

The Process of Acquiring Live Systems

Daniel DeFreez
Jonathan McCoy
2009

Grant funding from Southern Oregon University

Abstract

This paper describes a process for acquiring a live system independent of the method of deployment. We discuss how an acquisition may be staged so as to minimize the destruction of potential evidence and the insertion of artifacts. We also discuss the tools used and how they can be packaged to perform an acquisition. Attention is paid to how dependence on the target system may jeopardize the acquisition process and what steps can be taken to mitigate such risk.

1 Introduction

In a digital investigation, a computer acquisition, either of a running system or after it has been powered off, is the primary tool for data collection and therefore the primary source of evidence. Post-mortem acquisition (collection of data after the target system has been powered off) is a well-established process, while live system acquisition is a relatively new and quickly evolving process. In this paper we describe a set of best known practices for acquiring a live system, common pitfalls, inherent limitations, and possible countermeasures.

The process of acquiring a live system is primarily of interest to digital investigators and incident responders, who must typically adhere to procedures and practices that create *forensically sound duplicates*. According to Craig Ball,

“A ‘forensically-sound’ duplicate of a drive is, first and foremost, one created by a method which does not, in any way, alter any data on the drive being duplicated. Second, a forensically-sound

duplicate must contain a copy of every bit, byte and sector of the source drive, including unallocated ‘empty’ space and slack space, precisely as such data appears on the source drive relative to the other data on the drive. Finally, a forensically-sound duplicate will not contain any data (except known filler characters) other than which was copied from the source drive.” (Ball)

This concept works well for post-mortem computer forensics, but the procedure for acquiring a live system is necessarily different than the procedure for post-mortem acquisition because the *act* of acquisition, as well as the passing of time, modifies the system being acquired. A great deal of discussion has been generated on the topic of what constitutes a forensically sound duplicate in the context of new sources of evidence, including data from a live system (Murr, 2006) (Carvey, What is "forensically sound", 2006) (Bejtlich, 2006). This paper subscribes to Mike Murr’s revised definition of a forensically sound duplicate:

“A forensically sound duplicate is a complete and accurate representation of the source evidence. A forensically sound duplicate is obtained in a manner that may inherently (due to acquisition tools, techniques, and process) alter the source evidence, but does not explicitly alter the source evidence. If data not directly contained in the source evidence is included in the duplicate, then the introduced data must be distinguishable from the representation of the source evidence. The use of the term complete refers to the components of the source evidence that are both relevant, and reasonably believed to be relevant.” (Ball)

The acquisition process laid forth in this paper is designed to fulfill the requirements of Murr's definition of a forensically sound duplicate, with one exception. We take the distinction between inherent modification and explicit modification to mean that while it is important to recognize that the target will inevitably be modified, and while it is important to minimize the impact of the collection process on the system, it is of paramount importance to ensure the accuracy of the data collected, even at the expense of the source evidence. We separate out the acquisition process into a series of stages in a definite order, what we call the acquisition process. While the acquisition process may significantly alter the source evidence, it is structured so that the data collected provides an accurate duplicate of the *original* source evidence as possible, neglecting the end state of the machine.

Section two of this paper details the stages of the acquisition process, what order they should be put in, and what tools are necessary to complete each stage. Section three discusses the various methods by which this process can be deployed to a target system. Section four engages the problem of acquiring data in an untrusted environment. Section five introduces some thoughts on how the acquisition process might be countered from an anti-forensic standpoint.

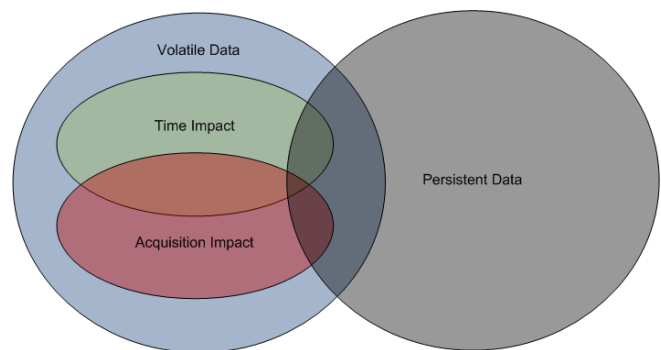
2 Acquisition

When collecting data from a live system for analysis, we are primarily concerned with volatile data that is irretrievable once the system is shut off. In an optimal procedure, each stage of the acquisition would not modify any data being acquired by a postcedent stage. Optimal procedures are tough to come by, but we can try.

