available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

Forensic discovery auditing of digital evidence containers

Golden G. Richard III, Vassil Roussev, Lodovico Marziale*

Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA

ARTICLE INFO

Article history:
Received 13 February 2007
Accepted 2 April 2007

Keywords:
Digital forensics
Operating systems internals
Filesystems
Digital evidence containers
Auditing

ABSTRACT

Current digital forensics methods capture, preserve, and analyze digital evidence in general-purpose electronic containers (typically, plain files) with no dedicated support to help establish that the evidence has been properly handled. Auditing of a digital investigation, from identification and seizure of evidence through duplication and investigation is, essentially, ad hoc, recorded in separate log files or in an investigator's case notebook. Auditing performed in this fashion is bound to be incomplete, because different tools provide widely disparate amounts of auditing information – including none at all – and there is ample room for human error. The latter is a particularly pressing concern given the fast growth of the size of forensic targets. Recently, there has been a serious community effort to develop an open standard for specialized digital evidence containers (DECs). A DEC differs from a general purpose container in that, in addition to the actual evidence, it bundles arbitrary metadata associated with it, such as logs and notes, and provides the basic means to detect evidence-tampering through digital signatures. Current approaches consist of defining a container format and providing a specialized library that can be used to manipulate it. While a big step in the right direction, this approach has some non-trivial shortcomings – it requires the retooling of existing forensic software and, thereby, limits the number of tools available to the investigator. More importantly, however, it does not provide a complete solution since it only records snapshots of the state of the DEC without being able to provide a trusted log of all data operations *actually* performed on the evidence. Without a trusted log the question of whether a tool worked exactly as advertised cannot be answered with certainty, which opens the door to challenges (both legitimate and frivolous) of the results.

In this paper, we propose a complementary mechanism, called the *Forensic Discovery Auditing Module* (*FDAM*), aimed at closing this loophole in the discovery process. FDAM can be thought of as a 'clean-room' environment for the manipulation of digital evidence, where evidence from containers is placed for controlled manipulation. It functions as an operating system component, which monitors and logs all access to the evidence and enforces policy restrictions. This allows the immediate, safe, and verifiable use of *any* tool deemed necessary by the examiner. In addition, the module can provide transparent support for multiple DEC formats, thereby greatly simplifying the adoption of open standards.

1. Introduction

Information discovered during an investigative inquiry becomes usable evidence only when it passes the rigorous test of admissability. One of the crucial aspects of arguing admissability is demonstrating that the information has been obtained and handled properly and, therefore, the obtained results can be trusted. In traditional forensics, where physical

* *Corresponding author*. Tel.: +1 504 957 5814.
E-mail addresses: golden@cs.uno.edu (G.G. Richard III), vassil@cs.uno.edu (V. Roussev), lmarzial@cs.uno.edu (L. Marziale).
1742-2876/$ – see front matter © 2007 Published by Elsevier Ltd.
doi:10.1016/j.diin.2007.04.002

artifacts are examined and interpreted, sealed evidence bags are a simple but indispensible tool in making that argument. In addition to tamper-evident design, such bags often provide writing space for identifying information, such as the name of the investigating officer, case identifiers, the suspect's name, a description of the item, and the date and time when the bag was sealed. Continuity sections on the bag allow tracking the movement of the bag, noting the chain of custodians who have undertaken the bag's care.

Digital forensic examiners have generally followed a similar approach, with one notable exception – so far they lack the equivalent of a tamper-evident bag for digital information. Currently, the state of the art digital forensic tools operate over standard operating systems' components and use plain files as containers for holding artifacts. One of the problems is that general-purpose tools used during the forensic process might potentially alter the artifact. Although this can be prevented by a write blocker, the issue of documenting which operations the tool performed remains unanswered. For example, if a tool was used to search for malware, was the whole disk searched, or just selected 'typical' places? To deal (in part) with this problem, best practices dictate that all work be performed on a copy of the evidence and all operations performed be recorded in an audit log. For the purposes of this article, we refer to the process of keeping a log of all operations on the evidence as *auditing*.

