FORENSIC RECOVERY OF EVIDENCE FROM DELETED VMWARE VSPHERE HYPERVISOR VIRTUAL MACHINES

by

Brendan Kinchla

A Capstone Project Submitted to the Faculty of

Utica College

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

May 2015

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Cybersecurity

Abstract

The purpose of this research was to analyze the potential for recovering evidence from deleted VMware vSphere Hypervisor (ESXi) virtual machines (VMs). There exists an absence of scholarly research on the topic of deleted VM forensic recovery. Research dedicated to forensic recovery of ESXi VMs and VMware's VM file system (VMFS) is nearly non-existent. This paper examined techniques to recover deleted ESXi VMs to a state where examination for forensic artifacts of user activity can occur. The paper examined the disk-provisioning methods for allocation of virtual disk files and the challenges for forensic recovery associated with each disk-provisioning type. The research determined that the two thick-provisioned virtual disk types provided the best opportunity for complete recovery, while certain characteristics of thin-provisioned virtual disk files made them less likely to recover in their entirety. Fragmentation of virtual disk files presented the greatest challenge for recovery of deleted VMs. Testing of alternate hypotheses attempting to reduce the likelihood of fragmentation within the virtual disk file met with mixed results, leaving fragmentation of virtual disk files as a significant challenge to successful VM recovery. The paper examined the techniques for recovering deleted files from VMFS volumes. Due to a lack of forensic tools with the ability to interpret the VMFS filesystem, forensic recovery focused on data stream searching through the VMFS volume image and file carving from consecutive disk sectors. This method proved to be inefficient, but ultimately successful in most of the test cases. Keywords: Cybersecurity, Professor Cynthia Gonnella, virtualization, VMDK.

Acknowledgements

I wish to express my deepest gratitude to my wife, Olga for her unending patience throughout this process. I could have never completed this without you and I'm humbled by the amount of support you provided me during my classwork and through the capstone process. I'd also like to thank Professor Cynthia Gonnella for her guidance through the capstone and her encouragement and positive feedback when I felt the paper was going nowhere. To my second reader, Cherilyn Neal, your capstone paper provided the spark for me to head down this road of virtualization forensics. I hope my paper has added something to the body of work you started. I greatly appreciate your willingness to perform second reader duties even through all of the changes I threw at you. I'd like to send an additional thank you to Professors Tony DeSarro and Vernon McCandlish for their genuine interest and effort in giving their students the tools and direction necessary to be successful.

Table of Contents

List of Illustrative Materials

Recovering Evidence from Deleted VMware vSphere Hypervisor Virtual Machines

Forensic examinations in virtualized environments pose challenges to examiners beyond those encountered in physical computer examinations. This fact stands in contrast to the perceived ease of use and rapid deployment capabilities associated with many of today's virtualization technology products. Much of the literature on virtualization technology focuses on disaster recovery, performance optimizations, and security considerations. There exists a small subset of research into virtualization forensics. The purpose of this research was to analyze the potential for recovering evidence from deleted VMware vSphere Hypervisor (ESXi) virtual machines (VMs).

According to a January 2012 Forrester Research Custom Technology Adoption Profile commissioned by Cisco Systems, 48% of servers in data centers were virtualized and, "85% of North American enterprises have already adopted x86 server virtualization or [were] planning to expand their implementation in the next 12 months" (pp.1-2). With 85% of North American enterprises using virtualization in the data center, virtualization is clearly mainstream technology in the Information Technology (IT) realm. Keith Ward, the Editor in Chief of Virtualization Review, quoted Andi Mann, Vice President of CA Technologies, as saying, "in 2015 I predict virtualization moves into broad mainstream business and consumer use cases, not just IT mainstream use cases" (Ward, 2015, p. 1, para. 12). With virtualization becoming omnipresent in the data center and moving into the consumer market, forensic capabilities in this space are lagging behind the momentum of virtualization technology. Cyber security expert and SANS Institute Instructor, Dr. Eric Cole (2010), explained the immature state of forensics within virtualized environments:

> If you ask a forensic expert to perform analysis on a compromised system, this is a straightforward task, and you will notice the confidence in which he or she can implement the task. However, if you mention it is a completely virtualized environment, the confidence changes to fear very quickly [….] Our ability to analyze these systems is still in its infancy even though the technology is quickly maturing. (p. xv, para. 2)

This research specifically probed the likelihood of recovering deleted VMs for forensic analysis. Through the exploration of these features within ESXi, the research sought to answer the following questions: What tools and methods can examiners employ to attempt recovery of entirely deleted VMs from a VMFS volume? Which disk-provisioning configuration poses the greatest opportunity for forensic recovery and which disk-provisioning method poses the greatest challenge for forensic recovery from a VMFS volume? What is the greatest challenge to forensic recovery of deleted ESXi VM files and what can increase the likelihood of successful forensic recovery? What tools and methods can examiners employ to attempt complete reconstruction of deleted ESXi VMs?

Virtualization Terminology

Virtualization, as IBM Global Education (2007) explained, is, "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications or end users interact with those resources" (p. 2, para. 3). Virtualized operating systems installed on complete virtual computers called VMs is the basis of x86 virtualization. As Diane Barrett and Gregory Kipper (2010) defined it, a VM is, "a software version of a physical computer that operates and executes like a physical machine" (p. 249, para. 5). The host system is the physical machine that the hypervisor runs on, as it hosts the VMs.

The component of virtualization technology responsible for running multiple VMs on a single host system, termed the hypervisor. As Diane Barrett (n.d.) from the University of Advancing Technology explained, "a hypervisor or VM monitor (VMM) is a virtualization platform that provides more than one operating systems to run on a host computer at the same time" (How Virtualization Works, para. 2). Bare-metal hypervisors, also known as Type 1 hypervisors, install directly onto the system hardware as opposed to type 2 hypervisors, which install on top of a host operating system (Lowe, Marshall, Atwell, Guthrie, & Liebowitz, 2014, p. 4, para. 2).

VMware vSphere Hypervisor (ESXi)

There are numerous implementations of virtualization technology available as either commercial or free products. The virtualization technology that this research focused on was VMware vSphere Hypervisor, also referred to as ESXi. VMware Inc. is a commercial entity, which developed vSphere, VMware's enterprise-grade virtualization product suite (Lowe et al., 2014, p.1 para. 1). VMware offers a limited functionality free license for the ESXi bare-metal hypervisor, which is the core of the vSphere product line. This research used the most current version of ESXi at the time, version 5.5 update 2, with an evaluation license, which enabled all features of ESXi.

Within VMware's vSphere product suite, ESXi provides the foundation of virtualization by way of the VMkernel. The VMkernel manages the VMs' access to the underlying physical hardware through CPU scheduling, memory management, and virtual switch data processing (Lowe et al., 2014, pp. 4-5). Due to the lack of a service console within ESXi, remote management utilities, of which VMware offers several options, perform the bulk of ESXi

configuration. The VMware vSphere Client, a free client application for ESXi host configuration and VM management, created and made changes to the VMs used in the research.

Virtualization Flexibility

The flexibility of VMs allows for dynamic reallocation of hardware, a major benefit over physical machines (Weadock, 2008, para. 2). This dynamic reallocation of hardware makes it quick and easy to shift hardware resources from one VM to another and to quickly create or delete VMs. The virtual isolation of VMs from one another allows each VM to contain its own operating system, applications, and networking configuration (Barrett & Kipper, 2010, p. 7, para. 3). This isolation is similar to physical machines, except that with virtualization, it can be accomplished using shared hardware resources, reducing the duration and cost of operations.

This flexibility inherent to virtualization makes deleting VMs, which may contain forensic artifacts of significance, trivial for a user to perform. ESXi is not different, providing administrators the ability to delete a VM from disk with a single mouse click. Performing a forensic analysis of the deleted VM files requires file recovery from the host disk.

Forensic Challenges of ESXi

ESXi is a proprietary system developed by VMware that has experienced widespread adoption in the virtualization community. Several of the features of ESXi and the proprietary nature of VMware's technology create challenges for forensic examiners. As Brett Shavers, digital forensics consultant and former adjunct instructor at the University of Washington's Digital Forensics program, explained (2008), "virtual machines can be a valuable tool in forensic investigations and can also be used to thwart forensics investigations just as easily" (p. 16, para. 1). Deleting VMs, snapshot rollbacks, and non-persistent changes to disk are examples of techniques used to thwart forensic investigations.

ESXi uses a proprietary file system, called Virtual Machine File System (VMFS), for all storage on the host system. There are several advantages to VMFS which enable the seamless operation of VMs in a distributed vSphere environment, such as allowing multiple VMs to read and write to shared volumes concurrently and direct-to-disk input/output operations, achieving high performance with low CPU overhead (Lowe et al., 2014, p. 310, para. 4). Despite the advantages to VMFS, since it is a proprietary file system, the vast majority of regularly accepted forensic tools are currently not able to interpret the VMFS file system.

At the time of this research, there were two versions of VMFS in regular use, VMFS3 and VMFS5. The most notable difference between VMFS3 and VMFS5 is VMFS5's support for disks greater than 2TB in size (VMware Inc., 2014a, p. 134). Similarly, the files that ESXi creates for suspended state and snapshot operations are VMware proprietary file formats. These proprietary file formats and the proprietary file system present multiple challenges for the forensic examiner seeking to carve deleted files from the file system and interpret recovered files' contents. While the methodology and techniques discussed in this research are specific to ESXi, the concepts are adaptable to other type-1 hypervisors and potentially type-2 hypervisor technologies as well.

Literature Review

Research of forensic analysis on VMs is limited when compared to the volume of research of forensic analysis on physical machines. The vast majority of research on forensic analysis of VMs is from the perspective of analyzing the VM itself. Very limited research exists which focuses on VM analysis from the hypervisor host system. Research into forensic analysis of ESXi as a host system is sparse, while research into the forensic analysis of the VMFS file system is nearly non-existent. This literature review presents data in four sections, forensic analysis of ESXi technology, forensic analysis of other VMware hypervisor technologies, file system forensic analysis, and VMFS forensic analysis.

Forensic Analysis of ESXi Technology

Lowe et al. (2014) state that a VM is comprised of several files on a storage device, with the two most common files being the virtual hard disk file and the configuration file. In the case of ESXi, the configuration file is a plain-text file referred to as the "VMX" file, and is identified by its extension, ".vmx." The type of information available within the VMX file includes, among many other parameters, the number of processors, the amount of RAM, network adapter configurations, and virtual hard disk configurations (p. 484, para. 2). Lowe et al. further describe virtual disk files, or "VMDK" files, which have the extension .vmdk. Each hard drive within a VM has an associated VMDK file, which holds the VM's stored data. The VMDK file is comprised of two files with the .vmdk extension. The VMDK header file contains a plain-text configuration and pointers to the VMDK flat file. The VMDK flat file is identified by the suffix "-flat.vmdk" and contains the actual virtual hard disk data (p. 487).

VMware, Inc. (2014) documented the complete set of files that comprise ESXi VMs within the "Introduction to vSphere Virtual Machines" section of the vSphere 5.5 Documentation

Center. (See *Table 10* in Appendix B for a list of files that make up an ESXi VM). Not all of the files are present for every VM.

ESXi supports several provisioning modes for virtual disks. The provisioning mode of the virtual disk is significant to the chances of forensic recovery. The three provisioning modes are thin-provisioned disks, thick-provisioned lazy zeroed, and thick-provisioned eager zeroed. Lowe et al. (2014) described a thin-provisioned disk as using only as much space on the datastore as the VM itself used. A thick-provisioned lazy zeroed disk uses as much space on the datastore as the configured size of the virtual disk. Unallocated space on the virtual disk remains untouched on the datastore file system until I/O occurs on the VM, at which time the VMkernel zeroes out the space needed for the I/O operation. A thick-provisioned eager zeroed disk uses as much space on the datastore as the configured size of the virtual disk. The VMkernel pre-zeroes unallocated space from the guest VM on the datastore file system (pp. 360-361).

Forensic Analysis of Other VMware Hypervisor Technologies

Brett Shavers' (2008) paper, "Virtual Forensics: A Discussion of Virtual Machines Related to Forensic Analysis," provides some detailed information on forensic analysis of VMware's type-2 hypervisor products, "in the context of VMware, unless otherwise noted, it is intended that VMware refers to the applications related to this paper to include VMware Workstation, VMware Player, and VMware Server" (pp. 9, para. 4). While Shavers' paper did not focus on ESXi, it did provide some insight into performing forensic analysis in virtualized systems. Shavers (2008) described the challenges of recovering a VM in full due to fragmentation of the files. Shavers stated that because of the large size of virtual disk files and a certain amount of fragmentation, fully recovering the contents of a deleted VM may not be possible (p. 8, para. 2).

Shavers' example assumed the use of Microsoft Windows as the host's operating system, which is not applicable to this research, but the potential for file fragmentation is a consideration for attempting to recover large deleted files. Brian Carrier is the author of some of the most widely used forensic tools, including The Sleuth Kit and the Autopsy Forensic Browser and holds a Ph.D. in Computer Science from Purdue University. Carrier has taught at SANS, FIRST, the @stake Academy, and SEARCH in the areas of computer forensics, file systems, and incident response. Carrier (2005) explained that an OS typically allocates consecutive data units, but when that is not possible, fragmentation, a condition when the data units that make up a file are not consecutive, can occur (p. 179, para. 4).

Scott Lowe is a technical architect with VMware's virtual networking business unit and author of six books on VMware vSphere technology. Lowe (2009) recognized the potential for fragmentation within the VMFS3 file system, but addressed the issue through the need for defragmentation, stating that VMFS generally does not need defragmentation to the degree that most other file systems do. Lowe attributed this to the type of files VMFS datastores typically store: typically a low number of very large files (p. 255, para. 2). While Lowe's explanation regarded the performance impact of fragmentation, from a forensic standpoint, the 1MB block size of VMFS5 volumes is significant because it reduces the number of blocks, or clusters, that a disk can contain when compared with other common file systems. Shavers' warning about the likelihood of fragmentation pertained to VMware's Windows based hypervisor products. As Microsoft (2002) documented, the default cluster size on the NTFS file system for disks over 2049MB in size is 4,096 bytes (More Information, para. 4).

The other concept referred to in VMware, Inc.'s performance study was that of thick-provisioned and thin-provisioned disks. VMware, Inc. (2009) explained the difference in disk

provisioning methods as a thick-provisioned disk has its entire capacity pre-allocated on the datastore. Meanwhile, a thin-provisioned disk only uses the amount of disk on the datastore equal to the amount of data allocated within the virtual disk (p.1, para. 5). Given this information, the research analyzed the likelihood of recovery of thick-provisioned disks against that of thin-provisioned disks.

File System Forensic Analysis

In his book *File System Forensics*, Carrier (2005) explained the challenges of performing file system analysis without documentation or source code:

> One of the biggest challenges that I have faced over the years while developing *The Sleuth Kit* (TSK) has been finding good file and volume system documentation [….] It is easy to find resources that describe file systems at a high level, but source code is typically needed to learn the details. (Preface, p. xv, para. 1)

Carrier (2005) explained the data necessary for a file system to function and referred to it as "essential file system data." In his text, *File System Forensic Analysis,* Carrier further described the essential file system data necessary to save and retrieve files. Data must be available to identify the name of a file, where that file's content is stored, and a pointer from the file's name to the metadata structure (p. 176, para. 1).

Cory Altheide has performed forensics and incident response work for numerous public and private sector agencies, including Google, Mandiant, and the National Nuclear Security Administration's Information Assurance Response Center. Altheide is also a contributing author of several books on the topic of digital forensics. Harlan Carvey is an InfoSec Research Senior Consultant at Dell SecureWorks and formerly Chief Forensics Scientist at Applied Security, Inc.

and Vice President of Advanced Security Projects with Terremark Worldwide, Inc. Carvey has developed several forensics tools, including the widely used RegRipper.

File carving, or carving, is a means of recovering deleted files that remain within unallocated space on a file system. Altheide and Carvey describe the practice of carving as searching through an unstructured stream of data for file headers or file signatures, determining the file end point, and saving the extracted data to a carved file (2011, p. 58, Carving, para. 1). Numerous programs exist to help automate the process of carving.

Scott Mueller is the president of Mueller Technical Research corporate training firm, teacher of seminars on PC repair, and author of the longest running PC repair book in the world, *Upgrading and Repairing PCs*, as well as numerous articles for various computer publications and newsletters. Mueller (2013), while describing the method of partition alignment on Windows Vista and later Microsoft operating systems, explained that because of the multiple types of storage devices with potentially variable block sizes, Windows Vista and later aligns partitions along 2,048-sector boundaries (p. 475 para. 6).

VMFS Forensic Analysis

The current version of VMFS, version 5, allows for read/write access from ESXi version 5.0 and above, but is not backward compatible with previous versions of ESXi. Table 1 shows the relationship between VMFS version and ESXi version.

Table 1

VMFS and ESXi Version Comparison

VMFS	ESX/ESXi 3.x host	ESX/ESXi 4.x host	ESXi 5.x host
VMFS2	Read-Only	Read-Only	Not Supported
VMFS3	Read-Write	Read-Write	Read-Write

VMFS5	Not Supported	Not Supported	Read-Write

Note. Adapted from " vSphere Storage," p. 134, Understanding VMFS datastores, table 16-1, by VMWare Inc., 2014a.

This research used VMFS5 for all scenarios, as ESXi v5.5 update 2 was the host operating system version used for testing, and VMFS5 was the most current version of VMFS at the time of the analysis.

Jason Perlow is a Senior Technology Editor at ZDnet, a technology news website, and a Partner Technology Strategist with Microsoft Corp. In March 2009, Perlow wrote a column named, "VMware's filesystem clustering cracked open." Within this article, Perlow alluded to the secrecy that VMware maintains with regard to the inner workings of VMFS, announcing that a company named Fluid Operations published an open-source implementation of VMFS on the Google Code website (Perlow, 2009, para. 1).

Perlow (2009) questioned whether VMware should open the VMFS specification as it had previously done with VMDK (para. 6). At the time of this research, VMFS was still a closed source file system. Fluid Operations' open source implementation of VMFS which Perlow wrote about was last updated over five years ago and only supported up to VMFS3 (Fluid Operations, 2010, para. 2). The Fluid Operations' VMFS driver was not able to read the VMFS5 volumes used for this research.

Christophe Fillot and Mike Hommey developed another open source VMFS implementation, based on the VMFS code from Fluid Operations, named vmfs-tools (Fillot & Hommey, 2012, Introduction, para. 1). Unlike the Fluid Operations VMFS implementation, vmfs-tools did have limited support for VMFS5 (Hommey, 2011, para. 1). The vmfs-tools source code showed the file system signature for VMFS volumes appears at offset 0x0100000 and has a value of, "0xc001d00d." The volume information structure immediately follows the VMFS signature and includes metadata about the volume such as the VMFS version number, label, size,

universally unique identifier (UUID), creation time, and modification time (Fillot & Hommey, 2012, vmfs_volume.h, lines 24-41).

Diane Barrett holds a Master of Science in IT with an information security specialization. Barrett has worked in the information technology industry for over 20 years and has been a primary or coauthor on several works on computer forensics. Gregory Kipper has over ten years of experience working in the field of digital forensics and has published numerous papers and spoken at industry events on the topic of digital forensics. In Barrett and Kipper's book, *Virtualization and Forensics* (2010), they provided a high-level overview of how essential file system data is stored in VMFS through the metadata files. (See *Table 11* in Appendix B for the VMFS metadata files and their purpose). Barrett and Kipper provided useful information for identifying the role of the metadata files of VMFS but provided no information about interpreting the metadata file contents.

Methodology

The methodology for the research analysis followed the process model described in the next section. Drawing information from the literature available on forensic analysis of ESXi, VMFS, and other VMware hypervisor technologies, the methodology sought to perform three tasks; recover deleted VM files from a VMFS volume, recover forensic artifacts of user activity from within those VMs, and reconstruct the deleted VMs within ESXi. The deleted VMs tested with this methodology differed only in their ESXi virtual disk-provisioning mode, with the same three recovery goals applied to each VM.

Process Model

This research followed the forensic process model outlined by the U.S. Department of Commerce, National Institute for Standards in Technology's (NIST) (2006) special publication 800-86, "Guide to Integrating Forensic Techniques into Incident Response." NIST (2006) detailed four phases of their process model: Collection, Examination, Analysis, and Reporting. This research process implemented all phases of the NIST process model.

Collection phase. Identification of potential data sources and acquisition of data from those sources occurs during the collection phase. The collection phase follows a plan that takes into account the likely value of each potential data source, the volatility of each data source, the amount of effort required to acquire each data source, a process to acquire the data, and verification of the acquired data's integrity (NIST, 2006, p. 3-3, Acquiring the Data, para. 1-5).

Examination Phase. Assessing and extracting relevant pieces of data from the collection phase occur during the examination phase. The examination phase includes steps taken reduce the amount of data to be analyzed and to mitigate features that obscure data, such as

compression, encryption, and access control mechanisms (NIST, 2006, p. 3-6, Examination, para. 1-2).

Analysis phase. Analysts study and draw conclusions from the extracted data during the analysis phase. The analysis phase uses a methodical approach to analyze the data to reach a conclusion or determine that no conclusion exists with the evidence available (NIST, 2006, p. 3-6, Analysis, para. 1).

Reporting phase. Preparing and presenting the information resulting from the analysis phase occurs during the reporting phase. The reporting phase identifies problems and addresses shortcomings or errors. Careful documentation of the findings and all steps taken is an essential part of the reporting phase (NIST, 2006, p. 3-6 – 3-7, Reporting, para. 1-5). The "Discussion of Findings" section represents the reporting phase of this research.

Data Collection and Analysis Tools

This section describes the tools used to carry out each phase of the NIST Process model. All tools used to perform the collection, examination, and analysis were freely available open source tools. "Ubuntu 14 Forensics VM" was the examination system on which each of these tools ran. The VMware vSphere client ran on the "Windows 8 x64" VM because the vSphere client software only runs on Microsoft Windows.

dc3dd v7.1.614. Dc3dd is an open source tool used during the analysis for forensic imaging with on-the-fly hashing and forensic disk wiping. The tool's authors describe dc3dd as a patch to the GNU dd program with several additional features, such as on-the-fly hashing (Medico, Cordovano, Kornblum, Lowe, & Levendoski, 2014, Description, para. 1).

dcfldd v1.3.4-1. Nicholas Harbour at the Department of Defense Computer Forensics Lab (DCFL) developed the open source tool dcfldd. Harbour described dcfldd as an

enhancement to GNU dd that includes security and forensics features (2006, Introduction, para. 1). The examination phase made use of dcfldd for extracting specific disk sectors from disk image files.

Fluid Ops VMFS driver. The open source VMFS driver from Fluid Ops provided a way to mount a VMFS volume as a read-only file system. The Fluid Ops VMFS driver provided support for only VMFS version 3 (Fluid Operations, 2010, para. 1-2). The examination phase made use of the Fluid Ops VMFS driver code to interpret the VMFS file system metadata files.

foremost v1.5.7. Foremost, as Altheide and Carvey (2011) described it is an open source file carving tool which uses configurable file headers and footers (p. 59, Foremost, para. 1). The examination and analysis phases made use of foremost to carve specific file types from the volume image and virtual disk images.

