Chapter 25

AN EVIDENCE ACQUISITION TOOL FOR LIVE SYSTEMS

Renico Koen and Martin Olivier

Abstract Evidence acquisition is concerned with the collection of evidence from digital devices for subsequent analysis and presentation. It is extremely important that the digital evidence is collected in a forensically-sound manner using acquisition tools that do not affect the integrity of the evidence. This paper describes a forensic acquisition tool that may be used to access files on a live system without compromising the state of the files in question. This is done in the context of the Reco Platform, an open source forensic framework that was used to develop the prototype evidence acquisition tool both quickly and efficiently. The paper also discusses the implementation of the prototype and the results obtained.

Keywords: Live systems, evidence acquisition, Reco Platform

1. Introduction

Traditional or "dead" forensics involves the recovery of evidence from computer systems that have been powered down [1, 3]. Unfortunately, shutting down a system results in the loss of important volatile data. Also, it may not be possible to shut down vital enterprise systems to conduct forensic investigations.

Live forensics [1, 3] is an attractive alternative to dead analysis, enabling an investigator to recover and analyze data while a computer system is running. However, this technique does have limitations due to the possible presence of an intermediary, such as a rootkit, which may modify data before it is presented to the investigator. Even if a rootkit is not present, the mere fact that an untrusted piece of code, in the form of a normal operating system service, was used to retrieve the forensic data may cast doubt on the validity of the data.

Please use the following format when citing this chapter:

Koen, R. and Olivier, M., 2008, in IFIP International Federation for Information Processing, Volume 285; *Advances in Digital Forensics IV*; Indrajit Ray, Sujeet Shenoi; (Boston: Springer), pp. 325–334.

Operating system services execute in various layers. Depending on its location and functionality, a rootkit may hijack services in any of these layers. In particular, a rootkit hides its presence by modifying system services. For example it may exclude itself from the list of processes displayed to users or it may remove the names of its own files from file lists. In order to do this effectively, a rootkit ideally operates at the lowest possible layer (kernel layer); if it knows what information has been requested, it is easier to remove traces of itself.

Given this fact, the reliability of digital evidence retrieved from a lower layer is potentially higher than that retrieved from a higher layer. Obtaining information from a lower layer not only bypasses rootkits in the higher layers, but also shortens the chain of services used to answer a query. If fewer services are involved, the probability that one of them has been modified is lower than in the case of a longer chain of services. However, if data is to be retrieved from a lower layer, it is necessary to reconstruct the higher-level information structures – ideally using code that is known to be reliable.

This paper describes a prototype forensic acquisition tool that accesses low level information from a disk during live analysis. The tool uses its own code to reconstruct the logical files that exist above the low level information on disk. The implementation is based on the Reco Platform [7], which was designed to allow rapid prototyping of forensic tools, including tools that emerge from academic research and one-of-a-kind tools needed for special investigations. The platform ensures that as much code as possible is reused; this increases the reliability of evidence and its potential admissibility in legal proceedings. The Reco Platform also enables investigators to utilize other tools built using the platform, thereby increasing the range of collection and analysis possibilities.

2. Live Evidence Acquisition

It is important to ensure that digital evidence is not modified during a live acquisition process. Walker [17] observes that even a single file timestamp found to be later than the date of acquisition may cause digital evidence to be declared inadmissible in court. Any file accessed from a logical partition, which is mounted in standard read/write mode, may have some of its attributes (e.g., access time) modified by the operating system when it is accessed. The ability of the operating system to update the file access time is useful for system administrators, but it is highly undesirable for digital forensic investigators. The use of standard file access routines supplied by the operating system should, therefore, be avoided during live evidence acquisition.

Casey [5] notes that standard operating system copy routines should also be avoided due to presence of rootkits. Live acquisition software should, therefore, have the capability to perform low-level file access without the help of the operating system. Moreover, all files should be accessed in read-only mode to preserve the integrity of file data and metadata [5]. This is because the state of a file system mounted in read/write mode is implicitly modified whenever a file is accessed.

Another, more technical, requirement for a live acquisition tool is static compilation and storage of binaries used to perform acquisitions. According to Adelstein [1], an investigator should never trust binaries stored on the system in question; rather, the investigator should employ statically-compiled binaries that do not use external libraries. The binaries should, therefore, be stored on CD-ROM to ensure that they cannot be altered.

3. Reco Platform

The Reco Platform was designed to provide the low-level functionality required by digital forensic tools, thereby decreasing the time and expertise required to develop prototypes. The platform is written in C++ and compiles under Linux and Windows. It is published under the popular GNU license [6], which enables the code to be used freely in open source projects. Like other open source software [12], the Reco source code can be inspected by programmers, helping create a stable, forensically-sound platform that will compare favorably with expensive commercial toolkits in terms of quality.

