Digital Forensic Science

Issues, Methods, and Challenges

Synthesis Lectures on Information Security, Privacy, & Trust

Digital Forensic Science: Issues, Methods, and Challenges

Vassil Roussev

www.morganclaypool.com

ISBN: 9781627059596 paperback
ISBN: 9781627054652 ebook

DOI 10.2200/S00738ED1V01Y201610SPT019

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON INFORMATION SECURITY, PRIVACY, & TRUST

Lecture #19
Series Editors: Elisa Bertino, *Purdue University*
Ravi Sandhu, *University of Texas, San Antonio*
Series ISSN
Print 1945-9742 Electronic 1945-9750

Digital Forensic Science

Issues, Methods, and Challenges

Vassil Roussev
University of New Orleans

SYNTHESIS LECTURES ON INFORMATION SECURITY, PRIVACY, & TRUST #19

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

Digital forensic science, or *digital forensics*, is the application of scientific tools and methods to identify, collect, and analyze digital (data) artifacts in support of legal proceedings. From a more technical perspective, it is the process of reconstructing the relevant sequence of events that have led to the currently observable state of a target IT system or (digital) artifacts.

Over the last three decades, the importance of digital evidence has grown in lockstep with the fast societal adoption of information technology, which has resulted in the continuous accumulation of data at an exponential rate. Simultaneously, there has been a rapid growth in network connectivity and the complexity of IT systems, leading to more complex behavior that needs to be investigated.

The goal of this book is to provide a systematic *technical* overview of digital forensic techniques, primarily from the point of view of computer science. This allows us to put the field in the broader perspective of a host of related areas and gain better insight into the computational challenges facing forensics, as well as draw inspiration for addressing them. This is needed as some of the challenges faced by digital forensics, such as cloud computing, require *qualitatively* different approaches; the sheer volume of data to be examined also requires new means of processing it.

KEYWORDS

digital forensics, cyber forensics, cyber crime, incident response, data recovery

To my parents, *Reni* and *Rosen*,
for their love and for opening all of life's opportunities.

To my wife, *Laura*,
for her love and unconditional support.

To my advisor, *Prasun*,
for his patience and the enduring wisdom of his lessons.

Contents

CHAPTER 1
Introduction

In a word, the computer scientist is a *toolsmith*—no more, but no less. It is an honorable calling.

Frederick P. Brooks, Jr. [66]

Forensic science (or *forensics*) is dedicated to the systematic application of scientific methods to gather and analyze evidence for a legal purpose. *Digital forensics*—a.k.a. *cyber* or *computer* forensics—is a subfield within forensics, which deals specifically with digital artifacts, such as files, and computer systems and networks used to create, transform, transmit, and store them.

The rapid adoption of information technology (IT) in all aspects of modern life means that it bears witness to an ever expanding number of human- and machine-initiated interactions and transactions. It is increasingly the case that the *only* historical trace of such events exists in electronic form. At the same time, most IT systems are not *specifically* engineered to facilitate the forensic acquisition and analysis of their data. Therefore, there is the need to continuously develop forensic methods that keep up with the rapid growth in data volume and system complexity.

The main goal of this book is to provide a relatively brief, but systematic, *technical* overview of digital forensic methods, as they exist today, and to outline the main challenges that need to be addressed in the immediate future.

1.1 SCOPE OF THIS BOOK

By its nature, digital forensics is a multi-disciplinary undertaking, combining various expertise including software developers providing tools, investigators applying their analytical expertise, and lawyers framing the goals and bounds of the investigation. Nevertheless, the almost singular focus of this book is on the technical aspects of process—the algorithmic techniques used in the acquisition and analysis of the different systems and artifacts.

In other words, *the goal is to provide a computer science view of digital forensic methods*. This is in sync with Fred Brooks' thesis that the primary purpose of computer science research is to build computational tools to solve problems emanating from other domains: "Hitching our research to someone else's driving problems, and solving those problems on the owners' terms, leads us to richer computer science research." [66]

This means that we will only superficially touch upon the various legal concerns, or any of the issues regarding tool use, procedural training, and other important components of digital forensic practice. In part, this is due to the shortness of the book format, and the high quality coverage of these topics in existing literature.

However, the *primary* reason is that we seek to present digital forensics from a different perspective that has been missing. It is an effort to *systematize* the computational methods that we have acquired over the last three decades, and put them in a coherent and extensible framework.

Target audience. The treatment of the topics is based on the author's experience as a computer science educator and researcher. It is likely to fit better as part of a special topics course in a general computer science curriculum rather than as part of a specialized training toward certification, or digital forensics degree.

We expect this text to be most appropriate in an advanced, or a graduate, course in digital forensics; it could also be used as supplemental material in an introductory course, as some of the topic treatment is different from other textbooks. We hope that faculty and graduate students will find it helpful as a starting point in their research efforts, and as a good reference on a variety of topics.

Non-goals. It may be useful to point out explicitly what we are *not* trying to achieve. Broadly, we are not trying to replace any of the established texts. These come in two general categories:

- comprehensive introduction to the *profession* of the forensic investigator (often used as a primary textbook in introductory courses) such as Casey's *Digital Evidence and Computer Crime* [32];
- in-depth technical reference books on specialized topics of interest, such as Carrier's classic *File System Forensic Analysis* [23], *The Art of Memory Forensics* by Ligh et al. [108], or Carvey's go-to books on *Windows* [29] and registry analysis [30].