GNU dd v8.21. GNU dd is an open source tool present in most Linux distributions as part of the GNU core utilities. Brian Carrier (2005) described dd as a tool that copies data regardless of the data type. Dd reads data in chunks from its input source and copies to its output (p. 60, "A Case Study Using," para 3-4). During the analysis, dd carved the VMFS file system image from the physical disk image.

GPT fdisk (gdisk) v0.8.1. GPT fdisk, or gdisk, is an open source tool used to create, modify, or list information about GUID Partition Table (GPT) disks (Smith, 2014, Description, para. 1). During the analysis, gdisk gathered information about the partitions on the GPT (VM datastore) disk.

GNU md5sum v8.21. Md5sum is an open source tool, bundled as part of the GNU core utilities in most Linux distributions. The Free Software Foundation is the main sponsor of the GNU operating system project and describes the function of md5sum as computing an MD5

checksum for each specified file (Free Software Foundation, 2014, "6.4 md5sum: Print or," para. 1). During the collection, extraction, and analysis phase of each test, md5sum verified file integrity.

hexedit v1.2.12. Hexedit is an open source hexediting tool developed by Pascal Rigaux that displays files in both hexadecimal and ASCII output formats. Hexedit permits file viewing, editing, and searching within a text user interface (Rigaux, n.d. COMMANDS(full and detailed)).

Okular v0.14.3. The open source document viewer application, okular, is part of the KDE ("Kool Desktop Environment") project and can present ".pdf," postscript, ".doc," and other file formats (Free Software Foundation Europe, 2015, Okular, para. 1). During the analysis phase of each test, okular was the ".pdf" viewer used to open the control documents.

Regripper v2.5. Regripper is an open source tool developed by Harlan Carvey that extracts keys, values, and data from the Windows registry (Carvey, 2012, Regripper, para. 1). Regripper extracted the computer name from the system hive on each of the recovered virtual disks.

Ssdeep v2.7. Jesse Kornblum developed the open source tool ssdeep to provide Context Triggered Piecewise Hashing, or "fuzzy hashing". Fuzzy hashing, as Kornblum (2006) explained, is a technique for identifying sequences of identical bytes in the same order for the purpose of finding almost identical files (p. 92, para. 5). In this research, fuzzy hashing with ssdeep helped to show that all the ".nvram" files carved from the VMFS volume were nearly identical.

The Sleuth Kit v4.1.3. The Sleuth Kit is an open source set of command line tools developed by Brian Carrier that is used for analyzing disk images (Carrier, 2015, The Sleuth Kit,

para. 1). The Sleuth Kit tools analyzed the carved virtual disk images to identify partition extents within the virtual disks.

mmls. The tool mmls is one of The Sleuth Kit tools, commonly used to list the contents of the partition table (Carrier, 2010, para. 1). The use of mmls during this research was to identify partition extents to use with dcfldd for volume carving.

vmfs-tools. VMFS-tools is an open-source project, based on the Fluid Operations' VMFS code, and allows for access to VMFS with Linux virtual file systems through the FUSE (file system in userspace) framework (Fillot & Hommey, 2012, Introduction, para. 1). Use of the vmfs-tools suite allowed for mounting of the VMFS volume image and accessing active files on the VMFS file system for analysis.

VMware ESXi (vSphere Hypervisor) v5.5 update 2. ESXi is the common name for the VMware's vSphere Hypervisor. Mike DiPetrillo, Global Cloud Services Architect & Principal Software Engineer for VMware explained that the acronym ESX stood for, "Elastic Sky," and the X was added to, "make it sound more technical" (DiPetrillo, 2010). The "i" was added, according to Mohammed Raffic Kajamoideen, author of VMware ESXi 5.1 Cookbook, to signify, "integrated" (Kajamoideen, 2013, What is VMware ESXi?, para. 1).

xxd V1.10. Written by Juergen Weigert, xxd is an open source tool which outputs a hex dump for a given file or input stream (Moolenaar & Nugent, 1996, Description, para. 1). During the analysis, xxd provided a means of outputting variable length ASCII text representations of hexadecimal data streams to standard output or to a plain text files.

Testing Environment

The testing environment was comprised of two physical machines and six VMs. An ESXi host system hosted four of the VMs used as test systems for the forensic analysis. The other

physical machine was an Apple MacBook Pro, which hosted two VMware Fusion VMs that performed the forensic analysis of the test VMs and the ESXi host system's datastore. The details of each of those systems follow.

ESXi host system. An ESXi host system, named "vmhost21" was the physical machine that hosted four VMs and was the target of the forensic analysis tests in this research. The ESXi host system had the following software and main hardware components:

- Operating system: VMware vSphere Hypervisor (ESXi) 5.5 update 2
- CPU: Intel Core i5-3550, BX806237i53550, quad-core, 6MB L3 cache, 3.30GHz
- RAM: Kingston HyperX Blu, 16GB DDR3, 1600MHz
- Motherboard: Gigabyte GA-Z77X-UD3H, ATX
- System hard drive: OCZ Technology Vertex Plus R2 MLC SATA II 2.5" 120GB SSD
- VM datastore drive: Western Digital Caviar WD2000JD-00HBB0 SATA 3.5" 200GB

The VM datastore drive, forensically wiped with zeroes using dc3dd prior to configuring the datastore, held a single 80GB VMFS5 volume labeled "Boston." The "Boston" volume had a formatted capacity of 79.75GB.

Test VMs. Four VMs had their files stored on the "Boston" volume. Three of the VMs existed for specific tests focusing on recovery of artifacts after deleting the VMs from disk, while one VM existed to generate disk activity to observe the effects on the three deleted VMs' files. Each of the VMs had the same software and virtual hardware configuration with the exception of the virtual disk allocation method. The "Alpha" VM, used for test #1, had a thick-provisioned, eager zeroed virtual disk. The "Bravo" VM, used for test #2, had the ESXi default method of thick-provisioned, lazy zeroed virtual disk. The "Charlie" VM, used for test #3, had a thin-

provisioned virtual disk. The "Delta" VM had a thick-provisioned, lazy zeroed virtual disk. Table 2 lists the configuration of the four test VMs.

Table 2

Test VM Configurations

	VM 1	**VM 2**	**VM 3**	**VM 4**
VM Name	Alpha	Bravo	Charlie	Delta
VM working location	[Boston]/Alpha	[Boston]/Bravo	[Boston]/Charlie	[Boston]/Delta
CPUs	1	1	1	1
RAM	1GB	1GB	1GB	1GB
HDD size	12GB	12GB	12GB	12GB
Virtual disk-provisioning method	Thick-provisioned, eager zeroed	Thick-provisioned, lazy zeroed	Thin-provisioned	Thick-provisioned, lazy zeroed
Virtual disk file	[Boston]/Alpha/Alpha-flat.vmdk	[Boston]/Bravo/Bravo-flat.vmdk	[Boston]/Charlie/Charlie-flat.vmdk	[Boston]/Delta/Delta-flat.vmdk
VM configuration file	[Boston]/Alpha/Alpha.vmx	[Boston]/Bravo/Bravo.vmx	[Boston]/Charlie/Charlie.vmx	[Boston]/Delta/Delta.vmx
Operating system	Windows 7 Professional x64	Windows 7 Professional x64	Windows 7 Professional x64	Windows 7 Professional x64

Note. This table lists the virtual hardware, configuration files and operating system of each of the VMs on the "Boston" volume of, "vmhost21."

Forensic analysis systems. The forensic analysis used two VMware Fusion VMs hosted on a single physical machine. The physical machine host system had the following characteristics:

- Manufacturer and Model: Apple MacBook Pro 15-inch, mid-2012 edition
- RAM: 16GB 1333MHz DDR3
- Graphics: NVIDIA GeForce GT 650M 512 MB

- Operating system: OS X 10.9.5 (13F34)
- Software: VMware Fusion Professional version 7.1.1 (2498930)

The two analysis systems were VMs in VMware Fusion on the MacBook Pro physical machine. Table 3 lists the configuration of the two forensic analysis VMs.

Table 3

Forensic Analysis VMs Configuration

	Analysis VM 1	**Analysis VM 2**
VM Name	Ubuntu 14 Forensics VM	Windows 8 x64
Operating system	Ubuntu 14.04 LTS 64-bit	Microsoft Windows 8.1 Pro
CPU cores	4	4
RAM	4 GB	4GB
System hard disk capacity	60GB	60GB
Additional hard disk capacity	60GB	60GB
USB Compatibility	2.0	3.0

Note. This table lists the virtual hardware and operating system for each of the forensic analysis VMs.

Additional hardware. An Etekcity USB2.0/eSata Dual HDD docking station, model number 08-WS-ST320A-ETEK, served as the means by which the VM datastore hard drive connected to the forensic workstations and created image backups. Two Western Digital Green 3.0TB 3.5" SATA hard drives, model number WD30EZRX, served as the working-copy image media and preservation-copy image media.

Test Scenarios

Three test scenarios sought to identify the feasibility of VM recovery from a VMFS volume under different configurations. Each of the test scenarios contained identical VMs, with the exception of the virtual disk-provisioning mode, installed on the same VMFS datastore of the same ESXi host. The order of VM creation and virtual disk file provisioning was "Alpha,"

"Bravo," "Delta," "Charlie." All four VMs completed configuration according to the scenario preparation section.

Test #1: Thick-provisioned, eager zeroed virtual disk. The goal of test #1 was to recover the deleted VM, "Alpha," from the VMFS file system, and analyze to identify forensic artifacts of user activity. Four ".pdf" control files, part of the scenario preparation, served as the artifacts of user activity. "Alpha" had a single virtual disk configured in thick-provisioned, eager zeroed mode. As explained in the "Forensic Analysis of ESXi Technology" section, a thick-provisioned eager zeroed disk uses as much space on the datastore as the configured size of the virtual disk. The VMkernel pre-zeroes unallocated space from the guest VM on the datastore file system (Lowe et al., 2014, pp. 360-361).

Test #2: Thick-provisioned, lazy zeroed virtual disk. The goal of test #2 was to recover the VM, "Bravo," from the VMFS file system and analyze it to identify forensic artifacts of user activity. Four ".pdf" control files, part of the scenario preparation, served as the artifacts of user activity. "Bravo" had a single virtual disk configured in thick-provisioned, lazy zeroed mode, which is the default mode for ESXi.

Test #3: Thin-provisioned virtual disk. The goal of test #3 was to recover the VM, "Charlie," from the VMFS file system and analyze it to identify forensic artifacts of user activity. Four ".pdf" control files, part of the scenario preparation, served as the artifacts of user activity. "Charlie" had a single virtual disk configured in thin-provisioned mode.

Scenario preparation. The testing scenario consisted of the same series of actions for each VM. Table 4 lists the four ".pdf" control files used in the tests and the MD5 hash value of each file.

Table 4

Control File Names and MD5 Hash Values

File Name	MD5 Hash Value
ControlDoc1.pdf	809176d892d68b147bdeb8ad3aba80bd
ControlDoc2.pdf	41e40c445740b974ff428cfc772cea4b
ControlDoc3.pdf	cf6121ff90ad66330dc346e2d7455a63
ControlDoc4.pdf	40885cd433950c38ce4af661d1810c33

Note. This table lists the filename and MD5 hash value of each control file used in the testing.

The MD5 hash value in Table 4 refers to the output of the MD5 one-way cryptographic hash function. One-way hash functions, which Pfleeger and Pfleeger explain in the fourth edition of *Security in Computing* (2007), are functions that are easy to compute, but whose inverse is more difficult or impossible to compute (p. 79, Cryptographic Hash Functions, para. 4). Hashing, or applying a one-way hash routing to a data stream, always produces the same result as long as the data stream and the hash algorithm remain unchanged. During the testing, hashing the image files before and after analysis ensured that, if the hash values matched, then no changes occurred to the image files.

The following sequence of events prepared the test VMs:

1. Executed Internet Explorer from the Windows desktop;
2. Microsoft Bing search for, "Google Chrome";
3. Downloaded and installed Google Chrome;
4. Google search for, "PDF reader," using Google Chrome;
5. Downloaded and installed Foxit Reader 7.0.6.2216.
6. Using Google Chrome, saved "ControlDoc1.pdf" from http://10.32.128.142/control to C:\Users\User\Desktop;

7. Using Google Chrome, saved "ControlDoc3.pdf" and "ControlDoc4.pdf" from http://10.32.128.142/control to C:\Users\User\Downloads;
8. Using Google Chrome, accessed "ControlDoc2.pdf" from http://10.32.128.142/control and opened with the Foxit Reader extension for Google Chrome;
9. "ControlDoc1.pdf" sent to the Recycle Bin;
10. Deleted "ControlDoc3.pdf" and "ControlDoc4.pdf."

Following the completion of the VM configuration, the VMware vSphere Client gracefully shutdown the ESXi host. Dc3dd created working-copy and preservation-copy raw images, calculated the MD5 hash value of the original source media, and both images, verified the MD5 hash values after image creation, and saved the log output to the file, "/media/hdd/capstone/preserve/VMD_physical_20150305_restore.log. The restoration images, named "VMD_physical_20150305_restore.raw" for the working-copy and "VMD_physical_restore.preserve" for the preservation-copy, served as a comparison and restore point in the event further testing needed to occur which compared the disk image prior to the VMs' deletion to data recovered after the VMs' deletion. The ESXi host booted up following the restore image creation and the VMware vSphere Client deleted the VMs, "Alpha," "Bravo," and "Charlie," from the VM datastore.

Analysis

Three phases comprised the research analysis: the collection phase, examination phase, and analysis phase. This section presents the execution of the methodology throughout these three phases and the findings observed. Based on the findings, testing of additional hypotheses offered an opportunity to identify alternate methods of accomplishing the goals of the analysis where shortcomings were noted.

Collection Phase

The steps to collect the disk image of the VM datastore physical disk included: shutting down "vmhost21," removing the VM datastore physical disk and connecting the disk to the Etekcity USB2.0/eSata Dual HDD docking station. The Etekcity docking station connected to the forensic workstation VM, "Ubuntu 14 Forensics VM," via USB2.0. The mount point /media/hdd on the "Ubuntu 14 Forensics VM" mounted the target disk, WD Green, 3.0TB. Under the "/media/hdd/capstone" parent directory, the "preserve" path stored all preservation-copies of evidence, while the "working" path stored all the working-copies of evidence.

The collection phase for the three tests included the collection of a single data source, a bit-level image of the VM datastore physical disk from, "vmhost21." Dc3dd created working-copy and preservation-copy raw images, calculated the MD5 hash value of the original source media, and both images, verified the MD5 hash values after image creation, and saved the log output to the file, "/media/hdd/capstone/preserve/VMD_physical_20150305.log". (See *Figure 16* in Appendix A for the contents of "VMD_physical_20150305.log").

Examination Phase

Reducing the amount of data for analysis is one of the main purposes of the examination phase. Extracting a VMFS volume image from the physical image, "VMD_physical_20150305.raw," reduced the amount of data needed to examine for file carving.

Gdisk identified the partition boundaries of the partitions within the VMFS datastore physical disk image. Gdisk identified the sector size as 512 bytes and the only partition on the disk with a starting sector of 2,048 and ending sector of 167,774,207 as shown in Figure 1.

```
Found valid GPT with protective MBR; using GPT.
Disk VMD_physical_20150305.raw: 390721968 sectors, 186.3 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): A3EB8578-B863-41CD-AFAC-5DFB090A4B53
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 390721934
Partitions will be aligned on 2048-sector boundaries
Total free space is 222949741 sectors (106.3 GiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048       167774207   80.0 GiB    FFFF
```

Figure 1. VMFS volume geometry. This figure shows the output of the command, " gdisk –l /media/hdd/capstone/working/VMD_physical_20150305.raw", with 512-byte logical sectors, a volume start sector of 2048 and volume end sector of 167774207.

A review of the contents of sector 2,048 in the "VMD_physical_20150305.raw" disk image with xxd showed that it was blank, as was every sector from 2048 until 4096. Next, xxd allowed for a manual review of the VMFS starting sector. As discussed in the literature review section, "Forensic Analysis of VMFS," the signature, "0xc001d00d," for the VMFS volume appeared at sector 2048 of the volume. Using a system with little-endian architecture, such as the test ESXi host, "vmhost21," the signature used the little-endian convention, meaning that the least significant byte was in the left-most, or largest, address location. The little-endian representation of 0xc001d00d was, "0d d0 01 c0," so the signature value to identify the VMFS volume within the "VMD_physical_20150305.raw" image was, "0x0dd001c0." Note that the

hexadecimal characters used to create the signature in big-endian notation resemble the phrase, "cool dood," which can make it easier for examiners to remember the signature value of a VMFS volume.

According to the gdisk output from the "VMD_physical_20150305.raw" image, the VMFS volume started at sector 2048, or offset 0x100000. Offset 0x200000 of the "VMD_physical_20150305.raw" physical disk image represented offset 0x100000 of the VMFS volume. The VMFS signature appeared at offset 0x100000 of the volume image, or 0x200000 of the physical image. The remainder of the VMFS volume info header followed the signature string immediately. Xxd retrieved volume information from "VMD_physical_20150305.raw" by referencing the VMFS volume info structure from the vmfs-tools source code, as explained in the Methodology section under Data Collection and Analysis Tools, vmfs-tools. (See *Table 9* in Appendix B for the volume info retrieved from "VMD_physical_20150305.raw").

The partition's starting sector of 2048 was the starting point for the volume image extract. To determine the partition length in sectors, the calculation of subtracting the starting-sector value from the ending sector value, resulted in a partition length of 167,772,159 sectors. The tool dcfldd carved the VMFS volume from "VMD_physical_20150305.raw" and saved preservation-copy image, "Boston_volume_20150305.raw.preserve," and working-copy volume image files, "Boston_volume_20150305.raw." Dcfldd calculated the MD5 hash value of the bit stream synchronously with the image file creation (See *Figure 17* in Appendix A for the dcfldd command syntax and output). MD5sum calculated the md5 hash values of the working-copy and preservation-copy volume images and confirmed to match the dcfldd output MD5 hash value. The output of the md5sum calculations confirmed the volume image's MD5 hash value matched the original source MD5 hash value, as calculated by dcfldd, confirming that the image files

matched the bit stream of 167,772,159 sectors starting at sector 2,048 within the physical disk image, "VMD_physical_20150305.raw."

Several files of importance were necessary to extract from the volume image in order to recreate or analyze a VM. The files, listed by order of importance, with number one being most important, were:

1. "-flat.vmdk" virtual disk file;
2. ".vmdk" virtual disk configuration file;
3. ".vmx" configuration file;
4. VM supporting files (if present): ".vmsd," ".vmsn," ".vswp," ".vmss," ".nvram," and ".log;"

An analysis of the remaining active VM files gave clues to help extract the deleted VM files. The volume image file was mounted on "Ubuntu 14 Forensics VM" using a tool from the vmfs-tools suite, vmfs-fuse. The command, "vmfs-fuse /media/hdd/capstone/working/Boston_volume_20150305.raw /media/vmfs," mounted the "Boston_volume_20150305.raw" image as a read-only volume to /media/vmfs.

Each VM file type contained specific characteristics that helped identify the deleted files from the deleted VMs. The ".log," ".vmx," ".vmxf," ".vmdk," and ".vmsd" files were text based and did not contain any file type signature. Log files usually hold the most information about the VM, so the ".log" format was the first file type analyzed.

When reviewing the log files from the Delta VM, the first line of each log file contained similar text, not repeated anywhere else in the file. The first line of each log file contained the phrase, ": Log for VMware ESX pid=", which was sufficiently unique to use as a log file signature. Similarly, each log file ended with a string believed to be unique enough to use as an

end-of-file identifier. Each log file ended with the text "VMX has left the building: 0". Assuming the zero at the end of the phrase was an error code that could contain values other than zero, the end-of-file phrase used for carving allowed for other values as the last digit by specifying a wildcard, "?" in place of the last digit in the foremost configuration.

Using a similar method to define the start and end of each log file, and a custom foremost configuration file that contained configurations for VMware ".log," ".vmx," ".vmxf," ".nvram," and ".vmdk" files. Some trial-and-error carving led to the foremost configuration used to run foremost. (See *Figure 18* in Appendix B for a screenshot of the foremost_vmware.conf configuration). Using foremost and the custom configuration file, the command "foremost –c foremost_vmware.conf Boston_volume_20150305.raw" carved VMware ".log," ".vmx," ".vmxf," ".nvram," and ".vmdk" files, from the "Boston_volume_20150305.raw" image. Foremost carved 104 files from the "Boston_volume_20150305.raw" image. MD5sum calculated the MD5 hash value of each carved file. Files with matching MD5 hash values had all extra copies moved to a "duplicates" folder so only one version of each file remained in the working directory. Each of the carved files contained, somewhere in the text, the name of the VM that the file belonged to with the exception of some ".nvram" files. A search for each VM's name in each of the carved files enabled quick sorting of the files by VM. During the analysis phase for each VM, these files helped to reconstruct the VM within ESXi.

In each ".vmdk" file was a section named, "Extent description," which showed the size, in sectors, and file name, of each virtual disk file associated with the VM. Knowing the size of the virtual disk is useful when carving "-flat.vmdk" from the VMFS volume. In the "Extent description" section of "00490890.vmdk" the size of "Alpha-flat.vmdk" was shown as 25,165,824 logical sectors, which equaled exactly 12GB with 512-byte sectors.

Analyzing the "Delta-flat.vmdk" file showed that the "-flat.vmdk" file was a complete virtual hard drive image, starting with the master boot record and containing the partition table and all partitions within the virtual disk. Indications of master boot record (MBR) sectors helped to identify the beginning sector of "–flat.vmdk" files. Carrier (2005) detailed the data structure of the MBR as a 512-byte structure with the first 446 bytes reserved for the boot code and the final 66 bytes containing the partition table entries and the MBR signature value "0x55AA" (p. 88, para. 3). The MBR signature value identified sectors on the "Boston" volume that contained MBRs, as indicators of the presence of a virtual disk.

In order to eliminate as many matches to the "55AA" pattern as possible that were not associated with the MBR, the signature search only examined bytes 256 and 512 in each sector. The tool xxd produces an ASCII representation of a hexdump output for a data stream. The xxd output is configurable in the number of bytes to display in the hexdump. The last byte in the xxd output, followed by two spaces, is the only column in the xxd output with two spaces following it. This unique characteristic of the last byte in the xxd output allowed for easy identification of the last byte by searching for the trailing double-space.

Setting the number of displayed bytes to 256 and filtering the results through a grep basic expression searching for "\s55aa\s\s" ensured that only matches in bytes 255-256 and 511-512 in each sector could match the search criteria. The 256-byte search boundary aligned with the 512-byte sector boundary at every second occurrence, which greatly minimized the number of false matches, but still produced many false matches to the MBR signature. Filtering for likely matches within the partition table, the results narrowed to four MBR signatures on the "Boston" volume.

