ESXi server utilizes its own naming convention to name each datastore, creating and assigning each datastore its own Universally Unique Identifier (UUID). Although the user provides a name to identify the datastore, which can be see in Figure 44 under the identification column, these names are simply symbolic links on the backend to the volumes UUID, see Figure 46.

```

/vmfs/volumes # ls -la
total 4100
drwxr-xr-x 1 root root 512 Feb 24 14:24 .
drwxr-xr-x 1 root root 512 Feb 22 20:05 ..
drwxr-xr-x 1 root root 8 Jan 1 1970 5421ddff-8dcda020-9dd5-000c29f5bd69
drwxr-xr-x 1 root root 8 Jan 1 1970 5421ddff-1c795409-5d5e-000c29f5bd69
drwxr-xr-t 1 root root 2100 Nov 12 23:12 5421ddff-a37c5a1a-54dc-000c29f5bd69
drwxr-xr-t 1 root root 1400 Nov 12 23:16 5463e928-87759553-294f-000c29f5bd69
drwxr-xr-t 1 root root 1540 Feb 20 22:34 5478465f-2420367c-b11d-000c29f5bd69
drwxr-xr-x 1 root root 8 Jan 1 1970 698ed378-3180051e-cf52-ff610ea612d0
lrwxr-xr-x 1 root root 35 Feb 24 14:24 datastore1 -> 5421ddff-a37c5a1a-54dc-000c29f5bd69
lrwxr-xr-x 1 root root 35 Feb 24 14:24 datastore2 -> 5478465f-2420367c-b11d-000c29f5bd69
drwxr-xr-x 1 root root 8 Jan 1 1970 f723437c-32d4394f-baf9-d1e30aa134ca
lrwxr-xr-x 1 root root 35 Feb 24 14:24 iSCSIDatastore -> 5463e928-87759553-294f-000c29f5bd69
/vmfs/volumes #

```

Figure 46 - Datastore names are symbolic links to ESXi own naming convention

During the imaging process the network utilization was also monitored from the FreeNAS server, which was the source of the iSCSI disk, see Figure 47. With the FreeNAS server's particular set of hardware, about 200 Mbps of concurrent throughput was achieved through the imaging process. Also, when the dd copy is being created at 15:29 (3:29 PM) higher amounts of data is being read and written to the disk.

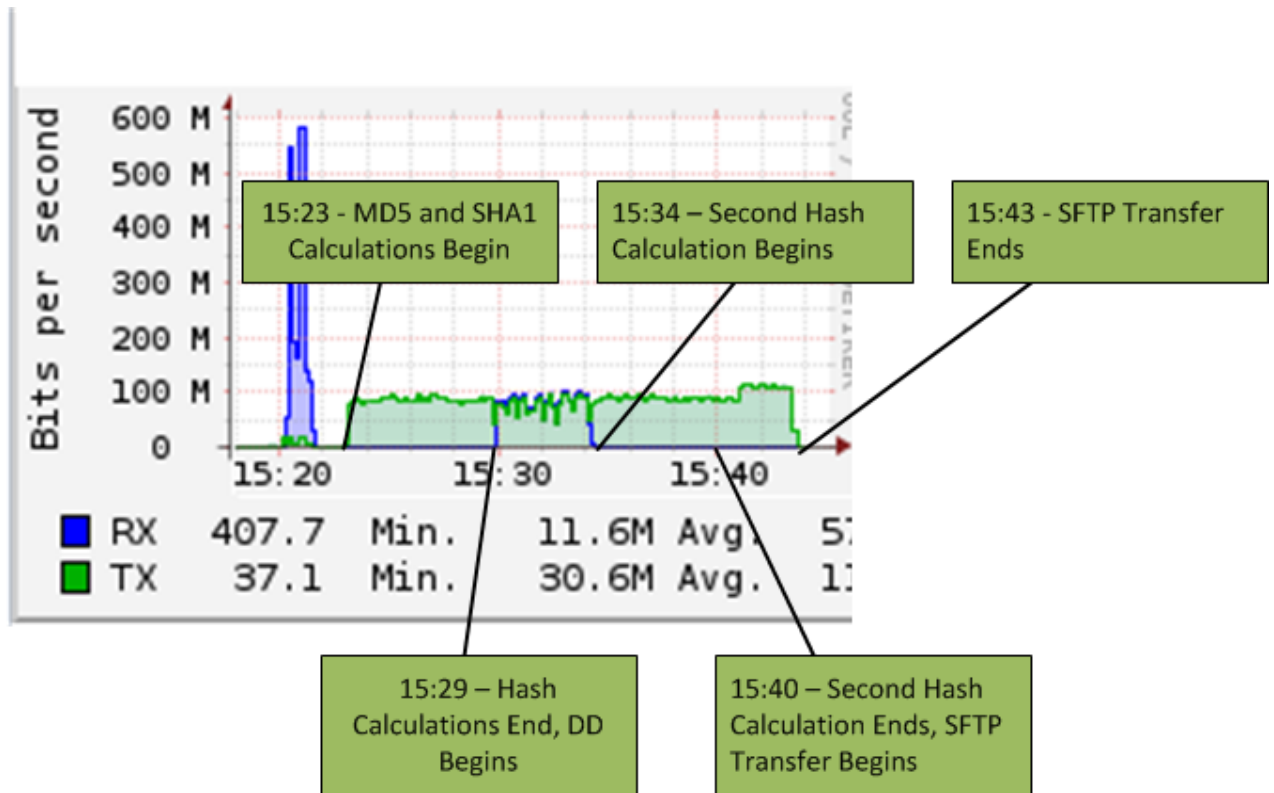


Figure 47 - Network traffic graph during image acquisition from iSCSI datastore

Scenario 3

Scenario 3 looks at how the imaging process affects the computing resources on both the ESXi server and the local imaging machine in addition to the length of time it takes to acquire different sized image. This scenario is broken down into two parts, the imaging of a 4 GB file and a 40 GB file.

Part 1

A 4 GB file was acquired in part one of this scenario. After selecting the file to be imaged and moving past all of the user input stages, the ESXimager program began working on the target file at 1:12 PM, see Figure 48. All imaging operations were completed at 1:48 PM including restarting the virtual machine, see Figure 49. The entire imaging process took about 36 minutes to complete, including all the intricate steps the ESXimager program performs.

```
20150225 13:12:57 [info] (Scenario3P1.2) Calculating md5 hash of file on ESXi server this may take a while be patient...
20150225 13:19:49 [info] (Scenario3P1.2) Done calculating md5 hash of file on ESXi server.
20150225 13:19:49 [info] (Scenario3P1.2) MD5 calculation of file /vmfs/volumes/5421ddff-a37c5ala-54dc-000c29f5bd69/DSL/DSL_1-flat.vmdk took 412 seconds
20150225 13:19:51 [info] (Scenario3P1.2) Calculating sha1 hash of file on ESXi server this may take a while be patient...
20150225 13:27:07 [info] (Scenario3P1.2) Done calculating sha1 hash of file on ESXi server.
20150225 13:27:07 [info] (Scenario3P1.2) SHA1 calculation of file /vmfs/volumes/5421ddff-a37c5ala-54dc-000c29f5bd69/DSL/DSL_1-flat.vmdk took 436 seconds
20150225 13:27:07 [info] (Scenario3P1.2) Hashes of /vmfs/volumes/5421ddff-a37c5ala-54dc-000c29f5bd69/DSL/DSL_1-flat.vmdk Before DD:
MD5: c9a5a6878d97b48cc965c1e41859f034
SHA1: 1bf99ee9f374e58e201e4dda4f474e570eb77229
```

Figure 48 - ESXimager begins working on target 4 GB file

```
20150225 13:48:30 [info] (Scenario3P1.2) No integrity problems found for file: /home/matt/ESXimager/Cases/Scenario3P1.2/DSL_1-flat.vmdk.d
20150225 13:48:30 [info] (Scenario3P1.2) Done performing image integrity verification
20150225 13:48:30 [info] (Scenario3P1.2) Attempting to restart suspended VMs.
20150225 13:48:30 [info] (Scenario3P1.2) Writing to integrity file
20150225 13:48:30 [info] (Scenario3P1.2) All imaging operations complete.
20150225 13:48:30 [info] (Scenario3P1.2) Imaging process took 2134 seconds
```

Figure 49 - ESXimager finishes imaging 4 GB file

In addition to measuring how long it took to image a 4 GB file, the impact on computing resources was also evaluated primarily on the ESXi server. Figure 50 shows the impact the imaging process of the 4 GB file had on the ESXi servers CPU utilization. Each major portion of the imaging process is annotated to show how each step of the imaging process affects the servers CPU utilization. The most prominent line, the orange line, represents the overall CPU utilization percentage. The `md5sum`, `sha1sum`, and `dd` commands do not make use of multi-core CPU's and only utilize one CPU core to the fullest extent. In this test environment, the ESXi server had access to two CPU cores and since these commands only make use of one CPU core, a single core on the ESXi server was maxed out at 100% utilization during the hash calculations and the `dd` copy. With one CPU core maxed at 100% utilization and the other between 1-5%, the average CPU usage for the ESXi server hovered at about 50% during the hash calculations and the `dd`

image creation. Once these steps were complete and the program began to copy the image to the imaging machine via SFTP, CPU utilization received a slight bump, going up to roughly 65% overall utilization. Once the SFTP transfer was complete, the imaging tasks were completed on the ESXi server and CPU utilization returned to normal.

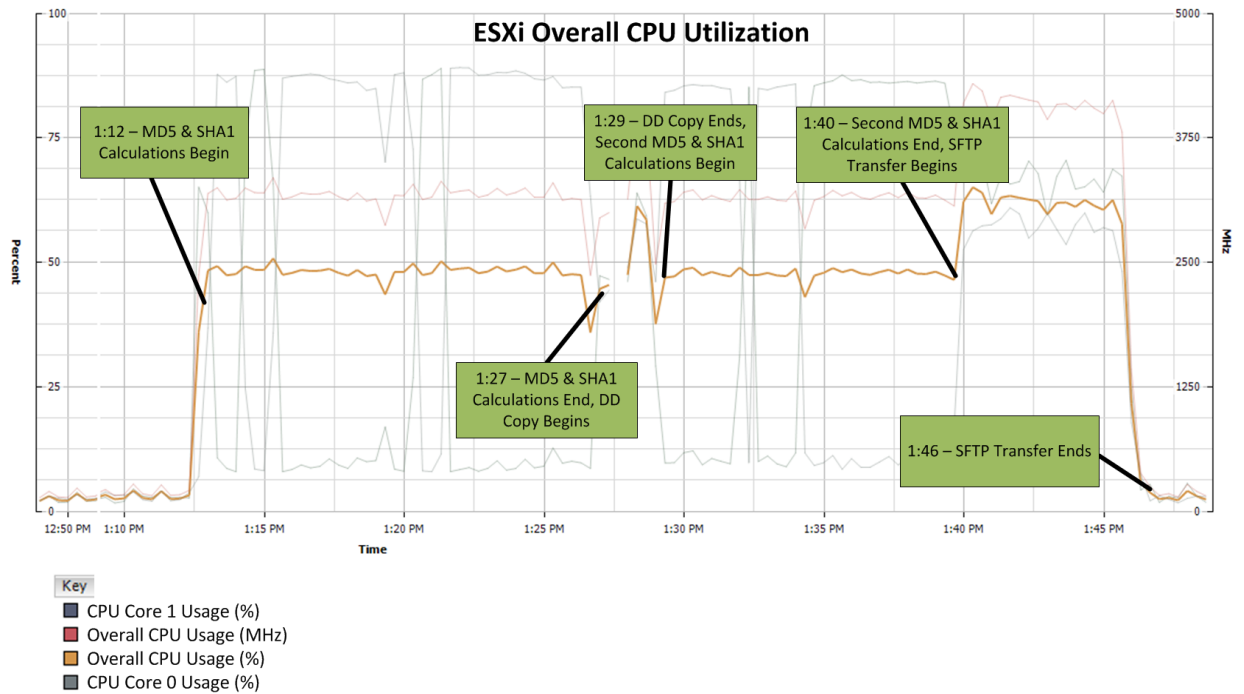


Figure 50 - ESXi CPU performance graph for 4 GB file acquisition

With the imaging tasks completed on the ESXi server and the image transferred, the rest of the processing took place on the imaging machine. The imaging machine calculated two more sets of hash values after the SFTP transfer is complete and the same CPU limitations apply to the `md5sum` and `sha1sum` commands on the imaging machine. As seen in Figure 51, the imaging machine has two CPU cores but the total utilization between the two cores does not surpass roughly 50%. This screenshot includes CPU utilization at the tail end of the SFTP transfer the

duration of the MD5 and SHA1 hash calculations.

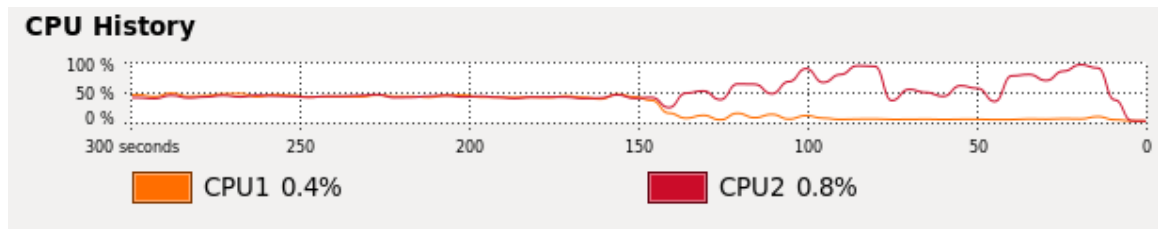


Figure 51 - Linux imaging CPU utilization during 4 GB file acquisition

Because the test environment relied heavily on the use of nested virtualization technologies being run entirely on a single host desktop machine, some final metrics related to the host machines hard drive performance were gathered. With one physical machine running 5 virtual machines, it was surmised that the host machine's hardware might have been a limiting factor in the performance of the ESXimager program particularly in the host machine's hard drive. Utilizing the Windows Performance Monitor application, a number of hard drive metrics were obtained. The most important metric obtained was the hard drives queue length over time, which according to Hua is an indication of a bottleneck on the system [15].