The Reco Platform provides multiple layers of software abstraction to support forensic applications development and rapid prototyping. The lower layers offer core functionality, giving developers limited abstraction but more control; the upper layers supply a higher degree of abstraction, but little control. Developers may choose the software layer that strikes the right balance between abstraction and control. This section provides an abbreviated discussion of the Reco architecture; interested readers are referred to [8] for additional details.

The Reco Platform currently consists of five layers: (i) physical, (ii) interpretation, (iii) abstraction, (iv) access, and (v) logging (Figure 1). The physical layer, which offers the lowest degree of abstraction, emulates the hardware devices from where digital evidence is collected. Digital evidence in a popular forensic format (e.g., disk image or TCP-Dump trace) is supplied to the physical layer. The physical layer uses this digital evidence to emulate the functionality expected by the device

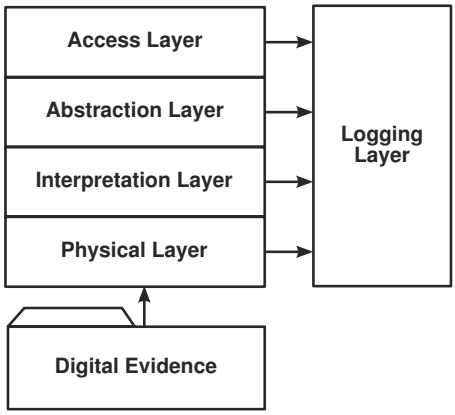

Figure 1. Reco Platform layers.

drivers that provide access to the evidence. This simplifies the task of modifying third-party software drivers for use with the Reco Platform.

The interpretation layer is the second layer in the Reco hierarchy. This layer typically models device drivers. The purpose of the interpretation layer is to read low-level data supplied by the physical layer in block or stream formats and convert it to a higher level of abstraction such as file-based information (for block devices) and temporal information (for stream-based devices).

The third layer is the abstraction layer. Its purpose is to supply functionality that is not specific to any operating system or computing platform. This is done in order to hide unnecessary details that may obscure an investigator's perception of the information conveyed by digital evidence. Another purpose is to enable investigators to identify relationships that may exist among different pieces of digital evidence. Tallard and Levitt [16] note that this functionality is crucial to filtering data that is not relevant and to creating abstract objects that can be interpreted in a relational manner with other objects.

The fourth Reco layer, the access layer, provides access to information generated by the lower layers. Searching, indexing and access control functionality are implemented at this layer. Visual abstraction may also be implemented at this layer to display digital information in a human-oriented format. Wang [18] has observed that digital evidence is not well perceived by the human senses. The access layer enables investigators to view digital evidence in an organized and understandable manner, helping increase their efficiency.

The uppermost logging layer provides the logging facilities needed in digital forensic environments. Logging is an important part of any

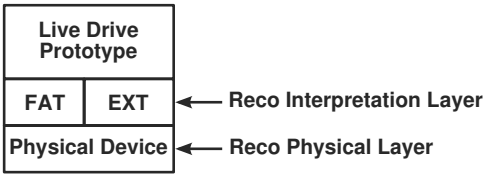

Figure 2. Dependence of the prototype on the Reco Platform.

digital forensics tool. According to NIST [11], tools used for backing-up disk data should log all errors and resolve the errors. The logging layer is used at every level of abstraction in the Reco hierarchy to document the actions applied to digital evidence.

4. Prototype Development

This section describes the prototype used for live analysis based on the Reco Platform. The prototype relies on the Reco physical layer to provide access to file system images and on the interpretation layer to supply file access routines for accessing files in read-only mode (Figure 2).

The Reco Platform is designed to work with Linux and Windows. In keeping with the Reco philosophy, the prototype was developed to permit source code to compile and run under both operating systems. The Reco Platform supports the FAT12, FAT16, FAT 32, EXT2 and EXT3 file systems, which were more than adequate for developing the prototype.

The next two subsections discuss issues related to the Linux and Windows prototypes. Note that the low-level implementation details are different for the two prototypes, but the higher-level algorithms are the same.

4.1 Linux-Based Prototype

A device that contains a Linux file system is referred to as a block device [15]. Block devices may be opened like any other file in the Linux environment except that administrative privileges are required. Since a block device can be opened as a file, it is possible to read data from the device; this process is similar to reading data from an acquired hard drive image.

The partition in which a file of interest exists is first located and opened in read-only mode. The Reco Platform is then instructed to use the opened file as the target for analysis. The easiest way to determine which block device represents the logical partition in question

is to inspect the contents of the file `/etc/mtab` [2]. This file contains information about mounted partition types, their mount points and the locations of the block devices containing the partitions. Unfortunately, this technique has certain disadvantages. In particular, administrator (root) privileges are required when a device is opened as a file and access to the `/etc/mtab` file changes the state of the file system. One way to access the `/etc/mtab` file without altering its access time is to open the device on which the file is located via the Reco Platform, access the file in read-only mode and then close the device. This method allows file access without compromising the integrity of the file system, but prior knowledge of the block device that maps to the mounted root partition is required.

