ANALYSIS OF TIME ACTIVITY DATA CHARACTERISTICS AND DATA DEGRADATION IN DIGITAL FORENSICS

by

Andrew C. Smallman
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

___________________________________	Dr. James Jones, Dissertation Director
___________________________________	Dr. Gheorghe Tecuci, Committee Member
___________________________________	Dr. Daniel Carr, Committee Member
___________________________________	Dr. Duminda Wijesekera, Committee Member
___________________________________	Mr. Robert Osgood, Committee Member
___________________________________	Dr. Deborah Goodings, Associate Dean
___________________________________	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date:______________________________	Summer Semester 2020 George Mason University Fairfax, VA

Analysis of Time Activity Data Characteristics and Data Degradation in Digital Forensics

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Andrew C. Smallman
Master of Science
Capitol College, 2005
Dual Bachelors of Science
University of Redlands, 1998

Director: Dr. James Jones, Associate Professor
Electrical and Computer Engineering

Summer Semester 2020
George Mason University
Fairfax, VA

Dedication

To Dar. Thank you for everything.

Table of Contents

List of Tables

List of Figures

List of Equations

Abstract

ANALYSIS OF TIME ACTIVITY DATA CHARACTERISTICS AND DATA DEGRADATION IN DIGITAL FORENSICS

Andrew C. Smallman, Ph.D.

George Mason University, 2020

Dissertation Director: Dr. James Jones

Activity analysis is an increasingly common task in complex investigative digital forensics examinations. This analysis relies on extracting data from a system and projecting backwards to identify and explain events that took place in the past. There have historically been two approaches: either examiners look at each log individually, or all individual records are extracted from all available sources and combined into a massive database for analysis. Either method ignores potentially relevant information about the context of the individual records as well as the characteristics of their sources. It is also challenging to identify if any records were once present but are now missing due to either intentional obfuscation or simply routine system operation and interactions.

This work presents a taxonomy for describing time-activity data (TAD) and TAD source characteristics and describes an inferential analysis strategy based on the characteristics of TAD sources. This enables examiners to identify and describe the characteristics for different sources and how they may enhance or complicate activity analysis conclusions. This work also presents a state-based approach to activity analysis. This model for system state changes over time in response to user actions provides a method for the analysis of TAD records from successive disk images. This method was then applied to a series of images from the M57-Patents dataset to analyze the degradation of TAD record data over time from a series of linked images from the same system. The data was analyzed to see if the degradation varies by record source or type and to look for variation across three separate systems.

Chapter One – Overview and Goals

All information, regardless of the medium, is fragile. Information transmitted by spoken word persists only until the sound falls below the background noise and becomes unintelligible. Words written on paper seem permanent, but on a long enough timeline the ink will break down and the paper itself may degrades due to the environment, the activities of other people, or simply the passage of time. Even engravings in stone tablets eventually succumb to the passage of time.

At first, the degradation of the information medium does not interfere with our ability to retrieve the information stored on the paper. As more time passes, some portions may be harder to read and some content inferred, but most of the original information can still be extracted. Specialized tools and techniques can potentially recover more information than is visible to the unaided eye.

Left to its own devices the information on the paper will eventually be unreadable due to natural degradation. It can also occur much sooner as a result of human actions such as shredding, burning, or simply writing new information on top of the old information.

Data stored on computers is affected by analogous processes. Once written to a hard drive, data will persist until the media breaks down, is damaged, or until it is partially or completely overwritten by new data. At a high level, the information on a computer consists of the files the user places on the storage, plus the supporting information that the operating system and other programs create and modify as they perform actions as directed by the user.

Extracting and analyzing the time data from a computer system under examination can be essential in understanding the pattern of life for both the system and its users. This analysis of activities can give essential context to the content that is found on the system. One particular challenge in understanding the full history of a system is that data stored on the system is constantly changing, overwriting new files, changing attributes, deleting old files, and so on. If the critical actions took place days or months ago then some of the records are likely to be missing or otherwise changed. Different time data sources have different characteristics which may allow you to infer if particular records had ever existed on the system, confirming or invalidating your analytic hypothesis.

Certain records are more likely than others to have been overwritten in the intervening time. The fundamental question the case study attempts to answer is "how much of evidence of those past actions will still persist after particular periods of time?"

The overarching goal for this work is to improve activity analysis in digital forensics by formalizing existing ad-hoc concepts and building a state-change user activity model. I will then validate these results by analyzing a realistic forensic dataset. After building a foundation through a better understanding of the characteristics and logical structures of time data sources, this lifespan analysis should help to inform expectations for the survival of records of a particular type on a representative system.

Written Organization

Chapter two provides a brief overview of the field of digital forensics, and specifically the area of Investigative Digital Forensics (IDF), plus a more detailed description of timeline creation and activity analysis. I also survey some of the related and foundational work in the field as it relates to my work in later chapters.

Chapter three discusses general characteristics of metadata and looks at a particular type of metadata in great detail, the Time Activity Datum (TAD). I define and describe TADs and enumerate their characteristics.

Chapter four enumerates the details and characteristics of collections of TADs. Taken together, chapters three and four establish a functional taxonomy for describing time data records and sources in digital forensics.

Computers operate at the direction of their users. Chapter five details a conceptual model for system states and their decomposition into corresponding physical and data states. Chapter six expands this with user interactions into a system activity model. Each user action causes changes to the physical characteristics of the computer, which are translated by the operating system and applications into a cascade of actions and changes to the computer components and storage.

Chapter seven builds upon the previous work and describes a logical method for performing a differential analysis on data extracted from a series of images describing a single system. Past efforts have looked primarily at the sector or file level, so I extend these methods to analysis of all the TAD records extracted from a series of successive system states.

Chapter eight applies this method to a series of images from the Digital Corpora M57-Patents scenario. The data processing and extraction methodology is described, as well as the process to calculate the lifespan for each record contained within each source image. In chapter nine the data is dissected and analyzed in order to characterize the degradation of TAD records related to user activities over the lifetime of the scenario, and to see if any generalizations can be made regarding general and specific sources and actions.

Chapter ten discusses the results of this work and suggests areas for future research.

Chapter Two – Background and related work

Although this work is not intended to be a survey of the field of digital forensics, a brief overview is helpful before getting too deep into my specific research area. Digital forensics applies a scientific process to the analysis of digital items. It is also commonly referred to as computer forensics, but I will use the term digital forensics as the techniques can be applied to individual files, storage devices, phones, and many other non-computer items as well. Jones et al. (2016) describes it concisely: “The practice of digital forensics is the art and science of inferring and proving past activity given some set of residual digital artifacts and traces.”

The concepts described are generally applicable regardless of the particular operating systems and forensic tools used, and should remain so as technology progresses. Only the case study in chapter nine is particular to the underlying tool and operating system, although the findings should continue to be informative with newer operating systems and applications as long as there is not a paradigm shift in how records are managed by consumer operating systems.

First a note regarding verbiage. The descriptors “subject” and “suspect” are often used interchangeably in forensic writing. I will use “subject” in my own writing, but some quotes and references will use “suspect”. Subject in this context is “the subject of the investigation or analysis” whether the subject is a person or a computer.

I will use the term *investigator* as a general term to refer to a member of the investigative team. It could refer to a criminal investigator, technician, analyst, forensic examiner, linguist, or any necessary specialty. Essentially *investigator* is shorthand for "a member of the investigative team with the right training, knowledge, and experience to make an appropriate decision". Investigative digital forensics requires personnel with a range of knowledge and skills, and one person cannot reasonably be expected to be an expert in all necessary aspects of technology, law, investigations, and content analysis.

Categories of Digital Forensics Work

There are three broad but relatively distinct types of digital forensics work:

- Incident response and intrusion analysis
- Device and application analysis and characterization
- Investigative digital forensics

There is not a formal separation between these categories, and some matters may have components from two or even all three types. These categories function as a logical separation based on the particular approach taken by the practitioner in each and the standard approach to the work.

Incident response is what most closely matches the Hollywood image of hackers intruding into a corporate or government computer network. On the inside, specialists identify and mitigate an intrusion by collecting and analyzing logs and other data from network devices and compromised systems. An intrusion is tracked from the initial foothold through multiple systems and devices. Incident response has an inherent tension between the desire to mitigate the intrusion and get the network functioning properly and securely while also collecting and analyzing the information to understand the attack method and identify a specific attacker. Malware analysis is often performed as a part of intrusion analysis, but it can also be performed as a separate function.

Device and application analysis and characterization is commonly performed to understand the capabilities and implementation of a device such as a cell phone, embedded system, or software application. It may also address how to extract data from a device in a forensically sound manner, as well as to characterize the data that is stored and can be recovered for analysis. These techniques are almost always performed on an exemplar or artificially created system.

Investigative Digital Forensics

In investigative digital forensics (IDF) the computer or other device is believed to contain data (evidentiary or otherwise) about someone or something that is related to an investigation. The computer itself could have been used to commit a crime, or it could contain evidence related to a crime. In the first instance the computer could have been used to send a threatening email or launch an attack on a government system. In the second instance a business committing tax fraud could be keeping two sets of books, both stored separately in accounting software on the computer. In investigative digital forensics (IDF) the computer or other device is believed to contain data (evidentiary or otherwise) about someone or something that is related to an investigation. The computer itself could have been used to commit a crime, or it could contain evidence related to a crime. IDF is inherently an interdisciplinary endeavor, combining computer science, data analytics, psychology, and many other related fields and methods (Woods et al., 2011).

Even the most technical analysis needs to be able to be explained from the standpoint of "what does it mean and why is important" to a reviewer or some other decision maker, whether a jury, investigator, intelligence analyst, policymaker, etc. This work attempts to avoid most technical jargon and specific low-level technical implementations of the items discussed here. Most of the concepts can be applied across a wide range of systems and technical implementations, and it is a task left up to the particular investigator or analyst to determine exactly how these concepts map onto their particular technical problem set, implementation, and investigative needs.