Under current methodology, which places the burden entirely on the human investigator, auditing is bound to be incomplete, because different tools provide varying levels of auditing information (in a variety of formats), much of which must be recorded manually. Over the course of an investigation, a piece of digital evidence may be touched by many different tools, some of which generate no audit trail of their actions (e.g., *dd* or the command line tools of the Sleuthkit) and some that generate their own audit logs (e.g., FTK). At the end, an investigator is left to piece together these bits of audit trail to create a comprehensive view of what occurred during the investigation. Another potential issue with application-generated logs is that the application is charged with policing itself, which makes the possibility of both the application and its log being faulty much more likely than if they were developed separately. Finally, as a practical concern, failure to manually record a bit of auditing information, such as the MD5 hash generated by *md5sum* for a large disk image, could potentially result in a huge amount of lost time if the operation must be repeated. If generation of auditing details such as these is automated, so much the better.

Anecdotal evidence suggests that many examiners try to circumvent the above problem by using only the results produced by commercial tools of vendors prepared to vigorously defend their product in court (at own expense). There are at least two major problems with this approach. The first one is that investigators are limited to the capabilities of the chosen tools. In general, there is hardly anything special about the functions found in most forensic tool suites – almost all of these are based on functions existent in general-purpose tools (e.g., data transfer, text searches, file type identification, etc.) and, over time, forensic tools simply accumulate more of them in a convenient package. Nonetheless, there will always be a need to use other tools to discover and interpret evidence.

Another issue is that, just like any other piece of software, forensic tools invariably contain implementation errors (bugs) or are based on unsound assumptions. As a simple example, earlier versions of many tools could be fooled into believing that a text file was a Microsoft Windows executable by starting the file with the string 'MZ'. It is not always the vendor's fault – artifacts created in proprietary formats (e.g., many formats in Microsoft Windows) can change with no warning from version to version. While discovered bugs and shortcomings tend to be fixed, the continuous introduction of new tool features brings new potential problems. For example, recent trends towards multi-threading bring into the picture a whole new class of *potential* implementation problems related to synchronization that do not exist in a single-threaded implementation.

In short, both investigators and the tools they use are prone to errors and this can lead to challenges (both legitimate and frivolous) of the results. Since the possibility of error will never go away and is inherently difficult to quantify, the only practical way to genuinely improve the trustworthiness of the process is to have an independent auditing facility that is: (a) automated, to guard against human lapses and (b) tool-independent, to independently confirm/challenge the results of any tool used, including ones not specifically labeled 'forensic'.

We should emphasize that such an independent auditing facility is not charged with duplicating the forensic functions of any tool but is an impartial observer of all basic data operations *actually* performed on the evidence. The essential result is a consistent, trusted, and tamper-evident audit trail that can be examined after the fact to confirm/debunk challenges.

Recently, Turner (2005) coined the rather descriptive term Digital Evidence Bag (DEB) to describe his proposed approach to dealing with the above problems. More precisely, his work can be described as an effort to create a specialized container – an open, common file format – for storing digital evidence. DEBs bundle digital evidence, associated metadata, and audit logs into a single structure, providing an audit trail of operations performed on the digital evidence in the bag as well as integrity checks. In the following section, we describe DEBs and other *digital evidence containers* (or *DECs*) in more detail.

All current DEC specifications address the auditing problem to some degree. However, they all rely on an individual tool's 'voluntary' participation, i.e., forensic applications need to use a specialized API, which effectively replaces the filesystem API. Obviously, reengineering all existing forensic applications for that specific purpose is a tall order and there are no compelling incentives at the moment. Therefore, our work is targeted at the development of operating system-level mechanisms to support/enforce the use of DECs. As we will demonstrate, this approach can achieve both automation and tool-independence without additional development efforts for existing tools. Overall, our approach can also provide stronger auditing guarantees. We view this as a complementary mechanism to development and standardization of DEC formats, which aids in evidence-handling procedures and provides transparent support for a variety of digital evidence container formats.