Focusing on expected values within the partition table prior to the MBR footer, the LBA start sector of the first partition provided an opportunity to filter, given the limited possibilities of values within that LBS start sector field. As Mueller (2013) explained, Windows Vista and later aligns partitions along 2,048-sector boundaries (p. 475, para. 6). An additional Perl-compatible regular expression pattern match "00(([0-9A-F]8)|([1-9A-F]0))\s0000\s([0-9A-F]{4}\s){26}55aa\s\s" applied to the previous command's output. The added regular expression searched for matching word values between "0x0008" and "0x00F0" in multiples of 2048, beginning 56 bytes prior to the "55aa" MBR signature, which aligned with the LBA start sector of the first partition within the partition table. Figure 2 shows the command used to identify the byte locations of the MBR signatures within the "Boston" volume and the first result, matching a partition table entry and MBR footer.

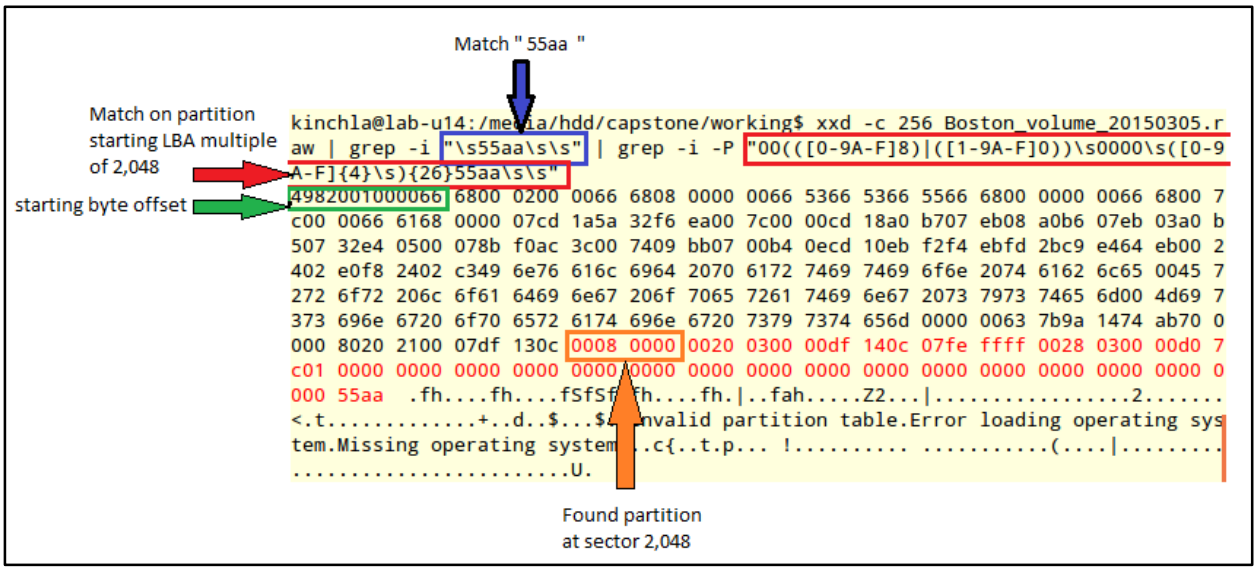

Figure 2. MBR sector identification. This figure shows the xxd command filtered through grep to produce output showing the last 256 bytes of an MBR sector. The red characters show the portion of the output that matched the last grep expression. The formatting of the xxd output made it appear that the starting byte (in the green box) was, " 0x4982001000066," but the starting byte was actually " 0x498200100" and the last four digits, "0066," were the displayed contents of bytes 0x498200100-0x498200101.

Analyzing the partition table within each MBR enabled the determination of disk extents within the VMFS volume. Each MBR match within the xxd output started at 256-bytes past the

sector start. Subtracting 256 bytes, or 0x100, from the starting position of the xxd output resulted in the MBR's starting byte. With 512-byte sectors on the "Boston" volume, dividing the starting byte by 512 produced the starting sector for the virtual disk. Figure 3 shows the locations of the partition-starting sector and partition length for the second partition within the partition table.

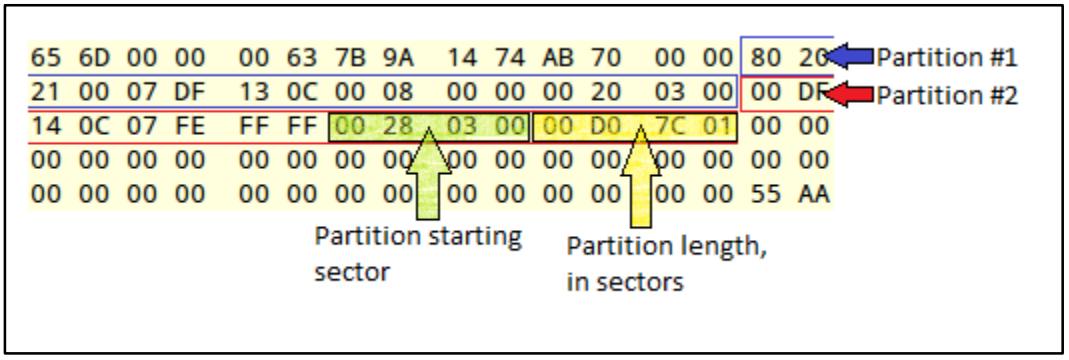

Figure 3. Partition table entries. This figure shows the partition-starting sector and partition length values for the second partition in the partition table, which identified the ending sector of the virtual disk file.

Adding the partition-starting sector to the partition length, then adding the MBR sector on the "Boston" volume produced the virtual disk-ending sector on the "Boston" volume. Subtracting the MBR sector from the virtual disk-ending sector produced the virtual disk length in sectors. The sector values displayed in each virtual disk file's partition table are relative to the virtual disk file, not the volume on which they reside. Dcfldd carved each of the four virtual disk files using the calculations described above and synchronously calculated the MD5 hash value of each virtual disk file with the carving process. Table 5 shows the extents of each virtual disk file within the "Boston" volume.

Table 5

Virtual disk file extents, carved file names

Volume starting sector	**Volume ending sector**	**Number of sectors**	**File carved as**	**Md5 Hash value**
38,539,264	63,703,040	25,163,776	38539264-flat.vmdk	9f37811117150dab01c178d0b5829894

63,705,088	88,868,864	25,163,776	63705088-flat.vmdk	f505f1086280e6cfbd293502e8ef024f
97,521,664	122,685,440	25,163,776	97521664-flat.vmdk	70f3b4801e19123c4e525e6dfccaf5de
131,338,240	156,502,016	25,163,776	131338240-flat.vmdk	651b2f876ae3e3782642cec8bf72ffb4

Note. This table shows the extents of each " –flat.vmdk" file on the " Boston" volume, the MD5 hash value of each virtual disk file, and the carved virtual disk file name.

Comparing the sector length of the virtual disks derived from the partition tables to the sector length shown in each ".vmdk" file showed that each ".vmdk" file was one VMFS5 block longer than the end of the last partition. Analyzing the active virtual disk file, "Delta-flat.vmdk," confirmed that there was an additional 2048 sectors, or one VMFS5 block, between the end of the last partition within the virtual disk file and the end of the virtual disk file.

At this point in the examination, it was unknown which virtual disk file belonged to which VM. It would have been possible to compare the number of sectors of the carved files from Table 5 to the values in the extent description section of the ".vmdk" files, but in the test scenarios, each VM had the same size virtual disk so the number of sectors could not distinguish the virtual disks used in this research.

Many of the files that foremost carved from disk overlapped disk locations where the VMFS metadata files ".fdc.sf" and ".sbc.sf" resided. Each of the carved ".vmdk" files resided within the metadata file ".fdc.sf," and each of the carved ".vmx" files resided within the ".sbc.sf" metadata file. With the "Boston_volume_20150305.raw" image mounted to "/media/vmfs/" using vmfs-fuse on "Ubuntu 14 Forensics VM," ".fdc.sf" and ".sbc.sf" files, xxd reviewed the contents of each system, or metadata, file and confirmed the presence of each carved ".vmdk" and ".vmx" resident within the metadata files. After deleting the VMs, the ".vmdk" and ".vmx" files remained resident within the ".fdc.sf" and ".sdc.sf" on the VMFS file system.

Figure 4 shows a disk view chart with the locations of the carved files on the "Boston" volume. The figure shows that the virtual disk files "Alpha-flat.vmdk," "Bravo-flat.vmdk," and "Delta-flat.vmdk" had uninterrupted contiguous blocks allocated to the files, which enabled successful recovery of the files. Using dcfldd to carve consecutive sectors from each file's starting point resulted in complete recovery of the virtual disk files. "Charlie-flat.vmdk," on the other hand, had fragmentation throughout the virtual disk file. As a result, the "Charlie-flat.vmdk" file failed to recover successfully when carving consecutive sectors from the "Boston" volume.

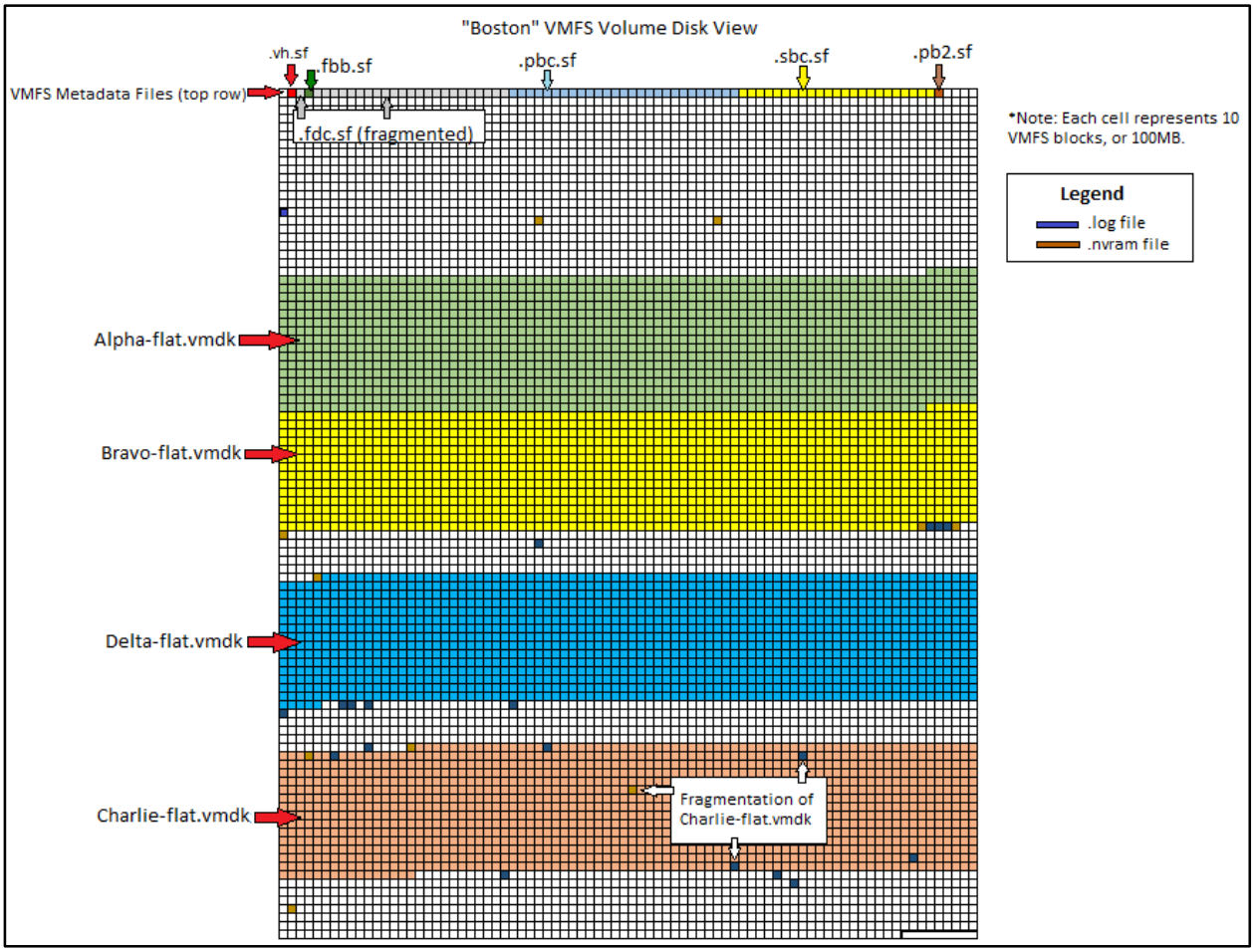

Figure 4. Disk view of "Boston" volume. This figure shows a visual representation of the file allocation across the "Boston" VMFS volume. The figure shows the contiguous sectors that comprised the "Alpha-flat.vmdk," "Bravo-flat.vmdk," and "Delta-flat.vmdk" virtual disk files and the fragmentation of "Charlie-flat.vmdk." Each cell within the figure represents 10 VMFS blocks.

Xxd retrieved the file descriptor details and sub-block metadata for "Charlie-flat.vmdk." The metadata files contained block usage information about the blocks used by each file within the volume. The vmfs-tools source code, provided the information necessary to identify the blocks used by the "Charlie-flat.vmdk" file from within the restoration image, prior to deleting the "Charlie" VM from the VMFS volume. (See *Table 12* in Appendix E for the file descriptor and sub-block metadata associated with "Charlie-flat.vmdk").

Figure 5 shows the values within the file descriptor, ".fdc.sf," system file and sub-block system file, ".sbc.sf," from the restoration image of the "Boston" volume. The values represent the file descriptor and sub-block system entries for "Charlie-flat.vmfs" before deletion of the "Charlie" VM. The file named "Charlie-flat.vmdk within ".sbc.sf," had an inode ID of "44 8A 80 04." The inode ID "44 8A 80 04" within ".fdc.sf" provided the file size and number of VMFS blocks allocated to "Charlie-flat.vmdk."

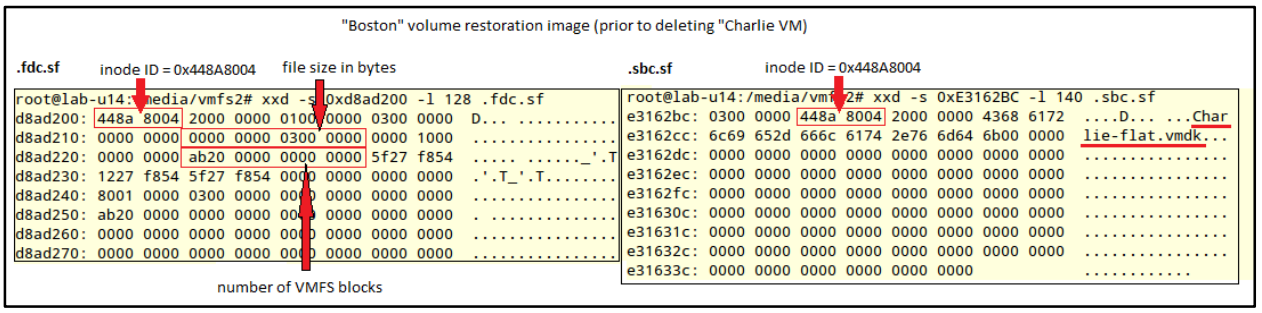

Figure 5. Metadata of active file. This figure shows the metadata retrieved for " Charlie-flat.vmdk" from the " .fdc.sf" and " sbc.sf" system files. This data showed the values prior to deleting the "Charlie" VM from the VMFS volume.

The ".fdc.sf" and ".sbc.sf" system files from the restoration image compared to the ".fdc.sf" and "sbc.sf" system files from the "Boston_volume_20150305.raw" image showed that deleting a file purges the entry for the file within ".sbc.sf." Further, deleting the "Charlie-flat.vmdk," file kept the inode ID entry in the "fdc.sf" file, but zeroed out the values for the file size and number of blocks allocated to the file, among other fields. Figure 6 shows the ".fdc.sf" contents for inode ID "44 8A 80 04" following the deletion of "Charlie-flat.vmdk."

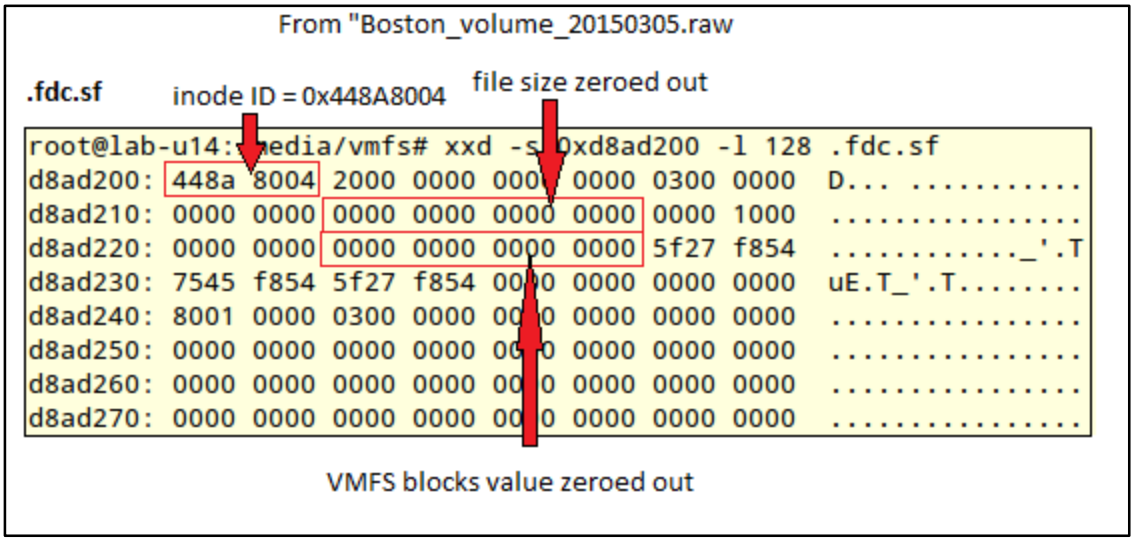

Figure 6. Metadata of deleted file. This figure shows that the file size and number of VMFS blocks fields, among other fields, zeroed out following the deletion of "Charlie-flat.vmdk."

The changes made to the VMFS system files following a file's deletion from the file system make it highly unlikely to recover a file comprised on a non-contiguous data stream. Once the file system deletes the file and updates the supporting system files, VMFS clears many of the fields necessary to recover non-contiguous files. During the examination phase, the "Charlie-flat.vmdk" file was unrecoverable using consecutive sector file carving techniques and could not be recovered based on information obtained through the VMFS system files. For this research, the "Charlie-flat.vmdk" file was unrecoverable.

Analysis Phase

Test #1: Thick-provisioned, eager zeroed virtual disk. The goal of test #1 was to recover the VM, "Alpha," from the VMFS file system and analyze it to recover the four control files from the test scenario. "Alpha" had a single virtual disk configured in thick-provisioned, eager zeroed mode, which is the default mode for ESXi.

Virtual disk identification. A tool from The Sleuth Kit, mmls, inspected each "–flat.vmdk" carved file and determined that each had the same partition layout, with a 24,956,928-sector, or 11.9 GB NTFS volume starting at sector 206,848. Each "-flat.vmdk" file, mounted one

at a time to "/media/test" as a read-only volume at offset 105,906,176, provided file system browsing of the active files in the 11.9 GB NTFS volume in each virtual disk file. Regripper analyzed the system registry hive contained within the NTFS volume of each "–flat.vmdk" file to determine the computer name associated with each file. Each "-flat.vmdk" file was unmounted from "/media/test" following the identification of the computer name from the system registry hive. The computer name found in the "Windows\System32\config\SYSTEM" registry hive of "38539264-flat.vmdk" was, "Alpha," as shown in Figure 7. The "Alpha" VM was the subject of test #1.

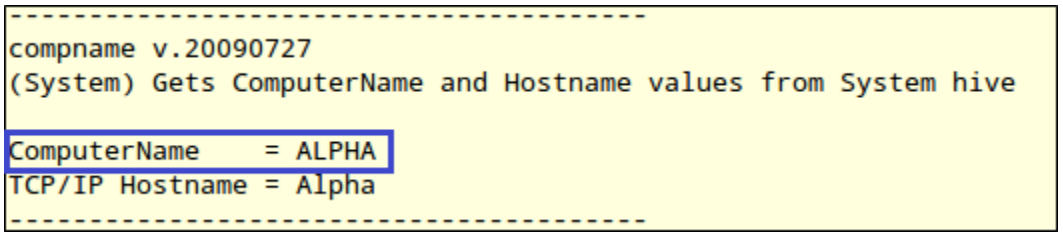

Figure 7. Regripper identified "Alpha" VM. This figure shows the Regripper output of the compname plugin run on the "Windows\System32\config\SYSTEM" file in the "38539264-flat.vmdk" virtual disk. The computer name configured within the registry was, "ALPHA."

Control file recovery. The second partition from the "38539264-flat.vmdk" virtual disk file, mounted as a read-only volume at "/media/alpha" on "Ubuntu 14 Forensics VM," enabled file system browsing of the "Alpha" VMs' system drive. The command "mount –o ro,offset=$((206848*512)) /media/hdd/capstone/working/38539264-flat.vmdk /media/alpha" mounted the volume to "/media/alpha". The partition start sector returned by mmls, multiplied by the sector size in bytes, produced the offset value.

Browsing to the relative path of "/$Recycle.Bin/ S-1-5-21-1394914191-3772142883-1012848598-1000/" showed a ".pdf" file, "$RMWUZ26.pdf." MD5sum calculated the MD5 hash value of "$RMWUZ26.pdf" as "809176d892d68b147bdeb8ad3aba80bd," the same hash value as ControlDoc1.pdf. The file, opened in okular using the command "okular

$RMWUZ26.pdf" displayed the contents of "ControlDoc1.pdf." The contents simply read, "Control Document #1".

Having the known MD5 hash value for "ControlDoc2.pdf" allowed for the use of MD5sum to search for a matching MD5 hash value within Alpha's Google Chrome cache directory. From within the "/media/alpha/Users/User/AppData/Local/Google/Chrome/User Data/Default/Cache" folder, the command "md5sum * | grep –i "41e40c445740b974ff428cfc772cea4b" returned a single file name, "f_00004a". The file, opened in okular using the command "okular f_00004a" displayed the contents of "ControlDoc2.pdf." The contents simply read, "Control Document #2".

With a single file definition, for "pdf" files, enabled in the foremost configuration, foremost attempted to recover the deleted files, "ControlDoc3.pdf," and "ControlDoc4.pdf," from unallocated space on the Alpha virtual disk. Foremost completed with 94 ".pdf" files recovered, but none of the recovered ".pdf" files matched the MD5 hash values of "ControlDoc3.pdf" or "ControlDoc4.pdf."