Historically most research effort related to IDF has been focused on evidence preservation, extraction, and presentation (Garfinkel, Farrella, Roussev & Dinolt, 2009). Extensive work has been published on deleted item recovery, file carving, decryption, and other techniques that are necessary to collect, extract, and present the data for content review. Some effort has been made towards developing more efficient triage (Roussev & Quates, 2012) as well as statistical sampling techniques and feature extraction (Garfinkel, 2010), but efforts to make the content review process more efficient and effective remain a fertile ground for research. In much of the published forensic research, there seems to be a preference to attempt to answer a particular technical question or build a process to perform a particular task. Chapters three through seven build a conceptual framework for analysis before performing a case study of particular data in chapters eight and nine.

Phases of Digital Forensics Work

The digital forensics lifecycle can be broken down into three broad phases:

1. Response and collection

2. Extraction and transformation
3. Analysis, including technical analysis and content analysis

Response and Collection Phase

In the response and collection phase, personnel must identify items that may contain relevant digital information and preserve it in a manner that will enable later analysis. This phase is not always performed by the expert forensic examiner, rather in many situations it is performed by system administrators, technicians, law enforcement first responders, and the like. The term of art is to collect the data in "forensically sound" manner, which can be colloquially described as acquiring the data in such a way as to not prevent or interfere with any downstream forensics processing and analysis. Ideally this acquires a bit-by-bit copy of the original data, but is impossible or impracticable in many circumstances. Many devices and data sources require acquisition techniques that make some changes to the underlying system in the process of extracting the data. As long as this acquisition is well documented and performed using tools that are well understood, tested, and verified, then provenance of the data will be known.

One example of unavoidable changes to a subject system caused by the acquisition process occurs when system memory is acquired from a running system. Setting aside the fact that even an idle system will have background processes and the like that are making changes to the memory, storage, and system state, the act of collecting the data from the memory will also cause changes to the system. At a minimum, the responder must make their tools available to the subject system, either by connecting a storage device or accessing a remote resource over the network. Then a destination is required to store the extracted data, again either by connecting additional storage or accessing networked storage. Then the acquisition program must be executed which will place it in memory, overwriting what was previously there, as well as leaving a record within the operating system housekeeping metadata.

If the responder is using well established and understood processes and documents their actions (Carvey, 2009, pp. 99), these changes will be able to be identified and excluded as necessary during the extraction and analysis phases. Any data that was overwritten by these changes will be lost, so the goal is to use the lightest possible "touch" when acquiring data from a live system.

Extraction and Transformation Phase

Extraction and transformation occur when information is obtained from the collected forensic data and, processed, categorized, indexed, and otherwise made intelligible and understandable for analysis by the investigative team. In most instances these tasks are done by forensic suites or other automated or semi-automated processes or toolchains. File hashing and comparison to alert and ignore lists typically happens during this phase as well. Alert lists are used to identify files that have previously been associated with malware, contraband, anti-forensics tools, and the like. Ignore lists are used to identify files associated with operating systems and common applications to exclude them from further processing, as they likely have little probative value in more circumstances. The extraction and transformation phase consumes a significant amount of resources, including processing time, storage, and other infrastructure such as database servers. Full processing of a modern hard drive can take several days or longer. In spite of this, most of the time in a case is spent in the analysis phase.

Analysis Phase

Once processing is complete, or even while it is still ongoing in some instances, the data is made available to the investigative team for analysis and review. Broadly speaking there are two types of analysis that are performed: technical analysis and content analysis.

Technical analysis is when an appropriately knowledgeable technical reviewer will examine data on the system and determine some investigative meaning or relevance from it. Often this involves analyzing items such as system settings, time stamps, configuration files, and other system artifacts to determine a logical “real world” meaning to the technical data. An examiner could look at certain registry keys, time stamps on program files and folders, and an executable in the download folder and determine that a particular program was downloaded and installed at a particular time. Technical analysis can be thought of as “viewing and interpreting the data as the system or an application would”. Some items such as web history that previously required technical analysis can now be parsed and rendered “human viewable” by forensic software, allowing the investigators to review it directly. For critical items a second review by a technical expert is a good practice, rather than relying only on a tool’s interpretation of data.

Content analysis and review involves looking at the data and files as a user would. Specific content can be prioritized for review using searches for key terms, starting with likely drive locations, or filtering by particular types of content. Content analysis must be performed by somebody who is knowledgeable about the particulars of the investigation and some familiarity with computers, but it is not necessarily a technical expert. Depending on the content, additional subject matter experts may need to be brought in, such as linguists or forensic accountants.

Often technical analysis and content review may look at different facets of the same item. Content review may identify a relevant email, and technical analysis may show that the sending address for the email was spoofed to appear authentic when in fact it came from a different address.

The phases in digital forensics are not always strictly linear. A new relevant data source may be identified during the content analysis and lead the investigative team to collect that source, which must then be processed and analyzed. A file or file fragment may be encountered during content review that was damaged or otherwise not fully parsed. After manual reconstruction and repair, the remaining content of the damaged file is made available for content review.

At the conclusion of the analysis phase, some type of report is typically produced.

Volatility and Data Degradation in the Forensic Context

In digital forensics the distinction between volatile and non-volatile information is typically made between data (evidence) that will be lost when a system is shut down such as memory contents, and information that will persist on storage after loss of power (Carvey, 2009, pp. 3-4). After the system is powered down it can be transported back to a controlled laboratory environment so the storage can be safely acquired using standard forensic procedures and equipment. In this context data volatility guides the investigative team in prioritizing the acquisition of data from memory. In an intrusion response scenario, the bulk of the pertinent information may come from the system memory, particularly running processes and open connections.

The concept of a relative “order of volatility” is brought up multiple times in the literature, with slightly different meanings. Casey (2009, pp. 4-7) discusses this as rough order to collect data during a live response based on the expected lifespan of the data and the risk of inducing changes to that data while interacting with the system to acquire it. Farmer and Venema (2006) also discuss an ordering of volatility, but also associates the particular data lifespan with the underlying physical storage medium. Shavers (2013, pp. 11-12) identifies that “all data is considered volatile, in that, at a point in time, that data may not be available for recovery” and “data in RAM is susceptible to be destroyed by natural processes of the operating system or user intervention in nanoseconds”. Although in the text the author was specifically referring to the need to capture memory quickly in a live response scenario, the statement is true for all data on a system, both in non-persistent and persistent storage. Even if not directly changed non-persistent storage will be lost if power is interrupted to the system, whereas persistent storage requires an affirmative action to be changed.

Casey also conceptually applies Locard’s exchange principle to digital forensics. In essence the principle states “when two objects come into contact, material is exchanged or transferred between them.” The same principle applies to users and computers, digital devices, and even subcomponents in a system.

Rather than two things coming "into contact" the exchange principle for digital forensics can be rephrased as "when any two parts of a user-computer system interact, they are likely to induce changes in each other, both obvious and subtle." Change due to user interaction with a system can occur whether the user is the owner of the system or an investigator attempting to collect data. Presumably the investigator is using forensic best practices to minimize the changes introduced to the system while interacting with it. No similar assumption can be made about the regular user of a system. They may be attempting to induce significant changes, particularly if they are attempting to erase information or otherwise tamper with potential evidence.

In practice as a system responds to the interaction with a user, the user facing components of the system will then interact with other components and system processes, inducing changes in those components and their corresponding states, and so on. This will be addressed in greater detail in chapter six with the activity model for a computer system and activity cascades. Advances in virtual machines have made it technically possible to immediately capture most parts of a virtualized system all at once, although there is always some interaction with the host hardware and software that is more challenging to completely capture.

Some of these interactions can be subtle, and it is the essential when acquiring data to minimize the effects of these changes on the collected data. A write blocker will prevent changes from being made to the content of a source hard drive, but the "Self-Monitoring, Analysis and Reporting Technology" (SMART) on the drive controller will continue to increment the "Power-On Hours" counter as long as the drive is powered.

Such a change is unlikely to affect the results of the forensic analysis, but the change is still being made nonetheless.

Farmer and Venema (2006, p7) identify that in practice it is impossible to capture all the data on a computer as the act of capturing it will necessitate changing other parts of it. This is certainly true when performing data acquisition from a live running system, and particularly in memory acquisition. They describe this as a version of "Heisenberg's uncertainty principle applied to data gathering and system analysis".

One of the challenges peculiar to IDF is that in most cases a significant amount of time has passed between the activity of interest and when the investigative team acquires the data from the subject computer. Whether days, weeks, or months have passed, if the computer was still being used then the data stored within is still being changed. Old files are deleted, web history is overwritten by new browsing activity, and so on.   In the example of the threatening email, if it was sent through a web client in a fit of anger one night and not repeated then it is likely that new activity over several months will overwrite most if not all of the traces on the system. That is not to say that the evidence will not exist on the system after a month or a year, as each case must be approached individually. Instead it becomes less likely that any particular piece of data will be found after a long interval. Most user actions trigger a cascade of changes to a system, so some evidence may be recovered later even if portions are missing.

Data Degradation over Time

The concept and function of data degradation on a system over time has been explored both conceptually and on exemplar datasets. Garfinkel (2012) explored the factors affecting data decay, but admitted that “we currently do not understand them well enough to come up with quantifiable numbers.” He primarily looks at the problem from a filesystem and disk sector approach, where actions and interactions with files will change the underlying storage, overwriting free space and leaving file residue behind.

A novel longitudinal study of filesystem metadata from Microsoft employee desktop computers found that the median file ages ranged from 80 to 160 days across the collected datasets (Agrawal, Bolosky, Douceur & Lorch, 2007). They also found that less than 20% of the files were less than 30 days old. It is likely that this is a result of the comparatively small amount of user-generated content on modern computer systems in comparison to the number of operating system and application files. A user may spend five minutes writing an email or an hour editing a document, but installing a program will add hundreds or thousands of new files. User generated content is also more likely to be modified or deleted during the system lifecycle than core system files.