Computer data can be broken down into two main areas: volatile data and persistent data. Volatile data is all data that will be lost, altered, or made inaccessible when the machine is powered off. Everything else is persistent data, which can

be retrieved after power off. Volatile data can be further modified by the passage of time or the acquisition process itself. Volatile data is best exemplified by RAM, which loses its contents relatively quickly after a computer is powered off. Persistent data, on the other hand, is best exemplified by the data on a hard drive, which persists across power cycles.

In order to preserve the integrity of data during the acquisition process, we must understand the ways the acquisition process can alter it. Both time and acquisition activity can impact data on a system. As shown in the diagram below, not all data is susceptible to both of these impacts (and not all data is equally volatile).



If the evidence produced by the acquisition is to be presented in a court of law, a full understanding of the acquisition impacts is necessary. For example, inserting a USB drive creates a new key in the registry and alters the `setupapi.log`.

While the process of acquiring a live system necessarily modifies the system being acquired, we have tried to organize the stages of our acquisition such that any given stage minimally impacts later stages. Nearly every stage contaminates RAM, so it is necessary to capture RAM first. We begin by dumping physical memory, which may include unallocated memory. A physical memory dump does not include memory that has been swapped to the hard disk (the page file), so then we dump the page file. The next step is a logical dump of process memory, which prevents smearing and includes any process

space that has been swapped out. After this, we capture system information such as network configuration, the current process list, and user information.

2.1 Raw Memory

The first piece of data collected by our tool is a raw copy of physical memory. We are using ManTech's Memory DD¹ (hereafter referred to as MDD) to dump RAM. Because MDD requires kernel-level access, administrator privileges are required on the target to perform this step. MDD creates a single file that contains a copy of physical memory. However, this process can take some time to complete (typically on the order of tens of minutes), which is significant because it leads to an artifact known as *smearing*. Smearing happens when the data is changing as it is being read. When two related pieces of data are read at different times, the fact that one piece of data could have changed before the second is read can lead to an incoherent overall picture. Imagine taking a picture of the left half of a blue car, then, before you can take a picture of the right half, the car is painted red. When the two pictures are put together we are left with an inconsistent view of the car.

2.2 Page File

In order to capture a complete snapshot of memory we must include paged-out memory. Operating systems utilize page files – also known as swap files – to temporarily store infrequently accessed portions of memory on disk, allowing RAM to be reallocated. Page files are easily disturbed by system activity. In particular, dumping process memory will significantly modify page files. This is why it is necessary to copy page files before dumping process memory.

Windows keeps page files locked, preventing normal access; however, there are other methods of accessing them. We can either read the files raw or by using Shadow Copy. This is done by snapshotting the volume and retrieving the previously locked files from the snapshot. This has the benefit of avoiding smearing, but leaves footprints. Directly accessing the disk (a raw copy) would avoid these footprints but run the risk of smearing.

2.3 Process Memory

There are fundamental disadvantages inherent to a raw memory dump: smearing and difficulty of reconstruction. We can overcome these issues by dumping each process's memory individually. This is done by pausing each process before dumping its memory, and then restarting it. As that the process is paused while we are reading it, smearing is avoided.² This also makes post-analysis easier. This is done by utilizing the Windows API to pause the threads of each process. While the process is paused, *pmdump*³ is used to capture memory. The threads are restarted after the dump has finished.

There have been reports that the target machine may become unstable while dumping process memory in this fashion (Turini, 2002). We attempted to reproduce this instability by repeatedly pausing the processes on numerous Windows XP SP3 machines for varying periods of time. No system crashes were observed.

Another way to reduce smearing without pausing the process is to manipulate process priority levels so that the target process receives as little CPU time as possible, while the acquisition process receives as much CPU time as possible.

¹ <http://www.mantech.com/msma/mdd.asp>

² Other processes still have the ability to write into the paused process, which may cause smearing.

³ <http://ntsecurity.nu/toolbox/pmdump/>

2.4 System Information

In a live acquisition it is important to retrieve as much volatile system information as possible before the machine is powered off. As is usually the case, network information is the most volatile and therefore grabbed first. The next most volatile is the tasklist, followed by the rest of the data collected, which should remain relatively static during the acquisition.

Network Information

We capture a list of open network connections, the ARP table, the routing table, and interface configuration.

Tools used:

- netstat -ano
- arp -a
- netstat -r
- ipconfig /all

Process List

We capture the list of processes known to be running, and the services hosted by each process.