Using hexedit to search for the control file's identifiable hexadecimal string, "2F417574686F7228422E204B696E63686C6129", which is the hexadecimal representation of the ASCII string, "/Author(B. Kinchla)", a fragment of "ControlFile3.pdf" was located at sector offset 24,346,352 within the "Alpha-flat.vmdk" virtual disk file. Using dcfldd to extract each sector and hash the results with md5sum determined that "ControlFile3.pdf" contained 8,192 bytes, or two contiguous sectors, before the file fragmented to other blocks within the virtual file system. Figure 8 shows the commands used to determine the offset where fragmentation occurred.

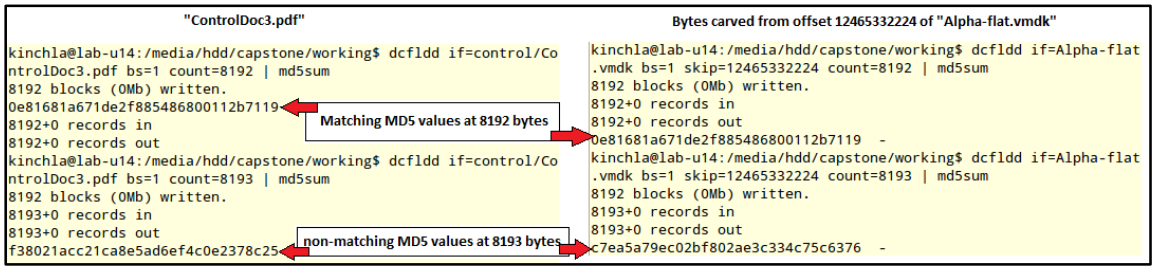

Figure 8. Determining fragmentation offset with dcfldd and md5sum. This figure shows md5sum calculating the MD5 hash value of dcfldd output. The commands on the left calculated the MD5 hash value of "ControlDoc3.pdf" for sectors 0-8191 and 0-8192. The commands on the left calculated the MD5 hash value of 8192 bytes beginning at byte offset 12,465,332,224 and of 8193 bytes beginning at byte offset 12,465,332,224. The results show matching MD5 hash values at 8192 bytes, but differing MD5 hash values at 8193 bytes.

Continuing to search with hexedit for the string "2F417574686F7228422E204B696E63686C6129" returned only three results for "ControlDoc1.pdf," "ControlDoc2.pdf," and the first fragment of "ControlDoc3.pdf." Manually searching within hexedit returned no matches for "ControlDoc4.pdf." The research did not continue further searching for fragments of "ControlDoc4.pdf."

VM reconstruction. Identification of the most current copies of each ".vmx," ".vmxf," ".vmdk," and "-flat.vmdk," was the next challenge to reconstructing the VM in its most recent state (See *Table 14* in Appendix C for an inventory of recovered files belonging to the Alpha VM). Because the ".log" files contained timestamps, they provided the best clue to finding the latest configuration of the VM prior to deletion. Using the command, "head –n 1 *.log", the output was the first line of each ".log" file. The log file 156733440.log had the latest timestamp, at "2015-03-05T09:44:16.321Z."

Of the files that foremost carved during the examination phase, there were nine ".vmx" files belonging to the "Alpha" VM. The contents of "15633440.log" identified the ".vmx" file that associated with the "15633440.log" file. Using a series of "diff -a" commands to find differences between the various ".vmx" files and comparing those differences to the contents of "156733440.log," identified the ".vmx" file that matched the log file. "01559408.vmx" matched

the contents of "156733440.log" and, therefore was the ".vmx" file last in use by the "Alpha" VM.

With the recovery of the latest ".vmx" file from the "Alpha" VM, combined with the ".vmxf," ".vmdk," and "-flat.vmdk" files, there existed the necessary files to reconstruct the VM on an ESXi system. To find the original filenames of the Alpha VM files, the ".log" file, ".vmx" file, and ".vmdk" file contained the names and file system paths for each of the deleted files. "156733440.log" provided the name of the VM directory and the ".vmx" file as "Alpha\Alpha.vmx." The ".vmx" file, "01559408.vmx" renamed to "Alpha.vmx," had its contents displayed in its entirety, within "156733440.log". "Alpha.vmx" provided filenames for the ".vmxf," file within the "extendedConfigFile" configuration parameter and ".vmdk," files, "Alpha.vmdk" and "Alpha.vmxf". "Alpha.vmdk" provided the filename for the "-flat.vmdk" file, "Alpha-flat.vmdk," within the "Extent description" section. The files "01559088.vmxf" and "38539264-flat.vmdk," renamed to "Alpha.vmxf" and "Alpha-flat.vmdk," were copied along with "Alpha.vmdk" and "Alpha.vmx" to a new folder named "Alpha" on the datastore volume, "Chicago," on the ESXi host, "vmhost21." After using scp to copy the VM files to "vmhost21," md5sum calculated the MD5 hash values for the copied files on "vmhost21." (See *Table 14* in Appendix C to reference the MD5 hash values of the Alpha VM recovered files).

Within the vSphere Client, using the Datastore Browser, right-clicking on the "Alpha.vmx" file name and selecting "Add to Inventory," invoked the "Add to Inventory" wizard. The Alpha VM was added to the ESXi server's inventory as an available VM after completing the "Add to Inventory" wizard. The "Alpha" VM failed to start, returning the error shown in Figure 9, indicating a problem with "Alpha.vmdk."

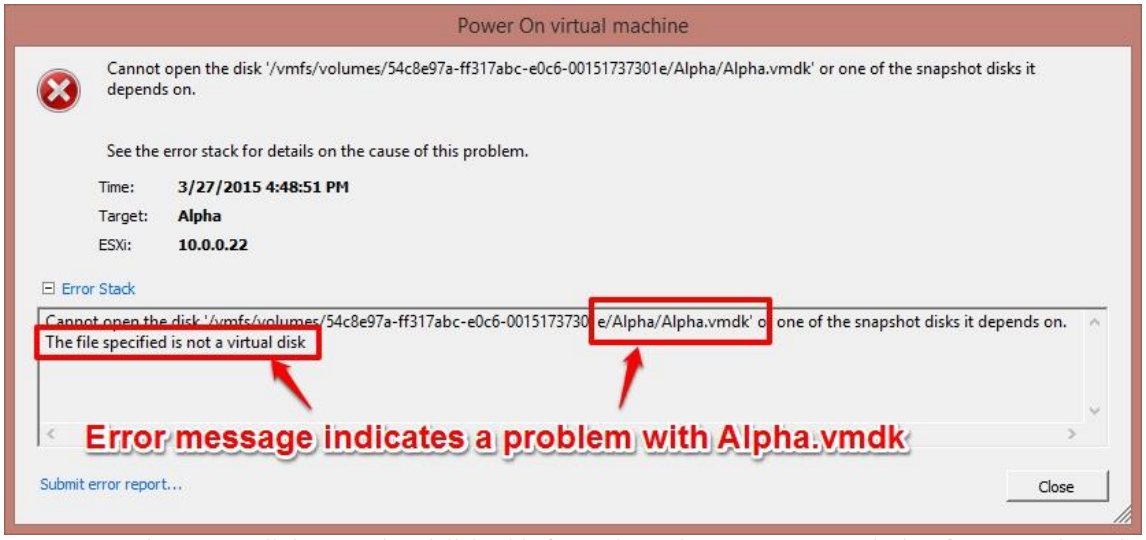

Figure 9. ESXi error - .vmdk is not a virtual disk. This figure shows the error encountered when first attempting to boot the "Alpha" VM from the reconstructed files.

Seeing no obvious problem with "Alpha.vmdk," an analysis of "Alpha-flat.vmdk," comparing "Alpha-flat.vmdk" to the active file, "Delta-flat.vmdk," from the "Boston_volume_20150305.raw" image ensued. The comparison revealed that "Delta-flat.vmdk" contained one VMFS5 block, or 2048 sectors, beyond the end of the last partition in the virtual disk image. The "Alpha-flat.vmdk" virtual disk file ended immediately following the last partition in the virtual disk and did not contain the extra block.

Comparing the sector length of the virtual disks derived from the partition tables to the sector length shown in the "extent description" section of each ".vmdk" file showed that each "-flat.vmdk" file was one VMFS5 block longer than the end of the last partition. Using dcfldd to re-carve the virtual disk file "Alpha-flat.vmdk" from "Boston_volume_20150305.raw," the command, "dcfldd bs=512 skip=38539264 count=$((25163776+2048)) hash=md5 if=Boston_volume_20150305.raw of=38539264+2048-flat.vmdk," produced a virtual disk with an additional 2048 sectors at the end of the file named, "38539264+2048-flat.vmdk." Dcfldd calculated the MD5 hash value of the extracted bit stream simultaneously with the image creation process.

The new "-flat.vmdk" file for the "Alpha" VM, "38539264+2048-flat.vmdk," was copied to the Alpha directory on the datastore volume, "Chicago," on the ESXi host, "vmhost21." Renaming the existing "Alpha-flat.vmdk" to "Alpha-flat.vmdk.old" and renaming "38539264+2048-flat.vmdk" to "Alpha-flat.vmdk," allowed the VM to boot up properly. Figure 10 shows the fully booted VM within the vSphere Client console.

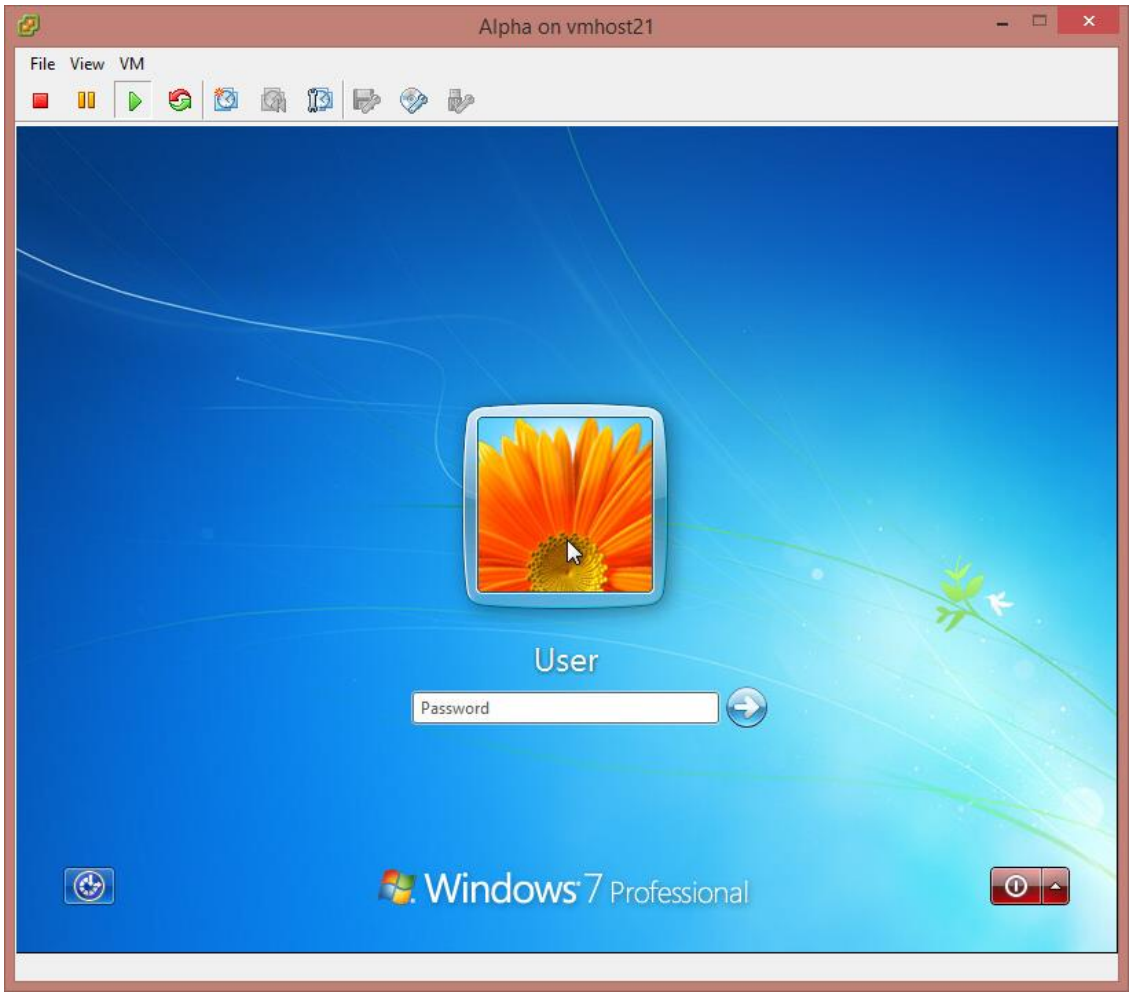

Figure 10. Successful reconstruction of "Alpha" VM. This figure shows the fully booted "Alpha" VM to illustrate the successful recovery of the VM from the VMFS volume.

Following the same process used to carve "38539264+2048-flat.vmdk," the "-flat.vmdk" files from sectors 63705088, 97521664, and 131338240, on the "Boston_volume_20150305.raw" image were re-carved with the additional 2048 sectors appended to the end of each virtual disk file. Dcfldd calculated the MD5 hash value of each

carved file simultaneously with the image file creation (See *Table 13* in Appendix B for a revised table of the MD5 hash values for the "-flat.vmdk" files with the appended 2048 sectors).

Test #2: Thick-provisioned, lazy zeroed virtual disk. The goal of test #2 was to recover the VM, "Bravo," from the VMFS file system and analyze to recover the four control files from the test scenario. "Bravo" had a single virtual disk configured in thick-provisioned, lazy zeroed mode, which is the default mode for ESXi.

Virtual disk identification. During the analysis phase of test #1, Regripper analyzed the system registry hive contained within each NTFS volume from the carved virtual disk images to determine the computer name associated with each file "-flat.vmdk" file. The computer name found in the "Windows\System32\config\SYSTEM" registry hive of "63705088+2048-flat.vmdk" was, "Bravo," as shown in Figure 11. The "Bravo" VM was the subject of test #2.

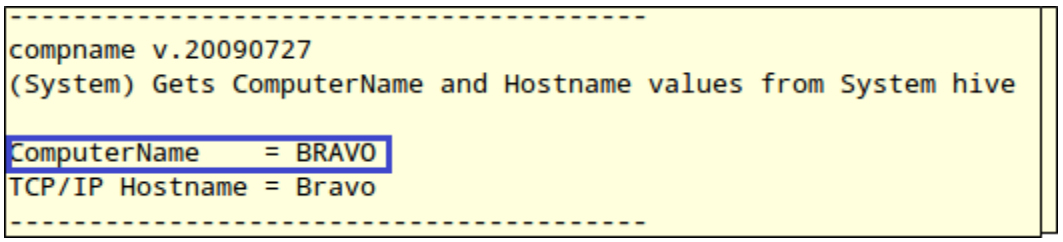

Figure 11. Regripper identified "Bravo" virtual disk. This figure shows the Regripper output of the compname plugin run on the "Windows\System32\config\SYSTEM" file in the "63705088+2048-flat.vmdk" virtual disk. The computer name configured within the registry was, "BRAVO."

Control file recovery. The second partition from the "63705088+2048-flat.vmdk" virtual disk file, mounted as a read-only volume at "/media/brav/" on "Ubuntu 14 Forensics VM," enabled file system browsing of the "Bravo" VMs' system drive. The command "mount –o ro,offset=$((206848*512)) /media/hdd/capstone/working/63705088+2048-flat.vmdk /media/bravo/" mounted the volume to "/media/bravo". The partition start sector returned by mmls, multiplied by the sector size in bytes, produced the offset value.

Browsing to the relative path of, "/$Recycle.Bin/ S-1-5-21-2592085848-1128064398-1000/", showed a ".pdf" file, "$RDMRHC9.pdf." MD5sum calculated the MD5 hash value of

"$RDMRHC9.pdf" as "809176d892d68b147bdeb8ad3aba80bd," the same hash value as ControlDoc1.pdf. The file, opened in okular using the command, "okular $RDMRHC9.pdf", displayed the contents of "ControlDoc1.pdf." The contents simply read, "Control Document #1."

Having the known MD5 hash value for "ControlDoc2.pdf" allowed for the use of MD5sum to search for a matching MD5 hash value within Beta's Google Chrome cache directory. From within the "/media/bravo/Users/User/AppData/Local/Google/Chrome/User Data/Default/Cache" folder, the command "md5sum * | grep –i "41e40c445740b974ff428cfc772cea4b" returned a single file name, "f_00004b". The file, opened in okular using the command "okular f_00004b" displayed the contents of "ControlDoc2.pdf." The contents simply read, "Control Document #2".

With a single file definition, for "pdf" files, enabled in the foremost configuration, foremost attempted to recover the deleted files, "ControlDoc3.pdf," and "ControlDoc4.pdf," from unallocated space on the Alpha virtual disk. Foremost completed with 102 ".pdf" files recovered, but none of the recovered ".pdf" files matched the MD5 hash values of "ControlDoc3.pdf" or "ControlDoc4.pdf."

Using hexedit to search for the control file's identifiable hexadecimal string, "2F417574686F7228422E204B696E63686C6129", returned only three results for "ControlDoc1.pdf," "ControlDoc2.pdf," and the first fragment of "ControlDoc3.pdf." Using the same technique described in Analysis, Analysis Phase, Test #1: Thick-provisioned, eager zeroed virtual disk, Control file recovery, determined that the search string identified only the first NTFS block of "ControlDoc3.pdf". The fragment of "ControlDoc3.pdf" differed from the original control file at byte 4097. Manually searching within hexedit returned no matches for

"ControlDoc4.pdf." The research did not continue further searching for fragments of "ControlDoc4.pdf."

VM reconstruction. Identification of the most current copies of each ".vmx," ".vmxf," ".vmdk," and "-flat.vmdk," was the next challenge to reconstructing the VM in its most recent state. (See *Table 16* in Appendix D for an inventory of recovered files belonging to the Bravo VM). Using the command, "head –n 1 *.log", the output was the first line of each ".log" file. The log file "159092736.log" had the latest timestamp at "2015-03-05T09:44:33.798Z."

Of the files that foremost carved during the examination phase, there were nine ".vmx" files belonging to the "Bravo" VM. The contents of "159092736.log" identified the ".vmx" file that associated with the "159092736.log" file. Using a series of "diff -a" commands to find differences between the various ".vmx" files and comparing those differences to the contents of "159092736.log," identified the ".vmx" file that matched the log file. "01559424.vmx" matched the contents of "159092736.log" and, therefore was the ".vmx" file last in use by the "Bravo" VM.

With the recovery of the latest ".vmx" file from the "Bravo" VM, combined with the ".vmxf," ".vmdk," and "-flat.vmdk" files, there existed the necessary files to reconstruct the VM on an ESXi system. To find the original filenames of the Bravo VM files, the ".log" file, ".vmx" file, and ".vmdk" file contained the names and file system paths for each of the deleted files. "159092736.log" provided the name of the VM directory and the ".vmx" file as "Bravo\Bravo.vmx." The ".vmx" file, "01559424.vmx" renamed to "Bravo.vmx," had its contents displayed in its entirety, within "159092736.log". "Bravo.vmx" provided filenames for the ".vmxf," file within the "extendedConfigFile" configuration parameter and ".vmdk," file within the "scsi0:0.fileName" configuration parameter, "Bravo.vmdk" and "Bravo.vmxf".

"Bravo.vmdk" provided the filename for the "-flat.vmdk" file, "Bravo-flat.vmdk" within the "Extent description" section. The files "01559264.vmxf" and "63705088+2048-flat.vmdk," renamed to "Bravo.vmxf" and "Bravo-flat.vmdk," were copied along with "Bravo.vmdk" and "Bravo.vmx" to a new folder named "Bravo" on the datastore volume, "Chicago," on the ESXi host, "vmhost21." After using scp to copy the VM files to "vmhost21," md5sum calculated the MD5 hash values for the copied files on "vmhost21." (See *Table 16* in Appendix D to reference the MD5 hash values of the Bravo VM recovered files).

Within the vSphere Client, using the Datastore Browser, right-clicking on the "Bravo.vmx" file name and selecting "Add to Inventory" invoked the "Add to Inventory" wizard. The "Bravo" VM was added to the ESXi server's inventory as an available VM after completing the "Add to Inventory" wizard. The "Bravo" VM started without error. Figure 12 shows the fully booted VM within the vSphere Client console

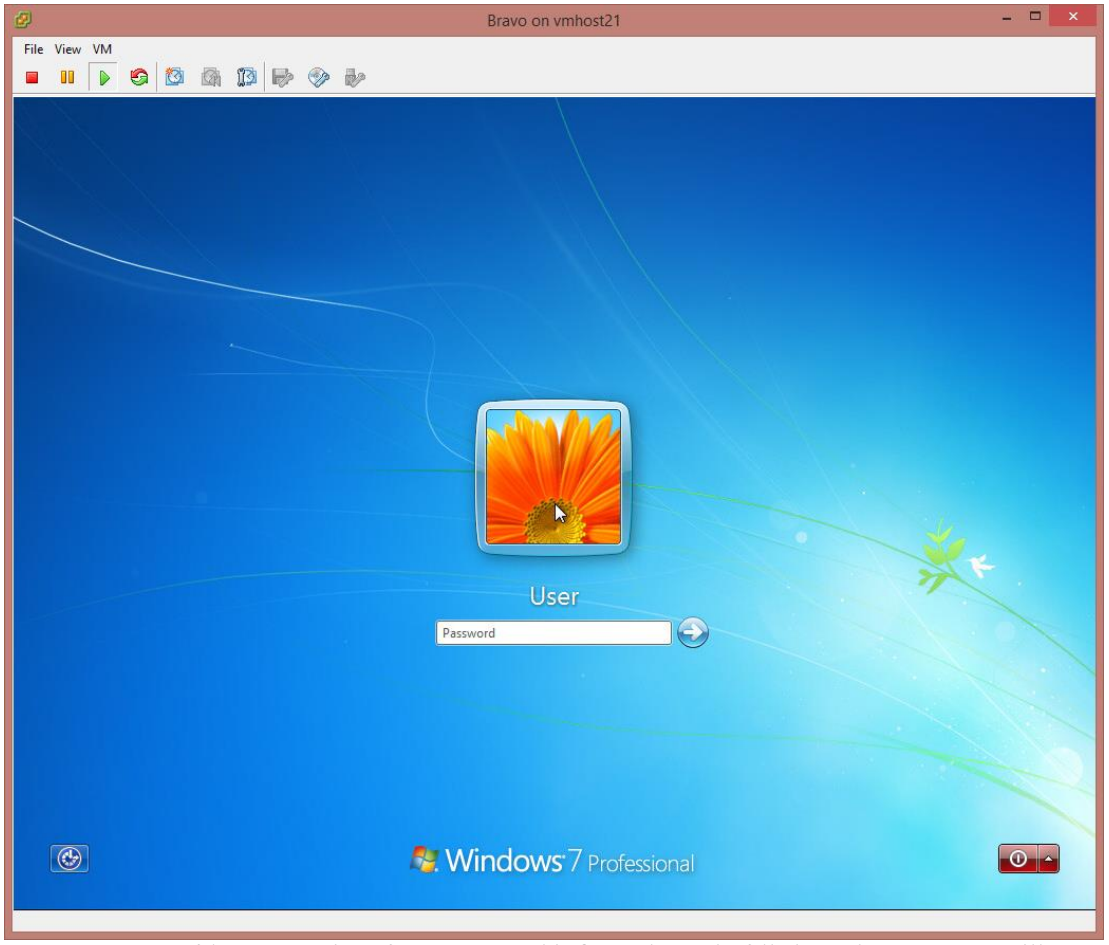

Figure 12. Successful reconstruction of "Bravo" VM. This figure shows the fully booted "Bravo" VM to illustrate the successful recovery of the VM from the VMFS volume.