A direct analysis approach looked at the traces left behind when applications are installed and then uninstalled from an exemplar system (Jones et al., 2016). This approach used the technique of sector matching, as well as removing the pre-usage drive state from the analyzed dataset. They also use the M57-patents dataset in their work, declaring them “sufficiently realistic”.

A different study (Roussev & Quates, 2012) also used the M57-patents dataset to evaluate a triage technique based on similarity digests. In their work they identify a focus on metadata as being a particularly efficient means of triage due to the relatively small amount of metadata compared to content on a typical system. I agree with this, which is why my work focuses on a particular subset of metadata, time-activity data. They also state that “most metadata is for human use and tends to contain higher-level logical information. The downside of metadata analysis is that it may not be complete, or trustworthy.” I partially disagree with this, in that most metadata is not intended for “human use”, at least not directly. Metadata is often used by the operating system to provide assistance or information for the user in some manner, and it does typically contain inherently meaningful content. Data manipulation is a concern for all forensic data, whether metadata or traditional file content. Due to the way activity cascades and secondary effects function, intentionally tampering with metadata is non-trivial and is difficult to accomplish without leaving indications. In many situations metadata can simply be overwritten with newer data. In the event there is intentional tampering, it will be hard to conceal fully. Although the original data is likely to be unrecoverable, the evidence of tampering is harder to successfully conceal. Jones et al (2016) notes:

> In most cases, however, digital artifacts and traces are altered, destroyed, and disassociated over time due to normal system operation and deliberate obfuscation activity. As a result, analysts are often presented with partial and incomplete artifacts and traces from which defensible conclusions must be drawn.

Timeline Analysis

Timeline analysis blends technical and content review in an attempt to create a sequential list of relevant actions and events from the system being analyzed. Timeline analysis can show patterns and trends on a computer system that may not be apparent when performing a basic content review of the files contained on the system. Although it is not necessary to conduct a timeline analysis in support of every investigation, many complex investigations benefit from an understanding of the overall pattern of activity on the system. The timeline is useful both for understanding the lifecycle of the system itself, as well as placing relevant items identified during the review of files and other content within the larger context of actions on the system.

Basic timeline creation and file system activity analysis in computer forensics dates back to scripts created in 2000 by Rob Lee (Carvey, 2012, p. 196), and was significantly advanced in 2010 with the creation of the Log2timeline tool and publication of the corresponding whitepaper (Guðjónsson, 2010). Although there is some difference of opinion regarding exactly how to integrate timeline creation and activity analysis with traditional forensic content extraction and analysis (Carvey, 2012, p197), it is well accepted within the forensic community that this analysis is an integral component of many complex forensic analysis challenges.

One ongoing challenge in timeline creation and analysis is the sheer amount of time records that are present on a typical personal computer.   A narrowly targeted analysis of a subset of records is relatively simple to analyze and understand and explain, but runs the risk of missing relevant and potentially exculpatory information from other sources. Most tools provide some capability to search, filter, and display a subset of the collected time records, although finding relevant items within this large dataset is a non-trivial task.

Current timeline creation tools like Plaso (formerly Log2timeline) are excellent at identifying and extracting time-data records from a wide range of sources and compiling them into a “SuperTimeline” (Guðjónsson, 2010), which is a monolithic combination of time events from all sources within a single forensic image. Although this is an essential and necessary step in assembling the time data understanding the complete lifecycle of a system, simply combining all the records from multiple distinct sources discards critical contextual information that can be leveraged to make conclusions and inferences than can be made by only looking at the individual records. One technique that leverages this contextual information to draw conclusions is called *relative ordering* which is described in chapter four.

Errors and Inconsistencies in Time Records

A significant amount of research has focused on seeming inconstant time stamps on files and in other records. One interesting work attempts to identify logical inconsistencies in sequences of time records such as a record of a user executing a program on a system without a prior corresponding record of a successful login (Marrington, Baggili, Mohay & Clark, 2011). There are applications of this formalized logic to systems and processes where actions are fully logged and those records are well maintained, but on a typical end-user system there will be insufficiently robust logs for this type of analysis. The authors also identify that:

> The most common inconsistency in digital evidence is naturally occurring, that is to say, inconsistency which is not the result of deliberate tampering. This includes data pertaining to an event or file which simply is not recorded, or may have been over-written during the normal operation of the computer system. It also includes “naturally” erroneous or inaccurate data, perhaps due to a hardware characteristic,

> software misconfiguration or bug. Such natural inconsistencies pose difficulties for investigators, even if they are not the result of deliberate action taken by a suspect.

Registry settings can affect system states and as well as the changes that result from activities (Carvey, 2009, pp. 43). One setting disables the updating of the stored access times when files are touched. Another setting causes the page file to be cleared on shutdown. If enabled, this has two potential effects. First if a live system is encountered during the acquisition phase, shutting it down would result in clearing the page file. As the page file may contain older data than is present in RAM, this may result in the loss of some historic information that is otherwise unavailable.

An analysis of file times from a corpus of over 1000 drives (Rowe & Garfinkel, 2010) found that:

> While timestamps provide valuable clues about the kind of usage of the drive and the nature of the user, they pose challenges: There are default times, scheduled times, bursts of events, erroneous clocks, inconsistencies in the meaning of creation time, and a variety of other phenomena.

Temporal information can be particularly attractive for bulk or automated analysis because it has relatively comparable values even across a wide variety of different files and file types. Rowe and Garfinkel also identified that in some instances the usage of a particular file could be inferred by the relative values of its various timestamps:

> Our triage also identifies (but does not exclude from further analysis) those files with special relationships between their timestamps: files modified before being

> created, files accessed before being created, and files modified before being accessed. Modification before creation is typical of purchased software, where the modification time originates from the vendor. Access times before creation indicate files that have been accessed without changing the metadata, perhaps to reduce operating-system bookkeeping. Modification before access (when modification is not before creation) indicates files that are probably read-only.

Other authors also emphasize the risk of severe clock skew or intentional tampering as a potential cause for time stamps that appear incorrect or "out of order" (Raghavan, 2014).

Although tampering is technically possible and needs to be considered in complex analysis, many of these supposed time inconsistencies are usually the result of a human analyst interpreting time records differently from the actual purpose behind the recorded time. In plain English this means the error lies in "what we think it should mean" rather than "what the computer means with this time record, based on our training and experience". Unless extremely narrow intervals are critical for understanding particular actions then for practical purposes clock skew can generally be ignored when examining a single system, as well as the drift of the clock over time. Extreme clock changes should be identifiable in the system logs or through other means. These changes are often the result of time zone changes, loss of battery power for the onboard clock, or other loss of the stored time value.

Activity Modeling and Differential Forensic Analysis

Modeling system interactions is common in many fields, but has not been extensively applied to IDF. One paper describes a method to model user actions captured in logs in an attempt to identify anomalous actions (Marrington, Baggili, Mohay & Clark, 2011). This does not attempt to analyze the full scope of time data from a system, rather it looks for logged behavior without the corresponding pre-condition being logged, such as a user executing a program without first logging onto the system. The authors also admit that "there are, of course, many instances where an event may be missing as a result of non-suspicious computer activity." The tool they developed, CAT detect, leans heavily on the characteristics of formal Windows logs, and thus can draw strong inferences from that data. For data outside of formal logs, such as file MAC times, the outcomes can only be inferred. They also had to modify the system configuration of their exemplar for "maximum logging". Most users will not do this, particularly if they are conducting activities that may result in the attention of the authorities. Servers are a more likely host to have more robust logging options enabled.

One effort attempts to formally model a system for event reconstruction through state transitions and process algebra (Soltani & Seno, 2019). Even limiting the model to a narrowly focused compromise of simplified system quickly runs into "state space explosion" as the transitions make changes to the underlying storage. This model is backwards looking, in that it takes the system in the final collected state and attempts to project backwards and determine the actions that brought the system from a secure state into the compromised state. The authors state that "although there are several attempts to formally reconstruct events, it is still in its infancy stages. Many aspects of modeling systems and specifying digital forensic requirements should be further explored.”

A different effort describes a formal model for intrusion analysis to support timeline reconstruction and analysis (Chabot, Bertaux, Nicolle & Kechadi, 2014), which attempts to project backwards to user actions from time records through an “event reconstruction” process. Their model uses temporal closeness of particular events to infer actions by a particular user, which is a technique used by most investigators, although not a formally defined.

A model for differential forensic analysis and image differencing (Garfinkel, Nelson & Young, 2012) establishes the basic model of user actions transforming a system between two distinct states. They acknowledge that the “practice has been increasingly used by digital forensic examiners but not formalized until now.” There are several differencing strategies described in the article, including disk image comparison and registry hive comparison. The general strategy described is in line with the sequential state analysis described in chapter seven and the case study in chapters eight and nine.

Chapter Three – Time Activity Data

Introduction

Before examine examining individual records, it is worthwhile to look at their characteristics in a generalized manner, and similarities and differences between different items and sources. This chapter will discuss characteristics that apply to individual records as well as those that apply to collections of these records.

This chapter and the next provides a thorough taxonomy for describing temporal record characteristics and how to leverage these characteristics when conducting activity analysis. This work will also touch on some analysis techniques that integrate activity analysis with the traditional forensic process to provide new insights for the overall investigative digital forensic process. After first defining and describing the characteristics of metadata and time activity data (TAD), a taxonomy will be established for TAD collection types commonly encountered in computer forensics.

Metadata Analysis and Characterization

In the field of computer forensics, metadata is often identified as an essential element in many investigations. Reviewing the literature provides an extensive discussion of examples of metadata, emphasis on the importance of it, and techniques for preserving, extracting, and analyzing it. There is little discussion of the characteristics, classification, or categorization of metadata in generalized terms.

Carvey (2009, p299-300) reinforces the most common description, “the term metadata refers to data about data” and “the most commonly known metadata about files on Windows systems are the file MAC times.” MAC times refer to file modified, accessed, and created times.