Due to the limitations of the series format, we have also chosen to forego a discussion on multimedia data and device forensics, which is a topic worth its own book, such as the one edited by Ho and Li [92].

1.2 ORGANIZATION

The book's structure is relatively flat with almost no dependencies among the chapters. The two exceptions are Chapter 3, which should be a prerequisite for any of the subsequent chapters, and Chapter 6, which will make most sense as the closing discussion.

Chapter 2 provides a brief history of digital forensics, with an emphasis on technology trends that have driven forensic development. The purpose is to supply a historical context for the methods and tools that have emerged, and to allow us to reason about current challenges, and near-term developments.

Chapter 3 looks at the digital forensics process from several different perspectives—legal, procedural, technical, and cognitive—in an effort to provide a full picture of the field. Later, these models are referenced to provide a framework to reason about a variety of challenges, from managing data volumes to improving the user interface of forensic tools.

Chapter 4 is focused on *system* forensics; that is, on the types of evidentiary artifacts that are produced during the normal operation of computer systems. Most of these are operating system and application data structures, but we also discuss the emerging problem of cloud system forensics.

Chapter 5 discusses *artifact* forensics: the analysis of autonomous data objects, usually files, that have self-contained representation and meaningful intepretation outside the scope of a specific computer system. These include text, images, audio, video, and a wide variety of composite document formats.

Chapter 6 is an effort to outline a medium-term research agenda that is emerging from the broader trends in IT, such as fast data growth, cloud computing, and IoT. The focus is on the difficult problems that need to be addressed in the next five years rather than on the specific engineering concerns of today, such as finding ways to get around encryption mechanisms.

CHAPTER 2
Brief History

The beginning of the modern era of digital forensics can be dated to the mid-1980s, which saw the adoption of 18 U.S.C. § 1030 [1] as part of the *Comprehensive Crime Control Act of 1984* [33]. The *Computer Fraud and Abuse Act* of 1986 was enacted by the U.S. Congress as the first of several amendements to clarify and expand the scope of the provisions. In 1984, the FBI initiatated its *Magnetic Media Program* [72], which can be viewed as a watershed moment in recognizing the importance of digital evidence, and the need for professionalization of the field.

Prior to that, computer professionals used ad-hoc methods and tools primarily for the purposes of data extraction and recovery after unforeseen failures and human errors; to this day, data recovery remains a cornerstone of digital forensic methodology. In the pre-1984 days, there was little effort to build a systematic body of knowledge, or specialized expertise. This is not surprising as there was little societal need—computers were centralized timeshare systems used by businesses, and had little useful information for the legal system. This all began to change with the massive surge in popularity of personal computers, and the introduction of dial-up networking for consumers.

2.1 EARLY YEARS (1984–1996)

The following dozen years (1984–1996) saw a rapid increase in personal computer use, along with fast growth in private network services like *CompuServe*, *Prodigy*, and *AOL*. This exploration period is characterized by substantial diversity of hardware and software, and saw the emergence of early *de facto* standard file formats (e.g., GIF [45]), most of which were poorly documented and rarely formally described [72].

Toward the end of the period, the meteoric rise in popularity of the *Netscape Navigator* web browser marked the tipping point for the transition to standards-based internetworking. At the same time, the combination of the *Intel x86* architecture and *Microsoft Windows* operating system became the dominant software on the PC desktop. Taken together, these developments rapidly reduced the platform diversity and enabled a coherent view of the digital forensic process to gradually emerge. It also became feasible to become an expert by focusing on just one platform that (at the time) had minimal security and privacy provisions to impede the analysis. For example, one of the major forensic vendors today, *AccessData*, advertised itself as "leaders in cryptography and password recovery since 1987," and offered a set of tools for that purpose [2].

In other words, the main uses of the forensic techniques of the time was to provide locksmith and disaster recovery services. Accordingly, the main thrust of the efforts was reverse-engineering/brute-forcing of the (weak) application-level encryption techniques employed by various vendors, and filesystem "undelete," which enabled the (partial) recovery of ostensibly deleted information.

2.2 GOLDEN AGE (1997–2007)

Around 1997, we saw the emergence of the first commercial tools, like *EnCase Forensic*, that specifically targeted law enforcement use cases and provided an integrated environment for managing a case. This marked the beginning of a decade (1997–2007) of rapid expansion of forensic capabilities, both commercial and open source, against a backdrop of growing use of the Internet for business transactions, and a relatively weak understanding and use of privacy and security mechanisms. Garfinkel refers to this period as a "Golden Age" of digital forensics [72]. During this time, we saw the establishment of the first academic conference—*DFRWS* (dfrws.org) in 2001—with the exclusive focus on basic and applied digital forensic research.