Test #3: Thin-provisioned virtual disk. The goal of test #3 was to recover the VM, "Charlie," from the VMFS file system and analyze to recover the four control files from the test scenario. "Charlie" had a single virtual disk configured in thin-provisioned mode, which is the default mode for ESXi.

Virtual disk identification. During the analysis phase of test #1, the researcher mounted each of the carved "-flat.vmdk" files at sector 206848 as an NTFS volume to /media/test on "Ubuntu 14 Forensics VM." Three of the four virtual disk files mounted successfully as an NTFS volume at a sector 206848 of the virtual disk. The fourth disk file contained no data at sector 206848 and therefore was unable to be analyzed as a virtual disk file. Regripper analyzed the system registry hive contained within each NTFS volume to determine the computer name

associated with each file "-flat.vmdk" file. The registry hives for "Alpha," "Bravo," and "Delta" were found, but the virtual disk file belonging to the "Charlie" VM was the disk file that was missing the Windows system partition.

Using xxd to open "VMD_physical_20150305_restore.raw" to offset 67,246,227,456, the first byte of the "Charlie" virtual disk file from the physical volume, showed an MBR sector with a complete partition table. The partition table contained two partitions; the first partition started at sector 2,048 (relative to byte 67,246,227,456 as the start of sector 0) and had a length of 204,800 sectors. The second partition started at sector 206,848 and had a length of 24,956,928 sectors, or 11.9 GB.

Using xxd to review relative sector 206,848, offset 67,352,133,632 within the restoration image, showed that no partition existed at that location. Because no partition start sector existed at relative offset 206,848 within the restoration image, non-contiguous sectors on the VMFS volume comprised the "Charlie" VM's virtual disk prior to deleting the virtual disk file. Therefore, file-carving techniques that rely on contiguous allocation of data would not be able to recover the "Charlie" virtual disk file after deletion.

As discussed in the analysis, examination phase, deleting the "Charlie-flat.vmdk" file caused the clearing of metadata from the system files for the "Charlie-flat.vmdk" file. The blocks allocated to the "Charlie-flat.vmdk" file were unknown with the metadata cleared. Therefore, file carving from the individual blocks was not reasonably possible to recover the complete virtual disk file for the "Charlie" VM from the "Boston" volume.

Control file recovery. Each of the control files resided within the "Charlie-flat.vmdk" virtual disk file prior to deleting the "Charlie" VM. Without successful recovery of the "Charlie-flat.vmdk," virtual disk file, there was no attempt to recover the control files. Unknown file

extents for "Charlie-flat.vmdk" made it impossible to know if a file carved directly from the VMFS volume had previously been contained within the extents of "Charlie-flat.vmdk.

VM Reconstruction. To reconstruct each VM, the virtual disk file was necessary. The "Charlie-flat.vmdk" was unrecoverable using the techniques in this research. Therefore, the "Charlie" VM was not reconstructed.

Hypotheses. Testing of three hypotheses occurred to explain some of the results observed. The cause of fragmentation of "Charlie-flat.vmdk" prompted the testing of two hypotheses. The first hypothesis, referred to as the "non-consecutive writes" hypothesis offered that thin-provisioned virtual disk files have a higher likelihood of fragmentation due to non-consecutive writes to disk during virtual disk file expansion. For thin-provisioned virtual disks which have not been fully allocated within the VMFS volume, unallocated blocks contiguous to, or addressed closely to the last allocated block of the virtual disk file are candidates for any ESXi process to write to. Doing so would cause fragmentation of the thin-provisioned virtual disk when it requires allocation of more blocks. The second hypothesis, referred to as the "allocation percentage" hypothesis offered that any virtual disk file stands a higher chance of encountering fragmentation as less allocated blocks remain within the volume. The inability to recover the deleted control files from any of the deleted VMs prompted the testing of a third hypothesis, referred to as the "control file" hypothesis. The "control file" hypothesis offered that chances for recovery of deleted files within the VM decreased as unallocated space within the virtual disk file decreased.

The testing of the "allocation percentage" hypothesis required an additional testing environment be created using the same scenario as the analysis testing scenario, except with the VMs provisioned in a different order. The testing of the "control file" hypothesis

required an additional testing environment be created using the same scenario as the original testing scenario, except the VMFS volume was doubled in capacity and each VM's virtual disk file was configured with 50% more capacity. The testing of the "non-consecutive writes" hypothesis used data from each of the three testing environments to draw conclusions.

Non-consecutive writes hypothesis. The non-consecutive writes hypothesis stated that thin-provisioned virtual disk files have a higher likelihood of fragmentation due to non-consecutive writes to disk during virtual disk file expansion. Testing of this scenario required an additional testing environment, which was identical to the analysis-testing environment except the provisioning of VMs, occurred in a different order to change the physical location of the virtual disk files within the VMFS volume.

The VM datastore drive, forensically wiped with zeroes using dc3dd prior to configuring the datastore, held a single 80GB VMFS5 volume labeled "Dallas." The "Dallas" volume had a formatted capacity of 79.75GB. Four VMs had their files stored on the "Dallas" volume. Each of the VMs had the same software and virtual hardware configuration with the exception of the virtual disk allocation method. The "Charlie2" VM had a thin-provisioned virtual disk. The "Bravo2" VM had the ESXi default method of thick-provisioned, lazy zeroed virtual disk. The "Alpha2" VM had a thick-provisioned, eager zeroed virtual disk. The "Delta2" VM had a thick-provisioned, lazy zeroed virtual disk. Table 6 lists the configuration of the four test VMs.

Table 6

Non-Consecutive Writes Test VM Configurations

	VM 1	**VM 2**	**VM 3**	**VM 4**
VM Name	Charlie2	Bravo2	Alpha2	Delta2
VM working location	[Dallas]/Charlie2	[Dallas]/Bravo2	[Dallas]/Alpha2	[Dallas]/Delta2

CPUs	1	1	1	1
RAM	1GB	1GB	1GB	1GB
HDD size	12GB	12GB	12GB	12GB
Virtual disk-provisioning method	Thin-provisioned	Thick-provisioned, lazy zeroed	Thick-provisioned, eager zeroed	Thick-provisioned, lazy zeroed
Virtual disk file	[Dallas]/Charlie2/Charlie2-flat.vmdk	[Dallas]/Bravo2/Bravo2-flat.vmdk	[Dallas]/Alpha2/Alpha2-flat.vmdk	[Dallas]/Delta2/Delta2-flat.vmdk
VM configuration file	[Dallas]/Charlie2/Charlie2.vmx	[Dallas]/Bravo2/Bravo2.vmx	[Dallas]/Alpha2/Alpha2.vmx	[Dallas]/Delta2/Delta2.vmx
Operating system	Windows 7 Professional x64	Windows 7 Professional x64	Windows 7 Professional x64	Windows 7 Professional x64

Note. This table lists the virtual hardware, configuration files and operating system of each of the VMs on the "Dallas" volume of "vmhost21."

The expected results of the testing if the hypothesis held true were two fold. Swapping the provisioning order of the "Alpha" and "Charlie" VMs, named "Alpha2" and "Charlie2" in this scenario, positioned "Charlie2-flat.vmdk" in a similar physical location to "Alpha-flat.vmdk's" position within the original testing scenario. Given that "Alpha-flat.vmdk" had no fragmentation within the "Boston" volume, the expectation was that "Charlie-flat2.vmdk" would similarly have no fragmentation within the "Dallas" volume. With further provisioning of VMs to the "Dallas" volume, the expectation was that ESXi would allocate blocks contiguous to, or within the maximum consecutive length on disk of the "Charlie2-flat.vmdk" virtual disk, to other VM files. That scenario, if shown to be true, would be highly likely to cause fragmentation of the "Charlie2-flat.vmdk" virtual disk as it expands in size over time.

After completing the scenario setup and deleting the "Alpha2," "Bravo2," and "Charlie2" VMs, dcfldd captured a working-copy and preservation-copy image of the "Dallas" VMFS volume. Using the command, "dcfldd bs=512 skip=2048 count=167772159 if=/dev/sdc

of=/media/hdd/capstone/hypo1/working/Dallas_volume_20150411.raw of=
of=/media/hdd/capstone/hypo1/preserve/Dallas_volume_20150411.raw.preserve", dcfldd created the working-copy and preservation-copy images of the "Dallas" VMFS volume. Following the image creation, the foremost configuration file from the original scenario carved the VMware ".log," ".vmx," ".vmxf," ".nvram," and ".vmdk" files from the "Dallas" volume. Using xxd in the same manner as the original testing scenario, xxd identified the MBR sectors for the four virtual disk files. Using the same technique detailed in the original testing scenario, Dcfldd carved each of the "-flat.vmdk" files from the "Dallas" volume according to the starting position and length of each virtual disk's partitions within the partition table. Each "-flat.vmdk", mounted to /media/temp on "Ubuntu 14 Forensic VM", produced the name of the VM to which it belonged using regripper's compname plugin to parse the "SYSTEM" registry hive at "/media/temp/Windows/System32/config/SYSTEM". The identified locations and length of each VMware ".log," ".vmx," ".vmxf," ".nvram," and ".vmdk" file as well as the location and length of each "-flat.vmdk" file, revealed the disk layout of the "Dallas" volume shown in Figure 13.

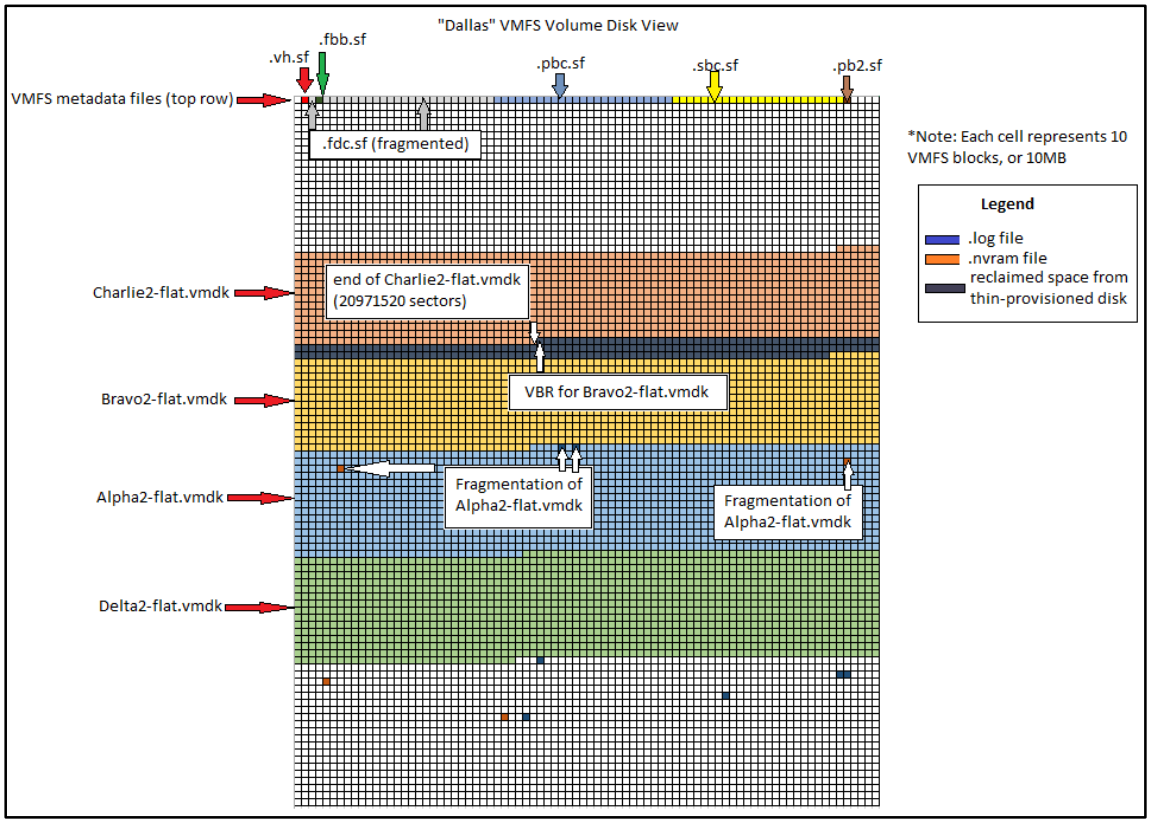

Figure 13. Disk view of "Dallas" volume. This figure shows a visual representation of the file allocation across the "Dallas" VMFS volume. The figure shows the contiguous sectors that comprised the "Charlie2-flat.vmdk," "Bravo2-flat.vmdk," and "Delta-flat.vmdk" virtual disk files and the fragmentation of "Alpha2-flat.vmdk." Each cell within the figure represents 10 VMFS blocks.

The first finding of the scenario confirmed the expectation of the "non-consecutive writes" hypothesis that further provisioning of VMs to the "Dallas" volume after the allocation of the "Charlie2-flat.vmdk" file, ESXi would allocate blocks contiguous to, or within the maximum consecutive length on disk of the "Charlie2-flat.vmdk" virtual disk, to other VM files. As can be seen in Figure 13, marked by dark purple cells, due to "Charlie2-flat.vmdk" being a thin-provisioned virtual disk file, ESXi did not allocate the full-configured length of 25,165,824 sectors to "Charlie2-flat.vmdk." Sector 0 of "Bravo2-flat.vmdk" occurred on disk at 20,971,520 sectors after sector 0 of "Charlie2-flat.vmdk." Therefore, 4,194,304 sectors, or 2GB, of "Charlie2-flat.vmdk" were unallocated and could not be allocated as contiguous blocks to the

existing "Charlie2-flat.vmdk because ESXi had already allocated those blocks to "Bravo2-flat.vmdk."

The second finding of the scenario confirmed the expectation of the "non-consecutive writes" hypothesis that no fragmentation occurred within the "Charlie2-flat.vmdk" file. Within the "Dallas" volume's disk layout was the absence of fragmentation within the "Charlie2-flat.vmdk" virtual disk file and the introduction of fragmentation within the virtual disk files that were located at higher addresses on the "Dallas" volume. This finding met the expectation of the "non-consecutive writes" hypothesis that no fragmentation occurred within the "Charlie2-flat.vmdk" file.

The most recent ".vmx," ".vmxf," and ".vmdk" files from the Charlie2 VM, identified using the same technique described in the Analysis, Analysis Phase, Test #1: Thick-provisioned, eager zeroed virtual disk, VM reconstruction section, were copied along with "Charlie2-flat.vmdk" using scp to the "Charlie2" folder on the "Chicago" VMFS volume on "vmhost21." Within the vSphere Client, using the Datastore Browser, right-clicking on the "Charlie2.vmx" file name and selecting "Add to Inventory," invoked the "Add to Inventory" wizard. After adding the "Charlie2" VM to the VM inventory on "vmhost21," it was powered on and successfully booted the "Charlie2" VM as shown in Figure 14 In the original test scenario, each of the thick-provisioned virtual disk files needed an additional 1MB block added to the end of each virtual disk file for ESXi to accept it as a valid virtual disk. For the thin-provisioned virtual disk file, no additional blocks were needed.

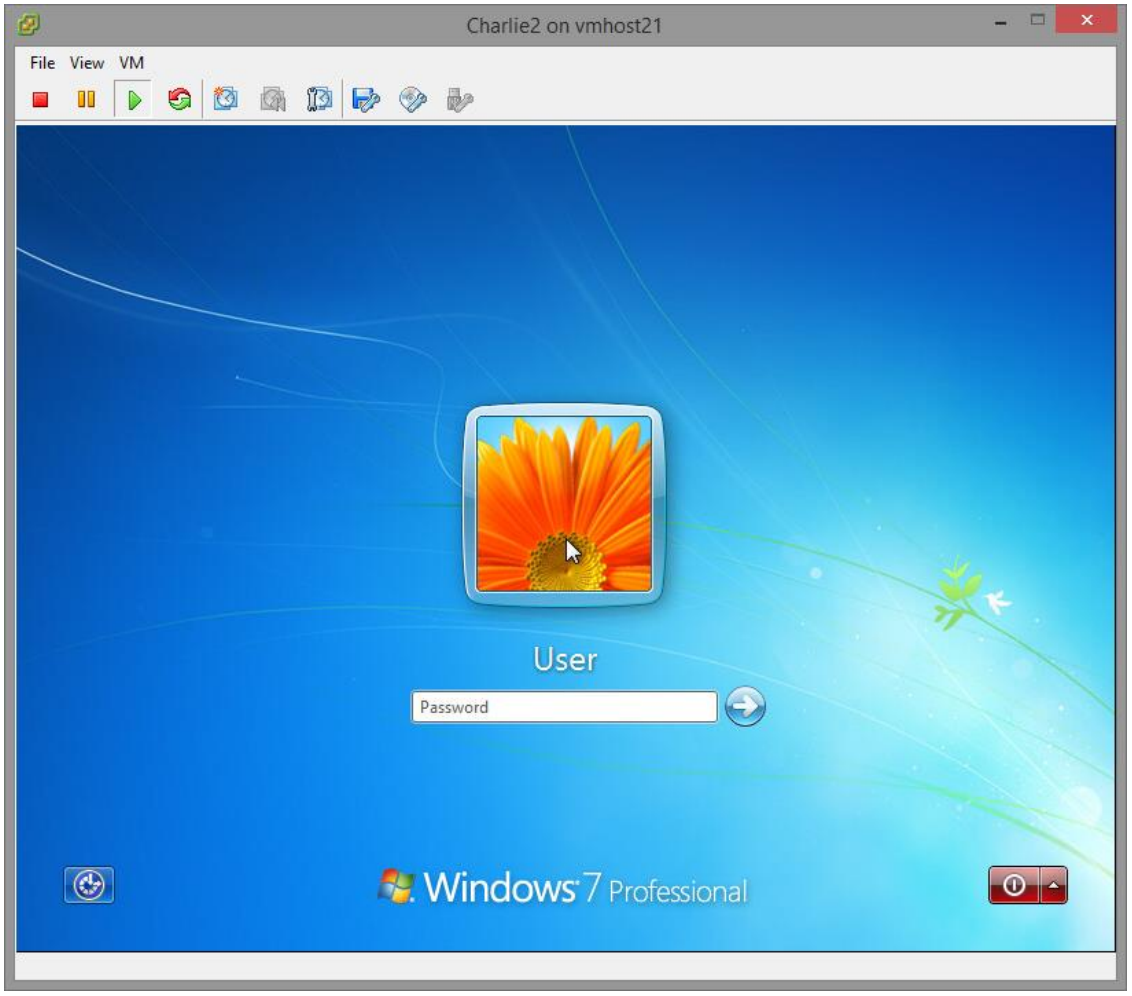

Figure 14. Successful reconstruction of "Charlie2" VM. This figure shows the fully booted "Charlie2" VM to illustrate the successful recovery of the VM from the VMFS volume "Dallas" as part of the "non-consecutive writes" hypothesis testing.

Unallocated percentage hypothesis. The "unallocated percentage" hypothesis offered that the likelihood of fragmentation within the VM virtual disk files decreased as the percentage of unallocated space on the VMFS volume increased. The testing of the "unallocated percentage" hypothesis required an additional testing environment, created using the same scenario as the original testing scenario, except the VMFS volume had twice as much capacity and each VM's virtual disk file was configured with 50% more capacity.

The VM datastore drive, forensically wiped with zeroes using dc3dd prior to configuring the datastore, held a single 160GB VMFS5 volume labeled "Edmonton." Four VMs had their

files stored on the "Edmonton" volume. Each of the VMs had the same software and virtual hardware configuration with the exception of the virtual disk allocation method. The "Charlie3" VM had a thin-provisioned virtual disk. The "Bravo3" VM had the ESXi default method of thick-provisioned, lazy zeroed virtual disk. The "Alpha3" VM had a thick-provisioned, eager zeroed virtual disk. The "Delta3" VM had a thick-provisioned, lazy zeroed virtual disk. Table 7 lists the configuration of the four test VMs.

Table 7

Control File Hypothesis Test VM Configurations

	VM 1	VM 2	VM 3	VM 4
VM Name	Alpha3	Bravo3	Charlie3	Delta3
VM working location	[Edmonton]/Alpha3	[Edmonton]/Bravo3	[Edmonton]/Charlie3	[Edmonton]/Delta3
CPUs	1	1	1	1
RAM	1GB	1GB	1GB	1GB
HDD size	18GB	18GB	18GB	18GB
Virtual disk allocation method	Thick-provisioned, eager zeroed	Thick-provisioned, lazy zeroed	Thin-provisioned	Thick-provisioned, lazy zeroed
Virtual disk file	[Edmonton]/Alpha3/Alpha3-flat.vmdk	[Edmonton]/Bravo3/Bravo3-flat.vmdk	[Edmonton]/Charlie3/Charlie3-flat.vmdk	[Edmonton]/Delta3/Delta3-flat.vmdk
VM configuration file	[Edmonton]/Alpha3/Alpha3.vmx	[Edmonton]/Bravo3/Bravo3.vmx	[Edmonton]/Charlie3/Charlie3.vmx	[Edmonton]/Delta3/Delta3.vmx
Operating system	Windows 7 Professional x64	Windows 7 Professional x64	Windows 7 Professional x64	Windows 7 Professional x64

Note. This table lists the virtual hardware, configuration files and operating system of each of the VMs on the " Edmonton" volume of" vmhost21."

Increasing the amount of unallocated space on the VMFS volume in this scenario allowed for provisioning of the same number of VMs as the original test scenario, but with unallocated

streams of greater length available to avoid fragmentation of the virtual disk files. This test environment also increased the disk capacity of each VM's virtual disk file by 50% in order to test the "control file" hypothesis with the same testing environment. The net increase in disk capacity was an additional 24GB of configured virtual disk space for VMs cumulatively and an additional 80GB of disk space available on the VMFS volume. In the original testing scenario, fragmentation of the "Charlie-flat.vmdk" virtual disk file was significant, rendering the file unrecoverable using consecutive sector carving techniques. The expectation for the "allocation percentage" hypothesis scenario is less, or no fragmentation of the virtual disk files due to the higher percentage of unallocated space on the VMFS volume.