Disk queue length represents the number of Input/Output (I/O) requests that are queued and according to a Microsoft TechNet article, an average disk queue length exceeding twice the number of spindles means there is likely a bottleneck [16]. The host machines storage consists of two 500 GB hard drives in a Redundant Array of Independent Disks (RAID) 0 configuration. Utilizing the formula from the Microsoft TechNet article, a disk queue length higher than 4 could mean there is a potential bottleneck. Figure 52 shows the host machine disk queue length during the 4 GB imaging test. Interestingly, the graph shows a much higher disk queue while the SFTP transfer was taking place between 1:40 PM (13:40) and 1:46 PM (13:46). Throughout the whole imaging process, the disk queue remains under 5 but during the SFTP transfer, the disk queue spikes to almost 40. This potentially means the SFTP transfer process is being limited by the host

machine's disks Input Output Operations per Second (IOPS). With faster storage, the SFTP process may take less time to complete. Furthermore, Figure 53 shows the host machine's disk queue length over an 18-hour period, which includes imaging tests that were conducted as well as normal computer usage (web browsing, word processing, Excel, email etc...). For the most part the disk queue length over the 18-hour period remains below 5, but during the multiple imaging tests the queue length spikes to above 30.

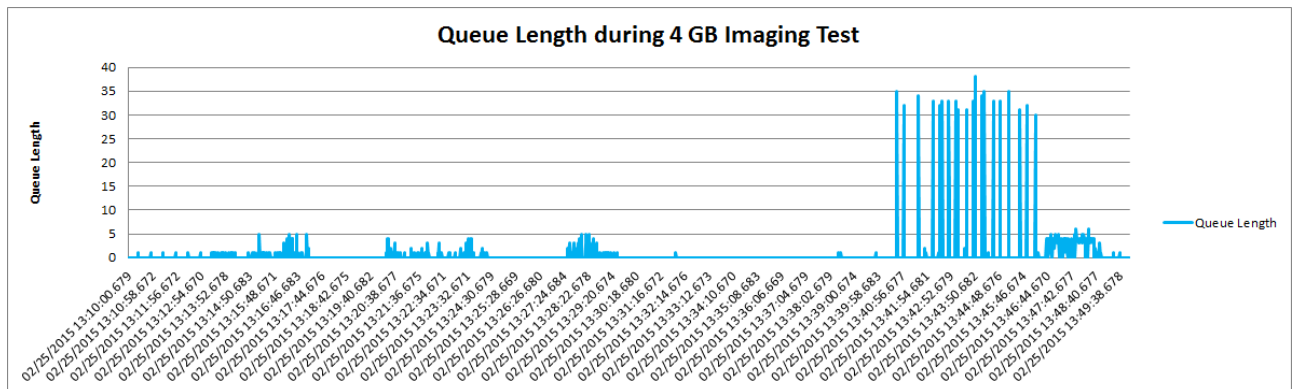


Figure 52 - Host machine disk queue length during 4 GB imaging test

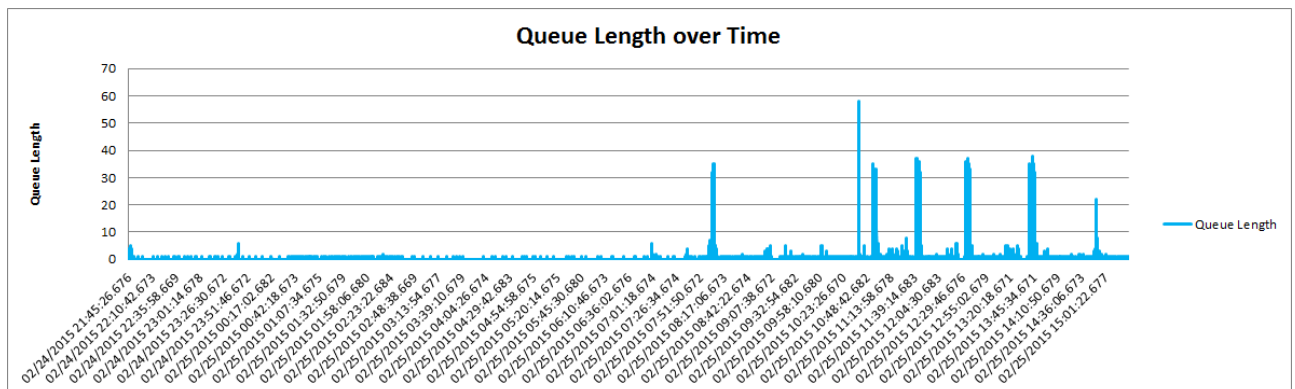


Figure 53 - Host machine disk queue over 18-hour period

Part 2

In Part 2, a 40 GB file was imaged. From start to finish the imaging process took just under 10.5 hours. The process began at 7:54 PM (19:54) and ended at 6:21 AM (6:21), see Figure 54 and Figure 55 respectively.

```

20150225 19:54:11 [info] (Scenario3P2.1) Calculating md5 hash of file on ESXi server this may take a while be patient...
20150225 20:55:38 [info] (Scenario3P2.1) Done calculating md5 hash of file on ESXi server.
20150225 20:55:38 [info] (Scenario3P2.1) MD5 calculation of file /vmfs/volumes/5421ddff-a37c5ala-54dc-000c29f5bd69/Win2k8R2TestLocal/Win2k8R2TestLocal-flat.vmdk took 3687 seconds
20150225 20:55:40 [info] (Scenario3P2.1) Calculating sha1 hash of file on ESXi server this may take a while be patient...
20150225 22:15:28 [info] (Scenario3P2.1) Done calculating sha1 hash of file on ESXi server.
20150225 22:15:28 [info] (Scenario3P2.1) SHA1 calculation of file /vmfs/volumes/5421ddff-a37c5ala-54dc-000c29f5bd69/Win2k8R2TestLocal/Win2k8R2TestLocal-flat.vmdk took 4788 seconds
20150225 22:15:28 [info] (Scenario3P2.1) Hashes of /vmfs/volumes/5421ddff-a37c5ala-54dc-000c29f5bd69/Win2k8R2TestLocal/Win2k8R2TestLocal-flat.vmdk Before DD:
MD5: b8b97861955cle58819868b480128b3d
SHA1: 0ea5b5d0bfff6d7239f0ef57ddbc57e8fddcb4a9

```

Figure 54 - ESXImager begins imaging 40 GB file

```

20150226 06:21:31 [info] (Scenario3P2.1) No integrity problems found for file: /home/matt/ESXImager/Cases/Scenario3P2.1/Win2k8R2TestLocal-flat.vmdk.dd
20150226 06:21:31 [info] (Scenario3P2.1) Done performing image integrity verification
20150226 06:21:31 [info] (Scenario3P2.1) Attempting to restart suspended VMs.
20150226 06:21:31 [info] (Scenario3P2.1) Writing to integrity file
20150226 06:21:31 [info] (Scenario3P2.1) All imaging operations complete.
20150226 06:21:31 [info] (Scenario3P2.1) Imaging process took 37641 seconds

```

Figure 55 – ESXImager finishes imaging 40 GB file

The unusual part about this test is that each hash calculation took an extremely long time on the ESXi server. Even more strangely, for the same file, the hash calculations finished exceedingly quickly on the Linux imaging machine. To give an idea of how long each step of the imaging process took for this 40 GB file, see Table 9.

Process	Where Processing Occurred	Start Time	End Time	Duration
MD5 Calculation 1	ESXi	7:54 PM	8:55 PM	1h 1m
SHA1 Calculation 1	ESXi	8:55 PM	10:15 PM	1h 20m
dd Image Creation	ESXi	10:15 PM	10:45PM	30m
MD5 Calculation 2	ESXi	10:45 PM	1:12 AM	2h 27m
SHA1 Calculation 2	ESXi	1:12 AM	4:03 AM	2h 51m
SFTP Transfer	ESXi & Imaging Machine	4:03 AM	6:05 AM	2h 2m
MD5 Calculation 3	Imaging Machine	6:05 AM	6:09 AM	4m
SHA1 Calculation 3	Imaging Machine	6:09 AM	6:13 AM	4m
MD5 Calculation 4	Imaging Machine	6:13 AM	6:17 AM	4m
SHA1 Calculation 4	Imaging Machine	6:17 AM	6:21 AM	4m

Table 9 - Timeline breakdown during 40 GB file imaging

There is a colossal disparity between the time it takes the ESXi server to calculate hash values and the time it takes for the Linux imaging machine to calculate the same hash values.

Referring back to Table 4 and Table 5, these two virtual machines have access to almost

comparable computing resources. Each have two vCPUs and the ESXi server has 4 GB of RAM where the CentOS imaging machine only has 2 GB of RAM.

In terms of performance, the ESXi server had comparable CPU utilization as in the previous 4 GB imaging test, see Figure 50. No hard evidence or documentation has been found that could possibly explain why the ESXi server takes such a long time to calculate these hash values. There could be a potential number of limitations with the current setup of the test environment such as utilizing a single set of physical hardware to run so many virtual machines doing both CPU and disk intensive tasks. Or perhaps using nested virtualization and running the ESXi hypervisor within VMware Workstation caused an unforeseen bottleneck. These questions cannot be answered until a more robust and realistic test environment is created.

Chapter 6

Limitations and Future Work

There are a number of limitations that exist in the ESXimager program's current state. Most of these limitations are solvable and require additional development time. First, a virtual machine can have multiple `.vmdk` (virtual machine hard drive) files. These files can exist on one or multiple different datastores. Currently ESXimager assumes the `.vmdk` file exists in the same location as the `.vmx` file is found. Second, when creating the `dd` image the image file is stored in the same directory as the original file. The program does not check first to see if there is enough space available on the volume before executing the `dd` copy. Third, a large enterprise environment will most likely contain multiple ESXi servers connected to the same central storage location such as a SAN or NAS. ESXimager currently searches for `.vmx` files to find virtual machines. However, combining this method with shared storage will potentially cause the program to find virtual machines that are not running on the ESXi server the program is currently connected to. A more reliable way needs to be created to only find virtual machines running on the ESXi server currently connected to. Fourth, during the hash calculations and `dd` copy, the program becomes unresponsive to user input. The way it is currently coded, the program waits for the hash or `dd` commands to finish before being able to accept new user inputs. This problem can be fixed by making use of the `fork` command. Fifth, when using `dd`, knowing the correct block size can drastically improve the time it takes for the copy to finish. Currently the program assumes the datastores use 1M block sizes. The block size can be determined by parsing the output of the `vmfstools` command. Sixth, the program currently has only been tested to work with ESXi 5.5. There are differences between ESXi 5.5 and previous versions that will cause the

program to not function correctly. There will obviously be additional bugs and more potential limitations found from additional testing in larger, more realistic environments.

An additional limitation or system stability concern should also be mentioned.

ESXimager must utilize root user credentials in order to properly function and the tool executes shell commands to accomplish its imaging tasks. This indicates there is a possibility that ESXimager could compromise the stability or security of an ESXi server. Although unlikely, the tool could malfunction and cause an errant command to be executed on the ESXi server, which could harm or damage the ESXi server and the virtual machines running on it.

Future work could include fixing the limitations mentioned above and adding more robust features. A potentially useful and unique feature would be to allow ESXimager to capture network traffic directly from an ESXi server.

Companies and organizations may utilize a firewall or a network tap hooked up to packet capture software to save and examine network traffic if deemed necessary. VMware even has a Knowledge Base (KB) article detailing how to configure a virtual machine to monitor network traffic on a specific ESXi portgroup [17]. While the concept of collecting network traffic for analysis is not new, to the best of my knowledge a tool has not yet been created to easily allow network traffic captures to occur directly from an ESXi server. Like the imaging process explained above, an ESXi server's ability to execute shell commands could be leveraged to allow for direct network traffic captures to take place. The `pktcap-uw` shell command allows for live network traffic captures to take place directly from the ESXi server, available in ESXi version 5.5 [18]. Previous versions of ESXi utilize the `tcpdump-uw` command which has been replaced by the `pktcap-uw` command [19]. This feature would be particularly useful if an investigator wanted to obtain a network traffic capture from a particular virtual machine. Perhaps the virtual machine is compromised or an investigator wants to check if the virtual machine is making any

suspicious network connections. Instead of having to setup and utilize another technology such as a network tap, a network capture could be initiated directly from the ESXi server, effectively removing a step and simplifying the process. The network capture could further be filtered by source, destination, protocol, or port. It is important to note that an ESXi server will only be able to capture traffic of virtual machines running on the server. The `pktcap-uw` command would output into a `.pcap` file which could then be opened and examined with a tool like Wireshark [20].

This tool fills a specific niche to help grow the number of digital forensics tools designed to work in virtual environments. The hope is that ESXimager in its current state will make digital forensic investigations easier to conduct in virtual and cloud environments. In order to ensure the continued development of this program and in the hopes of making it into a widely used and helpful tool it has been made available on GitHub [21]. With the help and input of the computer forensics community, this program has the potential to be of vital importance to computer forensics investigations.

References

- [1] T. J. Bittman, M. A. Margevicius, and P. Dawson, “Magic Quadrant for x86 Server Virtualization Infrastructure,” *Gartner*, 02-Jul-2014. [Online]. Available: <http://www.gartner.com/technology/reprints.do?id=1-1WR7CAC&ct=140703&st=sb>. [Accessed: 08-Jan-2015].
- [2] “VMware Named a Leader in 2014 Magic Quadrant for x86 Server Virtualization Infrastructure for Fifth Consecutive Year,” 09-Jul-2014. [Online]. Available: <http://www.vmware.com/company/news/releases/vmw-newsfeed/VMware-Named-a-Leader-in-2014-Magic-Quadrant-for-x86-Server-Virtualization-Infrastructure-for-Fifth-Consecutive-Year/1859233>. [Accessed: 08-Jan-2015].
- [3] B. Nelson, A. Phillips, and C. Steuart, *Guide to computer forensics and investigations*, 4th Ed. Boston, MA: Cengage Learning, 2009.
- [4] “EnCase: Cybersecurity, E-Discovery, Digital Forensics.” [Online]. Available: <https://www.guidancesoftware.com/>. [Accessed: 02-Mar-2015].
- [5] “Forensic Toolkit (FTK) | AccessData.” [Online]. Available: <http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk>. [Accessed: 03-Mar-2015].
- [6] “The Sleuth Kit (TSK) & Autopsy: Open Source Digital Forensics Tools.” [Online]. Available: <http://www.sleuthkit.org/>. [Accessed: 02-Mar-2015].
- [7] W. Delport, M. Köhn, and M. S. Olivier, “Isolating a cloud instance for a digital forensic investigation.,” in *ISSA*, 2011.