Occasionally metadata will be directly created or modified by a user, but most often it will be generated by operating system processes or application settings and incorporated into files and disk structures without specific user direction. Metadata can be embedded within the structure of the file it describes, or it can be separate as part of the filesystem structure. Metadata can describe a wide variety of characteristics of an item. For this work I will focus primarily on items that contain a time reference or are otherwise relevant to our activity analysis.

One particularly subtle insight on metadata comes from the ForensicsWiki website (Https://Forensicswiki.Xyz):

> Since metadata is fundamentally data, it suffers all of the data quality and pedigree issues as any other form of data. Nevertheless, because metadata isn't generally visible unless you use a special tool, more skill is required to alter or otherwise manipulate it.

In most instances metadata is abstracted away from the user and managed by operating system or application processes. User actions can and will change metadata, most often this will be a side effect of other actions.

For the purposes of this work, I define *metadata* as data that contains information either about itself or another distinct data item. A *data item* is defined as a discrete piece of data in the total corpus of data being analyzed. This is typically a file, but could also be a logical sub-section of a file, a portion of unallocated space on a drive or any arbitrary portion of the source image bitstream. A particular data item does not necessarily need to be contiguous on the drive image, but the investigator must be able to provide a rationale or justification for collating disparate portions of the source image into a single logical item.

Consider a JPEG file (JPEG / Exif) created with a digital camera and now contained on a hard disk image under examination. The file itself is a data item. The file also contains two separate data items: the JPEG image stored with a particular compression algorithm and the Exif tag structure that describes the JPEG and how it was captured. The Exif data item contains metadata which describes the JPEG image sub item. As the JPEG file itself is stored within a file system, there will be additional metadata contained within the file system structure that describes it, such as created, modified, and accessed times (MAC times). We will return to this JPEG example as we continue to enumerate metadata characteristics.

Metadata characteristics describe features or qualities of individual pieces of metadata without necessarily being contained within the metadata itself. Two metadata characteristics that are particularly relevant to activity analysis are source and location.

Table 1

Summary of Metadata Characteristics

Characteristic	Possible Values
Metadata Source	Primary or Secondary
Metadata Location	Internal or External

Metadata Source

The *metadata source* describes where the metadata is contained relative to what it describes, and can be either primary or secondary. *Primary metadata* is metadata that is contained within the item(s) or container(s) being investigated or examined, or algorithmically generated from it. This metadata is generally extracted from the item using one or more forensic tools or processes, or it may be viewable using standard system tools.

Secondary metadata describes a container or data item being examined that has been generated or applied through a user-based process, usually by an investigator based on analysis or outside knowledge. Examples of secondary metadata include the collector's biographic information, case number, date and time collected, image filename, acquisition log, and surveillance log. When applying secondary metadata to an item it is important to ensure that it remains non-prejudicial. When basing conclusions on secondary metadata one must ensure both that the information is given proper weight based on its source and that critical items are verified through other means whenever possible.

Any dependencies for secondary metadata must also be noted. The calculated MD5 or SHA1 hash of a file is primary metadata. When that hash is compared against a hash database for contraband or malware then any positive or negative matches are secondary metadata. The calculated hash itself insufficient to determine the characteristics or content of a file without some external reference.

For the purposes of this work any data or metadata items that are contained within the image file metadata or acquisition notes but were not part of the collected source drive are considered secondary metadata. Examples of this may include the collecting technician's biographic information, date and time collected, or the physical description and condition as found.

Metadata Location

Metadata location describes where the metadata itself is stored relative to the data that it describes, whether internal to the file or elsewhere within the storage. *Internal metadata* is metadata about a specific data item that is contained within that file, or is generated algorithmically by analyzing only the data within the file. *External metadata* is metadata about a file that is contained elsewhere.

Consider a JPEG file contained on a hard disk image under examination. The EXIF information contained within is characterized as primary and internal metadata, as is the calculated MD5 hash of the file. The filesystem-level created, modified, and accessed time stamps stored in the directory listing are characterized as primary external metadata. An investigator's bookmark of the file and a comment "Subject's Car" within the forensic tool would be secondary external metadata. It is assumed in this instance that the bookmark information is stored in the working database of the particular forensic tool or in a report.

Time Activity Data Definition and Structures

A single *time-activity datum* (TAD) is a record that contains at a minimum a time stamp (pre-normalization) and record source as mandatory components. Without a date an item may still be metadata but it cannot be placed on a timeline. It is possible for a TAD to consist of no other data other than a timestamp if the meaning of the timestamp if the meaning is implied by the existence of the record, which will be shown in the example in table two. Most TADs contain additional raw content from which additional data components may be extracted or derived.

The *time-activity data corpus* is defined as the set of all collected and available *time-activity datums (TADs)***.** This set can be constructed from multiple data sources, including but not limited to:

- Computer hard drive image(s)
- External Servers and services (File server, email server, syslog service)
- Loose Media (disk images and individual files)
- Network Devices and Services (firewall logs, VPN concentrator)
- Peripherals and devices (printers, cameras, cell phones)
- Network captures
- Subject / Victim / Witness interviews, voice mails
- Subpoena Returns (business records, emails, provider logs)
- Alarm Systems / Door access controls

It is worth noting that this listing contains several items that are not typically considered part of traditional computer forensics, though they are familiar to many criminal investigators. Nonetheless they may still be useful collections of relevant activity data, to which the techniques described below can be applied.

Why Computers Keep Records

Before going deeper into activity analysis, it is worth thinking about why computers log or record time-activity data in the first place. In some circumstances the operating system or applications create and maintain actual logs for security, audit, or troubleshooting purposes. In most end-user systems these actual logs are the minority of the time-activity data available to be analyzed. Contrast this with servers and network devices where much of the available TADs come from actual logs**.** User-focused operating systems are generally not designed with logging or security in mind. Creating and maintaining logs takes storage space and CPU cycles that could otherwise be used for data processing, user interface animations, and cat videos. In chapter four we will discuss some distinctions between different types of TAD sources.

Most TAD sources on a user machine are not intended to directly log or track user activities. They may store relevant temporal metadata to provide some functionality to the user, or there may be seemingly innocuous data that is peripherally stored by low-level system functions, such as filesystem timestamps. Storing web browsing history allows a user to recall a site that was visited in the past, even though they may not remember the exact date or web address. File modified-accessed-created (MAC) times allow users to search for and find recently used documents that they have misplaced on their increasingly large hard drives. Keeping track of modification dates allows software to synchronize the contents of two directories (typically located on different computers networked together) and update any changed files with the latest version.

Metadata fragility generally describes how likely a particular type of metadata is to be changed through normal system activity. This type of fragility is distinct from loss of data through file deletion where the metadata contained in a file is lost when the file itself is lost. Metadata that is very fragile is easily changed through routine user actions and system processes, such as updating file modified or access times. Slightly less fragile would be the printed record in the of a document, which is only updated when the user takes a specific relevant action. Even less fragile metadata is unlikely to be changed without specific intentional user actions or special tools, such as the 'Author' field of a PDF document. A combination of very fragile and less fragile metadata can describe a particular file, and even coexist within a single file or source. Fragility does not have a specific numeric value or scale, instead relative ordering is more appropriate. Fragility itself and the relative ordering will also vary based on the particular system, user actions, and the technical savvy of the system user.

A Simple Time-Activity Datum Source

Starting with an example of an extremely simple log will provide a base for understanding source characteristics. This log is from an electronic lock on a secured door at a business. There is only one code, shared between all authorized employees. The lock is not connected to any external systems, and the log is stored in internal memory. There is a clock chip inside which keeps accurate time according to the manufacturer. Data is extracted using a serial cable and special maintenance software. The contents of “SecureDoor1.csv” are below:

Table 2

Data from SecureDoor1.csv

Line	Value
1	7/17/2017 07:54:33
2	7/17/2017 16:22:17
3	7/18/2017 08:03:57
4	7/18/2017 17:05:32
5	7/18/2017 22:16:03
6	7/19/2017 07:45:31

After collecting the data, the technician tests the lock with a known event by opening and closing the door then extracting the record and determines that the internal clock is accurate to within one second and that the times stored are Eastern Standard Time. Reviewing the technical documentation revealed that the lock only records door opening events, not closing events, and that only the six most recent events are stored. Changing the door code or internal clock causes the log to be erased. The lock does not record how long the door was open or how many people passed through while it was open. Although it is not possible to tell from the logs if the door was left open for an extended period of time, while collecting the data the technician notes that a separate switch causes a loud and annoying buzzer to sound if the door is open for longer than 30 seconds.

Note that in this example there is no identifier for the specific door contained in the log. Instead it is inferred from the collection source. If the identifier is left off or marked incorrectly during collection there can be significant problems with later analysis. The characteristics and capabilities of the door logging process were determined by reviewing technical materials and verified experimentally. The specific time offset for the door had to be determined experimentally. Even though this log was a simple TAD source, multiple investigative, technical, and analytic steps were necessary in order to extract and preserve the data, as well as to determine the additional information necessary to place the extracted TADs in the proper context for the investigation.

A Slightly More Complex Log

Now consider a centrally monitored alarm panel with cellular network connectivity and multiple users with unique codes. The alarm company provides the log displayed in the following table.

