Digital Investigation 20 (2017) S107–S115

Contents lists available at ScienceDirect

Digital Investigation

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

DFRWS 2017 Europe — Proceedings of the Fourth Annual DFRWS Europe

# Characterizing loss of digital evidence due to abstraction layers

Felix Freiling [a, *], Thomas Glanzmann [b, **], Hans P. Reiser [c, ***]

[a] Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Martensstr. 3, 91058 Erlangen, Germany
[b] Rathsberger Str. 28, 91054 Erlangen, Germany
[c] Faculty of Computer Science and Mathematics, University of Passau, Innstraße 43, 94032 Passau, Germany

ARTICLE INFO

*Article history:*
Received 26 January 2017
Accepted 26 January 2017

*Keywords:*
Abstraction layer
Virtualization
Forensic analysis
File system
Virtual memory

ABSTRACT

We study the problem of evidence collection in environments where abstraction layers are used to organize data storage. Based on a formal model, the problem of evidence collection is defined as the task to reconstruct high-level from low-level storage. We investigate the conditions under which different levels of evidence collection can be performed and show that abstraction layers, in general, make it harder to acquire evidence. We illustrate our findings by describing several practical scenarios from file systems, memory management, and disk volume management.


## Introduction

Abstraction layers are a ubiquitous concept in modern computer systems. Like in computer networks (Tanenbaum, 2002) or file systems (Carrier, 2005), they usually come as a hierarchy where resources on a "higher" layer of abstraction are mapped to a set of resources on a "lower" layer of abstraction. Abstraction layers usually come with the benefit of isolation and more efficient resource usage stemming from separation of concerns.

When accessing upper layers, however, details of lower layers are "hidden". This observation is relevant to the process of evidence collection for forensic purposes: It is usually easier to establish the authenticity of digital evidence when it is accessed at lower layers of abstraction, but the items of interest are at higher levels of abstraction. Therefore, the items need to be reconstructed taking one abstraction layer at a time. But every abstraction layer introduces ambiguity and the possibility of interpretation errors (Casey, 2011). Even worse, sometimes it is impossible to access the lower layer because it is physically out of reach, such as in many cloud computing environments (Roussev et al., 2016).

### *Loss of digital evidence due to abstraction layers*

Given modern systems with complex hierarchies of abstraction layers, we ask the following question: Under which conditions is it possible to reconstruct higher levels of abstraction from data given at lower layers of abstraction? This question may appear trivial at first sight since all evidence that exists at the higher abstraction layer *must* also exist on the lower layer. However, from a digital forensics point of view we not only target actively used data but also deleted or otherwise unused data. This includes fragments of deleted files or the remains of process data structures in unused parts of physical memory. Not being able to interpret the data at the lower layer means that—at least from the viewpoint of the forensic expert—such data is "lost" through abstraction. Furthermore, there are many examples where abstraction layers combined with concurrency (as in the case of virtualization) may change data at the lower layer in ways that do not exist without abstraction. One example is resource pooling in cloud data centers: disk space that is not used by one virtual machine may be re-allocated to another virtual machine and overwritten, which may destroy evidence at higher layers. Our aim, therefore, is to better understand the different ways in which digital evidence is lost due to abstraction layers.

### *Related work*

The concept of abstraction underlies most techniques to build computer systems. Many early software and hardware engineering techniques are based on functional (or code) abstraction, as for example in structured programming with stepwise refinement. Later, data abstraction became a major paradigm for the construction of software systems, for example in object-oriented programming or other types of current formal modeling languages.

* Corresponding author.
** Corresponding author.
*** Corresponding author.
*E-mail addresses:* felix.freiling@cs.fau.de (F. Freiling), thomas@glanzmann.de (T. Glanzmann), hans.reiser@uni-passau.de (H.P. Reiser).

http://dx.doi.org/10.1016/j.diin.2017.01.012

Abstraction layers are not only a basic principle of constructing computer systems but also to reason about them. This is apparent in the area of (formal) verification where abstraction layers have been investigated in great depth. Formally, an abstraction layer relates to two system descriptions, one at a higher layer and one at a lower layer. Proving that the lower layer "implements" the higher layer, requires to map the state space (and therefore also the executions) of the lower layer to the higher layer. This can be done by constructing a *refinement mapping* (Abadi and Lamport, 1991) between the two layers.

The relations between "primitive" and "complex" event sequences were also influential to early attempts to formalize (digital) investigative techniques. For example, Carrier introduces the idea of constructing analysis tools along the abstraction levels of computer systems (Carrier, 2003). Later, Carrier and Spafford introduce *abstraction transformation functions* (Carrier and Spafford, 2006, p. S126) that allow the formulation high-level hypotheses and mapping them to low-level system operations. This is done to structure the area of investigative techniques and not to study the eventual loss of evidence through abstraction.