- [8] B. Martini and K.-K. R. Choo, "Remote programmatic vCloud forensics: a six-step collection process and a proof of concept," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, 2014, pp. 935–942.
- [9] V. Urias, J. Young, and S. Hatcher, "Implications of Cloud Computing on Digital Forensics," *J. Comput. JoC*, vol. 1, no. 1, 2014.
- [10] T. Atkison and J. C. F. Cruz, "Digital Forensics on a Virtual Machine."
- [11] M. Hirwani, "Forensic analysis of VMware hard disks," Rochester Institute of Technology, 2011.
- [12] P. Henry, "SANS Digital Forensics and Incident Response Blog | How To Digital Forensic Imaging In VMware ESXi | SANS Institute," *SANS Digital Forensics and Incident Response Blog*, 04-Oct-2010. .
- [13] "What Files Make Up a Virtual Machine?" [Online]. Available: https://www.vmware.com/support/ws55/doc/ws_learning_files_in_a_vm.html. [Accessed: 25-Feb-2015].
- [14] "VMware vSphere 4 ESXi Installable and vCenter Server Documentation Center." [Online]. Available: http://pubs.vmware.com/vsphere-4-esxi-installable-vcenter/index.jsp?topic=/com.vmware.vsphere.esxi_server_config.doc_41/esx_server_config/introduction_to_storage/c_networked_storage.html. [Accessed: 17-Feb-2015].
- [15] F. Hua, "How to measure IOPS for Windows," *The Official Synology Blog*, 03-Apr-2013. .
- [16] "Monitoring Queue Length." [Online]. Available: <https://technet.microsoft.com/en-us/library/cc938625.aspx>. [Accessed: 25-Feb-2015].
- [17] "VMware KB: Monitoring network traffic from within a virtual machine on a VMware vSphere ESX/ESXi server." [Online]. Available:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1038847. [Accessed: 02-Mar-2015].

[18]“VMware KB: Using the pktcap-uw tool in ESXi 5.5.” [Online]. Available:

http://kb.vmware.com/selfservice/search.do?cmd=displayKC&docType=kc&docTypeID=DT_KB_1_1&externalId=2051814. [Accessed: 03-Mar-2015].

[19]“VMware KB: Capturing a network trace in ESXi using Tech Support Mode or ESXi Shell.”

[Online]. Available:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1031186. [Accessed: 02-Mar-2015].

[20]“Wireshark · Go Deep.” [Online]. Available: <https://www.wireshark.org/>. [Accessed: 03-Mar-2015].

[21]“GitHub.” [Online]. Available: <https://github.com/>. [Accessed: 04-Mar-2015].

Appendix: A

Source Code

ESXimager2.9A.pl

```
#!/usr/bin/perl
use strict;
#use warnings;
use Tk;
use Tk::ProgressBar;
use Tk::MsgBox;
use Tk::DirTree;
use Tk::Pane;
use Tk::Font;
use Net::OpenSSH;
use Net::SFTP::Foreign;
use Data::Dumper;
use Time::HiRes;

#####
#ESXimager2.9.pl
#Matt Tentilucci
#2-22-2014
#
#V2.1 - Adding in user confirmation of VM choices and passing them back
to sshToESXi sub, removing lots of misc. lines from debugging/trial and
error
#V2.2 - Redesign user selection of VMs window and switched from grid to
pack geometry manager. Instead of having a sub window,
# there will be a frame within main window that will be updated with
the VM choices for the user to image. This should be a much
# cleaner look and prevent multiple windows from popping up. Also added
configuration file and Tools->Settings menu bar for editing it
```

#V2.3 - Added in menu item to open an existing case, variable cleanup, create new case, open case

#V2.4 - Created dirTreeFrame to show case directory listing to use once a case has been opened -> future, allow user to click on files and get info(size, hash, etc...)

moved some boxes around, the connect frame is now horizontal at the top of the window

#V2.5 - Improved VM imaging window. Asks the user what VM they want to image, then what files from that VM, then confirms selection. Added windows telling the user

what the program is doing, when the file gets DD, or hashes are being calculated because the program will not respond to user inputs when those things are being executed

Controlled where subwindows are shown on the screen, they show up in the middle of the main window. Changed console log box to scrolled and tied STDOUT to print in the box.

Now using print will print in the consoleLog, also \$consoleLog->see('end') shows the bottom of the console log and essentially makes it scroll automatically as it grows

After the image has been dd'd and SFTP'd the script will cleanup the .dd files is created on the ESXi server

Made the checkbutton frame scrolled when user has to select what vms/files they want to image

#V2.6 - Integrating buttons into file listing listbox to put selected file through strings and hexdump -C. Case names will not allow spaces, will =~ s/ //;

#V2.7 - Improved logging capabilities. Added Overall log file to keep track if everything, added case log file that keeps track of things related to a particular case

created logIt sub to simplify logging since log messages need to go to multiple places

#V2.8 - Create xml data structure to store a hash log of file acquired. Each case will have its own "hash log" xml file

Implemented a way to verify the integrity of the images takes both automatically after the imaging process is complete and also manually through

```

# the Tools->Verify Integrity menu button
# Added in way to view an image files information, hash history, size,
etc... view->file information
#V2.8A - Removed unnecessary comments, removed unnecessary lines,
condensed code = ~250 lines
#V2.9A - Misc. spelling fixes, create case window now appears in middle
of screen, changed dd to use bs=1M b/c the datastores default to 1M
block sizes(Should find the
# block size by executing vmfstools --query -h /path/to/volume and
parsing output)
# Also since MD5 and SHA1 values are calculated, tried to give some
indication as to which one is currently being calculated by slightly
changing the window depending on which is being calculated
# Added in elapsed time it takes each operation to complete. Gives user
about how long it takes to calculate hashes and for dd to complete.
These times are displayed and logged
# once the task has completed. Updating in real time would require
forking of the dd and hash tasks which would require possibly major
code changes this late
#####

#Before the config file is read and the desired log file location is
determined, I want to log debug messages so I will utilize this array
my @debugMessages;
push @debugMessages, logIt("[debug] (main) Program opened.",0,0,0);
push @debugMessages, logIt("[debug] (main) Initializing some
variables.",0,0,0);
#variable so ssh session to esxi can be accessible outside of sub
my $ssh;
my $checkFrame1;
my $checkFrame2;
my $buttonFrame1;
my $buttonFrame2;
#used for storing the data structure of the case integrity file
my $hashRef;

```



```

#hash ref used to save hashes of a file as it is being imaged. This
will be added into $hasRef and reset once a particular file has been
imaged

my $processingHashRef;
my $preMD5;
my $preSHA1;

#Variables for location of working directory, case directory,
configuration file, and log file
my $configFileLocation = $ENV{"HOME"} . "/ESXimager/ESXimager.cfg";
my $ESXiWorkingDir;
my $ESXiCasesDir;
my $logFileDestination;
my $currentCaseName = "No Case Opened Yet";
my $currentCaseLocation;
my $currentCaseLog;
my $currentCaseIntegrityFile;

push @debugMessages, logIt("[debug] (main) Done initializing
variables.",0,0,0);

#Creates main window
my $mw = MainWindow->new;
push @debugMessages, logIt("[debug] (main) Creating
MainWindow.",0,0,0);
$mw->title("ESXimager 2.9");
$mw->geometry("1400x600");

#Create menu bar
push @debugMessages, logIt("[debug] (main) Creating menu bar.",0,0,0);
$mw->configure(-menu => my $menubar = $mw->Menu);
my $file = $menubar->cascade(-label => '~File');
my $tools = $menubar->cascade(-label => '~Tools');
my $view = $menubar->cascade(-label => '~View');
my $help = $menubar->cascade(-label => '~Help');

```

```

$file->command(-label => 'New Case', -underline => 0, -command =>
\&createNewCase);
$file->command(-label => 'Open Case', -underline => 0, -command =>
\&openExistingCase);
$file->separator;
$file->command(-label => "Quit", -underline => 0, -command => \&exit);

$tools->command(-label => "Verify Integrity", -command =>
\&checkImageIntegrity);
$tools->command(-label => "Settings", -command => \&editSettings);

$help->command(-label => "About", -command => \&showHelp);

#console window
#Anytime print is used, it will output to the $consoleLog window
push @debugMessages, logIt("[debug] (main) Creating ConsoleLog
window.",0,0,0);
my $consoleLog = $mw->Scrolled('Text',-height => 10, -width => 125)-
>pack(-side => 'bottom', -fill => 'both');
tie *STDOUT, 'Tk::Text', $consoleLog->Subwidget('scrolled');

##Connection Frame##
push @debugMessages, logIt("[debug] (main) Creating Connection
Frame.",0,0,0);
#Create top left frame for holding username, password, server IP,
connect button widgets
my $connectionFrame = $mw->Frame(-borderwidth => 2, -relief =>
'groove');
$connectionFrame->pack;#(-side => 'left', -anchor => 'nw');
#label for IP
$connectionFrame->Label(-text => "Server IP")->pack(-side => 'left', -
anchor => 'n');
#entry for IP
my $ESXip = $connectionFrame->Entry( -width => 20, -text =>
"192.168.100.142")->pack(-side => 'left', -anchor => 'n');
#Label for user

```

```

$connectionFrame->Label(-text => "Username")->pack(-side => 'left', -
anchor => 'n');
#entry for user
my $username = $connectionFrame->Entry( -width => 20, -text =>
"root")->pack(-side => 'left', -anchor => 'n');
#label for pass
$connectionFrame->Label(-text => "Password")->pack(-side => 'left', -
anchor => 'n');
#entry for pass
my $password = $connectionFrame->Entry( -width => 20, -show => "*", -
text => "netsys01")->pack(-side => 'left', -anchor => 'n');
#connect button, first calls sub to do input sanitization and checking
on ip, username, and password boxes then
#either falls out with an error, or another sub is called to connect to
the ESXi server
$connectionFrame->Button(-text => "Connect", -command =>
\&sanitizeInputs )->pack(-side => 'left', -anchor => 'n');
##End Connection Frame##

#Creates a label in the top left display the case currently "open"
my $caseLabel = $mw->Label(-text => "$currentCaseName. You must open a
case before imaging a VM")->pack;#(-side => 'left', -anchor => 'nw');

##Dir File Frame##
push @debugMessages, logIt("[debug] (main) Creating Dir File
Frame.",0,0,0);
my $dirFileFrame = $mw->Frame(-borderwidth => 2, -relief => 'groove');
$dirFileFrame->pack(-side => 'right', -fill => 'both');
###Dir Tree Frame##
push @debugMessages, logIt("[debug] (main) Creating Dir Tree
Frame.",0,0,0);
my $dirTreeFrame = $dirFileFrame->Frame(-borderwidth => 2, -relief =>
'groove');
$dirTreeFrame->pack(-side => 'left', -fill => 'both');

```

```

my $dirTree = $dirTreeFrame->Scrolled('DirTree', -scrollbars => 'e', -
directory => $ESXiCasesDir, -width => 35, -height => 20, -browsecmd =>
\&listFiles)->pack(-side => 'left', -anchor => 'n', -fill => 'both');
###End Dir Tree Frame##

###File List Frame##

push @debugMessages, logIt("[debug] (main) Creating File List
Frame.",0,0,0);

my $fileListFrame = $dirFileFrame->Frame(-borderwidth => 2, -relief =>
'groove');

$fileListFrame->pack(-side => 'right', -fill => 'both');

my $fileList = $fileListFrame->Scrolled('Listbox', -scrollbars => 'e',
-width => 40, -height => 15)->pack(-side => 'top', -anchor => 'n', -
fill => 'both', -expand => 1);

listFiles($ESXiCasesDir);

$fileListFrame->Label(-text => "Display Selected File In: ")->pack(-
side => 'left', -anchor => 's', -fill => 'both');

my $stringsButton = $fileListFrame->Button(-text => "Strings", -command
=> [\&runThroughStrings, \$fileList])->pack(-side => 'left', -anchor =>
's', -fill => 'both', -expand => 1);

my $hexdumpButton = $fileListFrame->Button(-text => "Hexdump", -command
=> [\&runThroughHexdump, \$fileList])->pack(-side => 'left', -anchor =>
's', -fill => 'both', -expand => 1);

###End File List Frame##

##End Dir Tree Frame##

#This needs to go after $fileList is defined

$view->command(-label => "File Information", -command =>
[\&viewFileInfo, \$fileList]);

###VM Choices Frame##

push @debugMessages, logIt("[debug] (main) Creating VM Choices
Frame.",0,0,0);

my $vmChoicesFrame = $mw->Frame(-borderwidth => 2, -relief =>
'groove');

$vmChoicesFrame->pack(-side => 'left', -fill => 'both', -expand => 1);

my $vmChoicesLabel = $vmChoicesFrame->Label(-text => "Connect to an
ESXi server to populate\n")->pack;

```

```

##EndVM Choices Frame##

$consoleLog->see('end');

push @debugMessages, logIt("[debug] (main) Done creating Main
Window.",0,0,0);
checkOS();
readConfigFile();

#In addition to opening the program log file, we can open and print out
everything to our debug log file
my $debugLogLocation = $logFileDestination;
$debugLogLocation =~ s/\.log/Debug\.log/;
open (DEBUGLOGFILE, ">>$debugLogLocation");
{ my $ofh = select DEBUGLOGFILE;
  $| = 1;
  select $ofh;
}
foreach(@debugMessages)
{
    print DEBUGLOGFILE $_
}
#With the config file read, we now know where the overall
$logFileDestination is so we can open a file handle
open (PROGRAMLOGFILE, ">>$logFileDestination");
#Make file handle 'hot' so lines don't get buffered before printing
http://perl.plover.com/FAQs/Buffering.html
{ my $ofh = select PROGRAMLOGFILE;
  $| = 1;
  select $ofh;
}
logIt("[info] (main) Initialized... GREETINGS PROFESSOR FALKEN.", 1, 0,
1);

#Reads the configuration file for this program. It looks in
/home/user/ESXimager/ESXimager.cfg