Table 3

Data from Alarm Company

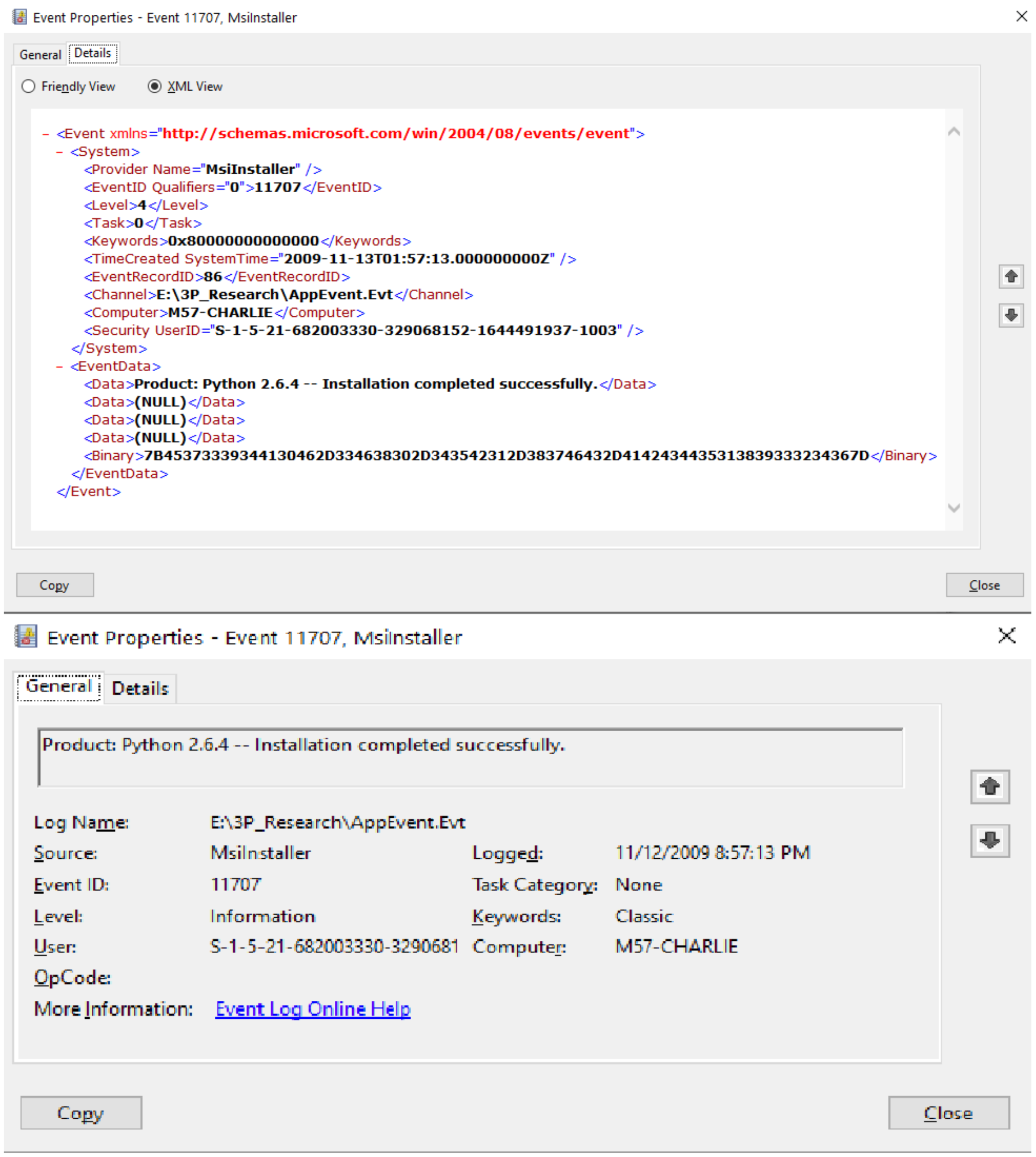
Line	Time	Action	User
1	6/17/2016 15:40:54	Disarm	Alice
2	6/17/2016 20:08:22	Arm	Alice
3	6/18/2016 16:22:03	Disarm	Alice
4	6/18/2016 20:39:05	Arm	Alice
5	6/19/2016 01:22:16	Disarm	Bob
6	6/19/2016 01:35:45	Arm	Bob

The alarm company informs the investigator that the records are stored in UTC. The usernames were provided by the client when the system was installed. Cellular network connectivity, third party monitoring, and offsite storage of logs provide a level of external validation that was not present in our first example.

Windows Event Log Time Datum Example

Consider the following TAD extracted from a Windows 7 AppEvent log. The source file was in the ‘charlie-2009-12-10.E01’ image, part of the M57-Patents dataset. Within the disk image, the record was originally located in the Windows\System32\Config folder on partition one in the AppEvent.Evt file.

Most TAD records and can be viewed with multiple different tools, on both the live system and when extracted from a forensic image. The figure below shows the TAD record as displayed using the native Windows event viewer software.


Event Properties - Event 11707, MsiInstaller
General
Details
Friendly View
XML View

```
- <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  - <System>
      <Provider Name="MsiInstaller" />
      <EventID Qualifiers="0">11707</EventID>
      <Level>4</Level>
      <Task>0</Task>
      <Keywords>0x80000000000000</Keywords>
      <TimeCreated SystemTime="2009-11-13T01:57:13.000000000Z" />
      <EventRecordID>86</EventRecordID>
      <Channel>E:\3P_Research\AppEvent.Evt</Channel>
      <Computer>M57-CHARLIE</Computer>
      <Security UserID="S-1-5-21-682003330-329068152-1644491937-1003" />
    </System>
  - <EventData>
      <Data>Product: Python 2.6.4 -- Installation completed successfully.</Data>
      <Data>(NULL)</Data>
      <Data>(NULL)</Data>
      <Data>(NULL)</Data>
      <Binary>7B45373339344130462D334638302D343542312D383746432D4142434435313839333234367D</Binary>
    </EventData>
  </Event>
```

Copy
Close
Event Properties - Event 11707, MsiInstaller
General
Details

Product: Python 2.6.4 -- Installation completed successfully.

Log Name:	E:\3P_Research\AppEvent.Evt		
Source:	MsiInstaller	Logged:	11/12/2009 8:57:13 PM
Event ID:	11707	Task Category:	None
Level:	Information	Keywords:	Classic
User:	S-1-5-21-682003330-3290681	Computer:	M57-CHARLIE
OpCode:			
More Information:	Event Log Online Help		

Figure 1

Example Event in Native View and XML

Most modern computer systems store time data in a UTC-based format and then dynamically convert it to the local time for user display based off system settings or geolocation. Time data extracted directly from logs will often be in a standardized time format, while items displayed for user information will use local time, such as a chat window. It is important to keep this in mind when cross correlating records from different sources.

Internally most computer systems typically use an epoch-based time system, where the system counts incrementally forward from a specific starting point. Each integer in the count is one unit of time, and represents the maximum granularity of that implementation. Two common epoch implementations used in modern systems are the NT time epoch which counts forward from January 1st, 1601 and the Unix epoch (also known as POSIX time) which counts forward from January 1st, 1970. Both have a base granularity of one second but can use additional digits to subdivide into smaller intervals. Microsoft Windows uses a 64-bit FILETIME structure that counts the number of 100 nanosecond intervals from January 1st, 1601. The IBM PC BIOS, DOS, FAT12, FAT16, and FAT32 implementations use an epoch that starts on January 1st, 1980. We will see an effect of this in chapter 5.

In this example, the TimeCreated entry for the event is stored in UTC time represented by the ‘Z’ in the XML entry, but is displayed to the user in system local time by converting it to EST, shown in the lower half of the figure. The same actual time can be converted and displayed in different time formats shown below:

- Unix epoch: 1258077433
- Unix epoch with milliseconds: 1258077433000
- NT time epoch: 12902551033
- FILETIME epoch: 129025510330000000
- UTC: Friday, November 13, 2009 1:57:13 a.m.
- Local time: Thursday, November 12, 2009 8:57:13 PM GMT-05:00

Processing the disk image and extracting the record with Plaso yields the data in the following table.

Table 4

Example Event Extracted and Parsed with Plaso

Field Name	Value
Timezone	UTC
Source	EVT
Sourcetype	WinEVT
Type	Creation Time
User	Charlie
Host	M57-CHARLIE
description	[11707 / 0x2dbb] Source Name: MsiInstaller Message string: Product: Python 2.6.4 -- Installation completed successfully. Strings: ['Product: Python 2.6.4 -- Installation completed successfully.', '(NULL)', '(NULL)', '(NULL)'] Computer Name: M57-CHARLIE Severity: Success Record Number: 86 Event Type: Failure Audit event Event Category: 0
filename	/WINDOWS/system32/config/AppEvent.Evt
inode	3665
format	winevt
extra	facility: 0 message_identifier: 11707 recovered: False sha256_hash: 61478990af1284ce01e5bf894cdaaa54f1f71075fd44a6c0a0a15f6c0c26ca4d
datetime	2009-11-13 1:57:13
offset	14616
record_number	86
event_identifier	11707
event_type	4
source_name	MsiInstaller
user_sid	S-1-5-21-682003330-329068152-1644491937-1003

Normalized to the TAD record structure for analysis, the following fields would be added to the record so that it can be combined with records from other data sources:

Table 5

Event Data Normalized to TAD Structure with Minimal Fields Shown

Field Name	Value
Unique Record ID	32435457
Date-Time Group (epoch converted, ms)	1258077433 000
Data Source Container	charlie-2009-12-10.E01
Data Source Path	\Partition1\Root\Windows\System32\Config\
Data Source Filename	AppEvent.Evt

The mandatory data source field is broken up into three sub-components: container, path, and filename, to facilitate later analysis and review. The unique record ID is programmatically generated by the software that combines records from multiple data sources. The date-time is generated by converting the record's specific time stamp value into the Unix epoch format. Even though the stored value is only accurate to the second, the converted epoch time is padded out to milliseconds so that it can be readily analyzed along with other records that do have millisecond precision. The Windows FILETIME epoch could also be used instead, as long as it is standardized throughout the analysis dataset.

Chapter Four – Time Activity Data Collections

When combined with the proceeding chapter, the work in this chapter establishes a functional taxonomy of time activity data for the field of investigative digital forensics.

TAD Collection Definition

A logical group of one or more *time activity datums* is called *TAD collection***.** TAD collections are often colloquially referred to as "logs". In many instances these TAD collections consist of metadata items, in others they contain a list of activities or events that occurred on, by, or to the system.