The most important source of forensic data during the period became local storage in the form of internal HDDs and removable media—CD/DVD and USB-attached flash/disk drives. This reflects an IT environment in which most computations were performed on workstations by standalone applications. Although the importance of the Internet was increasing dramatically, most of the related evidence could still be found in the local email client, or (web) browser caches. Thus, filesystem analysis and reconstruction (Section 4.1.4) became the main focus of forensic tools and investigations, and Carrier's definitive book on the subject [23] can be seen as a symbol of the age.

At the time, RAM was not considered a worthy source of evidence and there were no analytical tools beyond `grep` and `hexdump` to make sense of it. This began to change in 2005 with the first *DFRWS* memory forensics challenge [53], which led to the development of a number of tools for *Microsoft Windows* (discussed in Section 4.2); a follow-up challenge [76] focused research efforts on developing *Linux* memory analysis tools.

Between 2004 and 2007 several technology developments hinted that a new chapter in the history of the field was getting started. In 2004, *Google* announced the *Gmail* service [79]; its main significance is to show that a web application can be deployed on an Internet scale. Web apps are an implementation of the *software as a service* (SaaS) delivery model in which the client device needs no application-specific installation locally; most of the computation is performed on the provider's server infrastructure and only a small amount of user interface (UI) code is downloaded on the fly to manage the interaction with the user. Forensically, this is a big shift as most of the artifacts of interest are resident on the *server* side.

In 2006, Amazon announced its public cloud service [3], which greatly democratized access to large-scale computational resources. It suddenly became possible for *any* web app—not just the ones from companies with big IT infrastructure—to work at scale; there was no conceptual impediment for all software vendors to go the SaaS route. In practice, it took several years for this movement to become mainstream but, with the benefit of hindsight, it is easy to identify this as a critical moment in IT development.

In 2007, it was *Apple*'s turn to announce a major technology development—the first *smartphone* [6]; this was quickly followed by a Google-led effort to build a competing device using open source code, and the first *Android* device was announced in 2008 [183]. Mobile computing had been around for decades, but the smartphone combined a pocket-portable form factor with general purpose compute platform and ubiquitous network communication, to become—in less than a decade—the indispensible daily companion for the vast majority of people. Accordingly, it has become a witness of their actions, and a major source of forensic evidence.

2.3 PRESENT (2007–)

The current period is likely to be viewed as transitional. On the one hand, we have very mature techniques for analyzing persistent storage (Section 4.1) and main memory (Section 4.2) for all three of the main operating systems (OS) for the desktop/server environments—*Microsoft Windows*, *MacOS*, and *Linux*. Similarly, there are well-developed forensic tools for analyzing the two main mobile OS environments—*Android* and *iOS*.

On the other hand, we see exponential growth in the volume of forensic data in need of processing (Section 6.1) and the accelerating transition to cloud-centric IT (Section 4.6). As our discussion will show, the latter presents a qualitatively new target and requires a new set of tools to be developed. Separately, we are also seeing a maturing use of security and privacy techniques, such as encryption and media sanitization, that eliminate some traditional sources of evidence, and make access to others problematic.

It is difficult to predict what will be the event(s) that will mark the logical beginning of the next period, but one early candidate is the announcement of the *AWS Lambda* platform [9]. There is a broad consensus that the next major technology shift (over the next 10–15 years) will be the widespread adoption of the *Internet of Things* (IoT) [7]. It is expected that it will bring online between 10 and 100 times more Internet-connected devices of all kinds. The fast growing adoption of *AWS Lambda* as a means of working with these devices suggests that it could have a similar impact on the IT landscape to that of the original introduction of *AWS*.

Lambda provides a platform, in which customers write event-handling functions that require no explicit provisioning of resources. In a typical workflow, a device uploads a piece of data to a storage service, like *AWS S3*. This triggers an event, which is automatically dispatched to an instance of a user-defined handler; the result may be the generation of a series of subsequent events in a processing pipeline. From a forensic perspective, such an IT model renders existing techniques obsolete, as there is no meaningful data to be extracted from the embedded device itself.

2.4 SUMMARY

The main point of this brief walk through the history of digital forensics is to link the predominant forensic methods to the predominant IT environment. Almost all techniques in widespread use today are predicated on access to the full environment in which the relevant computations were performed. This started with standalone personal computers, which first became connected to the network, then became mobile, and eventually became portable and universally connected. Although each step introduced incremental challenges, the overall approach continued to work well.

However, IT is undergoing a rapid and dramatic shift from using software *products* to employing software *services*. Unlike prior developments, this one has *major* forensic implications; in simple terms, tools no longer have access to the full compute environment of the forensic target, which is a service hosted somewhere in a shared data center. Complicating things further is the fact that most computations are ephemeral (and do not leave the customary traces) and storage devices are routinely sanitized.

We will return to this discussion in several places throughout the text, especially in Chapter 6. For now, the main takeaway is that the future of forensics is likely to be different than its past and present. That being said, the bulk of the content will naturally focus on systematizing what we already know, but we will also point out the new challenges that may require completely new solutions.

CHAPTER 3

Definitions and Models

Forensic science is the application of scientific methods to collect, preserve, and analyze evidence related to legal cases. Historically, this involved the systematic analysis of (samples of) *physical* material in order to establish causal relationships among various events, as well as to address issues of provenance and authenticity.[1] The rationale behind it—*Locard's exchange principle*—is that physical contact between objects inevitably results in the exchange of matter leaving *traces* that can be analyzed to (partially) reconstruct the event.