4.2 Windows-Based Prototype

In the case of a Windows environment, a logical device is opened as a file and the Reco Platform is instructed to mount the open file as the forensic target. Specifically, the logical device is opened as a file using the `CreateFile()` API call with the filename "\\.\N" where N is the drive letter representing the logical partition [13]. Note that administrator privileges are required to perform this operation.

Next, the `GetLogicalDrives()` API call is used to determine which logical drives are mounted [14]. The call returns a bitmap representing the drive letters of the mounted logical partitions. Using this information in combination with the `CreateFile()` method, it is possible to obtain access to a logical partition as in the case of the Linux prototype. Having determined the partition containing the file of interest, the file is located and opened. The Reco Platform is then instructed to use the opened file as the source of analysis, after which the files stored on the partition become accessible to applications using the Reco framework.

4.3 Implementation

The prototype was written in the C++ programming language; its graphical user interface was implemented using the wxWidgets framework [19]. The prototype involves very little code (not including the Reco code) and was developed in a very short time.

Source files were designed to compile in both Linux and Windows without requiring a special `makefile` or changing the project source code. Platform-specific sections of code were marked for compilation using preprocessor flags specific to the operating system in question. The combination of the two approaches allowed for the development of code

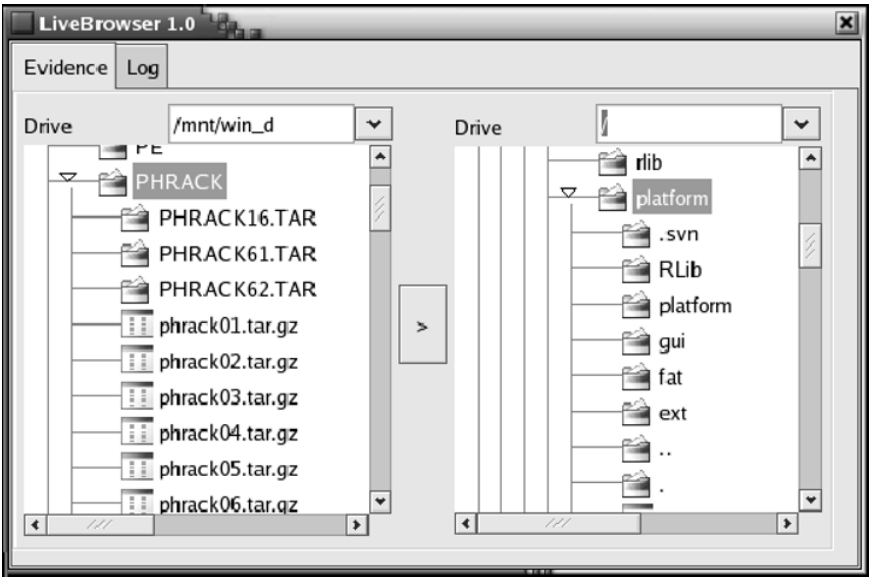

Figure 3. Screenshot of the prototype.

that runs under Linux and Windows without any major compatibility issues.

5. Results

The tool was tested on Linux Fedora Core 4 and Windows XP. Executables were generated that statically linked to the Reco library; this was done to minimize dependence on external libraries.

The results obtained with the two operating systems were similar: regular files could be accessed without modifying them or their metadata. A mounted logical partition could be opened by the prototype, the directories in the partition in question could be browsed and files could be copied to another partition to allow forensic examiners to inspect their contents. Figure 3 shows a screenshot of the prototype.

A comparison was conducted of the access times required by the file system drivers used by the prototype. An application was developed that created images of different sizes on the logical partitions targeted by the file system drivers. The created files were then read, and the time taken to read each consecutive file was recorded for each distinct logical partition.

The graph in Figure 4 shows the efficiency of the drivers used by the Reco Platform. The EXT file system driver shows a linear increase in

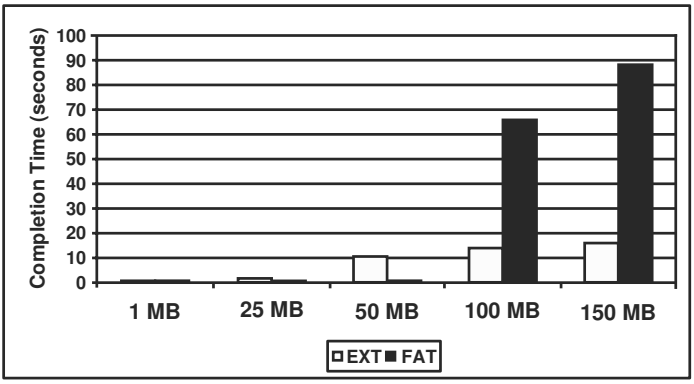

Figure 4. Access times for Reco file system drivers.

access time as the file size increases. This result is expected because more work is performed when more data is accessed.