After completing the scenario setup and deleting the "Alpha3," "Bravo3," and "Charlie3" VMs, dcfldd captured a working-copy and preservation-copy image of the "Edmonton" VMFS volume. Using the command, "dcfldd bs=512 skip=2048 count=335544320 if=/dev/sdc of=/media/hdd/capstone/hypo2/working/Edmonton_volume_20150412.raw of= of=/media/hdd/capstone/hypo2/preserve/Edmonton_volume_20150412.raw.preserve", dcfldd created the working-copy and preservation-copy images of the "Edmonton" VMFS volume. Following the image creation, the foremost configuration file from the original scenario carved the VMware ".log," ".vmx," ".vmxf," ".nvram," and ".vmdk" files from the "Edmonton" volume. Using xxd in the same manner as the original testing scenario, xxd identified the MBR sectors for the four virtual disk files. Using the same technique detailed in the original testing scenario, Dcfldd carved each of the "-flat.vmdk" files from the "Edmonton" volume according to the starting position and length of each virtual disk's partitions within the partition table. The

While each VM had a configured virtual disk file of 37539840 sectors, there only existed 24090624 sectors between sector 0 of "286019584-flat.vmdk" and sector 0 of "310110208-

flat.vmdk." This indicated that 286019584-flat.vmdk was thin-provisioned and, as the only thin-provisioned disk in the test, belonged to the "Charlie3" VM. The virtual disk file "310110208-flat.vmdk" extended beyond the bounds of the "Edmonton" volume. Xxd located the NTFS trailer for the truncated partition within "310110208-flat.vmdk" at sector offset 14,827,519. Because the truncation of "310110208-flat.vmdk" left 12,312,576 sectors unaccounted for, dcfldd merged the 12,312,576 sectors prior to sector 14,827,519 with "310110208-flat.vmdk" to form a complete "-flat.vmdk" file using the command, "dcfldd bs=512 skip=2514944 count=12312576 seek=25434112 if=Edmonton_20150412.raw of=310110208-flat.vmdk". Each "-flat.vmdk", mounted to /media/temp on "Ubuntu 14 Forensic VM", produced the name of the VM to which it belonged using regripper's compname plugin to parse the "SYSTEM" registry hive at "/media/temp/Windows/System32/config/SYSTEM". The identified locations and length of each VMware ".log," ".vmx," ".vmxf," ".nvram," and ".vmdk" file as well as the location and length of each "-flat.vmdk" file, revealed the disk layout of the "Edmonton" volume shown in Figure 15.

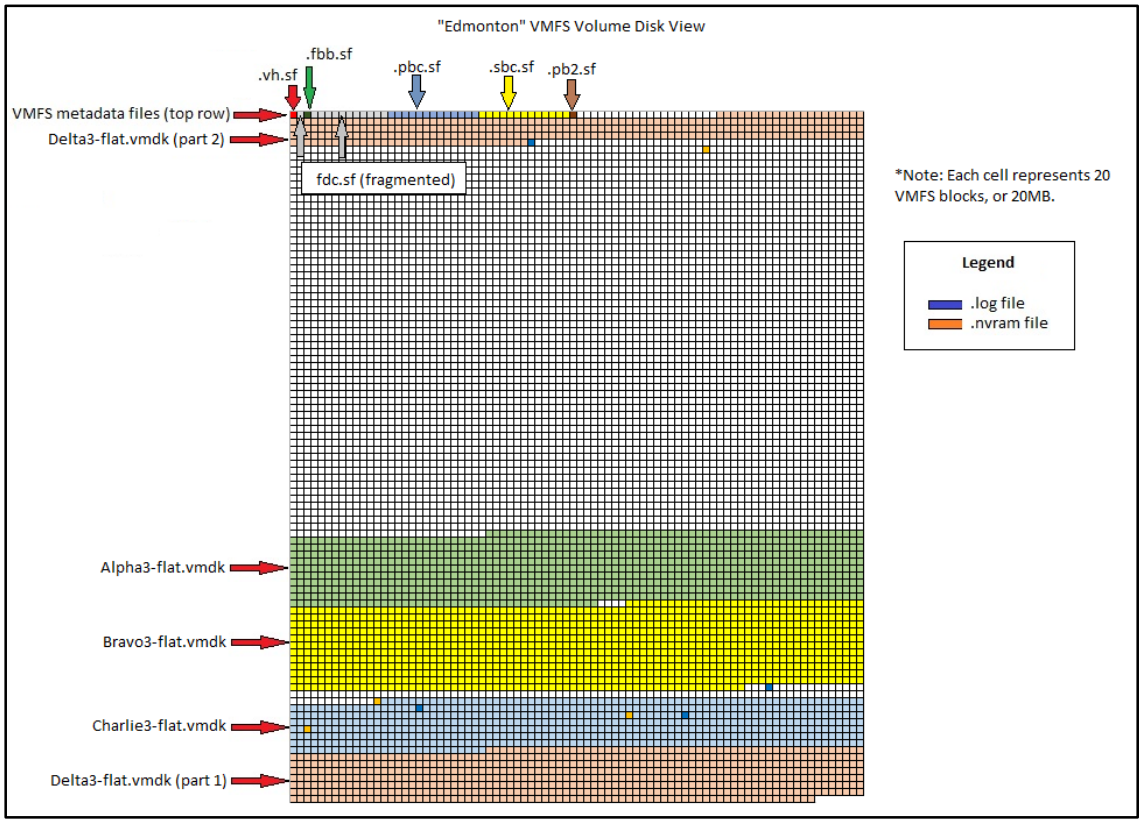

Figure 15. Disk view of "Edmonton" volume. This figure shows a visual representation of the file allocation across the "Edmonton" VMFS volume. The figure shows the contiguous sectors that comprised the "Alpha3-flat.vmdk" and "Bravo3-flat.vmdk," the fragmentation of "Charlie3-flat.vmdk," and "Delta3-flat.vmdk" with a truncated data stream at the end of the "Edmonton" volume, which resumed at lower addresses within the "Edmonton" volume. Each cell within the figure represents 20 VMFS blocks.

The expectation that less, or no, fragmentation of the virtual disk files would occur as part of the "unallocated percentage" hypothesis testing was unsupported. "Charlie3-flat.vmdk" experienced a similar amount of fragmentation during the test as in the original scenario testing. Further, the provisioning of "Delta3-flat.vmdk" occurred in such a way that the contiguous data stream of the virtual disk file reached the end of the "Edmonton" volume and resumed at lower addressed blocks. Although significant streams of contiguous unallocated blocks existed on the "Edmonton" volume, ESXi's method of provisioning the virtual disk files did not seek to avoid fragmentation by using available unallocated data streams. The disk allocation method in use by ESXi did not appear to place priority on avoiding fragmentation of the "-flat.vmdk" virtual disk files.

Control file hypothesis. The "control file" hypothesis offered that chances for recovery of deleted files within the VM decreased as unallocated space within the virtual disk file decreased. The testing of the "control file" hypothesis reviewed results of the original testing scenario against results of the "unallocated percentage" testing scenario to determine if an increase in unallocated space within the VM virtual disk file increased the chances for recovery of the deleted control files. The difference between the two scenarios for this testing was the configured size of the VM virtual disk files. Within the "unallocated percentage" hypothesis testing, each VM's virtual disk capacity increased by 50% over the original testing scenario VMs.

Using the same foremost-pdf control file used in the Analysis, Analysis Phase, Test #1: Thick-provisioned, eager zeroed virtual disk, Control file recovery section, named "foremost-pdf.conf," foremost carved 104 ".pdf" files from "Alpha3-flat.vmdk," 103 ".pdf" files from "Bravo3-flat.vmdk," and 29 ".pdf" files from "Charlie3-flat.vmdk." Foremost carved a file matching the MD5 hash value of "ControlDoc3.pdf" from "Alpha3-flat.vmdk" at sector 05938584. None of the files carved from "Bravo3-flat.vmdk" or "Charlie3-flat.vmdk" matched the MD5 hash value of "ControlDoc3.pdf" or "ControlDoc4.pdf".

Hexedit searched for the control file's identifiable hexadecimal string, "2F417574686F7228422E204B696E63686C6129", in each of the "Alpha3-flat.vmdk," "Bravo3-flat.vmdk," and "Charlie3-flat.vmdk" virtual disk files. Each of the virtual disk files contained fragments of "ControlFile3.pdf" and "ControlDoc4.pdf", but only "ControlDoc3.pdf" from "Alpha3-flat.vmdk", which foremost successfully recovered, remained intact with no fragmentation. The remaining fragmented control files, listed in Table 8, fragmented following

either the first or second NTFS block, leaving the majority of each file overwritten or located elsewhere within each virtual disk file.

Table 8

Control file fragments identified within virtual disk files

Virtual disk file	Control file name	Starting byte offset	Length
Alpha3-flat.vmdk	ControlDoc4.pdf	3,058,921,472	4096 Bytes
Bravo3-flat.vmdk	ControlDoc3.pdf	6,201,188,352	4096 Bytes
Bravo3-flat.vmdk	ControlDoc4.pdf	3,030,011,904	8192 Bytes
Charlie3-flat.vmdk	ControlDoc3.pdf	7,172,034,560	4096 Bytes
Charlie3-flat.vmdk	ControlDoc4.pdf	6,989,713,408	8192 Bytes

Note. This table lists the fragments of each control file and size of the fragments identified. Each of the control files fragmented following either the first of second NTFS block.

The result of the test that increasing the amount of unallocated space within the virtual disk file increases the likelihood of recovery of deleted artifacts of user activity was inconclusive. Among the three VMs each with two deleted control files, only one deleted control file recovered fully and the remaining control files experienced fragmentation at an early point within the file, either after one or two NTFS blocks. The test produced one complete control file and fragments of all other control files sought. This result met the expectation that the likelihood of user artifact recovery increased as the amount of unallocated space on the virtual disk file increased. The result did not provide a significant increase in the chance of recovery, however. Given that the recoverability of the deleted control files as part of the "control file" hypothesis testing was not significantly greater than the recovery experienced in the original testing scenario, the hypothesis requires further testing to determine if it holds true.

Discussion of Findings

Major Findings

The purpose of this research was to analyze the potential for recovering evidence from deleted VMware vSphere Hypervisor (ESXi) VMs under a variety of conditions. Specifically, the conditions tested were the three different virtual disk-provisioning methods available within ESXi: thick-provisioned, lazy zeroed virtual disk; thick-provisioned eager zeroed virtual disk; and thin-provisioned virtual disk. Using three VMs, each with a virtual disk configured in one of the three disk-provisioning modes, an identical scenario applied to each VM with the results recorded for comparison.

The sources selected for the literature review dealt primarily with one or more of the main themes of the research or provided background information necessary to understand the themes. The first theme of the research, understanding the VMFS file system to the point where a complete file system could be extracted from a physical disk image, built on information obtained from sources that documented features of VMFS5, differences between VMFS5 and VMFS3, and source code from the open source VMFS driver, vmfs-tools. Additionally, information from sources that discussed general file system forensics lent to the background information necessary to perform the collection and extraction phases of the analysis.

The second theme of the research, comparing the forensic recoverability of virtual disk files using the three disk provisioning modes in ESXi, built on information obtained from sources that specifically discussed the disk provisioning modes of ESXi. Additionally, this theme relied on information from sources that documented file fragmentation within VMFS and other VMware hypervisor products. This theme relied on and further built on the research performed during the first theme, understanding the VMFS file system.

The third theme of the research, recovering deleted VMware VM files from a VMFS volume by carving from contiguous disk sectors, built on information obtained from sources that discussed data carving and the various files that make up VMs on ESXi. The research focused on the VMware ".log," ".vmx," ".vmxf," ".nvram," ".vmdk," and "-flat.vmdk" file types, as those were the file types associated with the basic VMs built for the three tests. This theme relied on research performed for the first two themes and built on existing file carving research to define techniques for carving each of the VMware ESXi file types used in this research.

The fourth, and final, theme of the research, reconstructing deleted VMs, relied on the research performed for all three other themes. The research for the fourth theme additionally relied on analysis of the open source VMFS driver, vmfs-tools, in order to correct and verify some assumptions made during the analysis. Referencing sources on disk partitioning and interpreting partition tables assisted with explaining and correcting problems encountered while reconstructing deleted VMs.

Theme 1: The VMFS file system. The research performed on the VMFS file system revealed several critical pieces of information of use by anyone attempting to perform VMFS analysis. There are several characteristics of VMFS and of the VMFS system files encountered during this research that were unexpected. Those characteristics are the use of file headers offset from the beginning of system files and the file system itself, the static block size within VMFS, and resident VM files within VMFS system files.

The VMFS file system and the ".vh.sf" system file both have a single VMFS block offset between the beginning of the volume or file and the volume or file signature. This initially caused confusion during the analysis because the VMFS volume header was not located at the physical disk offset where the partition started according to the disk's partition table. A review of

the source code for vmfs-tools revealed the added block at the start of the VMFS volume prior to the volume signature. The ".vh.sf" volume header system file contained a two-block offset between the start of the file and the file header. The purpose of these offset headers was unknown and not explored further during the research.

VMFS5 has a static block size of 1MB. This is not a configurable setting in VMFS5 as it was in VMFS3 and previous versions. For comparison, the default block size of an NTFS volume is 4KB, so a VMFS block is 256 times the size of an NTFS block. The effect of the larger block size on forensic recovery is that fragmentation cannot occur on a VMFS volume for files smaller than 1MB. There are, however, only a few file types belonging to VMs that are typically smaller than 1MB in size. Those smaller files are the ".log," ".vmx," ".vmxf," ".nvram," and ".vmdk" files, although ".log" files have the potential to grow larger than 1MB.

The third unexpected characteristic of VMFS was the presence of ".vmx" and ".vmdk" files resident within the ".fdc.sf" system file. The ".vmx" and ".vmdk" files are both essential files for reconstructing deleted VMs. The files remained in the ".fdc.sf" file even after deleting the parent VM, which is helpful to examiners seeking to recover deleted VMs or even to catalog the VMs present and deleted on the VMFS volume.

Theme 2: Forensic recoverability of disk-provisioning modes. The research performed an analysis of the likelihood for recoverability of each of the three virtual disk-provisioning modes available for ESXi VMs. The three virtual disk-provisioning modes are thin-provisioned disks, thick-provisioned lazy zeroed, and thick-provisioned eager zeroed. The three modes differ in the manner that allocation of blocks occurs within the VMFS volume and the method used to zero space needed for I/O operations.

The largest hurdle to carving virtual disk files from the VMFS volume was identification of the MBR and each partition of the virtual disk. Using xxd to pass 256 bytes at a time of the VMFS volume's data stream through grep, searching for the master boot record (MBR) trailer value of "0x55AA" in the position of bytes 255 and 256, immediately preceded by values likely to occur in a partition table, produced the rough location of each virtual disk file's MBR. The start of each virtual disk file occurred 256 bytes prior to each search result. Analyzing each virtual disk file's partition table produced the extents of the virtual disk file.

This technique led to the discovery of a caveat with carving virtual disk files in this manner. The thick-provisioned disks required an additional VMFS block appended to the virtual disk file following the end of the last partition in the virtual disk file. Without the additional block appended to the end of the virtual disk file, ESXi failed to recognize the file for VM reconstruction. This is not the case for thin-provisioned virtual disks. With thin-provisioned virtual disks, no additional data needs appending to the end of the virtual disk file for ESXi to recognize the virtual disk.

Thick-provisioned eager zeroed virtual disks. When a VM has a disk configured in thick-provisioned eager zeroed mode, the full capacity of the virtual disk is allocateed within the datastore and ESXi zeroes all allocated blocks for the virtual disk at the time of virtual disk file creation. Immediately following the virtual disk creation, ESXi writes the full capacity of the virtual disk file to the datastore with zeroes. The "Alpha" VM, used in test #1, made use of a thick-provisioned eager zeroed virtual disk file, named "Alpha-flat.vmdk".

As Carrier (2005) explained, an OS typically allocates consecutive data units, but when that is not possible, fragmentation, a condition when the data units that make up a file are not consecutive, can occur (p. 179, para. 4). "Alpha-flat.vmdk" was a 12GB virtual disk file assigned

to the "Alpha" VM. The way ESXi provisioned "Alpha-flat.vmdk" to the VMFS volume, allocation of 12GB of contiguous zeroed VMFS blocks occurred in a single operation. After deleting the "Alpha" VM, including the "Alpha-flat.vmdk" virtual disk file, dd carved the deleted "Alpha-flat.vmdk" file from the VMFS volume. Carving 25,165,824 consecutive sectors from sector 38,539,264 to sector 63,705,088 on the VMFS volume produced a complete "Alpha-flat.vmdk" virtual disk file.

Thick-provisioned lazy zeroed virtual disks. When a VM has a disk configured in thick-provisioned lazy zeroed mode, the full capacity of the virtual disk is allocated within the datastore, but ESXi does not zero allocated sectors for the virtual disk at the time of virtual disk file creation. Zeroing of allocated sectors instead occurs when file I/O operations occur. Immediately following the virtual disk creation, ESXi allocates the full capacity of the virtual disk file to the datastore and leaves any existing data within the allocated sectors intact. The "Bravo" VM, used in test#2, made use of a thick-provisioned lazy zeroed virtual disk file, named "Bravo-flat.vmdk".

"Bravo-flat.vmdk" was a 12GB virtual disk file assigned to the "Bravo" VM. The way ESXi provisioned "Bravo-flat.vmdk" to the VMFS volume, allocation of 12GB of contiguous VMFS blocks occurred in a single operation. After deleting the "Bravo" VM, including the "Bravo-flat.vmdk" virtual disk file, dd carved the deleted "Bravo-flat.vmdk" file from the VMFS volume. Carving 25,165,824 consecutive sectors from sector 63,705,088 to sector 88,870,912 on the VMFS volume produced a complete "Bravo-flat.vmdk" virtual disk file.

Thin-provisioned virtual disks. When a VM has a disk configured in thin-provisioned mode, ESXi only allocates the capacity of the virtual disk file's active data within the datastore. Allocating additional blocks to the virtual disk file and zeroing the used sectors within those

blocks occurs when file I/O operations occur. The "Charlie" VM, used in test #3, made use of a thin-provisioned virtual disk file, named "Charlie-flat.vmdk".

"Charlie-flat.vmdk" was a 12GB virtual disk file assigned to the "Charlie" VM. The way ESXi provisioned "Charlie-flat.vmdk" to the VMFS volume, allocation of VMFS blocks occurred as files were written to the virtual disk file. After deleting the "Charlie" VM, including the "Charlie-flat.vmdk" virtual disk file, dd carved 25,165,824 consecutive sectors from sector 131,338,240 to sector 156,504,064 on the VMFS volume. The carved file was not a complete virtual disk file. As seen in the disk view of the "Boston" VMFS volume in Figure 4, fragmentation occurred within the "Charlie-flat.vmdk" file. At an undetermined point within the allocation of blocks to the "Charlie-flat.vmdk" file, ESXi fragmented the file and continued writing to unknown blocks within VMFS. The fragmentation of the "Charlie-flat.vmdk" file led to the formulation of two hypotheses regarding the likelihood for fragmentation within ESXi and VMFS5.

The first hypothesis, called the "non-consecutive writes" hypothesis offered that thin-provisioned virtual disk files have a higher likelihood of fragmentation due to non-consecutive writes to the VMFS volume during virtual disk file expansion. Testing of the hypothesis occurred through a recreation of the original test environment and test scenario with a change only in the provisioning order the VMs. There were two expected results of the "non-consecutive writes" hypothesis testing. The first expected result was no fragmentation would occur within the "Charlie2-flat.vmdk" file on the "Dallas" VMFS volume. The second expected result was future writes to the "Charlie2-flat.vmdk" file, resulting in expansion of the disk file on the "Dallas" volume, would cause fragmentation.

The "Charlie2-flat.vmdk" virtual disk file was the first virtual disk file written to the "Dallas" VMFS volume. This change in provisioning order created a virtual disk file of contiguous blocks with no fragmentation. The next virtual disk file provisioned, "Bravo2-flat.vmdk" had its first block located consecutively after the last block of "Charlie2-flat.vmdk." Because "Bravo2-flat.vmdk" occupied the next contiguous block after the end of "Charlie2-flat.vmdk," any writes to the "Charlie" VM resulting in virtual disk file expansion would inevitably cause fragmentation of the virtual disk file. As long as the "Bravo2-flat.vmdk" file existed as an active file on the file system, the blocks contiguous to "Charlie2-flat.vmdk" would be unavailable for writing other files to.

The second hypothesis, called the "unallocated percentage" hypothesis offered that as unallocated space on the VMFS volume increased, fragmentation of the virtual disk files would be less likely to occur. Testing of the hypothesis occurred through a recreation of the original test environment and test scenario with a change in the size of the VMFS volume, changing from 80GB to 160GB, and a change in each VM's virtual disk size, changing from 12GB to 18GB. Another hypothesis, "control file" hypothesis used the same test environment, which was the reason for increasing the VM virtual disk size. The "Edmonton" VMFS volume used in this test contained 70% unallocated space. The "Boston" VMFS volume used in the original test environment contained 40% unallocated space.

The expected testing result was that the virtual disk files contained less fragmentation on the "Edmonton" volume than on the "Boston" volume. The test produced inconclusive results because while less fragmentation occurred within the "Charlie3-flat.vmdk" virtual disk file, the file still fragmented and the "Delta3-flat.vmdk" virtual disk file wrapped from the end of the "Edmonton" volume to lower addressed blocks toward the beginning of the volume. Despite

large areas of contiguous unallocated space available, as shown by the disk view in Figure 15, ESXi wrote both "Charlie3-flat.vmdk" and "Delta3-flat.vmdk" to areas of the "Edmonton" volume where there was not enough contiguous unallocated space to avoid fragmentation.

Theme 3: Recovering deleted VM files from a VMFS volume. The research performed an examination of the VMFS volume for recovery of deleted VM files. The test placed four ".pdf" control files on each of the test VMs. The control files were named "ControlDoc1.pdf," "ControlDoc2.pdf," "ControlDoc3.pdf," and "ControlDoc4.pdf." Within each VM, each of the control files had the same actions performed on them.

The scenario preparation downloaded "ControlDoc1.pdf" to the user's desktop folder, opened the file in Foxit Reader, and then sent the file to the Windows recycle bin. "The scenario preparation opened "ControlDoc2.pdf" directly with the Foxit Reader Google Chrome plugin and downloaded "ControlDoc3.pdf" and "ControlDoc4.pdf" to the user's "Downloads" folder, opened in Foxit Reader, then deleted from the VM's disk.

During the original testing scenario, successful recovery of "ControlDoc1.pdf" and "ControlDoc2.pdf" occurred from both the "Alpha-flat.vmdk" and "Bravo-flat.vmdk" virtual disk files. Mounting the virtual disk files as loopback devices on "Ubuntu 14 Forensics VM" allowed for browsing of the active files within each of the virtual disk file systems. Successful recovery of "ControlDoc1.pdf" occurred through browsing to the scenario preparation user's recycle bin folder. Successful recovery of "ControlDoc2.pdf" occurred by performing a MD5 hash calculation of each file within the relative path of /Users/User/AppData/Local/Google/Chrome/User Data/Default/Cache/ and comparing to the known MD5 hash value of "ControlFile2.pdf".