Tools used:

- tasklist /v
- tasklist /svc

System and User Information

We capture the currently logged on user(s) and general system information, including environment variables, the computer name, uptime, OS version, registered owner, etc.

Tools used:

- psloggedon /accepteula
- set
- psinfo /accepteula

It may also be worthwhile to capture any clipboard data, the keyboard buffer, and take a screen capture, but we did not include this functionality in our prototype.

2.5 Encrypted Disk Detection

The easiest way for a person to thwart post-mortem forensic techniques is by using full disk encryption. As such, it is imperative that it be determined whether a number of disk encryption solutions exist, of which the most popular are Truecrypt, Microsoft Bitlocker, and PGP whole disk encryption. Because the vast majority of people employing full disk encryption will be using one of these pieces of software, the most practical way to determine whether or not a disk is encrypted is to look for signatures of these pieces of software. JADsoftware's Encrypted Disk Detector (EDD) can effectively detect these three pieces of software. It is freely available, and can easily be added to any live system acquisition process.

Of course, if the disk is encrypted in a different fashion, or if the encrypted volume is not currently mounted, then EDD will not detect it. Moreover, if the disk is encrypted at the hardware level, then EDD will not detect it. Hardware level encryption may be detected by comparing the hard drive model number in Windows with a list of known encrypted hard drives, but the hard drive model information in Windows could conceivably be faked. If the non-standard encryption software is used, it could conceivably be detected by requesting direct access to the hardware, but the authors know of no practical way of implementing this in a portable fashion for all drives.

3 Deployment Methods

It is desirable to place the toolset on a portable medium such as a CD, a conventional USB drive, or a U3 stick. Each method has advantages and disadvantages, and it is up to the person developing the toolkit to decide what delivery method is most appropriate. Often having both a CD and USB drive available is a good idea.

CDs have long been a standard delivery method for forensic response toolkits (Mandia & Prosis, 2003). CDs provide ample room to store

all of the necessary tools, are inherently write-protected, and can be used with the vast majority of systems. The write-protection feature ensures that the tools used during an incident have not been altered. This same feature, however, causes some difficulty because it does not provide the investigator any room to store the data collected. Either a separate storage device such as an external hard drive must be brought along to store the data, or the results need to be piped over the network with a utility such as netcat or cryptcat (Mandia & Prorise, 2003). Bringing along a separate storage device can be moderately inconvenient, and the network approach does not lend itself to an easily scriptable acquisition.

If the data is going to be stored on an external device, then it makes sense to use a USB drive for both the tools and data storage. A conventional USB drive can be used, but there are certain advantages afforded by a U3 device. SanDisk developed the "U3 smart drive" as a platform for portable computing. A U3 drive presents itself to Windows as a USB hub that has both a CD drive and a mass storage device attached. From SanDisk's point of view, the advantage of emulating a cd-rom is that Windows enables autorun for CDs by default. When a user plugs in a U3 device, the "U3 Launchpad," which comes preloaded on the CD portion, is automatically started. The Launchpad is used to launch specially packaged portable applications such as a web browser or antivirus. This Launchpad can be replaced with a forensic acquisition toolset. Using the autorun features of U3, an entire acquisition can be scripted and automatically started when the drive is inserted in the target system. The results of the acquisition can be stored on the data portion of the U3 drive.

4 On Trusted Tools and Trusted Shells

It is obvious that the output of tools supplied by a suspect system is, itself, suspect. If we open up a command shell using the copy of cmd.exe supplied by the system we are acquiring,

any commands initiated by that shell might be tainted by malicious code. Furthermore, cmd.exe itself might be intact, but the tools themselves might be tainted, the libraries on which our tools depend on might contain malicious code, even the operating system kernel itself cannot be trusted.

The natural response is for the forensic analyst to bring trusted tools along and hash them before and after, but this is more difficult than it might seem at first glance. People have long relied on "trusted shells," i.e. known, good copies of a shell, to maneuver around dubious environments. In Linux forensics this is often accomplished by statically compiling all of the necessary tools beforehand (Forensic Analysis of a Live Linux System, Pt. 2, 2004). Without access to the source code of many of the Windows utilities we are using, this is not a viable solution. In Windows, we rely on the fact that, unless a Dynamic-Link Library (DLL) path is explicitly defined, the directory from which the application is loaded is the first place Windows looks when loading DLLs (Microsoft, 2009). Bringing along all of the required DLLs, however, can be a cumbersome process. Not only can the number of necessary DLLs be rather large, but using system DLLs across different versions of Windows introduces issues of instability.