As early as 2010 it was observed that abstraction in form of virtualization poses problems to incident handling and forensics (Grobauer and Schreck, 2010). Similarly, Martini and Choo (2012) embed these issues into a process model framework for the forensic analysis of cloud computing platforms.

Birk and Wegener (2011) and later Kolb (2012) raised many detailed technical issues in the context of virtual machine analysis, especially the problem of correct resource attribution under resource pooling. Dykstra and Sherman (2012) investigated the problems of recovering deleted files or processes from memory in cloud computing environments. Zawoad and Hasan (2016a, 2016b) take a different approach and design an environment to aid in the forensic analysis of cloud systems.

Roussev and McCulley (2016) recently characterized the reason for problems in cloud forensic analysis, namely *cloud-native artifacts*. These artifacts encompass evidence that can only be acquired by physical access to the server, thereby pointing to a fundamental limitation of evidence collection in virtualized environments. While it is well-known that evidence collection on higher layers for multiple reasons is usually inferior to collection at lower layers, little research has been done to characterize the information loss that results from having abstraction layers at all.

### *Contributions*

We approach the research question using a formal model and derive conditions describing certain types of evidence collection for different conditions. The model covers systems that use abstraction layers to organize block-based storage. Based on our model, we derive conditions enabling the reconstruction of active and inactive data at lower layers. We can show, that these conditions are only partly satisfied by real data structures and that therefore data is "lost" through abstraction.

Our model formalizes page-based or block-based storage abstractions that may be fragmented or not. API-based storage abstractions like network file systems are excluded, because they hide state information. A further issue of the paper concerns reconstructing *state* instead of computation. The term *storage* is used to describe content of volatile physical memory as well as persistent physical disks alike, i.e., main memory and disks do not have to be distinguished. Our formalization covers a wide range of relevant data organization techniques like file systems, virtual memory, guest physical memory, logical volumes, and RAID systems.

### *Paper outline*

We present our model of abstraction layers in Section "Model", definitions of different evidence collection and reconstruction tasks in Section "Forensic reconstruction problems", the illustration of these classes using practical examples in Section "Abstraction layers in practice", and, finally, our conclusions in Section "Conclusions".

## Model

### *Layers*

We consider a single level of abstraction at a time, and for each level we distinguish two layers, the *upper* layer and the *lower* layer. Each layer provides computation and storage resources. However, the upper layer is implemented using resources from the lower layer.

### *Storage*

We consider a block-based storage abstraction (storage can be either memory or disk). Both layers consist of a finite ordered sequence of storage blocks, but blocks can be of different sizes and sequences can have different lengths in both layers. More precisely, on the lower layer, we model storage as a finite ordered sequence of storage blocks $l[0],l[1],\ldots,l[n]$, and on the upper layer we model storage similarly as a finite ordered sequence of storage blocks $u[0],u[1],\ldots,u[m]$. Each storage block is identified by a unique index, where $L$ indicates the range of indexes into $l$, i.e., the set $\{0,\ldots,n\}$, and $U$ the range of indexes into $u$, i.e., the set $\{0,\ldots,m\}$. Note that $m$ must not necessarily be equal to $n$, and that our model can be used to "simulate" multiple block sequences on both upper and lower layers.

### *Mappings*

Between upper and lower layers exists an entity that manages the lower storage so that upper storage is available at the upper layer. This entity uses a management data structure to map the set of upper storage blocks to lower storage blocks over time. In our model, this structure can map any subset of blocks in $u$ to any subset of blocks in $l$. Since the mapping is time dependent, we need to model its evolution over time. For simplicity, we use the set of natural numbers as time domain, i.e., $T = \{0,1,2,\ldots\}$. We then formally model the mapping at a certain time $t \in T$ as a relation $\varphi_t \subseteq U \times L$ that associates elements of $u$ to elements of $l$, i.e., $(x,y) \in \varphi_t$ iff $u[x]$ is mapped to $l[y]$ at a certain time $t$.

The definitions are illustrated in Fig. 1 where the assignment of blocks from $u$ to blocks from $l$ are depicted at a certain point in time $t$. Accordingly, at time $t$, upper layer block 1 ($u[1]$) is mapped to lower layer block 1 ($l[1]$). Furthermore, upper layer blocks $u[2]$ and $u[5]$ are both mapped to lower layer block $l[4]$, and $u[7]$ is mapped to $l[8]$ and $l[11]$.

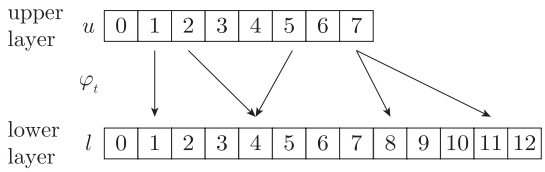


**Fig. 1.** Model of storage on different abstraction layers and mappings between layers at time $t$.