The FAT file system driver yielded less desirable results. Signs of an exponential increase are seen when the amount of data accessed increases. This is unfortunate because it shows that the Reco Platform is unable to provide fast access to large files stored in a FAT partition.

Because the higher-level operating system layers were bypassed when the acquisitions were performed, it can be assumed that the results obtained would be immune to most rootkits. Note that although rootkits are bypassed using this method, it is by no means a comprehensive way to neutralize rootkits in general. This is largely due to the limited involvement that an operating system has in controlling logical devices.

When access is required to a logical device, the prototype sends a request to the underlying operating system for permission to open a logical drive as a file. When data needs to be read from the logical drive, a read request is sent to the operating system to perform the task. A sophisticated kernel rootkit that has the same file processing capabilities as the Reco Platform could, in theory, return blocks of code that were maliciously engineered to hide traces of data, or it could inject falsified information. Although such a rootkit would be rare due to its complexity, it might be possible for a malicious programmer to develop one using enabling tools like the Reco Platform.

6. Conclusions

The live evidence acquisition tool described in this paper can be used to access files on a live target without compromising the state of the files. The evidence acquisition tool leverages the Reco Platform, an open

source framework designed for the rapid prototyping of forensic tools. With only a few lines of code, it was possible to quickly and efficiently develop Linux and Windows prototypes that provide true read-only access to FAT12/16/36 and EXT2/3 partitions without modifying files and their metadata.

However, two principal limitations exist, both of which should be considered in the context of live analysis. First, access to files stored in a logical partition requires administrator privileges. Second, access to file system data is by no means absolute – the low-level data access mechanisms can be bypassed by sophisticated kernel rootkits.

References

[1] F. Adelstein, Live forensics: Diagnosing your system without killing it first, *Communications of the ACM*, vol. 49(2), pp. 63–66, 2006.

[2] BrunoLinux.com, FSTAB and MTAB (www.brunolinux.com/02-The_Terminal/Fstab_and_Mtab.html).

[3] B. Carrier, Risks of live digital forensic analysis, *Communications of the ACM*, vol. 49(2), pp. 56–61, 2006.

[4] E. Casey, Error, uncertainty and loss in digital evidence, *International Journal of Digital Evidence*, vol. 1(2), 2002.

[5] E. Casey and A. Stanley, Tool review – Remote forensic preservation and examination tools, *Digital Investigation*, vol. 1(4), pp. 284–297, 2006.

[6] Free Software Foundation, GNU general public license, Boston, Massachusetts (www.gnu.org/copyleft/gpl.html).

[7] R. Koen, Reco Platform (sourceforge.net/projects/reco).

[8] R. Koen and M. Olivier, An open-source forensics platform, *Proceedings of the Southern African Telecommunication Network and Applications Conference*, 2007.

[9] W. Kuhnhauser, Root kits: An operating systems viewpoint, *ACM SIGOPS Operating Systems Review*, vol. 38(1), pp. 12–23, 2004.

[10] Linux Journal Staff, Take command: What is dd? *Linux Journal*, vol. 1996(32es), no. 11, 1996.

[11] J. Lyle, NIST CFTT: Testing disk imaging tools, *International Journal of Digital Evidence*, vol. 1(4), 2003.

[12] D. Manson, A. Carlin, S. Ramos, A. Gyger, M. Kaufman and J. Treichelt, Is the open way a better way? Digital forensics using open source tools, *Proceedings of the Fortieth Annual Hawaii International Conference on System Sciences*, p. 266b, 2007.

[13] Microsoft Corporation, CreateFile Function, Redmond, Washington (msdn2.microsoft.com/en-us/library/aa363858.aspx).

[14] Microsoft Corporation, GetLogicalDrives Function, Redmond, Washington (msdn2.microsoft.com/en-us/library/aa364972.aspx).

[15] D. Rusling, The File System (www.science.unitn.it/∼fiorella/guide linux/tlk/node94.html).

[16] T. Stallard and K. Levitt, Automated analysis for digital forensic science: Semantic integrity checking, *Proceedings of the Nineteenth Annual Computer Security Applications Conference*, pp. 160–167, 2003.

[17] C. Walker, Computer forensics: Bringing the evidence to court (www.infosecwriters.com/text_resources/pdf/Computer_Forensics_to_Court.pdf), 2007.

[18] S. Wang, Measures of retaining digital evidence to prosecute computer-based cyber-crimes, *Computer Standards and Interfaces*, vol. 29(2), pp. 216–223, 2007.

[19] wxWidgets, What is wxWidgets? (www.wxwidgets.org).