To recover the remaining control files, "ControlFile3.pdf" and "ControlFile4.pdf," deleted from disk during the scenario preparation, foremost ran with a configuration to carve only ".pdf" documents. Foremost recovered ".pdf" files, whose MD5 hash values, compared to the known MD5 hash values of "ControlDoc3.pdf" and "ControlDoc4.pdf," produced no matches on either the "Alpha-flat.vmdk" or "Bravo-flat.vmdk" virtual disk files. The deleted control files "ControlDoc3.pdf" and "ControlDoc4.pdf" were unrecoverable using foremost and manually searching for known strings within the files.

A hypothesis devised to test whether the control files stood a better chance of recovery had there been more free space on each of the virtual disk files. The "control file" hypothesis tested the recoverability of the same four control files, prepared using the same test scenario as detailed in the original testing methodology, with the difference of an increase in virtual disk size from 12GB to 18GB. The test resulted in a marginal improvement in control file recoverability. Of the six control files deleted from disk in the "control file" hypothesis testing, foremost recovered one control file in its entirety. Manual searching of each virtual disk file for a known unique string in the control files resulted in identification of fragments of the other five deleted control files. The hypothesis test results were not strong enough to prove the hypothesis. To prove or disprove the "control file" hypothesis requires further testing.

Theme 4: Deleted VM reconstruction. The research performed a reconstruction of the deleted files that made up the deleted VMs and imported those files into ESXi as a complete VM. The deleted file types recovered were the VM's ".log," ".nvram," ".vmdk," ".vmx," ".vmxf," and "-flat.vmdk" files. Each of the ".log," ".vmdk," ".vmx," and ".vmxf" files types are text file and therefore do not contain file type signatures to identify the specific file type within a data stream. Analyzing each of the text files led to strings of text whose uniqueness served as file

signatures for carving the files from the VMFS volume. (See *Figure 18* in Appendix B for the custom foremost configuration that successfully carved ".log," ".nvram," ".vmdk," ".vmx," and ".vmxf" files from each of the VMFS volumes in this research).

Starting with each ".log" file, the parent VM, and the most current copy of each file required identification. The ".log" files made for a logical starting point because the ".log" files contained timestamps on each ".log" entry, which made it simple to identify the latest version of a VM's ".log" file. Each ".log" file contained the VM name and a complete copy of the ".vmx," ".vmxf," and ".vmdk" file used for that VM. Comparing the ".vmx," ".vmxf," and ".vmdk" files to the log entries enabled the identification of the ".vmx," ".vmxf," and ".vmdk" files that the VM used during the same boot sequence as shown in the ".log" file.

The ".vmx" and ".vmdk" files can, and frequently do, reside within the VMFS system file, ".fdc.sf." This can be helpful during forensic analysis because these resident files remain intact even with a deleted parent VM. Referencing the ".fdc.sf" system file can give the examiner excellent clues about deleted VMs from the VMFS volume and configuration parameters, including virtual disk sizes of any deleted VMs.

The virtual disk file type, "-flat.vmdk", as described in Discussion of Findings, Major Findings, Theme 2: Forensic recoverability of disk-provisioning modes can be a challenge to recover fully from the VMFS volume. A main obstacle to recovery of the virtual disk file volume is fragmentation within the disk file. Fragmentation was particularly prevalent in the thin-provisioned virtual disk files. The testing revealed two different types of fragmentation and additional hypotheses sought to identify methods to reduce the likelihood of fragmented virtual disk files. The findings of those hypotheses are detailed in the Discussion of Findings, Major

Findings, Theme 2: Forensic recoverability of disk provisioning modes, thin-provisioned virtual disks.

Comparison of the Findings

Forensic recovery of evidence from ESXi VMs was a field of study with a lack of available research. Particularly lacking was research on the VMFS file system, which ESXi uses as the datastore for VMs. This research discovered techniques for recovering forensic artifacts from deleted ESXi VMs, as well as areas where additional holes in research of ESXi forensics exist.

As Carrier (2005) explained, essential file system data is necessary to save and retrieve files. Data must be available to identify the name of a file, where that file's content is stored, and a pointer from the file's name to the metadata structure (p. 176, para. 1). This research briefly looked at the mechanics and data layout of the VMFS file system. VMFS maintains the essential file system data that Carrier referred to within a series of system files. The ".fdc.sf" and ".sbc.sf" system files were of particular relevance to this research. The treatment of deleted files within the VMFS file system was an area of interest for this research. The research discovered that certain file types, notably the ".vmx" and ".vmdk" files, remain resident within the ".fdc.sf" system file and recoverable after deleting the parent VM. Other files that are not resident within a VMFS system file have their size, starting block, allocation bitmap, and other metadata wiped from the VMFS system files upon deletion. For these other file types, traditional file carving techniques and tools can aid in their recovery.

Shavers (2008,) in research on VMware's type-2 hypervisor products, described the challenges of recovering a VM in full due to fragmentation of the files. Shavers stated that because of the large size of virtual disk files and a certain amount of fragmentation, fully

recovering the contents of a deleted VM may not be possible (pp. 9, para. 4). Shavers' description of the challenges with fragmentation in recovering virtual disk files was a main challenge in this research as well. Fragmentation was the single biggest challenge encountered during the recovery of ESXi VM virtual disk files. Two hypotheses attempted to determine methods for reducing fragmentation, with mixed results. Fragmentation remains the biggest challenge to virtual disk file recovery from VMFS volumes.

Limitations of the Study

This research had several limitations due to both time and scope restraints. There remains a vast field of research to perform in the area of VMware ESXi forensics. Restraints imposed during the research for the sake of time requirements limited the ability to further research the forensic capabilities of the VMFS file system, the VM BIOS setting to ".nvram" address value mapping, and additional hypothesis testing of methods to reduce fragmentation within virtual disk files.

The VMFS file system remains largely undiscovered in academic research. This research looked briefly into some components of the VMFS file system as a means to an end. Dedicated research into the VMFS file system's operation and forensic capabilities is an area that is important to support efforts like this research going forward. A thorough understanding of the VMFS system files, the data contained in each system file, and the layout of that data within the file is essential to understanding how the VMFS file system works. Due to scope restraints, this research took a surface level look at the ".fdc.sf" and ".sbc.sf" system files.

Many possible file types make up an ESXi VM. This research focused on VMs that did not contain any snapshots, or suspended state information. This research reviewed the ".log," ".vmdk," ".vmx," ".vmxf," and "-flat.vmdk" file types for methods to carve each deleted file

from the file system, identify the parent VM of each file, identify the original file name of each file, and identify the directory structure and location within the VMFS volume necessary for VM reconstruction of each file. The ".nvram" file type requires additional research into methods for identifying the parent VM. This research did not map the ".nvram" address settings to BIOS settings.

The scope and time requirements of this research excluded any analysis of VM snapshot, snapshot data, suspended state, and swap files as part of the research. This research focused on the recovery of the minimum deleted VM files necessary to reconstruct a VM and have it boot back into Windows under ESXi. Expansion on this research could include recovery of all VM file types and offline analysis of VM snapshot, suspended state, and swap files.

Future Research Recommendations

The area of ESXi forensics remains fertile ground for researchers. Four areas in particular, closely related to this research, present opportunities to further the forensic community's ability to examine ESXi VMs. During this research, challenges arose with fragmentation of virtual disk files, lack of documentation of the VMFS file system, lack of documentation of the ".nvram" VM file type, and inefficiency in methodology due to lack of VMFS support in common forensic tools. An additional area not touched upon by this research, but needed by the forensic community is research into forensic analysis of VM snapshots and suspended state operations.

Fragmentation of virtual disk files emerged in this research as the most significant challenge to forensic recovery of evidence from ESXi VMs. Eliminating fragmentation of virtual disk files or determining a method of recovering the blocks assigned to deleted virtual disk files are two improvements that would aid examiners with recovery of ESXi virtual disks. Research into methods of recovery of fragmented virtual disk files and methods of avoiding fragmentation of virtual disk files would prove valuable to examiners seeking to recover deleted virtual disk files from a VMFS volume.

Contributing a detailed analysis of VMFS's operations would present examiners with the opportunity to more efficiently recover and analyze delete ESXi VMs. One of the challenges encountered during this research was the absence of proper documentation on VMFS operations. Throughout this research, the vmfs-tools source code and testing observations substituted for documentation, but added a great amount of time and effort to accomplish the research goals. Future research detailing VMFS operations will provide examiners with a useful reference for accurate and efficient analysis of VMFS volumes.

Analysis of the ".nvram" VM file type is necessary in order to understand the BIOS settings of the parent VM and identify the parent VM of a deleted ".nvram" file. The ".nvram" file type is a binary file type that contains the VM BIOS settings. As BIOS settings change, so does a particular address' setting within the ".nvram" file. Understanding what each address' setting in the ".nvram" file translates to as a BIOS setting on the VM can assist in identifying a parent VM.

Incorporation of VMFS parsing capabilities into data unit level forensic tools, such as blkls from The Sleuth Kit would improve examiners' ability to focus recovery efforts strictly on the unallocated blocks of the VMFS volume. During this research, data carving processes carved deleted as well as active files within the VMFS volume. Because the tools used to carve data or analyze the volume at the block level were unable to interpret the VMFS file system, all blocks were treated as unallocated, an inefficient way to approach data carving. Future research, which works with the authors of common open source tools to recognize and interpret the VMFS file system, will result in time savings for examiners and better handling of some fragmented files.

VM snapshots and suspended state operations are important areas for future research, as these features of ESXi can preserve volatile data at a point in time and create timelines of activity via multiple snapshots. These features of ESXi expand the possibilities of forensic recovery due to their unique characteristics that differentiate VM analysis from physical machine analysis. Research into both of these operations of ESXi will be a highly valuable contribution to the forensic community as virtualization technology continues to gain in popularity.

Conclusions

The purpose of this research was to analyze the potential for recovering evidence from deleted ESXi VMs. The research focused on ESXi version 5.5 update 2 and VMware's proprietary file system VMFS version 5. In pursuit of fulfilling the purpose, the research attempted to answer four questions: what tools and methods can examiners employ to attempt recovery of entirely deleted VMs from a VMFS volume? Which disk-provisioning configuration poses the greatest opportunity for forensic recovery and which disk-provisioning method poses the greatest challenge for forensic recovery from a VMFS volume? What is the greatest challenge to forensic recovery of deleted ESXi VM files and what can increase the likelihood of successful forensic recovery? What tools and methods can examiners employ to attempt complete reconstruction of deleted ESXi VMs?

At the time of the research, forensic tools that supported the VMFS file system were scarce. For this reason, many of the tools and methods to recover deleted VMs from a VMFS volume relied on recovery of individual or consecutive volume sectors. The tools during this research that performed the best for this purpose were foremost, for carving deleted files based on identified header and footer patterns, and dcfldd for carving deleted files based on identified start and end sectors. The method used to carve with foremost was to analyze known files of the same file types to carve, identify patterns at the start, or header, of the file, and the end, or footer, of the file. Incorporating the identified header and footer pattern into the foremost configuration file and running foremost against an image of the VMFS file system produced results that included both deleted and active files of the specified file types. Identifying start and end sectors of files to carve with dcfldd relied on xxd to search the volume for indications of the file start or

file end. Once identified, dcfldd was able to extract the specific consecutive sectors that comprised the complete carved file.

The research tested three VM disk-provisioning types; thick-provisioned, eager zeroed; thick-provisioned, lazy zeroed; and thin-provisioned. The two thick-provisioned types differed only in the method that the VMFS blocks were prepared prior to allocation. With eager zeroed disks, ESXi zeroes all VMFS blocks allocated to the virtual disk as part of the virtual disk creation process. With lazy zeroed disks, the VM zeroes allocated blocks within the VM file system as the allocation occurs. The difference between thick-provisioned and thin-provisioned disks that thick-provisioned disks have the entire length of the virtual disk allocated within VMFS at the time of the disk creation while thin-provisioned virtual disks have VMFS blocks allocated as needed by the virtual disk. As such, a thin-provisioned virtual disk may only have a percentage of its configured capacity allocated within VMFS at any time.

The research showed no difference in recoverability of the virtual disk file between the thick-provisioned, eager zeroed disk type and the thick-provisioned, lazy zeroed disk type. However, the research determined that thin-provisioned virtual disks experience fragmentation because of the thin-provisioning mode of operation, which is likely to interfere with the ability to recover deleted thin-provisioned virtual disk files. As a thin-provisioned virtual disk grows over time, the likelihood to expand to a contiguous block on the VMFS volume reduces, resulting in a fragmented virtual disk file. The research concluded that the two thick-provisioning modes provide the greatest potential for successful recovery of virtual disk files. Thin-provisioned virtual disk pose the greatest challenge for forensic recovery of virtual disk files from a VMFS volume.

The greatest challenge to forensic recovery of deleted ESXi VMs is the fragmentation within the VMFS volume of virtual disk files. Virtual disk files are typically large when compared to most common file types. In order to avoid fragmentation within the VMFS volume, the virtual disk file must be written to a portion of the volume with a continuous stream of unallocated space equal to or greater than the length of the virtual disk file. During the testing of scenarios that may increase the likelihood of avoiding fragmentation of virtual disk files, one finding was that ESXi does not seek to avoid fragmentation at the expense of consecutive allocation. In other words, ESXi seeks to write a new file to the next consecutive block as the previous write operation, regardless of whether fragmentation will result, even if there exists contiguous unallocated blocks with sufficient length to avoid fragmentation elsewhere on the volume. During this research, the best method discovered to avoid fragmentation of a virtual disk file was if allocation of the virtual disk file occurred as one of the first files on the volume. Commonly, allocation of virtual disk files occur well in advance of the need for forensic examination, which makes this method generally outside the control of the forensic examiner.

Complete reconstruction of a deleted VM relies on recovery of the essential ESXi VM file types, ".vmx," ".vmxf," ".vmdk," "-flat.vmdk," and ".nvram". Additionally, recovery of the VM's ".log" file(s) is instrumental in identifying the most recently active ".vmx" and ".vmxf" files. Each ".log" file contains the VM's complete configuration, recorded at the time of the VM's start, with each line time-stamped. This allows for identification of the most recent ".log" file and matching the configuration lines with the correct ".vmx" and ".vmxf" files. The contents of the ".vmx" file shows the parent VM name and the file names of the ".vmdk," ".vmxf," and ".nvram" files. The ".vmxf" file contained the file name of the ".vmx" file. The ".vmdk" file contains the directory structure of the parent VM within the VMFS file system, each "-

flat.vmdk" file name(s) and file size in disk sectors. Using the techniques described earlier in the Conclusions section to recover the "-flat.vmdk", or virtual disk file, combined with the techniques described here to identify the latest version of ".vmx" and ".vmxf" files types, re-assign file names to the recovered files, and recreate the directory structure of the deleted VM, deleted VMs can be fully reconstructed on a VMFS volume.

References

Altheide, C., & Carvey, H. (2011). *Digital forensics with open source tools*. Waltham, MA: Syngress.

Barrett, D. (n.d.). *Forensic challenges in virtualized environments*. Retrieved from University of Advancing Technology: http://www.uat.edu/academics/forensic_challenges_in_virtualized_environments.aspx

Barrett, D., & Kipper, G. (2010). *Virtualization and forensics: A digital forensic investigator's guide to virtual environments*. Burlington, MA: Syngress.

Carrier, B. (2005). *File System Forensic Analysis*. Boston, MA: Addison-Wesley.

Carrier, B. (2010, March 18). *Mmls*. Retrieved from Sleuth kit wiki: http://wiki.sleuthkit.org/index.php?title=Mmls

Carrier, B. (2015). *Open source digital forensics*. Retrieved from The sleuth kit: http://www.sleuthkit.org/

Carvey, H. (2012, July 27). *Regripper*. Retrieved from Google code project for regripper: https://code.google.com/p/regripper/wiki/RegRipper

Cole, D. E. (2010). Introduction. In D. Barrett, & G. Kipper, *Virtualization and forensics, A digital forensic investigator's guide to virtual environments* (p. xv). Burlington, MA: Syngress.

DiPetrillo, M. (2010, April 7). Mike DiPetrillo from VMware talks cloud. (R. Haywood, Interviewer) Retrieved from http://rodos.haywood.org/2010/04/mike-dipetrillo-from-vmware-talks-cloud.html

Fillot, C., & Hommey, M. (2012, March 25). *vmfs-tools*. Retrieved from Glandium: http://glandium.org/projects/vmfs-tools/

Fluid Operations. (2010, January 25). *Open source VMFS driver*. Retrieved from Google Project Hosting: https://code.google.com/p/vmfs/

Forrester Research, Inc. (2012, January). *Virtual security in the data center*. Retrieved from Cisco Systems: http://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/unified-fabric/tap_virtual_security_032012.pdf

Free Software Foundation. (2014, July 19). *Summarizing files*. Retrieved from GNU Coreutils: http://www.gnu.org/software/coreutils/manual/coreutils.html#md5sum-invocation

Free Software Foundation Europe. (2015, March 24). *Get a free software PDF reader*. Retrieved from FSFE: http://pdfreaders.org/

Harbour, N. (2006, December 19). *dcfldd - Latest version 1.3.4-1*. Retrieved from dcfldd: http://dcfldd.sourceforge.net/

Hommey, M. (2011, September 09). *Initial VMFS 5 support*. Retrieved from glandium.org: http://glandium.org/blog/?p=2194

IBM Global Education. (2007, October). *Virtualization in education*. Retrieved from http://www-07.ibm.com/solutions/in/education/download/Virtualization%20in%20Education.pdf

Kajamoideen, M. R. (2013, July 2). *Mohammed Raffic Kajamoideen's blog*. Retrieved from VMware Community: https://communities.vmware.com/people/raffic_ncc/blog/2013/07/01/difference-between-vmware-esx-and-esxi

Kornblum, J. (2006). Identifying almost identical files using context triggered. *Digital Investigation*(3), 91-97.

Lowe, S. (2009). *Mastering VMware vSphere 4*. Indianapolis, IN: Wiley Publishing, Inc.

Lowe, S., Marshall, N., Guthrie, F., Liebowitz, M., & Atwell, J. (2014). *Mastering VMware vSphere 5.5*. Indianapolis: Sybex.

Medico, A., Cordovano, R., Kornblum, J., Lowe, J., & Levendoski, M. (2014, 06 06). *dc3dd*. Retrieved from sourceforge.net: http://sourceforge.net/projects/dc3dd/

Microsoft. (2002, January 31). *The default cluster size for the NTFS and FAT file systems*. Retrieved February 2015, 20, from Microsoft Support: http://support.microsoft.com/kb/314878

Moolenaar, B., & Nugent, T. (1996, August). *xxd*. Retrieved from LinuxCommand.org: http://linuxcommand.org/man_pages/xxd1.html

Mueller, S. (2013). *Upgrading and Repairing PCs*. (21, Ed.) Indianapolis, IN: Que Publishing.

Perlow, J. (2009, March 3). *VMWare's filesystem clustering cracked open* . Retrieved from ZDnet: http://www.zdnet.com/article/vmwares-filesystem-clustering-cracked-open/

Pfleeger, C. P., & Pfleeger, S. L. (2007). *Security in Computing*. (4, Ed.) Boston, MA: Pearson Education, Inc.

Rigaux, P. (n.d.). *Hexedit (1)*. Retrieved from Pixel's Home Page: http://rigaux.org/hexedit.html

Shavers, B. (2008). *Virtual forensics: a discussion of virtual machines related to forensic analysis*. Retrieved from http://www.forensicfocus.com/downloads/virtual-machines-forensics-analysis.pdf

Smith, R. W. (2014, March 02). *GPT fdisk*. Retrieved from sourceforge.net: http://sourceforge.net/projects/gptfdisk/

U.S. Department of Commerce, National Institute for Standards in Technology. (2006). *Guide to integrating forensic techniques into incident response*. (NIST Special Publication 800-86). Retrieved from http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf

VMWare Inc. (2014, October 9). *vSphere virtual machine administration*. Retrieved from VMware vSphere 5.5 Documentation Center: http://pubs.vmware.com/vsphere-55/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-552-virtual-machine-admin-guide.pdf

VMware Inc. (2014a, November 17). *vSphere storage*. Retrieved from VMware vSphere 5.5 Documentation Center: http://pubs.vmware.com/vsphere-55/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-552-storage-guide.pdf

VMware, Inc. (2009). *Performance study of VMware vStorage thin provisioning*. Retrieved from VMware.com: http://www.vmware.com/pdf/vsp_4_thinprov_perf.pdf

Ward, K. (2015, January 5). *2015 Virtualization predictions*. Retrieved from Virtualization Review: http://virtualizationreview.com/articles/2015/01/01/2015-virtualization-predictions.aspx

Weadock, G. (2008, October 22). *Biggest benefits of virtualization?* Retrieved from Network World: http://www.networkworld.com/article/2346841/microsoft-subnet/biggest-benefits-of-virtualization-.html

Appendices

Appendix A – Physical Datastore

```
dc3dd 7.1.614 started at 2015-03-05 12:19:41 +0000
compiled options:
command line: dc3dd if=/dev/sdc hash=md5
log=/media/hdd/capstone/preserve/VMD_physical_20150305.log
hof=/media/hdd/capstone/preserve/VMD_physical_20150305.raw.preserve
hof=/media/hdd/capstone/working/VMD_physical_20150305.raw
device size: 390721968 sectors (probed)
sector size: 512 bytes (probed)
200049647616 bytes (186 G) copied (100%), 8158.38 s, 23 M/s
output hashing (100%)

input results for device `/dev/sdc':
   390721968 sectors in
   0 bad sectors replaced by zeros
   3b695cd3879e4e4b0922b3924a7a6ed1 (md5)

output results for file `/media/hdd/capstone/preserve/VMD_physical_20150305.raw.preserve':
   390721968 sectors out
   [ok] 3b695cd3879e4e4b0922b3924a7a6ed1 (md5)

output results for file `/media/hdd/capstone/working/VMD_physical_20150305.raw':
   390721968 sectors out
   [ok] 3b695cd3879e4e4b0922b3924a7a6ed1 (md5)

dc3dd completed at 2015-03-05 14:35:39 +0000
```

Figure 16. Contents of VMD_physical_20150305.log. This figure shows the output of the dc3dd command, verifying the MD5 hash value for the original source and both image copies matched.

```
kinchla@lab-u14:/$ sudo dcfldd if=/media/hdd/capstone/working/VMD_physical_20150
305.raw bs=512 skip=2048 count=167772159 of=/media/hdd/capstone/working/Boston_v
olume_20150305.raw of=/media/hdd/capstone/preserve/Boston_volume_20150305.raw.pr
eserve hash=md5
[sudo] password for kinchla:
167771904 blocks (81919Mb) written.Total (md5): 1b16f409837f77a2cba7588432804682

167772159+0 records in
167772159+0 records out
```

Figure 17. Carving the VMFS volume with dcfldd. This figure shows the dcfldd command used to extract the sectors of the VMFS volume and create the working-copy image, "Boston_volume_20150305.raw," and the preservation-copy image, "Boston_volume_20150305.raw.preserve." dcfldd calculated the MD5 hash value of the extracted data and displayed the hash value in the command output.