With the introduction of digital computing and communication, the same general assumptions were taken to the cyber world, largely unchallenged. Although a detailed conceptual discussion is outside the intent of this text, we should note that the presence of *persistent* "digital traces" (broadly defined) is neither inevitable nor is it a "natural" consequence of the processing and communication of digital information. Such records of cyber interactions are the result of concious engineering decisions, ones not usually taken *specifically* for forensic purposes. This is a point we will return to shortly, as we work toward a definition that is more directly applicable to *digital* forensics.

3.1 THE DAUBERT STANDARD

Any discussion on forensic evidence must inevitably begin with the *Daubert* standard—a reference to three landmark decisions by the Supreme Court of the United States: *Daubert v. Merrell Dow Pharmaceuticals*, 509 U.S. 579 (1993); *General Electric Co. v. Joiner*, 522 U.S. 136 (1997); and *Kumho Tire Co. v. Carmichael*, 526 U.S. 137 (1999).

In the words of Goodstein [78]: "The presentation of scientific evidence in a court of law is a kind of shotgun marriage between the two disciplines. ... The *Daubert* decision is an attempt (not the first, of course) to regulate that encounter."

These cases set a new standard for expert testimony [11], overhauling the previous *Frye* standard of 1923 (*Frye v. United States*, 293 F. 1013, D.C. Cir. 1923). In brief, the Supreme Court instructed trial judges to become gatekeepers of expert testimony, and gave four basic criteria to evaluate the admissability of forensic evidence:

1. The theoretical underpinnings of the methods must yield testable predictions by means of which the theory could be falsified.

2. The methods should preferably be published in a peer-reviewed journal.

[1]A more detailed definition and discussion of traditional forensics is beyond our scope.

3. There should be a known rate of error that can be used in evaluating the results.

4. The methods should be generally accepted within the relevant scientific community.

The court also emphasized that these standards are flexible and that the trial judge has a lot of leeway in determining admissability of forensic evidence and expert witness testimony. During legal proceedings, special *Daubert* hearings are often held in which the judge rules on the admissibility of expert witness testimony requested by the two sides.

In other words, scientific evidence becomes *forensic* only if the court deems it admissible. It is a somewhat paradoxic situation that an evaluation of the scientific merits of a specific method is rendered by a judge, not scientists. There is no guarantee that the legal decision, especially in the short term, will be in agreement with the ultimate scientific consensus on the subject. The courts have a tendency to be conservative and skeptical with respect to new types of forensic evidence. The admissability decision also depends on the specific case, the skill of the lawyers on both sides, the communication skills of the expert witnesses, and a host of other factors that have nothing to do with scientific merit.

The focus of this book is on the scientific aspect of the analytical methods and, therefore, we develop a more technical definition of *digital forensic science*.

3.2 DIGIAL FORENSIC SCIENCE DEFINITIONS

Early applications of digital forensic science emerged out of law enforcement agencies, and were initiated by investigators with *some* technical background, but no formal training as computer scientists. Through the 1990s, with the introduction and mass adoption of the Internet, the amount of data and the complexity of the systems investigated grew quickly. In response, digital forensic methods developed in an ad hoc, on-demand fashion, with no overarching methodology, or peer-reviewed venues. By the late 1990s, coordinated efforts emerged to formally define and organize the discipline, and to spell out best field practices in search, seizure, storage, and processing of digital evidence [126].

3.2.1 LAW-CENTRIC DEFINITIONS

In 2001, the first *Digital Research Forensic Workshop* was organized with the recognition that the ad hoc approach to digital evidence needed to be replaced by a systematic, multi-disciplinary effort to firmly establish digital forensic science as a rigorous discipline. The workshop produced an in-depth report outlining a research agenda and provided one of the most frequently cited definitions of *digital forensic science* [136]:

Digital forensics: The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations.

This definition, although primarily stressing the investigation of criminal actions, also includes an anticipatory element, which is typical of the notion of forensics in operational environments. The analysis there is performed primarily to identify the vector of attack and scope of a security incident; identifying adversary with any level of certainty is rare, and prosecution is not the typical outcome.

In contrast, the reference definition provided by NIST a few years later [100] is focused entirely on the legal aspects of forensics, and emphasizes the importance of strict chain of custody:

Digital forensics is considered the application of science to the identification, collection, examination, and analysis of data while preserving the integrity of the information and maintaining a strict chain of custody for the data. Data refers to distinct pieces of digital information that have been formatted in a specific way.

Another way to describe these law-centric definitions is that they provide a litmus test for determining whether specific investigative tools and techniques qualify as being *forensic*. From a legal perspective, this open-ended definition is normal and works well as the admissability of all evidence gets decided during the legal proceedings.

From the point of view of a technical discussion, however, such definitions are too generic to provide a meaningful starting point. Further, the chain of custody issues are primarily of procedural nature and do not bring up any notable technical problems. Since the goal of this book is to consider the technical aspects of digital forensics, it would be prudent to start with a working definition that is more directly related to our subject.