Most individual TADs are not independent items within a hard drive or disk image. They are typically either embedded within other data items or part of a larger collection of TADs, either physically or logically. Just as TADs have individual characteristics, collections of TADs have an additional set of common characteristics. The characteristics that are integral for activity analysis are listed in the following table.

Table 6

Summary of Time Activity Datum Collection Characteristics

Characteristic	Possible Values
TAD Collection Type	Actual, Functional, or Assembled logs
TAD Collection Location	Internal or External
TAD Collection Ordinal	Event Drive or Item driven
TAD Collection Depth	One or n (assembled logs), N/A (actual logs)
TAD Collection Breadth	Size or Time range (actual or functional logs only)
TAD Collection Lifespan	Start and end Values (all logs)
TAD Collection Granularity	Time Interval Value

TAD Collection Types

TAD Collections can be categorized into three distinct types: actual logs, functional logs, and assembled logs. By characterizing and analyzing the time data sources within specific collections we can often gain additional contextual information about the activities and data on the system, which would be lost if we simply examined the records individually or compiled them into a single monolithic list. Identifying the exact characteristics and values for a specific TAD collection will generally be determined by the analyst on a case-by-case basis at present. Tools could be developed to extract and present many of these parameters for review and to support complex analysis tasks. Each distinct source may have varying characteristics, and these will be influenced by the operating system, version, applications, configuration, and user activities.

Actual logs or formal logs are TAD collections that are maintained by applications or operating system services and designed to log activities or events. They are often stored in a structured database or other regular format such as a comma separated variable (CSV), extensible markup language (XML) or text file, with new TADs (log entries) appended sequentially to the end of the file. The Windows Event Log is an example of an actual log.

Functional logs or informal logs are commonly created by end-user applications. They have most of the characteristics and fidelity of actual logs, but were not necessarily intended to be used explicitly for activity logging. Functional logs are typically generated as a byproduct of enabling some user or system functionality, but the purpose of collecting the data was not to log or audit the activity. A saved chat session or internet history file is an example of a functional log. Functional logs are also commonly stored in a structured or regular format such as a database or text file,

Both actual and functional logs are generally stored on a system as a single file or logical entity. The data may be spread across multiple distinct files that are designed to be processed together, such as internet browser caches.

Assembled logs or derived logs are created by parsing and collating data from multiple sources and locations within a system or across multiple systems to create a logical TAD collection. Once extracted and collated using appropriate tools, assembled logs can be used for analysis in much the same manner as actual and practical logs.

Assembled logs are commonly created by extracting a common piece of metadata from multiple distinct files and constructing a list of a particular type of activity on a system, or activities across a subset of relevant or interesting files.

Some characteristics common to assembled logs can present challenges when analyzing and drawing conclusions based on the data in assembled logs, including coverage gaps and other issues typically not found in actual or functional logs. Even with these weaknesses, assembled logs can be invaluable in analysis because they can capture a wide array of information that would not typically be captured in an actual log on a system.

Examples of assembled logs include a listing of file system MAC times, e-mail time stamps, and registry key update times. Extracting and collating all of the last printed time metadata items from all the Microsoft Word documents on a system creates an assembled log of the printed documents. In these examples each metadata item with the same characteristic is collated into a logical TAD collection.

TAD Collection Characteristics

Each TAD collection has its own set of characteristics which often relate closely to the TAD collection type. These collection characteristics can be a powerful way to answer otherwise difficult questions in the analysis of a system. These characteristics help describe the data that is found within the TAD collection, as well as explain why other data is not present in the collection and corpus. Guðjónsson alludes to some of this with the term *temporal proximity*, which refers to records being close to other related records in time, even if the specific record is missing, overwritten, or otherwise untrustworthy. (2010)

The *TAD collection location* is comparable to the *metadata location* described earlier. *Internal TAD collections* describe the system or container they are stored within. *External TAD collections* describe a different, distinct system.

Lifespan

Each TAD Collection has a specific *lifespan*, which is the range of time that the log provides activity records for. The most basic way to determine the lifespan for a collection is to look at the earliest record and latest record chronologically. In some circumstances we may be able to slightly extend the lifespan for a collection earlier than the first record or later than the last record, based on an analysis of the overall system and the way the logging was performed.

Consider a system log that tracks attempted and successful user logins. If the last entry is on Tuesday but the investigator has determined through other analysis that the system was powered on and running normally until Friday, they can reasonably conclude that the lifespan of that log extends through Friday as it would have recorded any normal logins. This may also allow the investigator to conclude that no user logged in to that system through Friday using typical means.

Depth and Breadth

The *depth* of a collection describes the number of records that a TAD collection can have about each particular item that it describes. Most assembled TAD collections are *one-deep* because the data sources used to assemble the collections are usually overwritten by new activity, only storing the most recent action for each item. Rowe and Garfinkel (2010) note that information from log files "is more complete than file-directory metadata for some files since it reports more than the last event of a type", identifying the essential characteristic of one-deep logs without formally naming it.

Some functional logs may store up to a fixed number of records per item, such as the ten most recently accessed files or the last five times a particular website was visited. If there are only three records for an item in a five deep log then you may conclude that no other activities occurred relative to that item, at least within the constraints of the collection *breadth* and *lifespan*. Most actual TAD collections usually do not have a fixed depth per item, rather they maintain an overall size, number of records, or time limit, where excess records are periodically deleted from the log as an explicit or background function. In this case the TAD collection is not considered to have a meaningful depth characteristic.

The *breadth* of a TAD collection is important when the amount of data in a TAD Collection is affected by system or application settings. System settings or defaults may have a maximum size for actual and functional logs, causing the oldest data to be lost as new data is written and the log size is over a set amount or over a certain age, such as 50MB or six months. Other implementations may stop recording new data once the log hits a certain size. This is sometimes used in network devices to prevent new data from pushing out early items, and as a security measure to prevent an attacker from flooding the logs with activity after a system compromise and pushing out the initial attack. These are configurable settings in many systems which may require an analysis of the source item or configuration files to determine. Assembled TAD Collections do not have a meaningful breadth characteristic.

Granularity, Precision, and Accuracy

For a TAD collection the *granularity* is a measure of how precise the stored time values are. In this use case *precision*, which is the narrowness of the potential observation or occurrence window, is distinct from *accuracy*.

The *accuracy* of a TAD is the offset between the true occurrence time and the time that the system recorded as the time of the action or event. Accuracy in computer forensics is commonly influenced by misconfigured internal clocks, errors in initial system configuration, clock drift over time, or particular time zone settings on a source device. Accuracy is usually consistent within a particular TAD collection, but in some instances may change due to user actions or as a result of clock drift over a TAD Collection with a very long lifespan.

Drifting clocks or an incorrectly set time zone can cause accuracy issues, which can be corrected for using a source-derived offset when normalizing for analysis. A common way to observe the current accuracy of a given system is to boot it into BIOS and compare the internal clock to a known accurate source. Although this will not confirm the accuracy of the clock at any particular point in the past it can still be a useful reference point.

In some investigations the true accuracy of a source may never be known. The relative accuracy of a system at a particular time can be observed from storage alone by comparing a stored time with a closely associated event with an external time reference, such as an email with headers from an external mail server.

The granularity of a data source is not always obvious when looking at the TAD collection values. Sometimes it can be deduced by looking at the underlying source time stamps and how they are stored, other times it can be discovered by reviewing reference materials and standards. In some instances, it must be experimentally discovered by setting up a comparable exemplar system and testing specific relevant actions and events.

FAT-16 Granularity Example

In the FAT-16 filesystem standard, there are different granularities for stored TADs for different types of file activities (Carvey, 2009, pp. 300). Called *date resolution*, they are:

- 2 seconds for last modified time
- 10 milliseconds for creation time
- 1 day for access time

Suppose a FAT-16 source partition has a file with a stored Modified time of December 10th, 2007 at 6:16:02 p.m. It would not be uncommon for a forensic tool to prompt the analyst for the BIOS time zone setting of the system (EST in this case) and report the normalized time as December 10th, 2007 at 23:16:02.00 UTC. We are now faced with a dilemma regarding how to record this value and combine it with other TAD collections to perform activity analysis. Lacking any additional information from other sources, we have no way of knowing if the file was last modified at exactly at 6:16:02.00 p.m. or as late as 6:16:03.99 p.m. One possible solution is to look for any relevant primary metadata (data inside the file) as that granularity and precision is controlled by the file type and the program that modified it, not the host file system implementation.

Importance of Accurate Precision and the Risk of False Precision

In many instances the granularity of records will be sufficient to answer relevant investigative questions, but if more precision is necessary then additional analysis will be needed before conclusions can be reached. Most forensic tools do not explicitly call out the granularity of the underlying data source. Conversion from one recorded timestamp to a normalized epoch time will typically pad the values with additional digits. In traditional science the concept of precision is tracked using scientific notation and significant figures (e.g. 6.02×10^{23}) but there is not an equivalent notation currently in computer forensics. Even if only part of the value is displayed for analyst review, the entire overly precise value is stored in the analysis tool and used when records are sorted by time or otherwise subject to computational processes.

Most modern operating systems implement time granularity as a floor function, where additional digits of precision are truncated. Times manually recorded by humans will often be rounded instead. This difference is worth keeping in mind if one or more of your data sources is human-derived, such as a surveillance log or physical sign-in sheet. A specific machine generated timestamp can be interpreted as “not earlier than the minimum possible value and not as late as the start of the next possible value.” Low granularity is commonly the result of an attempt by designers to save storage space, such as in FAT16 filesystems, or by parsing output intended for human review rather than machine consumption, such as a chat log.