```

```

#If it does not find a config file it will prompt the user to create
one, bringing them to the
#configuration window. Otherwise, the config file is loaded and the
storage locations defined in it are used
sub readConfigFile
{
    my $expectedConfigFileLoc = $ENV{"HOME"} .
"/ESXimager/ESXimager.cfg";
    #if config file exists
    if (-e $expectedConfigFileLoc)
    {
        push @debugMessages, logIt("[debug] (main) Found config
file in expected location, here: $expectedConfigFileLoc.",0,0,0);
        open (CONFIGFILE, $expectedConfigFileLoc);
        push @debugMessages, logIt("[debug] (main) Reading config
file.",0,0,0);
        while(<CONFIGFILE>)
        {
            chomp($_);
            if($_ =~ m/^WorkingDir=.+\/)
            {
                my @configFileSplit = split(/=/);
                chomp($configFileSplit[1]);
                $ESXiWorkingDir = $configFileSplit[1];
                push @debugMessages, logIt("[debug] (main)
Setting ESXi Working Dir to: $ESXiWorkingDir.",0,0,0);
            }
            elsif($_ =~ m/CaseDir=.+\/)
            {
                my @configFileSplit = split(/=/);
                chomp($configFileSplit[1]);
                $ESXiCasesDir = $configFileSplit[1];
                push @debugMessages, logIt("[debug] (main)
Setting ESXi cases directory to: $ESXiCasesDir.",0,0,0);
                $dirTree->chdir($ESXiCasesDir);
                listFiles($ESXiCasesDir);
            }
        }
    }
}

```

```

    }
    elseif($_ =~ m/LogFile=.+/)
    {
        my @configFileSplit = split(/=/);
        chomp($configFileSplit[1]);
        $logFileDestination = $configFileSplit[1];
        push @debugMessages, logIt("[debug] (main)
Setting log file destination to: $logFileDestination.",0,0,0);
    }
    else
    {
        push @debugMessages, logIt("[debug] (main)
Misformatted Config file, dont know what $_ is.",0,0,0);
        $consoleLog->insert('end', "Misformatted Config
file, dont know what $_ is\n");
        $consoleLog->see('end');
    }
}
close(CONFIGFILE);
}
#config file must not exist in the expected location
else
{
    push @debugMessages, logIt("[debug] (main) No config file
could be located. Going to create config file.",0,0,0);
    my $message = $mw->MsgBox(-title => "Info", -type => "ok",
-icon => "info", -message => "It appears this is the first time you are
running this program, a configuration file could not be located. The
following window will allow you to create a configuration file.");
    $message->Show;
    editSettings();
}
}

#sanitizes and checks inputs with connect button is clicked, then
presents and error or

```

```

#calls another sub to connect to the ESXi server
sub sanitizeInputs
{
    my $ip = $ESXip->get;
    my $user = $username->get;
    my $password = $password->get;
    logIt("[info] (main) Sanitizing inputs...", 1, 0, 1);

    my $validInput = 0;

    #Check if the a valid IP address was entered
    if(($ip =~ /^(\\d{1,3})\\. (\\d{1,3})\\. (\\d{1,3})\\. (\\d{1,3})$/ ) && ($1
<= 255 && $2 <= 255 && $3 <= 255 && $4 <= 255 ))
    { $validInput++; }
    else
    {
        logIt("[error] (main) Please enter a valid ip address.", 1,
0, 1);

        my $error = $mw->MsgBox(-title => "Error", -type => "ok", -
icon => "error", -message => "Please enter a valid ip address.");
        $error->Show;
    }
    #checks to see if username field has something typed in
    if(length($user) > 0)
    { $validInput++; }
    else
    {
        logIt("[error] (main) Please enter a username.", 1, 0, 1);
        my $error = $mw->MsgBox(-title => "Error", -type => "ok", -
icon => "error", -message => "Please enter a username.");
        $error->Show;
    }
    #checks to see if password field has something typed in
    if(length($password) > 0)
    { $validInput++; }
    else

```



```

        {
            logIt("[error] (main) Please enter a password.", 1, 0, 1);
            my $error = $mw->MsgBox(-title => "Error", -type => "ok", -
icon => "error", -message => "Please enter a password.");
            $error->Show;
        }

#Calls sub to connect only if input validation has passed
if($validInput == 3)
{
    logIt("[info] (main) Done sanitizing inputs.", 1, 0, 1);
    sshToESXi($ip, $user, $password)
}
}

#connects to the ESXi server via SSH, may need to make $ssh global so
commands can be run outside this sub
sub sshToESXi
{
    my $ip = $_[0];
    my $user = $_[1];
    my $password = $_[2];

    logIt("[info] (main) Attempting to connect to ESXi server at
$ip...", 1, 0, 1);
    $mw->update;
    $ssh = Net::OpenSSH->new("$user:$password@$ip", master_opts => [
-o => "StrictHostKeyChecking=no"]);
    if(!$ssh->error)
    {
        logIt("[info] (main) Successfully connected to $ip", 1, 0,
1);
        $mw->update;
        #Are all vm's stored here? Investigate what path is used
for each esxi host for iscsi or VMs on a SAN
        findVMs("/vmfs/volumes/", $ip);
    }
}

```

```

    }
    else
    {
        logIt("[error] (main) Failed to connect to $ip " . $ssh-
>error, 1, 0, 1);
        my $error = $mw->MsgBox(-title => "Error", -type => "ok", -
icon => "error", -message => "Failed to connect to $ip " . $ssh-
>error);
        $error->Show;
    }
}

#Step 1: $checkFrame1 and $buttonFrame1 - find VMs on ESXi server and
allows the user to select which VM(s) they want to image
sub findVMs
{
    #destroys the two frames from the selectVMFiles sub if they are
defined. Either the user hit the back button or they imaged a VM which
returns to this screen when completed
    if (defined $checkFrame2 && defined $buttonFrame2)
    {
        $checkFrame2->destroy();
        $buttonFrame2->destroy();
    }

    my @vmxFound;
    my @getVMs;
    my $vmstore = $_[0];
    my $ip = $_[1];

    #make the checkbox frame scrollable incase there are multiple
VMs/files that go beyond the window size
    $checkFrame1 = $vmChoicesFrame->Scrolled('Pane',-scrollbars =>
'osoe')->pack(-side => 'top', -fill => 'both', -expand => 1);
    if (defined $vmChoicesLabel)
    {

```

```

        $vmChoicesLabel->packForget;
    }
    $checkFrame1->Label(-text=>"Please select which virtual machines
you want to image:")->pack(-side => "top")->pack();

    #finds anything with .vmx extension meaning it is a VM
    my $stdout = $ssh->capture("find $vmstore -name \"*.vmx\"");
    @vmxFound = split(/\s+/, $stdout);

    my @checkButtons;
    my @checkButtonValues;
    my $counter = 0;

    #Creates check buttons depending on how many VMs are found on the
server
    foreach(@vmxFound)
    {
        $checkButtonValues[$counter] = '0';
        $checkButtons[$counter] = $checkFrame1->Checkbutton(-text
=> $_,-onvalue => $_,-offvalue => '0',-variable =>
\ $checkButtonValues[$counter])->pack();
        $counter++;
    }

    #Creates ok and cancel button to approve VM selections
    $buttonFrame1 = $vmChoicesFrame->Frame()->pack(-side =>
"bottom");
    my $okButton = $buttonFrame1->Button(-text => 'Next', -command =>
[\&selectVMFiles, \@checkButtonValues])->pack(-side => "left");
}

#Step 2: $checkFrame2 and $checkFrame2 - destroys the frames from the
findVMs sub and replaces them with files associated with the VMs they
want to image.
#Asks the user what files they want to acquire, .vmx .vmdk .vmem
etc.....

```

```

sub selectVMFiles
{
    my @vmsToRestart;
    if($currentCaseName =~ m/No Case Opened Yet/)
    {
        logIt("[error] (main) A case has not yet been opened. Open
a case before imaging a VM.", 1, 0, 1);
        my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-icon => "error", -message => "A case has not yet been opened. Open a
case before imaging a VM.\n");
        $message->Show;
    }
    else
    {
        my $choicesRef = shift; #$_[0];
        my @findVMFiles;
        foreach(@$choicesRef)
        {
            if($_ ne '0')
            {
                push @findVMFiles, $_;
            }
            else
            {}
        }
        my $count = @findVMFiles;
        if ($count == 0)
        {
            logIt("[error] ($currentCaseName) No VM's were
selected to be imaged.", 1, 1, 1);
            my $message = $mw->MsgBox(-title => "Error", -type =>
"ok", -icon => "error", -message => "No VM's were selected to be
imaged.\n");
            $message->Show;
        }
        else
    }
}

```

```

{
    foreach(@findVMFiles)
    {
        logIt("[info] ($currentCaseName) Checking state
of Virtual Machine: $_", 1, 1, 1);
        my $vmStatus = checkIfVMRunning($_);
        if ($vmStatus == 1)
        {
            logIt("[info] ($currentCaseName) Virtual
Machine: $_ is running.", 1, 1, 1);
            my $messageBoxAnswer = $mw->messageBox(-
title => "Suspend Virtual Machine?", -type => "YesNo", -icon =>
"question", -message => "$_ is currently powered on and running.\nIt is
strongly recommended the virtual machine be suspended before
imaging.\nDo you want to suspend it?\n", -default => "yes");
            if ($messageBoxAnswer eq 'Yes')
            {
                logIt("[info] ($currentCaseName)
User has selected to suspend $_. The virtual machine will be restarted
once imaging is complete.", 1,1,1);
                suspendVM($_);
                push @vmsToRestart, $_;
            }
        }
        else
        {
            logIt("[info] ($currentCaseName) Virtual
Machine: $_ is not running.", 1, 1, 1);
        }
    }
    $checkFrame1->destroy();
    $buttonFrame1->destroy();

    my @checkButtons;
    my @checkButtonValues;

```

```

my $counter = 0;
$checkFrame2 = $vmChoicesFrame->Scrolled('Pane', -
scrollbars => 'osoe')->pack(-side => 'top', -fill => 'both', -expand =>
1);

foreach(@findVMFiles)
{
    my $VMDirPath = getDirName($_);
    #lists (ls) the given directory on the esxi
server

    my $stdout = $ssh->capture("ls $VMDirPath");
    my @filesFound = split(/\s+/, $stdout);

    foreach(@filesFound)
    {
        my $filePath = $VMDirPath . $_;
        $checkButtonValues[$counter] = '0';
        $checkButtons[$counter] = $checkFrame2->
>Checkbutton(-text => $filePath,-onvalue => $filePath,-offvalue =>
'0',-variable => \$checkButtonValues[$counter])->pack();
        $counter++;
    }
}
#Creates ok and cancel button to approve VM
selections

$buttonFrame2 = $vmChoicesFrame->Frame()->pack(-side
=> "bottom");

my $backButton = $buttonFrame2->Button(-text =>
'Back',-command => [\&findVMs])->pack(-side => "left");
my $okButton = $buttonFrame2->Button(-text =>
'Next',-command => [\&confirmUserVMImageChoices, \@checkButtonValues,
\@vmsToRestart])->pack(-side => "left");
}
}
}

```

#Step 3: Confirms the users choices for which VMs they wish to acquire, expects a reference to the array of checkButton choices as well as a #reference to the array of VMs that need to be restarted once imaging is complete

```
sub confirmUserVMImageChoices
{
    if($currentCaseName =~ m/No Case Opened Yet/)
    {
        logIt("[error] (main) A case has not yet been opened. Open
a case before imaging a VM.", 1, 0, 1);
        my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-icon => "error", -message => "A case has not yet been opened. Open a
case before imaging a VM.\n");
        $message->Show;
    }
    else
    {
        my $choicesRef = $_[0]; #shift; #$_[0];
        my $vmsToRestartRef = $_[1];
        my @VMsToImage;
        foreach(@$choicesRef)
        {
            if($_ ne '0')
            {
                push @VMsToImage, $_;
            }
            else
            {}
        }

        my $count = @VMsToImage;
        if ($count == 0)
        {
            logIt("[error] ($currentCaseName) No files were
selected to be imaged.", 1, 1, 1);
```

```

        my $message = $mw->MsgBox(-title => "Error", -type =>
"ok", -icon => "error", -message => "No files were selected to be
imaged.\n");

        $message->Show;
    }
    else
    {
        my @shortVMFileNames;
        foreach(@VMsToImage)
        {
            push @shortVMFileNames, "\n" .
getFileName($_);
        }
        my $messageBoxAnswer = $mw->messageBox(-title =>
"Confirm File Selection", -type => "YesNo", -icon => "question", -
message => "Would you like to image the following VMs files?:
@shortVMFileNames", -default => "yes");
        if ($messageBoxAnswer eq 'Yes')
        {
            logIt("[info] ($currentCaseName) the following
files will be imaged: @shortVMFileNames", 1, 1, 1);
            my @fileNames;
            my $startTime = time();
            foreach(@VMsToImage)
            {
                #delete whatever is currently in the
processing hash ref

                #we want to only have hash values for a
particular file

                $processingHashRef = {};
                for (keys %$processingHashRef)
                {
                    delete $processingHashRef->{$_};
                }
            }
        }
    }
}