A somewhat cleaner, yet unverified approach is to virtualize the later stages of the acquisition process with an application virtualization suite such as VMware's Thinapp or Microsoft's App-V. We used Thinapp for a proof-of-concept which offered a few advantages. First, Thinapp allows the packager to define isolation levels that prevent the application from modifying the filesystem or registry unless given explicit access. Second, because the entire filesystem is virtualized, any attempts to load libraries off of the target system will fail unless they are provided with the package, ensuring that only known, good libraries are used. Third, Thinapp compresses the entire package.

It is important to realize that each of these solutions is still vulnerable to a kernel-level rootkit. All processes are inherently dependent

upon the kernel to provide access to operating system resources, and if the kernel itself has been tainted, all data generated will be suspect. This is thrown into sharp relief by the fact that not *all* DLLs can be included in the acquisition disk or virtualized. Take ntdll.dll as an example. This DLL, which contains a large number of NT kernel functions, is mapped by Windows into the context of *every* process, including the System process. This mapping is performed in kernel-mode (Function Pointers, 2007) and is particularly difficult to perform by hand. It stands to reason that it would be even more difficult to include a copy of ntdll.dll that would work across multiple versions of Windows.

It may also be equally important to gather compromised information from a system, as proposed by Harlan Carvey (Carvey, Thoughts on using a "trusted shell", 2009). Rootkits can be exceedingly difficult to find. One method of detecting a rootkit is symptomatically. Data can often be accessed in a multitude of ways, either directly or through the operating system. By asking the operating system for data, even when we already can retrieve it raw, we can look for discrepancies and infer the presence of a rootkit (Carvey, Thoughts on using a "trusted shell", 2009). The registry, for example, can be accessed either by Windows or raw off the hard drive. If the registry is accessed through Windows it allows for interception and can be modified by rootkits. By gathering both sets of data you can check for possible discrepancies, indicating the possible presence of malicious software.

4 Conclusion

Any Live System Acquisition will change the target system. However with proper methodology the impact can be mitigated and for the most part known.

Works Cited

Ball, C. (n.d.). *Computer Forensics for Lawyers Who Can't Set a Digital Clock*. Retrieved October 4, 2009, from Craig D. Ball, P.C.:
http://www.craigball.com/CF_0807-Digital%20Clock%20article%20only.pdf

Bejtilch, R. (2006, August 3). *Forensically Sound Evidence*. Retrieved November 4, 2009, from TaoSecurity:
<http://taosecurity.blogspot.com/2006/08/forensically-sound-evidence.html>

Carvey, H. (2009, August 26). *Thoughts on using a "trusted shell"*. Retrieved August 26, 2009, from Windows Incident Response Blog:
<http://windowsir.blogspot.com/2009/08/thoughts-on-using-trusted-shell.html>

Carvey, H. (2009, July 18). *Update on USB Devices*. Retrieved September 2, 2009, from Windows Incident Response Blog:
<http://windowsir.blogspot.com/2009/07/update-on-usb-devices.html>

Carvey, H. (2006, April 03). *What is "forensically sound"*. Retrieved November 4, 2009, from Windows Incident Response Blog:
<http://windowsir.blogspot.com/2006/08/what-is-forensically-sound.html>

Forensic Analysis of a Live Linux System, Pt. 2. (2004, April 12). Retrieved October 29, 2009, from Security Focus:
<http://www.securityfocus.com/print/infocus/1773>

Function Pointers. (2007, September). Retrieved October 28, 2009, from Uninformed:
<http://uninformed.org/index.cgi?v=8&a=2&p=14>

Mandia, K., & Proise, C. (2003). *Incident Response & Computer Forensics*. McGraw-Hill.

Microsoft. (2009, October 8). *Dynamic-Link Library Search Order*. Retrieved October 28, 2009, from MSDN:

<http://msdn.microsoft.com/en-us/library/ms682586%28VS.85%29.aspx>

Murr, M. (2006, August 2). *"Forensically Sound Duplicate"*. Retrieved November 4, 2009, from Forensic Computing:
<http://www.forensicblog.org/2006/08/02/forensically-sound-duplicate/>

Turini, D. (2002, September 29). *Win32 process suspend/resume tool*. Retrieved September 6, 2009, from The Code Project:
<http://www.codeproject.com/KB/threads/pausep.aspx>

Walters, A., & Petroni, N. L. (2007). *Volatools: Integrating Volatile Memory Forensics into the Digital Investigation Process*. *Blackhat DC 2007*.