3.2.2 WORKING TECHNICAL DEFINITION

We adopt the working definition first introduced in [154], which directly relates to the formal definition of computing in terms of *Turing* machines, and is in the spirit of Carrier's computer history model (Section 3.3.2):

Digital forensics is the process of reconstructing the relevant sequence of events that have led to the currently observable state of a target IT system or (digital) artifacts.

Notes

1. The notion of *relevance* is inherently case-specific, and a big part of a forensic analyst's expertise is the ability to identify case-relevant evidence.

2. Frequently, a critical component of the forensic analysis is the causal attribution of event sequence to specific human actors of the system (such as users and administrators).

3. The provenance, reliability, and integrity of the data used as evidence are of primary importance.

We view all efforts to perform system, or artifact, analysis after the fact as a form of forensics. This includes common activities, such as incident response and internal investigations, which almost never result in any legal actions. On balance, only a tiny fraction of forensic analyses make it to the courtroom as formal evidence; this should not constrain us from exploring the full spectrum of techniques for reconstructing the past of digital artifacts.

The benefit of employing a broader view of forensic computing is that it helps us to identify closely related tools and methods that can be adapted and incorporated into forensics.

3.3 MODELS OF FORENSIC ANALYSIS

In this section we discuss three models of the forensic analysis; each considers a different aspect of the analysis and uses different methods to describe the process. Garfinkel's *differential analysis* (Section 3.3.1) approach formalizes a common logical inference technique (similar, for example, to differential diagnosis in medicine) for the case of computer systems. In this context, diffential analysis is an incremental technique to reason about the likely prior state and/or subsequent events of individual artifacts (e.g., a file has been copied).

Carrier's *computer history model* (Section 3.3.2) takes a deeper mathematical approach in describing forensics by viewing the computer system under investigation as a *finite state machine*. Although it has few *direct* practical implications, it is a conceptually important model for the field. Some background in formal mathematical reasoning is needed to fully appreciate its contribution.

The final model of Pirolli and Card (Section 3.3.3) does not come from the digital forensics literature, but from cognitive studies performed on intelligence analysts. It is included because we believe that the analytical process is *very* similar and requires the same type of skills. Understanding how analysts perform the cognitive tasks is of critical importance to designing usable tools for the practice. It also helps in understanding and modeling the differences in the level of abstraction at which the three groups of experts—forensic researchers/developers, analysts, and lawyers—operate.

3.3.1 DIFFERENTIAL ANALYSIS

The vast majority of existing forensic techniques can be described as special cases of *differential analysis*—the comparison of two objects, A and B, in order to identify the differences between them. The ultimate goal is to infer the sequence of events that (likely) have transformed A into B (A preceeds B in time). In the context of *digital* forensics, this fundamental concept has only recently been formalized by Garfinkel et al. [75], and the rest of this section introduces the formal framework they put forward.

Terminology

Historically, differencing tools (such as the venerable *diff*) have been applied to a wide variety of artifacts, especially text and program code, long before they were employed for forensic use. The following definitions are introduced to formally generalize the process.

- *Image*. A byte stream from any data-carrying device representing the object under analysis. This includes all common evidence sources—disk/filesystem images, memory images, network captures, etc.

  Images can be *physical*, or *logical*. The former reflect (at least partially) the physical layout of the data on the data store. The latter consists of a collection of self-contained objects (such as files) along with the logical relationships among them without any reference to their physical storage layout.

- *Baseline image, A*. The image first acquired at time T_A.

- *Final image, B*. The last acquired image, taken at time T_B.

- *Intermediary images, I_n*. Zero, or more, images recorded between the baseline and final images; I_n is the n^{th} image acquired.

- *Common baseline* is a single image that is a common ancestor to multiple final images.

- *Image delta, $B - A$*, is the differences between two images, typically between the baseline image and the final image.

- The *differencing strategy* defines the rules for identifying and reporting the differences between two, or more, images.

- *Feature*, f, is a piece of data that is either directly extracted from the image (file name/size), or is computed from the content (crypto hash).

- *Feature in image, (A, f)*. Features are found in images; in this case, feature f is found in image A.

- *Feature name,* NAME (A, f). Every feature may have zero, one, or multiple names. For example, for a *file content* feature, we could use any of the file names and aliases under which it may be known in the host filesystem.

- *Feature location*, Loc(f), describes the address ranges from which the content of the particular feature can be extracted. The locations may be either physical, or logical, depending on the type of image acquired.

- A *feature extraction function*, $F()$, performs the extraction/computation of a feature based on its location and content.

- *Feature set,* $F(A)$, consists of the features extracted from an image A, using the extraction function $F()$.

- The *feature set delta*, $F(B) - F(A)$, contains the differences between the feature sets extracted from two images; the delta is not necessarily symmetric.

- *Transformation sequence*, R, consists of the sequence of operations that, when applied to A, produce B. For example, the Unix *diff* program can generate a *patch file* that can be used to transform a text file in this fashion. In general, R is not unique and there can be an infinite number of transformations that can turn A into B.