Relative Ordering

In many instances, the granularity of TAD collections is sufficient to order significant user events correctly. When the granularity causes the exact times of critical TAD items to be uncertain, leveraging external sources can provide more precision. *Relative ordering* is a technique that can be used when multiple TADs from the same TAD collection have identical timestamps. *Relative ordering* allows you to infer the order that different TADs actually occurred by their ordering in the file itself. This is often seen in functional logs where the time granularity may be in minutes. Further precision and specificity can be gained if some actions in the less granular TAD collection can be correlated with items from other data sources with higher granularity. These more precise external correlations can be used to narrow down the possible occurrence window for other TADs with more precision than allowed by the initial TAD collection analysis.

Analysis using *relative ordering* is typically only possible for actual or functional logs. The item ordering in assembled logs is usually an artifact of their extraction and compilation and not the actual recording order on the system being analyzed.

Relative Ordering Analysis Example

Suppose a chat log is stored as a text file where each message is prefaced by the received time with a granularity of one minute. There is a sequence of ten messages that are all logged as received at 8:33 UTC. Although each stored time stamp is identical, you can determine the *relative ordering* based on positioning in the log. Suppose the fifth message is "File 1234.txt received and saved to C:\Stuff", which also has a timestamp of

8:33 UTC. Locating the specified file within the disk image, the investigator observes a creation time of 8:33:34.5 UTC, stored on a FAT-32 partition which has 10ms granularity for creation times. They can now conclude that the file was received at 8:33:34.5 UTC, that the four prior messages in the log occurred between 8:33:00 and 8:33:34.5, and that the five messages afterwards occurred between 8:33:34.5 and 8:34:00. In this example we are discounting any delays for file transmission and waits for user interaction, such as a confirmation message. In a situation where these small clock differences are critical to reaching an analytic conclusion then an exemplar system could be configured to determine the probable system behavior under different scenarios.

Analytic refinement using relative ordering and other logic-based techniques will not always be possible or practicable to do for all items or in scale, but for critical questions it can prove invaluable. Differences in granularity between different sources and record types can also explain seemingly impossible actions, such as files being modified on a drive before they appear to have been created or received.

Challenges for Complex Activity Analysis

A fundamental challenge for activity analysis is how to collate and present data from multiple distinct TAD collections with different characteristics such as lifespan and granularity. Historically most activity analysis and related tools commit the dual errors of discarding useful information and introducing potentially erroneous information. Each log file would be reviewed individually, or all possible time records would be extracted and combined into a system- or investigation-wide SuperTimeline (Guðjónsson, 2010). This practice ignores limitations introduced by granularity by taking the event times extracted by each tool and converting that precise value to the equivalent in the time system being used for analysis.

TAD Collection Granularity can make analysis challenging if there is a large potential window when critical events may have occurred and the TAD collection sources lack sufficient granularity. For smaller amounts of data and limited spans of time, spreadsheet-style analysis where each TAD collection source is a distinct column can be used to deconflict differing granularity from different TAD collections.

In some instances critical events from TAD collections with low granularity (imprecise times) can be properly interleaved within the events from TAD collections with high granularity (precise times) through relative ordering, external data sources, or other case specific analysis, but this is a manual process that is not possible in all situations due to lack of necessary supporting data.

Using the established TAD taxonomy, investigators can make stronger conclusions in their analysis by leveraging the characteristics and making inferences about other data that may not be present. Previously TAD records were generally viewed as individual events, or at best this inference was done on an ad-hoc basis.

Chapter Five – System States in Digital Forensics

This chapter first establishes a conceptual model for system states and activity. I will then leverage that model to describe how user actions trigger changes in these states. In later chapters I will utilize a subset of the model to perform differential activity analysis on a series of linked images.

Although investigators often examine them in laboratory settings, computers do not exist in a vacuum in Investigative Digital Forensics, nor are they stationary and immutable. Computers exist in the real world and are acted upon by users, with both known and unknown (or unforeseen) interactions.

For the purposes of this work I define a *computer* a single personal computer, server, or other self-contained computing device, as well as all the components and peripherals that are directly connected to it. Connections made over a network (either wired or wireless) are made between two distinct *computers*. This seemingly simplistic definition does not limit the capability to conduct advanced analysis.

Computer States and Complexity

A computer is a complex machine. Components of a computer include the central processing unit (CPU), motherboard, memory, storage, input / output devices, display, and other items. Conceptually a computer is also an electromechanical device that consists of generally fixed logic and a diverse array of data storage capabilities, often referred to as either memory or storage, depending on the underlying technology. Each bit of data in memory or storage represents either a value of 1 or 0. The collection of all these data bits plus the physical characteristics of a computer represents the *state* of the computer at any given point in time.

A computer also is a deterministic device, meaning that performing an identical action on a computer in the same *state* will produce in the same results. Getting a real-world computer into a completely identical state in multiple distinct instances is non-trivial. Most actions do not interact with all the components of a computer. Two distinct computer states are said to be equivalent with respect a particular action or activity if the result of that action is the same, including any relevant state changes. Outside of digital forensics and testing, this is commonly used when troubleshooting system and software errors.

This work will not attempt to capture or model human actions or states directly but will consider how their actions change the computers they interact with.

Data States in Computers

The rough sizes and orders of magnitude of data capacities on a modern computer are shown in the following table.

Table 7

Rough Order of Magnitude of Data in Modern Computers

Component	Typical Data Capacity
CPU Registers	Bytes (8 bits per byte)
CPU Cache	Kilobytes (10^3) to Megabytes (10^6)
Device Buffers	Bytes to Megabytes (10^6)
Video Memory	Megabytes (10^6) to Gigabytes (10^9)
System Memory	Gigabytes (10^9)
Solid State Storage	Gigabytes (10^9) to Terabytes (10^{12})
Hard Drives	Terabytes (10^{12})
Removable Media	Kilobytes (10^3) to Terabytes (10^{12})

In addition to the locations listed above, there is also data capacity in system boards and add-in cards to hold BIOS, card logic, and to act as buffers. These typically require specialized tools and knowledge to access, and do not factor into most IDF examinations. If an investigator suspects that it may contain relevant information, then this storage can be extracted and included with the rest of the computer data. Garfinkel (2010) identifies these components as an ongoing challenge to forensics as “the persistent memory inside GPUs, RAID controllers, network interfaces … is routinely ignored during forensic investigations.”

The *data state* of a computer is the collection of all data within the computer at a particular point in time. The data state is subdivided into the *memory state* and the *storage state*. The memory state is made up of all the volatile data from the computer, typically memory, cache, buffers, registers, and the like. The data state is made up of all the persistent (or non-volatile) data.

Volatility as a Function of Storage Medium

Data in memory and memory-type locations is generally fast to access but contains less data capacity than storage. Most memory is volatile in that it retains its contents only as long as system power is maintained. Once power is interrupted the data contained in volatile storage degrades relatively quickly cannot be reliably recovered using traditional forensic techniques.

Storage is slower to access than memory, but has the advantage of persisting across reboots. Programs and data are generally loaded from storage into memory to be executed, displayed, or manipulated, and changes are saved back onto the storage. Storage and storage-type locations include hard drives, removable media, and various types of solid-state storage.

Removable media can be treated either as a component of a computer, or as its own separate computer system, at the discretion of the investigator. Connected but removable storage should generally be considered part of the system storage along with internal storage unless there is a compelling reason to treat it as distinct computer system connected across a system boundary. As a rule of thumb if there is any indication the media was connected to multiple systems then it should be logically handled as an independent system used for transferring data between two or more other systems for the purpose of activity analysis.

There are some technologies that offer memory-like speeds while also being non-volatile. If data can be recovered from these devices it can be treated as storage and analyzed, regardless of the underlying technology.

Physical and Data States

The *physical state* of the computer system is the combination of the all the physical components and the physical characteristics. The physical characteristics of a computer, such as the lifespan of capacitors on the motherboard and power supply, may cause a computer to behave differently at different times even under an identical data state. Capacitors that are near the end of their usable lifespan could cause a computer to malfunction under heavy load where a new computer in an identical or equivalent data state would function properly.

As described earlier, the collection of all discrete data storage inside a computer is the *data state*. In many instances adding a device to a computer will change both the physical and data state, as devices may have some amount of data storage capability. Newly connected devices will often be detected by the operating system, which will also trigger changes to configuration files and related items.

Equivalent States Example

As a child I had access to an IBM PC/XT desktop computer. There was a shareware program installed that was configured to automatically run on boot which asked for a "randomly generated" three digit code before proceeding. When booting the system from a cold state (completely powered off) the random number was always the same. When performing a warm reboot, which is restarting a system after it had already been powered on without interrupting power, I observed that the number asked for seemed random. As best as I could determine, this difference in behavior was caused by the lack of an onboard battery to store the current date and time when the system was powered down. When the computer was initially powered on, the system clock assumed the default time of midnight on January 1st, 1980 and began counting forward. When the shareware program executed, it used the value of the onboard clock as entropy to generate a random number. Since the initial state of the onboard clock was always the same after a cold boot, the algorithm produced the same number each time. Moving that particular piece of software to a different order in the startup batch file would cause it to generate a different (but still repeatable) random number, as a different amount of time had elapsed since the clock had started.

This example demonstrates the difficulty of getting a computer into an equivalent state, even on a comparatively simple computer by modern standards. These states are *equivalent* rather than identical because many items had changed, such as the data contained on the onboard storage. The two data states were *functionally equivalent* from the perspective of the piece of shareware, because it only looked at the value of the system clock. Changing the order of startup programs caused the time of execution to shift enough to produce a different generated random number.

The complexity of computer states is why a common and often successful computer troubleshooting step is to “clear the Internet cache, delete temporary files, and restart the system”. This has the effect of eliminating many variables caused by previous user activities. Powering off a router or similar electronic device for 30 seconds will allow onboard capacitors to discharge and clear any volatile storage.

Availability of Previous Computer States in Digital Forensics

In IDF investigators typically have access to a collected disk image, which is a portion of the data state of a computer from one point in time. The physical connections and condition of components may be directly observed or it may be inferred by analyzing the contents of the collected data. Depending on the specific computing environment there may also be system backups and relevant logs from other systems available.