```



```

logIt("[info] ($currentCaseName) Working
on $_", 1, 1, 1);

my $targetImageFile = ddTargetFile($_);
my $ip = $ESXi->get;
logIt("[info] ($currentCaseName) Going to
SFTP $targetImageFile to this computer from ESXi server at IP $ip", 1,
1, 1);

$mw->update;
sftpTargetFileImage($targetImageFile);
my $filename =
getFileName($targetImageFile);
push @fileNames, $filename;
#print "Hash Ref: $hashRef File:
$filename ProcessingHashReg: $processingHashRef\n";
$hashRef->{$filename} =
$processingHashRef;
}
my $arrayRef = \@fileNames;
checkImageIntegrity($arrayRef);

#Restart VMs that were suspended once the
imaging process is complete
logIt("[info] ($currentCaseName) Attempting to
restart suspended VMs.", 1, 1, 1);
foreach (@$vmsToRestartRef)
{
    startVM($_);
}
#After imaging is complete, we want to write
our $hashRef data structure containing all the hash history to the case
integrity file
my $caseIntegrityFileLocation =
$currentCaseLocation . "/" . $currentCaseName . ".integrity";
open ($currentCaseIntegrityFile,
">$caseIntegrityFileLocation");

```

```

        logIt("[info] ($currentCaseName) Writing to
integrity file", 1, 1, 1);
        print $currentCaseIntegrityFile Data::Dumper-
>Dump([$hashRef], [qw/digest/]);
        close($currentCaseIntegrityFile);
        #Maybe add more info to the "done" window
        listFiles($currentCaseLocation);
        logIt("[info] ($currentCaseName) All imaging
operations complete.", 1, 1, 1);
        my $endTime = time();
        my $outputTime = $endTime - $startTime;
        logIt("[info] ($currentCaseName) Imaging
process took $outputTime seconds", 1, 1, 1);

        my $message = $mw->MsgBox(-title => "Info", -
type => "ok", -icon => "info", -message => "\tDone!\nImaging process
took $outputTime seconds\n");
        $message->Show;
        #Return the vm selection window to what it was
originally in case user wants to image more VMs
        findVMs();
    }
    else
    {}
}

}

}

#Step 4: DD target VMs, expects the absolute path to the vm file on
ESXi server
sub ddTargetFile
{
    #!!check to see if ddimages directory exists!! figure out later
    my $absolutePathFileToDD = $_[0];
    #This was a much easier solution to determining where to dd the
file to

```

```

my $dddDestination = $absolutePathFileToDD . ".dd";

#Took out datastore1 b/c with esxi4 does not have a datastore1,
this may have to be changed depending on how it works with a NAS
involved

#Instead, place the dd image file in the same directory as the
orig. Could be changed

#to "find" the storage directory and make a directory by
splitting and popping the

#absolute path apart but this should work for now

#determine file size of file we are about to acquire
my $fileSize = $ssh->capture("ls -lah $absolutePathFileToDD");
$fileSize = returnFileSize($fileSize);
my $subWindow = $mw->Toplevel;
$subWindow->title("(Step 1/5) Calculating *MD5* and SHA1
Hashes");

#####debugging window position
my $mwX = $mw->x;
my $mwY = $mw->y;
my $mwHeight = $mw->height;
my $mwWidth = $mw->width;
my $swHeight = $subWindow->height;
my $swWidth = $subWindow->width;
#####debugging window position

#Adjusts the sub window to appear in the middle of the main
window
my $xpos = int((( $mw->width - $subWindow->width) / 2) + $mw->x);
my $ypos = int((( $mw->height - $subWindow->height) / 2) + $mw-
>y);
$subWindow->geometry("+ $xpos+$ypos");

#Tells the user what is happening b/c they will not have control
until they get to the SFTP step

```

```

        $subWindow->Label(-text => "File: $absolutePathFileToDD\nSize:
$fileSize\nCalculating *MD5* hash for $absolutePathFileToDD...\nThis
may take some time depending on the file size, please be patient\n")-
>pack;

        $mw->update;

        sleep(1);

        my $startTime = time();
        my $md5 = calculateMD5HashOnESX($absolutePathFileToDD);
        my $endTime = time();
        my $outputTime = $endTime - $startTime;
        chomp ($outputTime);
        $subWindow->Label(-text => "MD5 calculation took: $outputTime
seconds\n")->pack;

        logIt("[info] ($currentCaseName) MD5 calculation of file
$absolutePathFileToDD took $outputTime seconds", 1, 1, 1);

        #Try and give some idea when program is calculating each hash
        $subWindow->title("(Step 1/5) Calculating MD5 and *SHA1*
Hashes");

        $subWindow->Label(-text => "File: $absolutePathFileToDD\nSize:
$fileSize\nCalculating *SHA1* hash for $absolutePathFileToDD...\nThis
may take some time depending on the file size, please be patient\n")-
>pack;

        $mw->update;

        sleep(2);

        $startTime = time();
        my $sha1 = calculateSHA1HashOnESX($absolutePathFileToDD);
        $endTime = time();
        $outputTime = $endTime - $startTime;
        chomp ($outputTime);
        $subWindow->Label(-text => "SHA1 calculation took: $outputTime
seconds\n")->pack;

```

```

logIt("[info] ($currentCaseName) SHA1 calculation of file
$absolutePathFileToDD took $outputTime seconds", 1, 1, 1);

#Done telling the user some info, destroy the sub window b/c we
are about to create a new one with new info
$subWindow->destroy();

logIt("[info] ($currentCaseName) Hashes of $absolutePathFileToDD
Before DD:\n \tMD5: $md5\n \tSHA1: $sha1", 1, 1, 1);
$processingHashRef->{getLoggingTime() . " Before DD on remote
server MD5"} = $md5;
$processingHashRef->{getLoggingTime() . " Before DD on remote
server SHA1"} = $sha1;
$mw->update;
sleep(1);

my $subWindow = $mw->Toplevel;
$subWindow->title("(Step 2/5) Creating bit level copy with DD");

#Dont need to recalculate window position again b/c the main
window should not have been moved. Just using values calculated from
above
$subWindow->geometry("+ $xpos+$ypos");

$subWindow->Label(-text => "File: $absolutePathFileToDD\nSize:
$fileSize\nCreating a copy of $absolutePathFileToDD with DD...\nThis
may take some time depending on the file size, please be patient\n")->
>pack;
$mw->update;

logIt("[info] ($currentCaseName) Beginning DD of file:
$absolutePathFileToDD Destination: $ddDestination", 1, 1, 1);

sleep(5);

$startTime = time();

```

```

        my $stdout = $ssh->capture("dd if=$absolutePathFileToDD
of=$dddDestination bs=1M");
        $endTime = time();
        $outputTime = $endTime - $startTime;
        chomp ($outputTime);
        $subWindow->Label(-text => "DD took: $outputTime seconds\n")-
>pack;
        logIt("[info] ($currentCaseName) DD copy of file
$absolutePathFileToDD took $outputTime seconds", 1, 1, 1);

        sleep(5);

        $subWindow->destroy();
        $mw->update;
        logIt("[info] ($currentCaseName) DD of file:
$absolutePathFileToDD to Destination: $dddDestination Done.", 1, 1, 1);

        sleep(1);
        my $subWindow = $mw->Toplevel;
        $subWindow->title("(Step 3/5) Calculating *MD5* and SHA1 hashes
after DD");

        #Dont need to recalculate window position again b/c the main
window should not have been moved. Just using values calculated from
above
        $subWindow->geometry("+ $xpos+$ypos");
        $fileSize = $ssh->capture("ls -lah $dddDestination");
        $fileSize = returnFileSize($fileSize);

        $subWindow->Label(-text => "File: $dddDestination\nSize:
$fileSize\nCalculating *MD5* hash for $dddDestination...\nThis may take
some time depending on the file size, please be patient\n")->pack;
        $mw->update;

        my $pathToHash = $dddDestination;
        $startTime = time();

```

```

my $md5Check = calculateMD5HashOnESX($pathToHash);
$endTime = time();
$outputTime = $endTime - $startTime;
chomp ($outputTime);
$subWindow->Label(-text => "MD5 calculation took: $outputTime
seconds\n")->pack;

logIt("[info] ($currentCaseName) MD5 calculation of file
$pathToHash took $outputTime seconds", 1, 1, 1);

#Try and give some idea when program is calculating each hash
$subWindow->title("(Step 3/5) Calculating MD5 and *SHA1* hashes
after DD");
$subWindow->Label(-text => "File: $ddDestination\nSize:
$fileSize\nCalculating *SHA1* hash for $ddDestination...\nThis may take
some time depending on the file size, please be patient\n")->pack;
$mw->update;
sleep(2);

$startTime = time();
my $sha1Check = calculateSHA1HashOnESX($pathToHash);
$endTime = time();
$outputTime = $endTime - $startTime;
chomp ($outputTime);
$subWindow->Label(-text => "SHA1 calculation took: $outputTime
seconds\n")->pack;

logIt("[info] ($currentCaseName) SHA1 calculation of file
$pathToHash took $outputTime seconds", 1, 1, 1);

$preMD5 = $md5Check;
$preSHA1 = $sha1Check;

logIt("[info] ($currentCaseName) Hashes of $pathToHash After
DD:\n \tMD5: $md5Check\n \tSHA1: $sha1Check", 1, 1, 1);

$processingHashRef->{getLoggingTime() . " After DD on remote
server MD5"} = $md5Check;

$processingHashRef->{getLoggingTime() . " After DD on remote
server SHA1"} = $sha1Check;

```

```

        #Done telling the user some info, destroy the sub window b/c we
are about to create a new one with new info
        $subWindow->destroy();

        $mw->update;
        sleep(1);
        return $pathToHash;
    }

#Step 5: SFTP target VMs, expects absolute path to target dd file
sub sftpTargetFileImage
{
    my $fileToSFTP = $_[0];

    my %args; #= ( user => 'root',password => 'netsys01');
    my $serverIP = $ESXip->get;
    my $user = $username->get;
    my $password = $password->get;
    my $host= '192.168.100.141';

    #label program will go to if a file hash is different after SFTP
and the user wants to try and reacquire the file
    REACQUIRE:

        logIt("[info] ($currentCaseName) SFTP connecting to ESXi server
$serverIP", 1, 1, 1);
        $mw->update;
        my $sftp = Net::SFTP::Foreign->new($serverIP, user => $user,
password => $password);
        $sftp->die_on_error("SSH Connection Failed");
        logIt("[info] ($currentCaseName) Successfully connected to
$serverIP", 1, 1, 1);

        my $getFileName2 = getFileName($fileToSFTP);

```



```

        #Now that we have a cases directory, the images need to be saved
to that directory

        my $localDestination = $currentCaseLocation . "/" .
$getFileName2;

        logIt("[info] ($currentCaseName) Transferring $fileToSFTP from
ESXi server to this computer. Local Destination:$localDestination", 1,
1, 1);

        $mw->update;

        #Create progress bar to show user program is doing something
my $percentDone = 0;
my $subWindow = $mw->Toplevel;
$subWindow->title("(Step 4/5) Transferring image to local
computer via SFTP");
my $startTime = time();
$subWindow->geometry("300x30");

my $xpos = int((( $mw->width - $subWindow->width) / 2) + $mw->x);
my $ypos = int((( $mw->height - $subWindow->height) / 2) + $mw-
>y);
$subWindow->geometry("+ $xpos + $ypos");

my $progressBar = $subWindow->ProgressBar(-width => 30, -blocks
=> 50, -from => 0, -to => 100, -variable => \$percentDone)->pack(-fill
=> 'x');

    $sftp->get($fileToSFTP,$localDestination, callback => sub {
        my ($sftp, $data, $offset, $size) = @_;
        #For whatever reason if the file size is 0, we avoid
dividing by 0
        if ($size == 0)
        {$size = 1;}
        $percentDone = ($offset / $size) * 100;
        $subWindow->update;

    }); #or die "File transfer failed\n";

```

```

$subWindow->destroy;
#With transfer complete, destroy the progress bar window
logIt("[info] ($currentCaseName) SFTP transfer complete.", 1, 1,
1);

my $endTime = time();
my $outputTime = $endTime - $startTime;
chomp ($outputTime);
logIt("[info] ($currentCaseName) SFTP transfer of file
$fileToSFTP took $outputTime seconds", 1, 1, 1);
sleep(2);

#get the file size locally
my $fileSize = `ls -lah $localDestination`;
$fileSize = returnFileSize($fileSize);

#Create subwindow to tell use the program is calculating hashes
my $subWindow = $mw->Toplevel;
$subWindow->title("(Step 5/5) Calculating *MD5* and SHA1 hashes
after SFTP transfer");
#Adjusts the sub window to appear in the middle of the main
window
my $xpos = int((( $mw->width - $subWindow->width) / 2) + $mw->x);
my $ypos = int((( $mw->height - $subWindow->height) / 2) + $mw-
>y);
$subWindow->geometry("+ $xpos+ $ypos");

#Tells the user what is happening b/c they will not have control
while hashes are being calculated
$subWindow->Label(-text => "File: $localDestination\nSize:
$fileSize\nCalculating *MD5* hash for $localDestination...\nThis may
take some time depending on the file size, please be patient\n")->pack;
$mw->update;

logIt("[info] ($currentCaseName) Working on $localDestination",
1, 1, 1);

```

```

    $startTime = time();
    my $md5Check = calculateMD5HashLocal($localDestination);
    $endTime = time();
    $outputTime = $endTime - $startTime;
    chomp ($outputTime);
    $subWindow->Label(-text => "MD5 calculation took: $outputTime
seconds\n")->pack;
    logIt("[info] ($currentCaseName) MD5 calculation of file
$localDestination took $outputTime seconds", 1, 1, 1);

    #Try and give some idea when program is calculating each hash
    $subWindow->title("(Step 5/5) Calculating MD5 and *SHA1* hashes
after SFTP transfer");
    $subWindow->Label(-text => "File: $localDestination\nSize:
$fileSize\nCalculating *SHA1* hash for $localDestination...\nThis may
take some time depending on the file size, please be patient\n")->pack;
    $mw->update;
    sleep(2);

    $startTime = time();
    my $sha1Check = calculateSHA1HashLocal($localDestination);
    $endTime = time();
    $outputTime = $endTime - $startTime;
    chomp ($outputTime);
    $subWindow->Label(-text => "SHA1 calculation took: $outputTime
seconds\n")->pack;
    logIt("[info] ($currentCaseName) SHA1 calculation of file
$localDestination took $outputTime seconds", 1, 1, 1);

    logIt("[info] ($currentCaseName) Hashes of $localDestination
After SFTP Transfer:\n \tMD5: $md5Check\n \tSHA1: $sha1Check", 1, 1,
1);
    $processingHashRef->{getLoggingTime() . " After SFTP transfer
MD5"} = $md5Check;
    $processingHashRef->{getLoggingTime() . " After SFTP transfer
SHA1"} = $sha1Check;