Appendix B – VMFS Volume

Table 9

Known VMFS Volume Info Header Fields

Description	Offset	Length (bytes)	Value	Interpretation
Signature	0x100000	4	0D D0 01 C0	"0xc001d00d": signature value
Version	0x100004	4	05 00 00 00	VMFS Version 5
Name	0x10002E	20	57 44 2D 57 4D 41 4C 38 31 31 36 34 33 37 37 00 00 00 00 00 00	WD-WMAL81164377
Volume size		4	00 00 00 14	"0x14000000": This value, multiplied by 256 provides the total number of bytes in the volume, 85899345920
UUID	0x100082	16	8C 96 F7 54 51 9B 3C D9 07 2D 00 15 17 1E 3F 3C	54F7968C-D93C9B51-2D07-5C3F1E171500
Created Time	0x100092	8	93 C8 F6 E4 7E 10 05 00	Wed, 04 Mar 2015 23:34:36 GMT
Modified time	0x100100	8	73 50 45 40 89 10 05 00	Thu, 05 Mar 2015 11:55:58 GMT

Note: Adapted from vmfs-tools source code (Fillott & Hommey, 2012). This table lists the known volume info header fields from the VMFS volume contained within the "VMD_physical_20150305.raw" physical volume.

Table 10

VM File Types

File	Usage	Description
.vmx	Vmname.vmx	VM configuration file
.vmfx	Vmname.vmfx	Additional VM configuration files
.vmdk	Vmname.vmdk	Virtual disk characteristics
-flat.vmdk	Vmname-flat.vmdk	VM data disk
.nvram	Vmname.nvram or nvram	VM BIOS or EFI configuration
.vmsd	Vmname.vmsd	VM snapshots
.vmsn	Vmnams.vmsn	VM snapshot data file
.vswp	Vmname.vswp	VM swap file
.vmss	Vmname.vmss	VM suspend file
.log	Vmname.log	Current VM log file
-#.log	Vmware-#.log (where # is a number starting with 1)	Old VM log files

Note: From " vSphere virtual machine administration," p. 11, table 1-1, by VMware, Inc., 2014.

Table 11

VMFS File System Metadata Files

Filename	**Role**
fbb.sf	File block system file
fdc.sf	File descriptor system file
sbc.sf	Sub-block system file
pbc.sf	Pointer block system file
vh.sf	Volume header system file

Note: Adapted from *Virtualization and Forensics* (2010) by Barrett & Kipper (p, 179, para. 2).

Table 12

Ssdeep comparison results of carved ".nvram" files

File1	File2	Percent Match
25821184.nvram	131336192.nvram	99
26249238.nvram	131336192.nvram	99
26249238.nvram	25821184.nvram	99
88870934.nvram	131336192.nvram	99
88870934.nvram	25821184.nvram	97
88870934.nvram	26249238.nvram	97
162947072.nvram	131336192.nvram	97
162947072.nvram	25821184.nvram	97
162947072.nvram	26249238.nvram	99
162947072.nvram	88870934.nvram	99
132732950.nvram	131336192.nvram	99
132732950.nvram	25821184.nvram	96
132732950.nvram	26249238.nvram	96
132732950.nvram	88870934.nvram	99
132732950.nvram	162947072.nvram	96
88942592.nvram	131336192.nvram	99
88942592.nvram	25821184.nvram	99
88942592.nvram	26249238.nvram	99
88942592.nvram	88870934.nvram	97
88942592.nvram	162947072.nvram	97
88942592.nvram	132732950.nvram	96

Note: This table shows the results of the command " ssdeep –lrd nvram/", comparing the similarity of the files contained within the " nvram" folder.

Table 13

Revised virtual disk file extents

Volume starting sector	**Volume ending sector**	**Number of sectors**	**File carved as**	**MD5 Hash value**
38,539,264	63,705,088	25,165,824	38539264-flat.vmdk	ca1c70318679d2770f4a9a159366974e
63,705,088	88,870,912	25,165,824	63705088+2048-flat.vmdk	2f4ed440bad5c542341189688c0137da
97,521,664	122,687,488	25,165,824	97521664+2048-flat.vmdk	2aa522c9b165e444b0d19dd4666efe6c
131,338,240	156,504,064	25,165,824	131338240+2048-flat.vmdk	db21b80916b0f51dc9e6ad047339f72e

Note: This table is a revision of Table 5 from Methodology, Analysis, Examination Phase. ESXi virtual disks contain one empty VMFS block following the end of the last partition of the virtual disk. The first set of virtual disks had been carved from the VMFS volume prior to this discovery, necessitating the files to be carved a second time with the added VMFS block included.

```
kinchla@lab-u14:~/capstone$ cat foremost.conf
#---------------------------
# VMware log files
#---------------------------
log     y       1048510 ??????????????????????\x7c\x20\x76\x6d\x78\x7c\x20\x49???\x3a\x20\x4c\x6f\
x67\x20\x66\x6f\x72\x20\x56\x4d\x77\x61\x72\x65\x20\x45\x53\x58\x20\x70\x69\x64\x3d \x56\x4d\x58\x20
\x68\x61\x73\x20\x6c\x65\x66\x74\x20\x74\x68\x65\x20\x62\x75\x69\x6c\x64\x69\x6e\x67\x3a\x20?\x2e\x0
a

#---------------------------
#VMware VMX files
#---------------------------
vmx     y       1048510 \x2e\x65\x6e\x63\x6f\x64\x69\x6e\x67\x20\x3d\x20???????\x0a\x63\x6f\x6e\x66\
x69\x67\x2e\x76\x65\x72\x73\x69\x6f\x6e\x20\x3d\x20???\x0a\x76\x69\x72\x74\x75\x61\x6c\x48\x57\x2e\x
76\x65\x72\x73\x69\x6f\x6e\x20\x3d\x20 \x74\x6f\x6f\x6c\x73\x2e\x72\x65\x6d\x69\x6e\x64\x49\x6e\x73\
x74\x61\x6c\x6c\x20\x3d\x20\x22????????

#---------------------------
#VMware NVRAM files
#---------------------------
nvram   y       10240 \x4d\x52\x56\x4e\x01\x00\x00\x00\x43\x4d\x4f\x53  \x00\x00\x00\xd0

#---------------------------
#VMware VMXF files
#---------------------------
vmxf    y       1048510 \x3c\x3f\x78\x6d\x6c\x20\x76\x65\x72\x73\x69\x6f\x6e\x3d\x22???\x22\x3f\x3e\
x0a\x3c\x46\x6f\x75\x6e\x64\x72\x79\x3e\x0a \x3c\x2f\x46\x6f\x75\x6e\x64\x72\x79\x3e\x0a

#---------------------------
#VMDK Files
#---------------------------
vmdk    y       1024    \x23\x20\x44\x69\x73\x6b\x20\x44\x65\x73\x63\x72\x69\x70\x74\x6f\x72\x46\x69
\x6c\x65\x0a
```

Figure 18. Contents of foremost_vmware.conf. This figure shows the contents of " foremost_vmware.conf" used to carve the VM, " .log," " .vmx," " .nvram," " .vmxf," and " .vmdk," files.

Appendix C – Alpha VM

Table 14

Carved files from the "Alpha" VM

File Name	File type	Carved from (sector)	Size (bytes)	MD5 hash value
00490850.vmx	vmx	00490850	1048510	6b28f747a065cdd9057455082e823385
00490890.vmdk	vmdk	00490890	1024	7405e0d4cd450d8c1935e015b5a1879b
01558752.vmx	vmx	01558752	19361	775015ab5bbf4c31cace80a9c7552b9d
01558816.vmx	vmx	01558816	2978	5a7e000f6147dc522edfb9e17c8e30f0
01558832.vmx	vmx	01558832	2980	5485e4c97dc20f76dc3dbb059b24fab7
01558944.vmx	vmx	01558944	2979	9059145772a968f0c9f1067e2074c78e
01559088.vmx	vmxf	01559088	3260	0b3597e4370594f5e1b5f7da75b50317
01559328.vmx	vmx	01559328	2979	6ff10e7805973b8767bc93ff0759bc2d
01559392.vmx	vmx	01559392	2978	dd914915c566fb03fb9c5d0a37bd374a
01559408.vmx	vmx	01559808	2979	b1f27791e703d220b50fa669c1b0dce0
131627008.log	log	131627008	214185	03027c836572feabac267ee71675ecd9
156733440.log	log	156733440	213356	627c96923aef75f3f8fdd4b363de98f1
88938496.log	log	88938496	583595	98ff0dd3b4fa5804e197ec755d1d502c
88940544.log	log	88940544	383574	583f2fb37da169b2638584cd20f5d8ab
88956928.log	log	88956928	225850	683e120325d5e26d9fe9387bd40aeb40
38539264-flat.vmdk	-flat.vmdk	38539264	12883853312	9f37811117150dab01c178d0b5829894

Note: This table lists all carved files from the "Boston_volume_20150205.raw" image that could be associated with the "Alpha" VM. Red typeface denotes the files identified as the most current versions and used to reconstruct the "Alpha" VM.

Table 15

Carved files used to reconstruct the "Alpha" VM

File Name	File type	Carved from (sector)	Size (bytes)	MD5 hash value
Alpha.vmdk	vmdk	00490890	1024	7405e0d4cd450d8c1935e015b5a1879b
Alpha.vmx	vmxf	01559088	3260	0b3597e4370594f5e1b5f7da75b50317
Alpha.vmx	vmx	01559808	2979	b1f27791e703d220b50fa669c1b0dce0
Alpha-flat.vmdk	-flat.vmdk	38539264	12883853312	9f37811117150dab01c178d0b5829894
Alpha-flat.vmdk	-flat.vmdk	38539264	12884901888	ca1c70318679d2770f4a9a159366974e

Note: This table lists the four carved files from the " Boston_volume_20150205.raw" image used to reconstruct the "Alpha" VM on the ESXi system " vhost21." Red typeface denotes a corrected entry. " Alpha-flat.vmdk" was re-carved after discovering that "-flat.vmdk" files require an additional 2048 sectors following the last partition on the virtual disk.

Appendix D – Bravo VM

Table 16

Carved files from the "Bravo" VM

File Name	File type	Carved from (sector)	Size (bytes)	MD5 hash value
00490826.vmdk	vmdk	00490826	1024	4eded3c8f9d193e37b11646457add3a6
01558688.vmx	vmx	01558688	2980	de13aada3e12727f5a28c8914ec7f1c4
01558960.vmx	vmx	01558960	2979	26cb192840a22e6bec3be543057ce9c6
01558992.vmx	vmx	01558992	2980	dbaad0c6884729ea00207f62f5b9be50
01559008.vmx	vmx	01559008	2979	fedbd6700ae465bf317fe43f61c03067
01559024.vmx	vmx	01559024	2979	b89e2e73ab1a32a5eca521df02f3085e
01559040.vmx	vmx	01559040	2978	6c6670731cd7290ec531b74b2d8bbf8e
01559168.vmx	vmx	01559168	2979	b218a13ea0f77e65eae16b238ebc1dc4
01559264.vmxf	vmxf	01559264	3260	85a72f0c27f8b62dcb73cda2603dd092
01559424.vmx	vmx	01559424	2978	410c79a74182e10d9939580c255df447
122757120.log	log	122757120	402827	a82070151280c442df30bba25c3422fd
123150336.log	log	123150336	372585	392648383f172ed7302559b1e2372176
124286976.log	log	124286976	226770	2700fc2b81fb169c57a2de255d69f77a
131203072.log	log	131203072	113428	7139d9282dafa3a09e0decc28894029b
157386752.log	log	157386752	943282	9c16e122c8db044faa225b6b7b8ef046
159092736.log	log	159092736	227094	fle77b240b937b2313a5d36ba4819e45
88936448.log	log	88936448	300007	383a9a599a50d9c68fd480d7bf112d14
63705088+2048-flat.vmdk	-flat.vmdk	63705088	12884901888	2f4ed440bad5c542341189688c0137da

Note: This table lists all carved files from the "Boston_volume_20150205.raw" image that could be associated with the "Bravo" VM. Red typeface denotes the files identified as the most current versions and used to reconstruct the "Bravo" VM.

Appendix E – Charlie VM

Table 17

"Charlie-flat.vmdk" file descriptor metadata from volume restoration image:

Description	Metadata file name	Offset	Length (bytes)	Value	Interpretation
ID	.fdc.sf	0x0D8AD200	4	44 8A 80 04	inode ID = 0x04808A44
ID2	.fdc.sf	0x0D8AD204	4	20 00 00 00	inode2 ID2 = 0x20 (unknown purpose)
Nlink	.fdc.sf	0x0D8AD208	4	01 00 00 00	Nlink = 0x01
Type	.fdc.sf	0x0D8AD20C	4	03 00 00 00	Type=0x03 (file – from vmfs-file.h)
Flags	.fdc.sf	0x0D8AD210	4	00 00 00 00	
Size	.fdc.sf	0x0D8AD214	8	00 00 00 00 03 00 00 00	12884901888 Bytes
Block Size	.fdc.sf	0x0D8AD21C	8	00 00 10 00 00 00 00 00	1048576-Byte block size (1MB)
Block Count	.fdc.sf	0x0D8AD224	8	AB 20 00 00 00 00 00 00	43808 Blocks
MTime	.fdc.sf	0x0D8AD22C	4	5F 27 F8 54	Thu, 05 March 2015 09:52:31 UTC
CTime	.fdc.sf	0x0D8AD230	4	12 27 F8 54	Thu, 05 March 2015 09:51:14 UTC
ATime	.fdc.sf	0x0D8AD234	4	5F 27 F8 54	Thu, 05 March 2015 09:52:31 UTC
Mode	.fdc.sf	0x0D8AD240	4	80 01 00 00	0x0180
ZLA	.fdc.sf	0x0D8AD244	4	03 00 00 00	0x03
Type	.sbc.sf	0x0E3162BC	4	03 00 00 00	Type 3 (File)
Block ID	.sbc.sf	0x0E3162C0	4	44 8A 80 04	Block ID = " 448A8004", same as " Node ID" from .fdc.sf.
Record ID	.sbc.sf	0x0E3162C4	4	20 00 00 00	Record ID = 32
Name	.sbc.sf	0x0E3162C8	128	43 68 61 72 6C 69 65 2D 66 6C 61 74 2E 76 6D 64 6B 00 00 00 …	" Delta-flat.vmdk"

Note: Adapted from vmfs-tools source code (Fillott & Hommey, 2012). This table shows the metadata associated with the " Charlie-flat.vmdk" inode within the restoration image.

Appendix F – Non-consecutive Writes Hypothesis

Table 18

Carved files from "Dallas" volume

Filename	Length (Bytes)	MD5 Hash	Filename	Length (Bytes)	MD5 Hash
00490914.vmxf	263	c915a29ad72ad4c1c8760ef2b6e31aa3	01558992.vmx	3069	cada1d806a2bbe4cf79019c49ccc1d22
00490938.vmxf	261	978108079942c4cae493ab8f073c4b20	01559008.vmx	2856	fee31a2ff43569d2a60b53396e41885a
00491010.vmxf	261	bcd7190e68973a2658a0cc9e94e3deaf	01559024.vmx	3067	c3ff9a500a79ac7b82485922a515fcc6
00491030.vmxf	261	ada8ffd32096ae5a46f214b97eab48af	01559040.vmx	3080	b0bffd315b233ba53f9be0c2ab5abf8f
00490926.vmdk	1024	eed930657b633e615e69c74caf054041	38537733.vmdk	1024	a9a32a58fb8fdc87d6167617fc7a6daa
00490950.vmdk	1024	ea30e66924deb51ee2b0cd9855827d37	84742144.log	683345	bf0b3d6514ca29aaa446c1e5a3bfec60
00491042.vmdk	1024	b00a6c14b3a4b7a7777ac50457b7063c	84744192.log	683404	c247668f3a3fff1140fc71d501515c04
00491070.vmdk	1024	8342f7103f0dc71b5d0bd03adbf6afdb	88938496.nvram	1024	85de00b97ec3a2482c5db044683175cc
01558784.vmx	19451	877b2b1d54f50fb0a36a7a5d82b6c9cd	89131008.nvram	1024	7e1694ae44645881ae4e9a375cadbfc3
01558816.vmx	3067	c3ff9a500a79ac7b82485922a515fcc6	135081984.log	683726	80d1e581f1398c0287d51d5ce0d0b584
01558832.vmx	3080	b0bffd315b233ba53f9be0c2ab5abf8f	135084032.log	766747	23aa66efaa36bf127a68db8bd4e2a5b1
01558848.vmx	3079	b1ade7d794fe848e3b19f610cb3fd5bd	135086080.log	219376	606338f42caf450ea21c7d57f01f6904

01558864.vmx	3066	065e8f1faa0c21df65 68ce2083cfb547	139302912.log	207076	064d581fe69c9e281 01b170b9c409f19
01558912.vmx	3069	cada1d806a2bbe4cf 79019c49ccc1d22	139307008.log	221678	d2a18c93900b2a3d c53d88a94b442ff0
01558928.vmx	3068	341e6f14c2accc53b d0e54867bb1510e	139470848.nvram	1024	6a8bb0eec4f433a25 63dba71c3352f11
01558944.vmx	2855	47733a927c2b6f4c1 1cbe4e7cb2107f0	143994880.log	221665	f4e301780de71f503 e971672622f4c12
01558960.vmx	2856	fee31a2ff43569d2a6 0b53396e41885a	148455424.log	220171	01eb7d4f0d618fdc0 6db79fcfcdee5cd
01558976.vmx	3067	c3ff9a500a79ac7b82 485922a515fcc6	148383744.nvram	1024	7e1694ae44645881 ae4e9a375cadbfc3

Note: This table lists the VMware files carved from the "Dallas" volume as part of the testing for the "non-consecutive writes" hypothesis. Each filename's name is the first sector where the file was found. Duplicate items are denoted by a color, matching the color of each identical file.

Table 19

"Dallas" carved virtual disk files

Volume starting sector	**Volume ending sector**	**Number of sectors**	**File carved as**	**MD5 Hash value**
38,539,264	63,705,088	25,165,824	38539264-flat.vmdk	105c9d1d8df4878f46c6eb2f84d6107e
59,510,784	84,676,608	25,165,824	63705088+2048-flat.vmdk	f90090fa7cbc14b1a8a7e20f55792c94
84,676,608	109,842,432	25,165,824	97521664+2048-flat.vmdk	7f4539a3f9830e827c2f4d78cc707d79
109,850,624	135,016,448	25,165,824	131338240+2048-flat.vmdk	f4a45ac054e32036cace5bbe9297be83

Note: This table lists the virtual disk files identified and carved from the "Dallas" volume. Note the overlap between the first and second virtual disk files due to "38539264-flat.vmdk" being a thin-provisioned virtual disk file.

Appendix G – Unallocated Percentage Hypothesis

Table 20

Carved files from "Edmonton" volume

Filename	Length (Bytes)	MD5 Hash	Filename	Length (Bytes)	MD5 Hash
00490794.vmxf	261	50a1f4f2db82065e6 9fe8e0979ccb3d1	01558768.vmx	2985	40704101d33e4c02 a93abc0c5c45104a
00490814.vmxf	261	f1b78f61492a718e0f 70068e5b556480	01558784.vmx	2984	37f5eaaea855b2912 d0d4a839fce54b2
00490838.vmxf	263	ff9746f836863193a 8defc2e4558b69e	01558800.vmx	2982	303e8c1408ecea03e 2ad576ba493dbb8
00490938.vmxf	261	21b231a5c7ce4c450 91c6e7b488014a0	01558832.vmx	2983	0431beeef110b584c 8de4fd19b52ae9b
00490826.vmdk	1024	2029c98dbd533493a 5c3a2e2d454c365	01558848.vmx	2982	b1f13a6ffaf12bcbe7 fed2169dd06a6a
00490850.vmdk	1024	1aa17fe2d1d319844 a72de3cba51f902	14895104.log	683229	8a1e5c17ec2c86458 71768fe6c0377b7
00490878.vmdk	1024	4b9041c74342c5d05 ee322978bc1f31d	19283968.nvram	1024	cd1278de31696ef1b d41137a979351fa
00490950.vmdk	1024	f95efa6ad0bcf65455 c4a5f061a63856	281628672.log	685199	fa0cff3c535afce408 20c4722ae768cd
00490978.vmdk	1024	4617cb527cade13a0 f51f9920e5181c8	286017536.nvram	1024	6b53d2a4363d0e50 be8f83b4579cef07
01558688.vmx	11175	4b67f6c10d6cff1f6a 6d56b5713797f7	289654784.log	687859	f2b05cb89928c604e 49c49f80050ec29
01558704.vmx	2983	cd4985dbbd72e51dc d2b11365d42f2be	294240256.nvram	1024	cd1278de31696ef1b d41137a979351fa
01558736.vmx	2997	cc9e470ea4a975d56 9547dff73d2ac19	294553600.log	683229	9fb46d8f7b5201077 f1f3a3601418ee3

01558752.vmx	2996	5b08689b543adbd8 9934d4607e113d24	299036672.nvram	1024	da623cbcbb03113d b96d8b6fa960892e

Note. This table lists the files carved from the " Edmonton" VMFS volume as part of the " unallocated percentage" hypothesis. Each filename's name is the first sector where the file was found. Duplicate items are denoted by a color, matching the color of each identical file.

Table 21

"Edmonton" carved virtual disk files

Volume starting sector	**Volume ending sector**	**Number of sectors**	**File carved as**	**MD5 Hash value**
206,065,664	243,605,504	37,539,840	206065664-flat.vmdk	5411076220507dde418bc8823c209588
243,814,400	281,354,240	37,539,840	243814400-flat.vmdk	6fa2d00649b536bc4cf84c088fe58b28
286,019,584	323,559,424	37,539,840	286019584-flat.vmdk	c56b10c234ba03b3c1921b0af7b98e4d
310,110,208	347,650,048	37,539,840	310110208-flat.vmdk	c5f56eca6a43955c3df701685c8bc2a2
310,110,208 & 2,514,944	335,544,320 & 14,827,519	37,539,840	310110208_merged-flat.vmdk	44b5caf80368825aaa70a7c8a8d5e8bc

Note: This table lists the virtual disk files identified and carved from the " Edmonton" volume. Note the overlap between the third and fourth virtual disk files due to " 310110208-flat.vmdk" being a thin-provisioned virtual disk file. " 310110208-flat.vmdk" extended beyond the end of the " Edmonton" volume. The virtual disk file " 310110208_merged-flat.vmdk" was a concatenation of " 310110208-flat.vmdk" and sectors 2,514,944 through 14,827,519 of the "Edmonton" volume.