Generalized Differential Analysis

As per [75], each feature has three pieces of metadata:

Location: A mandatory attribute describing the address of the feature; each feature must have at least one location associated with it. *Name*: A human-readable identifier for the feature; this is an optional attribute. *Timestamp(s) and other metadata*: Features may have one, or more, timestamps associated with them, such as times of creation, modification, last access, etc. In many cases, other pieces of metadata (key-value pairs) are also present.

Given this framework, differential analysis is performed *not* on the data images A and B, but on their corresponding feature sets, $F(A)$ and $F(B)$. The goal is to identify the operations which transform $F(A)$ into $F(B)$. These are termed *change primitives*, and seek to explain/reproduce the feature set changes.

In the general case, such changes are not unique as the observation points may fail to reflect the effects of individual operations which are subsequently overridden (e.g., any access to a file will override the value of the last access time attribute). A simple set of change inference rules is defined (Table 3.1) and formalized (Table 3.2) in order to bring consistency to the process. The rules are *correct* in that they transform $F(A)$ into $F(B)$ but do not necessarily describe the actual operations that took place. This is a fundamental handicap for any differential method; however, in the absence of complete operational history, it is the best that can be accomplished.

If A and B are from the same system and $T_A < T_B$, it would appear that all *new* features in the feature set delta $F(B) - F(A)$ should be timestamped after T_A. In other words, if B were to contain features that predate T_A, or postdate T_B, then this would rightfully be considered an inconsistecy. An investigation should detect such anomalies and provide a sound explanation

Table 3.1: Change detection rules in plain English ([75], Table 1)

If something did not exist and now it does, it was created
If it is in a new location, it was moved
If it did exist before and now it does not, it was deleted
If more copies of it exist, it was copied
If fewer copies of it exist, something got deleted
Aliasing means names can be added or deleted

Table 3.2: Abstract rules for transforming $A \to B$ (A into B) based on observed changes to features, f, feature locations Loc (A, f), and feature names Name (A, f). Note: The Rename primitive is not strictly needed (as it can be modeled as AddName followed by DelName), but it is useful to convey higher-level semantics ([75], Table 2).

Rule	Change Primitive for $A \underset{R}{\to} B$
$f \in F(A) \wedge f \in F(B)$	(No change)
$f \notin F(A) \wedge f \in F(B)$	Create f
$f \in F(A) \wedge f \notin F(B)$	Delete f
$\lvert \textsc{Loc}(A,f) \rvert = 1 \wedge \lvert \textsc{Loc}(A,f) \rvert = 1 \wedge \textsc{Loc}(A,f) \neq \textsc{Loc}(B,f)$	Move $\textsc{Loc}(A,f) \to \textsc{Loc}(B,f)$
$\lvert \textsc{Loc}(A,f) \rvert < \lvert \textsc{Loc}(B,f) \rvert$	Copy $\textsc{Loc}(A,f) \to (\textsc{Loc}(B,f) \mid \textsc{Loc}(A,f))$
$\lvert \textsc{Loc}(A,f) \rvert > \lvert \textsc{Loc}(B,f) \rvert$	Delete $(\textsc{Loc}(A,f) \mid \textsc{Loc}(B,f))$
$\lvert \textsc{Name}(A,f) \rvert = 1 \wedge \lvert \textsc{Name}(B,f) \rvert = 1 \wedge \textsc{Name}(A,f) \neq \textsc{Name}(B,f)$	Rename $\textsc{Name}(A,f) \to \textsc{Loc}(B,f)$
$(\lvert \textsc{Name}(A,f) \rvert \neq 1 \vee \lvert \textsc{Name}(B,f) \rvert \neq 1) \vee (n \notin \textsc{Name}(A,f) \wedge n \in \textsc{Name}(B,f))$	AddName f, n
$(\lvert \textsc{Name}(A,f) \rvert \neq 1 \vee \lvert \textsc{Name}(B,f) \rvert \neq 1) \vee (n \in \textsc{Name}(A,f) \wedge n \notin \textsc{Name}(B,f))$	DelName f, n

based on knowledge of how the target system operates. There is a range of possible explanations, such as:

Tampering. This is the easiest and most obvious explanation although it is not necessarily the most likely one; common examples include planting of new files with old timestamps, and system clock manipulation.

System operation. The *full* effects of the underlying operation, even as simple as copying a file, are not always obvious and require careful consideration. For example, the Unix *cp* command sets the creation time of the new copy to the time of the operation but will keep the original modification time if the *-p* option is used.

Time tracking errors. It has been shown [127, 167] that operating systems can introduce inconsitencies during normal operation due to rounding and implementation errors. It is worth noting that, in many cases, the accuracy of a recorded timestamp is of little importance to the operation of the system; therefore, perfection should not be assumed blindly.

Tool error is always a possibility; like all software, forensic tools have bugs and these can manifest themselves in unexpected ways.

One important practical concern is how to *report* the extracted feature changes. Performing a comprehensive differential analysis, for example, of two hard disk snapshots is likely to result in an enormous number of individual results that can overwhelm the investigator. It is critical that differencing tools provide the means to improve the quality and relevance of the results. This can be accomplished in a number of ways: (a) *filtering* of irrelevant information; (b) *aggregation* of the results to highlight both the common and the exceptional cases; (c) *progressive disclosure* of information, where users can start at the aggregate level and use queries and hierchies to drill down to the needed level of detail; (d) *timelining*—provide a chronological ordering of the (relevant) events.