```

```

$subWindow->destroy();

if ($preMD5 ne $md5Check || $preSHA1 ne $sha1Check)
{
    logIt("[error] ($currentCaseName) Hashes do not match for
file $localDestination. Pre MD5:$preMD5 Post MD5:$md5Check Pre
SHA1:$preSHA1 Post SHA1:$sha1Check", 1, 1, 1);
    my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-icon => "error", -message => "Hashes do not match for file
$localDestination. Pre MD5:$preMD5 Post MD5:$md5Check Pre SHA1:$preSHA1
Post SHA1:$sha1Check\n");
    $message->Show;
    my $messageBoxAnswer = $mw->messageBox(-title => "Re-
acquire file?", -type => "YesNo", -icon => "question", -message =>
"Would you like to try and re-acquire file: $localDestination?", -
default => "yes");
    if ($messageBoxAnswer eq 'yes')
    {
        logIt("[info] ($currentCaseName) Attempting to re-
acquire file: $localDestination.", 1, 1, 1);
        goto REACQUIRE;
    }
}
sleep(1);

cleanup($fileToSFTP);
}

#Checks the integrity of all the files currently in the given case
integrity file, or checks a subset of files if an array reference
containing
#the file names to be checked is passed to the sub
sub checkImageIntegrity
{
    my $arrayOfSpecificFiles = $_[0];

```

```

        if($currentCaseName =~ m/No Case Opened Yet/)
        {
            logIt("[error] (main) A case has not yet been opened. Open
a case before verifying image integrity.", 1, 0, 1);
            my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-icon => "error", -message => "A case has not yet been opened. Open a
case before verifying image integrity.\n");
            $message->Show;
        }
        else
        {
            logIt("[info] ($currentCaseName) Verifying integrity of
image files...", 1, 1, 1);
            if (defined $arrayOfSpecificFiles && $arrayOfSpecificFiles
ne '')
            {
                foreach (@$arrayOfSpecificFiles)
                {
                    my $absolutePath = $currentCaseLocation . "/" .
$_;
                    logIt("[info] ($currentCaseName) Checking
integrity of $absolutePath", 1, 1, 1);

                    #determine file size of file
                    my $fileSize = `ls -lah $absolutePath`;
                    $fileSize = returnFileSize($fileSize);
                    my $subWindow = $mw->Toplevel;
                    $subWindow->title("Calculating *MD5* and SHA1
Hashes for Image Integrity Verification");

                    #####debugging window position
                    my $mwX = $mw->x;
                    my $mwY = $mw->y;
                    my $mwHeight = $mw->height;
                    my $mwWidth = $mw->width;

```

```

my $swHeight = $subWindow->height;
my $swWidth = $subWindow->width;

#Adjusts the sub window to appear in the middle
of the main window

my $xpos = int((( $mw->width - $subWindow-
>width) / 2) + $mw->x);
my $ypos = int((( $mw->height - $subWindow-
>height) / 2) + $mw->y);
$subWindow->geometry("$xpos+$ypos");

#Tells the user what is happening b/c they will
not have control while files are being hashed

$subWindow->Label(-text => "File:
$absolutePath\nSize: $fileSize\nCalculating *MD5* hash for
$absolutePath...\nThis may take some time depending on the file size,
please be patient\n")->pack;

$mw->update;

sleep(1);

my $startTime = time();
my $currentMD5Hash =
calculateMD5HashLocal($absolutePath);
my $endTime = time();
my $outputTime = $endTime - $startTime;
chomp ($outputTime);
$subWindow->Label(-text => "MD5 calculation
took: $outputTime seconds\n")->pack;
logIt("[info] ($currentCaseName) MD5
calculation of file $absolutePath took $outputTime seconds", 1, 1, 1);

#Try and give some idea when program is
calculating each hash

$subWindow->title("Calculating MD5 and *SHA1*
Hashes for Image Integrity Verification");

```

```

        $subWindow->Label(-text => "File:
$absolutePath\nSize: $fileSize\nCalculating *SHA1* hash for
$absolutePath...\nThis may take some time depending on the file size,
please be patient\n")->pack;

        $mw->update;
        sleep(2);

        $startTime = time();
        my $currentSHA1Hash =
calculateSHA1HashLocal($absolutePath);
        $endTime = time();
        $outputTime = $endTime - $startTime;
        chomp ($outputTime);
        $subWindow->Label(-text => "SHA1 calculation
took: $outputTime seconds\n")->pack;
        logIt("[info] ($currentCaseName) SHA1
calculation of file $absolutePath took $outputTime seconds", 1, 1, 1);

        #Done telling the user some info, destroy the
sub window b/c we are about to create a new one with new info
        $subWindow->destroy();

        logIt("[info] ($currentCaseName) Hashes of
$absolutePath:\n \tMD5: $currentMD5Hash\n \tSHA1: $currentSHA1Hash", 1,
1, 1);

        my $message = getLoggingTime() . " Verifying
Image Integrity MD5";

        $hashRef->{$_}->{$message} = $currentMD5Hash;
        $message = getLoggingTime() . " Verifying Image
Integrity SHA1";

        $hashRef->{$_}->{$message} = $currentSHA1Hash;

        my %derefHash = %$hashRef;
        my $savedHashRef = $derefHash{$_};
        my %HoH = %$savedHashRef;
        my $isDifferent;

```

```

        foreach my $key (keys %HoH)
        {
            if ($key =~ m/.*MD5.*/)
            {
                my $value =
compareHashes($currentMD5Hash, $HoH{$key});
                $isDifferent = $isDifferent +
$value;
            }
            elsif ($key =~ m/.*SHA1.*/)
            {
                my $value =
compareHashes($currentSHA1Hash, $HoH{$key});
                $isDifferent = $isDifferent +
$value;
            }
            else {}
        }
        if ($isDifferent > 0 )
        {
            logIt("[warning] ($currentCaseName) Image
corrupt! Hashes are different for file: $absolutePath", 1, 1, 1);
            my $message = $mw->MsgBox(-title =>
"Error", -type => "ok", -icon => "error", -message => "Image corrupt!
Current hash values are different from the hashes in the saved digest
file for file: $absolutePath\nIt is strongly suggested you re-acquire
this file!\n");
            $message->Show;
        }
        else
        {
            logIt("[info] ($currentCaseName) No
integrity problems found for file: $absolutePath", 1, 1, 1);
        }
    }
}
else

```



```

{
    my %digest = %$hashRef;
    foreach my $key (keys %digest)
    {
        if ($key =~ m/.*\.dd/)
        {
            my $absolutePath = $currentCaseLocation .
"/" . $key;

            logIt("[info] ($currentCaseName) Checking
integrity of $absolutePath", 1, 1, 1);

            #determine file size of file
            my $fileSize = `ls -lah $absolutePath`;
            $fileSize = returnFileSize($fileSize);
            my $subWindow = $mw->Toplevel;
            $subWindow->title("Calculating MD5 and
SHA1 Hashes for Image Integrity Verification");

            #####debugging window position
            my $mwX = $mw->x;
            my $mwY = $mw->y;
            my $mwHeight = $mw->height;
            my $mwWidth = $mw->width;
            my $swHeight = $subWindow->height;
            my $swWidth = $subWindow->width;

            #Adjusts the sub window to appear in the
middle of the main window
            my $xpos = int((( $mw->width - $subWindow->
width) / 2) + $mw->x);
            my $ypos = int((( $mw->height -
$subWindow->height) / 2) + $mw->y);
            $subWindow->geometry("$xpos+$ypos");

            #Tells the user what is happening b/c
they will not have control while files are being hashed

```

```

        $subWindow->Label(-text => "File:
$absolutePath\nSize: $fileSize\nCalculating MD5 and SHA1 hashes for
$absolutePath...\nThis may take some time depending on the file size,
please be patient\n")->pack;

        $mw->update;

        sleep(1);

        my $currentMD5Hash =
calculateMD5HashLocal($absolutePath);
        my $currentSHA1Hash =
calculateSHA1HashLocal($absolutePath);

        #Done telling the user some info, destroy
the sub window b/c we are about to create a new one with new info
        $subWindow->destroy();

        logIt("[info] ($currentCaseName) Hashes
of $absolutePath:\n \tMD5: $currentMD5Hash\n \tSHA1: $currentSHA1Hash",
1, 1, 1);

        my $message = getLoggingTime() . "
Verifying Image Integrity MD5";
        $hashRef->{$key}->{$message} =
$currentMD5Hash;

        $message = getLoggingTime() . " Verifying
Image Integrity SHA1";
        $hashRef->{$key}->{$message} =
$currentSHA1Hash;

        my $ref = $digest{$key};
        my %HoH = %$ref;
        my $isDifferent;
        foreach my $otherKey (keys %HoH)
        {
            if ($otherKey =~ m/.*MD5.*/)
            {

```

```

my $value =
compareHashes($currentMD5Hash, $HoH{$otherKey});
$isdifferent = $isdifferent +
$value;

}
elseif ($otherKey =~ m/.*SHA1.*/)
{
my $value =
compareHashes($currentSHA1Hash, $HoH{$otherKey});
$isdifferent = $isdifferent +
$value;

}
else {}
}
if ($isdifferent > 0 )
{
logIt("[warning] ($currentCaseName)
Image corrupt! Hashes are different for file: $absolutePath", 1, 1, 1);
my $message = $mw->MsgBox(-title =>
"Error", -type => "ok", -icon => "error", -message => "Image corrupt!
Current hash values are different from the hashes in the saved digest
file for file: $absolutePath\nIt is strongly suggested you re-acquire
this file!\n");

$message->Show;
}
else
{
logIt("[info] ($currentCaseName) No
integrity problems found for file: $absolutePath", 1, 1, 1);
}
}
else {}
}

my $caseIntegrityFileLocation =
$currentCaseLocation . "/" . $currentCaseName . ".integrity";

```

```

        open ($currentCaseIntegrityFile,
">$caseIntegrityFileLocation");
        logIt("[info] ($currentCaseName) Writing to
integrity file", 1, 1, 1);
        print $currentCaseIntegrityFile Data::Dumper-
>Dump([$hashRef], [qw/digest/]);
        close($currentCaseIntegrityFile);
    }
    logIt("[info] ($currentCaseName) Done performing image
integrity verification", 1, 1, 1);
}
}

#Checks if the virtual machine use wants to image is currently powered
on/running. Ideally you want to freeze the VM so nothing changes as you
acquire the VM
#Expects to be passed the absolute path to the VM in questions .vmx
file
sub checkIfVMRunning
{
    my $vmxPath = $_[0];
    my $stdout = $ssh->capture('/sbin/vmdumper -l');
    my @lines = split(/\n/, $stdout);

    foreach(@lines)
    {
        my @lineParts = split(/\s+/, $_);
        foreach (@lineParts)
        {
            if ($_ =~ m/$vmxPath/)
            {
                return 1;
                last;
            }
        }
    }
}

```

```

        return 0;
    }

#suspends a given VM, expects absolute path to vm's .vmx path
sub suspendVM
{
    my $vmxPath = $_[0];
    logIt("[info] ($currentCaseName) Working on $vmxPath ...",
1,1,1);
    my $stdout = $ssh->capture('/sbin/vmdumper -l');
    my @lines = split(/\n/, $stdout);
    foreach (@lines)
    {
        my @lineParts = split(/\s+/, $_);
        foreach (@lineParts)
        {
            if ($_ =~ m/$vmxPath/)
            {
                my $VMwid = $lineParts[0];
                $VMwid =~ s/= /g;
                my @VMwidSplit = split(/\s+/, $VMwid);
                my $stdout = $ssh->capture("/sbin/vmdumper
$VMwidSplit[1] suspend_vm") or warn "remote command failed " . $ssh-
>error;

                my $lineToPrint = getLoggingTime() . " [info]
($currentCaseName) Suspending VM...";
                print PROGRAMLOGFILE $lineToPrint;
                print $currentCaseLog $lineToPrint;
                print DEBUGLOGFILE $lineToPrint;
                $consoleLog->insert('end', $lineToPrint);
                $consoleLog->see('end');

                my $subWindow = $mw->Toplevel;
                $subWindow->title("Suspending VM: $vmxPath");
                #Adjusts the sub window to appear in the middle
of the main window

```

```

        my $xpos = int((( $mw->width - $subWindow-
>width) / 2) + $mw->x);
        my $ypos = int((( $mw->height - $subWindow-
>height) / 2) + $mw->y);
        $subWindow->geometry("+ $xpos+$ypos");

        my $size = 24;
        my $font = $subWindow->fontCreate(-size =>
$size);

        my $text = "Suspending VM...";
        my $suspendVMLabel = $subWindow->Label(-text =>
$text,-width => 20, -font => $font)->pack(-fill => 'both');
        $mw->update;
        my $vmStillRunning = 1;
        while ($vmStillRunning == 1)
        {
            $vmStillRunning =
checkIfVMRunning($vmxPath);

            print PROGRAMLOGFILE ".";
            print $currentCaseLog ".";
            print DEBUGLOGFILE ".";
            print ".";

            $text = $text . ".";
            $suspendVMLabel->configure(-text =>
$text);

            $mw->update;
            sleep(1);
        }
        print PROGRAMLOGFILE "Done!\n";
        print $currentCaseLog "Done!\n";
        print DEBUGLOGFILE "Done!\n";
        print "Done!\n";
        logIt("[info] $vmxPath suspended.", 1,1,1);
        $subWindow->destroy();

```

```

        return 0;
    }
}
return 0;
}

#starts a given VM, expects absolute path to .vmx file in question
sub startVM
{
    my $vmToRestart = $_[0];
    logIt("[info] ($currentCaseName) Going to try and restart this
VM: $vmToRestart", 1, 1, 1);
    my $vmxFile = getFileName($vmToRestart);
    my $stdout = $ssh->capture("vim-cmd vmsvc/getallvms |grep
$vmxFile");
    my @lines = split(/\n/, $stdout);
    my $count = @lines;
    #only one .vmx file was matched so we dont need to worry about
starting the wrong VM
    if ($count == 1)
    {
        my @parts = split(/\s+/, $lines[0]);
        my $stdout = $ssh->capture("vim-cmd vmsvc/power.on
$parts[0]") or warn "remote command failed " . $ssh->error;
        my $lineToPrint = getLoggingTime() . " [info]
($currentCaseName) Starting VM...";
        print PROGRAMLOGFILE $lineToPrint;
        print $currentCaseLog $lineToPrint;
        $consoleLog->insert('end', $lineToPrint);
        $consoleLog->see('end');

        my $subWindow = $mw->Toplevel;
        $subWindow->title("Starting VM: $vmToRestart");
        #Adjusts the sub window to appear in the middle of the main
window

```

```

my $xpos = int((( $mw->width - $subWindow->width) / 2) +
$mw->x);
my $ypos = int((( $mw->height - $subWindow->height) / 2) +
$mw->y);
$subWindow->geometry("$xpos+$ypos");

my $size = 24;
my $font = $subWindow->fontCreate(-size => $size);
my $text = "Starting VM...";
my $startVMLabel = $subWindow->Label(-text => $text,-width
=> 20, -font => $font)->pack(-fill => 'both');
$mw->update;
my $vmStillRunning = 0;
while ($vmStillRunning == 0)
{
    $vmStillRunning = checkIfVMRunning($vmToRestart);

    print PROGRAMLOGFILE ".";
    print $currentCaseLog ".";
    print ".";
    $text = $text . ".";
    $startVMLabel->configure(-text => $text);
    $mw->update;
    sleep(1);
}
print PROGRAMLOGFILE "Done!\n";
print $currentCaseLog "Done!\n";
print "Done!\n";
logIt("[info] ($currentCaseName) $vmToRestart restarted.",
1,1,1);

$subWindow->destroy();
return 0;
}
}

```



```

#because the dd images are being stored into the same directory the
file exists in on the ESXi server, we want to clean up these files when
we are done
#expects the absolute path of the file to delete on the esxi server and
will also check that the file has a .dd extension so the wrong file
does not get
#deleted which would be very bad
#ideally dd would "store" copies somewhere else but this is how I have
it setup for now
sub cleanup
{
    logIt("[info] ($currentCaseName) Cleaning up.", 1, 1, 1);
    my $fileToDel = $_[0];
    if ($fileToDel =~ m/\.+\.dd$/)
    {
        logIt("[info] ($currentCaseName) Going to delete remote
file $fileToDel.", 1, 1, 1);
        my $stdout = $ssh->capture("rm -f $fileToDel");
        logIt("[info] ($currentCaseName) Done deleting remote file
$fileToDel $stdout", 1, 1, 1);
        $consoleLog->see('end');
    }
    else
    {
        logIt("[info] ($currentCaseName) File ($fileToDel) does not
have a .dd extension, will not delete this file.", 1, 1, 1);
        $consoleLog->see('end');
    }
}

#Allows user to edit settings of program. Location where cases, log
files, etc are stored. Maybe additional configurable options later
#Will be run from the Tools->Settings menu bar or run from the
readConfigFile sub if no configuration file is found
sub editSettings
{

```

```

if (-e $configFileLocation)
{
    open (CONFIGFILE, $configFileLocation);
    while(<CONFIGFILE>)
    {
        chomp($_);
        if($_ =~ m/^WorkingDir=.+$/)
        {
            my @configFileSplit = split(/=/);
            chomp($configFileSplit[1]);
            $ESXiWorkingDir = $configFileSplit[1];
        }
        elsif($_ =~ m/CaseDir=.+$/)
        {
            my @configFileSplit = split(/=/);
            chomp($configFileSplit[1]);
            $ESXiCasesDir = $configFileSplit[1];
        }
        elsif($_ =~ m/LogFile=.+$/)
        {
            my @configFileSplit = split(/=/);
            chomp($configFileSplit[1]);
            $logFileDestination = $configFileSplit[1];
        }
        else
        {
            logIt("[error] (main) Misformatted Config file,
dont know what $_ is", 1,0,1);
        }
    }
    close(CONFIGFILE);
}
#config file must not exist in the expected location
else
{

```

```

        $configFileLocation = $ENV{"HOME"} .
"/ESXimager/ESXimager.cfg";
        $ESXiWorkingDir = $ENV{"HOME"} . "/ESXimager";
        $ESXiCasesDir = $ESXiWorkingDir . "/Cases/";
        $logFileDestination = $ESXiWorkingDir . "/ESXimager.log";
    }

    my $settingsWindow = $mw->Toplevel;
    $settingsWindow->title("Settings");
    #Adjusts the sub window to appear in the middle of the main
window
    my $xpos = int((( $mw->width - $settingsWindow->width) / 2) + $mw->x);
    my $ypos = int((( $mw->height - $settingsWindow->height) / 2) + $mw->y);
    $settingsWindow->geometry("+ $xpos+ $ypos");

    #label for configuration file location
    $settingsWindow->Label(-text => "Configuration File Location:
$configFileLocation")->grid(-row => 0, -column => 0);

    #label for working directory location
    $settingsWindow->Label(-text => "ESXimager Working Directory: ")>grid(-row => 1, -column => 0, -sticky => "e");
    #entry for working directory location
    my $workingDirLocation = $settingsWindow->Entry( -width => 40, -text => $ESXiWorkingDir)->grid(-row => 1, -column => 1);

    #label for cases location
    $settingsWindow->Label(-text => "Cases Directory: ")>grid(-row => 2, -column => 0, -sticky => "e");
    #entry for cases location
    my $caseLocation = $settingsWindow->Entry( -width => 40, -text => $ESXiCasesDir)->grid(-row => 2, -column => 1);

    #label for log file location

```

```

    $settingsWindow->Label(-text => "Log File: ")->grid(-row => 3, -
column => 0, -sticky => "e");
    #entry for log file location
    my $logFileLocation = $settingsWindow->Entry( -width => 40, -text
=> $logFileDestination)->grid(-row => 3, -column => 1);

    #connect button, calls sub to update the configuration file (or
create it if it does not exist yet)
    #puts the buttons in a frame at the bottom of the window
    my $settingsWindowBottomFrame = $settingsWindow->Frame;
    $settingsWindowBottomFrame->grid(-row => 4, -column => 0, -
columnspan => 2);
    $settingsWindowBottomFrame->Button(-text => "Save", -command =>
sub {
        $ESXiWorkingDir = $workingDirLocation->get;
        $ESXiCasesDir = $caseLocation->get;
        $logFileDestination = $logFileLocation->get;
        #checks to see if working directory structure exists then
creates it if necessary
        unless (-e $ESXiWorkingDir or mkdir($ESXiWorkingDir, 0755))
        {die "Unable to create $ESXiWorkingDir";}
        open (CONFIGFILE, ">$configFileLocation");
        print CONFIGFILE "WorkingDir=$ESXiWorkingDir\n";
        print CONFIGFILE "CaseDir=$ESXiCasesDir\n";
        print CONFIGFILE "LogFile=$logFileDestination\n";
        close(CONFIGFILE);
        logIt("[info] (main) Config File Saved", 1,0,1);
    }->pack(-side => "left");
    my $cancelButton = $settingsWindowBottomFrame->Button(-text =>
"Exit", -command => [$settingsWindow => 'destroy']->pack(-side =>
"left");
}

#sub to create a new case, essentially all it does is create a
directory under whatever $ESXiCasesDir is and updates the label in $mw
sub createNewCase

```

```

{
    my $createCaseWindow = $mw->Toplevel;
    $createCaseWindow->title("Create New Case");
    my $xpos = int((( $mw->width - $createCaseWindow->width) / 2) +
$mw->x);
    my $ypos = int((( $mw->height - $createCaseWindow->height) / 2) +
$mw->y);
    $createCaseWindow->geometry("+ $xpos+$ypos");

    #label for configuration file location
    $createCaseWindow->Label(-text => "Case Directory Location:
$ESXiCasesDir")->grid(-row => 0, -column => 0);

    #label for working directory location
    $createCaseWindow->Label(-text => "Case Name: ")>grid(-row => 1,
-column => 0, -sticky => "e");
    #entry for working directory location
    my $newCaseName = $createCaseWindow->Entry( -width => 40)>grid(-
row => 1, -column => 1);

    my $createCaseWindowBottomFrame = $createCaseWindow->Frame;
    $createCaseWindowBottomFrame->grid(-row => 2, -column => 0, -
columnspan => 2);
    $createCaseWindowBottomFrame->Button(-text => "Create", -command
=> sub {
        $currentCaseName = $newCaseName->get;
        #Dont want case names to have any spaces
        $currentCaseName =~ s/ //g;
        $mw->update;
        my $newCaseDirPath = $ESXiCasesDir .
$currentCaseName;
        #checks to see if cases directory structure exists
then creates it if necessary
        unless (-e $ESXiCasesDir or mkdir($ESXiCasesDir,
0755))
            {die "Unable to create $ESXiCasesDir";}
    })
}

```

```

#Checks to see if case name user wants to make
already exists

if (-e $newCaseDirPath)
{
    my $error = $mw->MsgBox(-title => "Error", -
type => "ok", -icon => "error", -message => "The case name:
$currentCaseName already exists. Please use another case name.");
    $error->Show;
}
else
{
    mkdir ($newCaseDirPath, 0755);
    $currentCaseLocation = $newCaseDirPath;
    my $caseLogFileLocation = $currentCaseLocation
. "/$currentCaseName.log";
    open ($currentCaseLog,
">>$caseLogFileLocation");
    logIt("[info] ($currentCaseName) Created new
case: $currentCaseName Location: $currentCaseLocation", 1, 1, 1);
    logIt("[info] ($currentCaseName) Opened case
log file $caseLogFileLocation", 1, 1, 1);
    my $caseIntegrityFileLocation =
$currentCaseLocation . "/" . $currentCaseName . ".integrity";
    open ($currentCaseIntegrityFile,
">>$caseIntegrityFileLocation");
    logIt("[info] ($currentCaseName) Opened case
integrity file $caseIntegrityFileLocation", 1, 1, 1);
    #delete whatever is currently in the hash ref
    $hashRef = {};
    for (keys %$hashRef)
    {
        delete $hashRef->{$_};
    }
    #Adds the case name and case location to our
hash

```

```

        $hashRef->{"casename"} = $currentCaseName;
        $hashRef->{"caselocation"} =
$currentCaseLocation;

        print $currentCaseIntegrityFile Data::Dumper-
>Dump([$hashRef], [qw/digest/]);

        $caseLabel->configure(-text => "Current Case:
$currentCaseName Location: $currentCaseLocation");

        $dirTree->chdir($currentCaseLocation);
        listFiles($currentCaseLocation);
        $mw->update;

        #Done telling the user some info, destroy the
sub window b/c we are about to create a new one with new info
        $createCaseWindow->destroy();

    }

    })->pack(-side => "left");

    my $cancelButton = $createCaseWindowBottomFrame->Button(-text =>
"Exit", -command => [$createCaseWindow => 'destroy'])->pack(-side =>
"left");
}

#Allows user to open existing case, brings up a popup directory
navigation window to allow the user to select the case they want to
open
sub openExistingCase
{
    ##More work to be done here

    my $directory = $mw->chooseDirectory(-initialdir=>$ESXiCasesDir,
-ttitle => "Select a case to open");

    if ($directory ne '')
    {
        $currentCaseLocation = $directory;
        $currentCaseName = getFileName($currentCaseLocation);
        my $caseLogFileLocation = $currentCaseLocation .
"/$currentCaseName.log";
        open ($currentCaseLog, ">>$caseLogFileLocation");
    }
}

```

```

        logIt("[info] ($currentCaseName) Opened an existing case:
$currentCaseName Location: $currentCaseLocation", 1, 1, 1);
        logIt("[info] ($currentCaseName) Opened log file:
$caseLogFileLocation For case: $currentCaseName", 1, 1, 1);
        my $caseIntegrityFileLocation = $currentCaseLocation . "/"
. $currentCaseName . ".integrity";
        $hashRef = {};
        for (keys %$hashRef)
        {
            delete $hashRef->{$_};
        }
        open (CASEINTEGRITY, "< $caseIntegrityFileLocation");
        my @lines = <CASEINTEGRITY>;
        close(CASEINTEGRITY);

        my $digest = "";
        my $perlsrc = join(" ", @lines);
        eval $perlsrc;
        $hashRef = $digest;
        logIt("[info] ($currentCaseName) Opened case integrity file
$caseIntegrityFileLocation", 1, 1, 1);
        $caseLabel->configure(-text => "Current Case:
$currentCaseName Location: $currentCaseLocation");
        $dirTree->chdir($currentCaseLocation);
        listFiles($currentCaseLocation);
        $mw->update;
    }
    else {#use must have pressed cancel when selecting a directory
    }
}

#given a directory path, will list all the files in given directory
into the $fileList listbox next to the dirTree
sub listFiles
{
    my $path = $_[0];

```



```

#deletes all the entries in the listBox before populating it
$fileList->delete(0,'end');

opendir (DIR, $path);
$fileList->insert('end', $path);
$fileList->insert('end', "-----
-----");
while (my $file = readdir(DIR))
{
    next if $file =~ /^[.]/;
    if (-f $file)
    {
        $fileList->insert('end', "its a file\n");
    }
    else
    {$fileList->insert('end', $file);}
}
closedir(DIR);
}

#Runs a selected file from the file listBox through the strings command
and shows the output in a new window, expects a reference to the
$fileList listBox
sub runThroughStrings
{
    my $fileListBoxRef = $_[0];
    #de-reference
    my $fileListBoxDeref = $$fileListBoxRef;
    #curselection (cursor selection) returns an array in case
multiple items are selected however, I only allow one item to be
selected with this implementation -selectionmode?
    my @cursorSelection = $fileListBoxDeref->curselection;
    #The current directory being listed is always shown at the top of
the fileList listBox, this is element 0 of the listBox
    my $currentDirectory = $fileListBoxDeref->get(0);