3.3.2 COMPUTER HISTORY MODEL

Differential analysis offers a relatively simple view of forensic inference, by focusing on the beginning and end state of the data, and by expressing the difference in terms of a very small set of primitive operations. The *computer history model* (CHM) [27]—one of Carrier's important contributions to the field—seeks to offer a more detailed and formal description of the process. The model employs finite state machines to capture the state of the system, as well as its (algorithmic) reaction to outside events. One of the main considerations is the development of an investigative model that avoids human bias by focusing on modeling the computation itself along with strict scientific hypothesis testing. The investigation is defined as a series of yes/no questions (predicates) that are evaluated with respect to the available history of the computation.

Primitive Computer History Model

This assumes that the computer being investigated can be represented as a *finite state machine* (FSM), which transitions from one state to another in reaction to events. Formally, the FSM is a quintuple $(Q, \Sigma, \delta, s0, F)$, where Q is a finite set of states, Σ is a finite alphabet of event symbols, δ is the transition function $\delta : Q \times \Sigma \rightarrow Q$, $s0 \in Q$ is the starting state of the machine, and $F \subseteq Q$ is the set of final states.

The *primitive history* of a system describes the lowest-level state transitions (such as the execution of individual instructions), and consists of the sequence of primitive states and events that occurred.

The *primitive state* of a system is defined by the discrete values of its primitive, uniquely addressable storage locations. These may include anything from a CPU register to the content of network traffic (which is treated as temporary storage). As an illustration, Figure 3.1 shows an event E_1 reading the values from storage locations R_3 and R_6 and writing to locations R_3 and R_4.

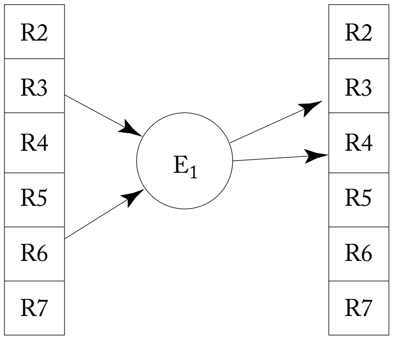

Figure 3.1: Primitive computer history model example: event E_1 is reading the values from storage locations R_3 and R_6 and writing to locations R_3 and R_4 [27].

The *primitive history* is the set T containing the times for which the system has a history. The duration between each time in T, Δt, must be shorter than the fastest state change in the system. The *primitive state history* is function $h_{ps} : T \to Q$ that maps a time $t \in T$ to the primitive state that existed at that time. The *primitive event history* is a function $h_{pe} : T \to \Sigma$ that maps a time $t \in T$ to a primitive event in the period $(t - \Delta t, t + \Delta t)$.

The model described so far is capable of describing a static computer system; in practice, this is insufficient as a modern computing system is dynamic—it can add resources (such as storage) and capabilities (code) on the fly. Therefore, the computer history model uses a dynamic FSM model with sets and functions to represent the changing system capabilities. Formally, each of the Q, Σ, and δ sets and functions can change for each $t \in T$.

Complex Computer History Model

The primitive model presented is rarely practical on contemporary computer systems executing billions of instructions per second (code reverse engineering would be an exceptional case). Also, there is a mismatch between the level of abstraction of the representation and that of the questions that an investigator would want to ask (e.g., *was this file downloaded?*). Therefore, the model provides the means to aggregate the state of the system and ask questions at the appropriate level of abstraction.

Complex events are state transitions that cause one or more lower-level complex or primitive events to occur; for example, copying a file triggers a large number of primitive events. *Complex storage locations* are virtual storage locations created by software; these are the ephemeral and persistent data structures used by software during normal execution. For example, a file is a complex storage location and the name value attribute pairs include the file name, several different timestamps, permissions, and content.

Figure 3.2 shows a complex event E_1 reading from complex storage locations D_1 and D_2 and writing a value to D_1. At a lower level, E_1 is performed using events E_{1a} and E_{1b}, such as CPU, or I/O instructions. The contents of D_1 and D_2 are stored in locations (D_{1a}, D_{1b}) and (D_{2a}, D_{2b}), respectively.

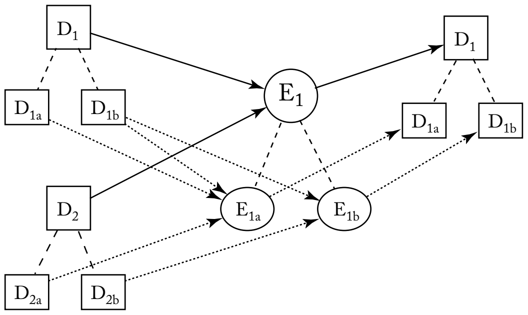

Figure 3.2: Complex history event examples: event E_1 with two complex cause locations and one complex effect location [27].

General Investigation Process

The sequence of queries pursued by the investigator will depend on the specific objectives of the inquiry, as well as the experience and training of the person performing it. The CHM is agnostic with respect to the overall process followed (we will discuss the cognitive perspective in Section 3.3.3) and does not assume a specific sequence of high-level phases. It does, however, postulate that the inquiry follow the general scientific method, which typically consists of four phases: *Observation*, *Hypothesis Formulation*, *Prediction*, and *Testing & Searching*.

Observation includes the running of appropriate tools to capture and observe aspects of the state of the system that are of interest, such as listing of files/processes, and rendering the content of files. During *Hypothesis Formulation* the investigators use the observed data, and combine it with their domain knowledge to formulate hypothesis that can be tested, and potentially falsified, in the history model. In the *Prediction* phase, the analyst identifies specific evidence that would be consistent, or would be in contradiction, with the hypothesis. Based on the predictions, experiments are performed in the *Testing* phase, and the outcomes are used to guide further iterations of the process.

Categories of Forensic Analysis

Based on the outlined framework, the CHM identifies seven categories of analytical techniques.

History duration. The sole techniques in this category and is *operational reconstruction*—it uses event reconstruction and temporal data from the storage devices to determine when events occurred and at what points in time the system was active. Primary sources for this analysis include log files, as well as the variety of timestamp attributes kept by the operating system and applications.

Primitive storage system configuration. The techniques in this category define the capabilities of the primitive storage system. These include the names of the storage devices, the number of addresses for each storage device, the domain of each address on each storage device, and when each storage device was connected. Together, these sets and functions define the set of possible states Q of the FSM.

Primitive event system configuration. Methods in this category define the capabilities of the primitive event system; that is, define the names of the event devices connected, the event symbols for each event device, the state change function for each event device, and when each event device was connected. Together, these sets and functions define the set of event symbols Σ and state change function δ. Since primitive events are almost never of direct interest to an investigation, these techniques are not generally performed.

Primitive state and event definition. Methods in this category define the primitive state history (h_{ps}) and event history (h_{es}) functions. There are five types of techniques that can be used to formulate and test this type of hypothesis and each class has a directional component. Since different approaches can be used to defining the *same* two functions, a hypothesis can be formulated using one technique and tested with another. Overall, these are impractical in real investigations, but are presented below for completeness.

Observation methods use direct observation of an output device to define its state in the inferred history, and are only applicable to output device controllers; they cannot work for internal devices, such as hard disks.

Capabilities techniques employ the primitive system capabilities to formulate and test state and event hypotheses. To formulate a hypothesis, the investigator chooses a possible state or event at random; this is impractical for almost all real systems as the state space is enormous.

Sample data techniques extract samples from observations of similar systems or from previous executions of the system being investigated; the results are metrics on the occurrence of events and states. To build a hypothesis, states and events are chosen based on how likely they are to occur. Testing the hypothesis reveals if there is evidence to support the state or event. Note that this is a conceptual class not used in practice as there are no relevant sample data.

Reconstruction techniques use a known state to formulate and test hypotheses about the event and state that existed immediately prior to the known state. This is not performed in practice, as questions are rarely formulated about primitive events.

Construction methods are the forward-looking techniques that use a known state to formulate and test hypotheses about the next event and state. This is not useful in practice as the typical starting point is an end state; further, any hypothesis about the future state would not be testable.

Complex storage system configuration. Techniques in this category define the complex storage capabilities of the system, and are needed to formulate and test hypotheses about complex states. The techniques define the names of the complex storage types (D_{cs}), the attribute names for each complex storage type (DAT_{cs}), the domain of each attribute (ADO_{cs}), the set of identifiers for the possible instances of each complex storage type (DAD_{cs}), the abstraction transformation functions for each complex storage type (ABS_{cs}), the materialization transformation functions for each complex storage type (MAT_{cs}), and the complex storage types that existed at each time and at each abstraction layer $X \in L$ (c_{cs-X}).

Two types of hypotheses are formulated in this category: the first one defines the names of the complex storage types and the states at which they existed; the second defines the attributes, domains, and transformation functions for each complex storage type. As discussed earlier, complex storage locations are program data structures. Consequently, to enumerate the complex storage types in existance at a particular point in time requires the reconstruction of the state of the computer, so that program state could be analyzed.

Identification of existing programs can be accomplished in one of two ways: *program identification*—by searching for programs on the system to be subsequently analyzed; and *data type observation*—by inferring the presence of complex storage types that existed based on the data types that are found. This latter technique may give false positives in that a complex type may have been created elsewhere and transferred to the system under investigation.

Three classes of techniques can be used to define the attributes, domains, and transformation functions for each complex storage type: (a) *complex storage specification observation*, which uses a specification to define a program's complex storage types; (b) *complex storage reverse engineering*, which uses design recovery reverse engineering to define complex storage locations; (c) *complex storage program analysis*, which uses static, or dynamic, code analysis of the programs to identify the instructions creating, or accessing, the complex storage locations and to infer their structure.

It is both impractical and unnecessary to fully enumerate the data structures used by programs; only a set of the most relevant and most frequently used ones are supported by investigative tools, and the identification process is part of the tool development process.

Complex event system configuration. These methods define the capabilities of the complex event system: the names of the *programs* that existed on the system (D_{ce}), the names of the *abstraction layers* (L), the symbols for the *complex events* in each program (DSY_{ce-X}), the *state change*