```

```

my $targetFile = $fileListBoxDeref->get(0) . "/" .
$fileListBoxDeref->get($cursorSelection[0]);

if (-f $targetFile)
{
    my $subWindow = $mw->Toplevel;
    $subWindow->title("Strings Output of File: $targetFile");
    my $stringsOutputWindow = $subWindow->Scrolled('Text')->pack(-fill => 'both');
    my $stringsFindLabel = $subWindow->Entry(-width => 20)->pack();

    $subWindow->Button(-text => "Find", -command => sub {
        my $searchString = $stringsFindLabel->get;
        my $stdout = `strings $targetFile | grep
$searchString`;

        my $findStringsSubWindow = $mw->Toplevel;
        my $stringsFindOutputWindow = $findStringsSubWindow->Scrolled('Text')->pack(-fill => 'both');
        $findStringsSubWindow->Button(-text => "Close
Window", -command => [$findStringsSubWindow => 'destroy'])->pack();
        $stringsFindOutputWindow->insert('end', $stdout);
    })->pack();

    $subWindow->Button(-text => "Close Window", -command =>
[$subWindow => 'destroy'])->pack();
    my $stdout = `strings $targetFile`;
    $stringsOutputWindow->insert('end', $stdout);
}

#To prevent anything from happening if the user selects the
divider line ----- or the current directory path
located at the top of the listBox
elseif($cursorSelection[0] == 1 | $cursorSelection[0] == 0)
{
    my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-icon => "error", -message => "No valid file selected. Please select a
file.\n");
    $message->Show;
}

```

```

    }
    #Otherwise they probably selected a directory
    else
    {
        my $message = $mw->MsgBox(-title => "Error", -type => "ok",
        -icon => "error", -message => "$targetFile is a directory. Please
        select a file.\n");
        $message->Show;
    }
}

#Runs a selected file from the file listbox through the strings command
and shows the output in a new window, expects a reference to the
$fileList listbox
sub runThroughHexdump
{
    my $fileListBoxRef = $_[0];
    #de-reference
    my $fileListBoxDeref = $$fileListBoxRef;
    #curselection (cursor selection) returns an array in case
multiple items are selected however, I only allow one item to be
selected with this implementation -selectionmode?
    my @cursorSelection = $fileListBoxDeref->curselection;
    #The current directory being listed is always shown at the top of
the fileList listbox, this is element 0 of the listbox
    my $currentDirectory = $fileListBoxDeref->get(0);
    my $targetFile = $fileListBoxDeref->get(0) . "/" .
$fileListBoxDeref->get($cursorSelection[0]);

    if (-f $targetFile)
    {
        my $subWindow = $mw->Toplevel;
        $subWindow->title("Hexdump Output of File: $targetFile");
        my $stringsOutputWindow = $subWindow->Scrolled('Text')-
>pack(-fill => 'both');

```

```

        $subWindow->Button(-text => "Close Window", -command =>
[$subWindow => 'destroy'])->pack();
        my $stdout = `hexdump -C $targetFile`;
        $stringsOutputWindow->insert('end', $stdout);
    }
    #To prevent anything from happening if the user selects the
divider line ----- or the current directory path
located at the top of the listBox
    elsif($cursorSelection[0] == 1 | $cursorSelection[0] == 0)
    {
        my $message = $mw->MsgBox(-title => "Error", -type => "ok",
- icon => "error", -message => "No valid file selected. Please select a
file.\n");
        $message->Show;
    }
    #Otherwise they probably selected a directory
    else
    {
        my $message = $mw->MsgBox(-title => "Error", -type => "ok",
- icon => "error", -message => "$targetFile is a directory. Please
select a file.\n");
        $message->Show;
    }
}

#Displays a window with information about one of the image files that
has been acquired
#Such as the date acquired, the hashes, file size
sub viewFileInfo
{
    my $fileListBoxRef = $_[0];
    #de-reference
    my $fileListBoxDeref = $$fileListBoxRef;
    #curselection (cursor selection) returns an array in case
multiple items are selected however, I only allow one item to be
selected with this implementation -selectionmode?

```

```

my @cursorSelection = $fileListBoxDeref->curselection;
#The current directory being listed is always shown at the top of
the fileList listBox, this is element 0 of the listBox
my $currentDirectory = $fileListBoxDeref->get(0);
my $targetFile = $fileListBoxDeref->get(0) . "/" .
$fileListBoxDeref->get($cursorSelection[0]);

if (-f $targetFile && $targetFile =~ m/.*\.\dd/)
{
    my %digest = %$hashRef;
    foreach my $key (keys %digest)
    {
        if ($key eq $fileListBoxDeref-
>get($cursorSelection[0]))
        {
            my $subWindow = $mw->Toplevel;
            $subWindow->title("View Information of File:
$targetFile");

            my $fileInfoOutputWindow = $subWindow-
>Scrolled('Text')->pack(-fill => 'both');
            $subWindow->Button(-text => "Close Window", -
command => [$subWindow => 'destroy'])->pack();
            $fileInfoOutputWindow->insert('end', "File: " .
$fileListBoxDeref->get($cursorSelection[0]) . "\n");
            my $filesize = `ls -lah $targetFile`;
            $fileInfoOutputWindow->insert('end', "Size: " .
returnFileSize($filesize) . "\n");
            $fileInfoOutputWindow->insert('end', "Hash
History:\n");

            my $absolutePath = $currentCaseLocation . "/" .
$key;

            my $ref = $digest{$key};
            my %HoH = %$ref;
            foreach my $otherKey (sort keys %HoH)
            {

```

```

                                $fileInfoOutputWindow->insert('end',
"\t$otherKey $HoH{$otherKey}\n");
                                }
                        }
                }
        }
        elseif($targetFile =~ m/.*\.log/ || $targetFile =~
m/.*\.integrity/)
        {
                my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-ICON => "error", -message => "No information available for .log or
.integrity files\n");
                $message->Show;
        }
        #To prevent anything from happening if the user selects the
divider line ----- or the current directory path
located at the top of the listbox
        elseif($cursorSelection[0] == 1 | $cursorSelection[0] == 0)
        {
                my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-ICON => "error", -message => "No valid file selected. Please select a
file.\n");
                $message->Show;
        }
        #Otherwise they probably selected a directory
        else
        {
                my $message = $mw->MsgBox(-title => "Error", -type => "ok",
-ICON => "error", -message => "$targetFile is a directory. Please
select a file.\n");
                $message->Show;
        }
}

#####
#####

```

```

#*****Start of Commonly Used Subs to Make Life
Better*****
*****#
#*****
*****#

#simplify logging. Will take what you want to print as an argument and
output to $consoleLog, the main program log, and the current case log
#Expects the exact message to be printed. ex. [info] (foo) case foo was
opened
#Also expects three values 0 or 1 to determine what to print out to
(for whatever reason a message need to only be printed to one log file)
# Args: message programLogFile caseLogFile consoleLog
sub logIt
{
    my $lineToPrint = $_[0];
    my $programLogPrint = $_[1];
    my $caseLogPrint = $_[2];
    my $consoleLogPrint = $_[3];

    my $lineToPrint = getLoggingTime() . " " . $lineToPrint . "\n";
    print PROGRAMLOGFILE $lineToPrint if $programLogPrint == 1;
    print $currentCaseLog $lineToPrint if $caseLogPrint == 1;
    $consoleLog->insert('end', $lineToPrint) if $consoleLogPrint ==
1;
    $consoleLog->see('end') if $consoleLogPrint == 1;
    print DEBUGLOGFILE $lineToPrint;
    return $lineToPrint;
}

#Gets the current time and returns a nice timestamp for logging
purposes
#http://stackoverflow.com/questions/12644322/how-to-write-the-current-
timestamp-in-a-file-perl
sub getLoggingTime
{

```

```

    my
($sec,$min,$hour,$mday,$mon,$year,$yday,$isdst)=localtime(time);
    my $nice_timestamp = sprintf ( "%04d%02d%02d %02d:%02d:%02d",

$year+1900,$mon+1,$mday,$hour,$min,$sec);
    return $nice_timestamp;
}

#Given the output of ls -lah of a single file, returns the file size,
*nix systems only
sub returnFileSize
{
    my $output = $_[0];
    my @split = split(/\s+/, $output);

    return $split[4];
}

#Sub is passed long absolute path of a file and returns the file name
and extension
#ex. sub is passed /var/storage/foo/bar.vmx and returns bar.vmx
sub getFileName
{
    my $absolutePath = $_[0];
    my $fileName;

    my @fileNameParts = split('/', $absolutePath);
    $fileName = pop @fileNameParts;
    return $fileName;
}

#Sub is passed long absolute path of a file and returns the files
parent directory
#ex. sub is passed /var/storage/foo/bar.vmx and returns
/var/storage/foo/

```



```

sub getDirName
{
    my $absolutePath = $_[0];
    my @fileNameParts = split('/', $absolutePath);
    #Because there is a leading / in the path we need to get rid of
    the first element in the array because it is nothing. print "--$_--" =
    ----

    shift @fileNameParts;
    pop @fileNameParts;

    my $parentDirPath;
    foreach(@fileNameParts)
    {
        $parentDirPath = $parentDirPath . "/" . $_;
    }
    $parentDirPath = $parentDirPath . "/";
    return $parentDirPath;
}

#Calculate MD5 hash of file about to be copied on ESX server
sub calculateMD5HashOnESX
{
    my $fileToHash = $_[0];

    logIt("[info] ($currentCaseName) Calculating md5 hash of file on
ESXi server this may take a while be patient...", 1, 1, 1);
    my $stdout = $ssh->capture("md5sum $fileToHash");
    my @split = split (/s+/, $stdout);
    $stdout = $split[0];
    logIt("[info] ($currentCaseName) Done calculating md5 hash of
file on ESXi server.", 1, 1, 1);
    chomp $stdout;
    return $stdout;
}

# Calculate SHA1 hash of the file about to be copied on ESX server

```

```

sub calculateSHA1HashOnESX
{
    my $fileToHash = $_[0];
    logIt("[info] ($currentCaseName) Calculating sha1 hash of file on
ESXi server this may take a while be patient...", 1, 1, 1);
    my $stdout = $ssh->capture("shasum $fileToHash");
    my @split = split (/\\s+/, $stdout);
    $stdout = $split[0];
    logIt("[info] ($currentCaseName) Done calculating sha1 hash of
file on ESXi server.", 1, 1, 1);
    chomp $stdout;
    return $stdout;
}

#Calculate MD5 hash of local file
sub calculateMD5HashLocal
{
    my $fileToHash = $_[0];
    my $operatingSystem = checkOS();

    logIt("[info] ($currentCaseName) Calculating md5 hash of local
file this may take a while be patient...", 1, 1, 1);
    my $stdout;
    $stdout = `md5sum $fileToHash` if $operatingSystem == 1;
    $stdout = `md5 $fileToHash` if $operatingSystem == 2;
    my @split = split (/\\s+/, $stdout);

    #[$#split] gives you the last element of an array
    $stdout = $split[0] if $operatingSystem == 1;
    $stdout = $split[$#split] if $operatingSystem == 2;
    logIt("[info] ($currentCaseName) Done calculating md5 hash of
local file.", 1, 1, 1);
    chomp $stdout;
    return $stdout;
}

```

```

# Calculate SHA1 hash of local file
sub calculateSHA1HashLocal
{
    my $fileToHash = $_[0];
    my $operatingSystem = checkOS();

    logIt("[info] ($currentCaseName) Calculating sha1 hash of local
file this may take a while be patient...", 1, 1, 1);
    my $stdout;
    $stdout = `shasum $fileToHash` if $operatingSystem == 1;
    $stdout = `shasum $fileToHash` if $operatingSystem == 2;
    #shasum on osx has same output as linux shasum
    my @split = split (/s+/, $stdout);

    #[$#split] gives you the last element of an array
    $stdout = $split[0] if $operatingSystem == 1;
    $stdout = $split[0] if $operatingSystem == 2;
    logIt("[info] ($currentCaseName) Done calculating sha1 hash of
local file.", 1, 1, 1);
    chomp $stdout;
    return $stdout;
}

#Because I am developing this on both OSX and linux I need to ensure
#the script would work on both linux and OSX. The reason being is that
linux
#used the command 'md5sum' whereas OSX just uses 'md5'
sub checkOS
{
    push @debugMessages, logIt("[debug] (main) Checking operating
system.", 0, 0, 0);
    my $OS = $^O;
    my $osValue;
    if($OS eq "linux")
    {

```

```

        push @debugMessages, logIt("[debug] (main) Operating system
is Linux.",0,0,0);
        $osValue = 1;
    }
    #darwin aka osx
    elsif($OS eq "darwin")
    {
        push @debugMessages, logIt("[debug] (main) Operating system
is Mac OSX (darwin).",0,0,0);
        $osValue = 2;
    }
    else
    {
        push @debugMessages, logIt("[debug] (main) Unsupported
operating system detected. ^O.",0,0,0);
        my $message = $mw->MsgBox(-title => "Error", -type => "ok",
        -icon => "error", -message => "You are running an operating system that
        this script is not designed to work for...\nYour operating system is:
        $^O\nSupported operating systems are Linux (linux) and OSX
        (darwin)\n");
        $message->Show;
        exit;
    }
    return $osValue;
}

#compares two hashes and returns 1 if they are different and 0 if they
are the same
sub compareHashes
{
    my $currentHash = $_[0];
    my    $savedHash = $_[1];

    if($currentHash ne $savedHash)
    {
        return 1;
    }
}

```

```

    }
    else
    {
        return 0;
    }
}

#*****
*****#
#*****End of Commonly Used Subs to Make Life
Better*****
*****#
#*****
*****#

#Wait for events. Required for the program to work
MainLoop;

#Close any file handles that may be open
close (PROGRAMLOGFILE);
close ($currentCaseLog);
close (DEBUGLOGFILE);
close ($currentCaseIntegrityFile);

```