Although the state of RAM, running programs, user displays, IO, and connected devices could be relevant to an investigation, they are usually not readily available in IDF. Most analysis is performed on a forensic image of a subject computer. An investigator may also have access to the physical computer itself to inspect, or at least photographs depicting how it was found and collected. Even if the physical and data state is meticulously documented during acquisition, with RAM captured, all open windows and running programs documented, and so on, that information may be of little value to answering many important investigative questions. In most instances the activity that is most relevant and led to the investigative interest in the system took place days, weeks, or even months prior. The computer's memory, storage, and configuration will have changed over that intervening time. As discussed earlier, this data did not spontaneously degrade or decay. Instead it was changed by user actions.

Some files on a system contain information that describes a previous state of that system. On a modern Windows system this may include system restore points, volume shadow copies, temporary files, hibernation files, memory swap files, and web history, among others. All of these items exist in the current data state of the system while also providing information about a prior state from some point in the past. Windows crash dumps are also a way for a prior memory state to exist on the collected storage state (Carvey, 2009, pp. 114). It is up to the investigative team to determine based on their particular needs whether or not to pull out these items and analyze them as part of a distinct past state separate from the final collected state.

Contrast this with a routine intrusion response incident where in-house or contracted security personnel are responding to a potentially compromised machine on the corporate network. Malicious software may still be resident in memory. Logs containing critical information are still present on the computer, not overwritten by months of additional activity. RAM can be captured, and in many situations a full forensic image of the storage is neither necessary nor practical. Instead a subset of files and metadata is extracted for further analysis. Data from the corporate router, firewall, and other security devices can be correlated with data extracted from the compromised machine.

Laboratory-based malware analysis provides for an even more controlled scenario (Carvey, 2009, pp. 366-369). A cloned virtual machine in a controlled environment is isolated from the host system and virtually powered on. Any external connections are simulated or carefully controlled. Tools running on the virtual machine monitor for any changes to files, registry settings, external connections, and so on. The virtual machine itself can be suspended and any point, cloned, or reverted back to its initial data state. The memory and storage can be accessed and extracted via the host system without interacting with the virtual machine running the malware. The system clock can even be manipulated.

In IDF the memory images, storage state, and configuration from particular states in the past would undoubtedly be interesting, useful, and relevant. The circumstances of investigative data acquisition make it unlikely that the computer will be found in the most relevant data state. Instead previous storage states can be inferred by analyzing files, logs, and metadata. Even if not collected directly, portions of the memory state can be inferred by analyzing swap space and temporary files as they are stored on the disk and may persist long after the memory has been overwritten.

Chapter Six – Activity Model for a Computer System

Definition and Boundaries

Creating a conceptual model for a complex system is useful for analytic purposes, as abstracting away some of the complexity allows us to focus on the more relevant aspects. I define a *user-computer system* herein for the purposes of activity analysis as the combination of a *computer* (defined previously) and a particular *user* interacting with the system. I define the *system boundary* as encompassing the user and computer, including all its subcomponents (Neelamkavil, 1994, p. 30).

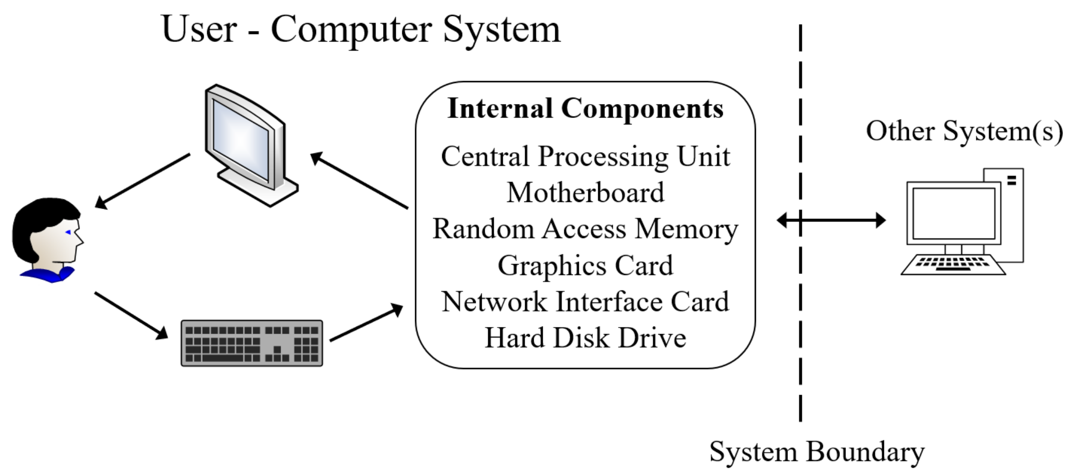


Figure 2

User-Computer System Model with Components

The *systems environment* includes all things outside the particular user-computer system that may interact with or otherwise be affected by the it. Any interactions with other computer systems, such as a mail server, router, and so on, occur within the overall systems environment. These other computer systems will have their own system boundary.

Not explicitly depicted in this diagram is the software that is running on this hardware. Software runs on the hardware, processing data and interacting with the user, system components, and other systems. These interactions are what changes the state of each component. The software can be thought of as a virtual or logical layer above the physical. The OSI reference model for protocol architectures depicts the software applications operating above the physical layers. Although this model was intended for network communications, it can be conceptually extended to communication within the directly attached system components as well.

Table 8

Rough Equivalents between OSI Model and Computer Activity Abstraction

OSI Model			Computer System
Host Layers	7	Application	Application
	6	Presentation	
	5	Session	Operating System
	4	Transport	
Media Layers	3	Network	IO / System Bus
	2	Data Link	Memory
	1	Physical	CPU

Abstraction of the User-Computer System Model for IDF Analysis

In this model I am going to focus on two specific areas that are particularly relevant in investigative digital forensics, user actions and storage contents. I will also only include computer system components that have storage which can be reasonably and reliably collected in the Investigative Digital Forensics context. As described earlier, this collected data makes up the storage state of the system at a particular time.

The memory state and other excluded items are still part of the computer system, but rather than attempt to analyze them directly I will look for their interactions with other components that we can more readily observe such as storage. This model is a simplified representation, but it is still useful to aid further analysis (Neelamkavil, 1994, p. 30). Rather than use the model to simulate the system, I will instead use it to abstract away many components to focus on ones that are both available and pertinent.

Data Degradation in TAD Collections

User actions often create records that provide an investigator with a wealth of historic and relevant information. After the initial record creation, subsequent system activities and user actions can also modify, overwrite, or completely erase these records of past actions.

Imagine a person walking along a beach, which leaves footprints in the sand. Beach activity, tides, time of day, and other factors will affect how clearly these footprints are recorded in the sand and how long they will remain on their initial condition. These steps may make fresh impressions that clearly stand out against the smooth sand of a receding tide, like TAD records written to the mostly empty hard drive from a fresh operating system installation. They may be mixed in among the existing footprints of hundreds of beachgoers on dry sand, like TAD records on a system shared by multiple users and running for years. Even footprints that are initially clear and easy to identify will become degraded over time due to wind, waves, and the actions of other beachgoers. A skilled tracker or naturalist with training, experience, and a bit of luck will still be able to identify particular tracks, age them, tell human from animal, and so on, although likely with less confidence as more time passes and the tracks continue to degrade.

Before moving on from this analogy, keep in mind that any individual TAD record does not degrade over time in the same sense as a footprint. A specific piece of data is either present on the computer system, or it is not. Fortunately, an individual user action typically causes multiple changes to the state of a computer system. The user may not have intended to cause multiple changes or even been aware of them, but most actions will cause a cascade of changes through different components of a computer system, some of which will result in changes to the data state. Changes from a single action are likely to be spread out across many different TAD collections. Records in different TAD collections with different characteristics (described in chapter four) will be affected differently as activities continue on the system. This may result in some records of an activity persisting even after additional actions occur on the system and other records have been overwritten. To circle back to the beach analogy, some of the footprints may still be visible after a period of time, allowing you to infer the path of the tracks.

User Actions Trigger an Activity Cascade

An *activity cascade* describes the observation that on a modern computer system, a single user action will tend to trigger multiple changes to the state of that system. A single user action will cause actions and state changes to cascade through the system they interacted with, and often other connected systems. These effects may be separated by both time and distance, but are all caused by the initial user action.

All activity on a computer has a human-initiated root cause. Examples of this include but are not limited to:

- A program installer executing because a user double clicked on the corresponding icon in the user interface.
- A program executing automatically as a scheduled task that was configured when the user initially installed it (e.g. scheduled to run by default, like an automatic daily virus scan).
- Operating system updates downloading and automatically installing at a pre-determined time, including system reboots.
- A user copies a file to a shared folder on a file server.
- A hostile user on a remote computer connects to a computer system, triggering remote code execution due to an unpatched operating system vulnerability, which downloads and installs a malicious payload without further user interaction.
- A hacker exploits a vulnerability to remotely deliver and execute their malware payload on a target computer system.

Automatic system updates may not initially appear to be triggered by a user, but consider that users configure their preferences and thus are responsible for causing the action to happen. Even if a user installs the operating system and leaves all settings at their default value, the user caused those settings to be in place by installing the software, even if it was only done passively. The purpose of identifying this causal relationship is not to necessarily assign blame or responsibility; rather it is to reinforce that computers do not spontaneously perform activities on their own, even if the user that caused it is far removed from the eventual action. The user who initiates the activity does not necessarily have to be the owner or authorized user of the computer, as shown by the file server and malware examples above, respectively.

Another way to conceptualize user-based causality on a computer system is through the plain English construction "but for this specific user action, that particular activity on the computer would not have happened". An activity cascade typically affects multiple components and sometimes multiple systems, causing changes to their respective data states. If some records are missing due to intentional destruction, overwritten by new data from further use, or the complete loss of the system, some of the missing records and corresponding user activity can often be inferred from the data that is available.

Example Activity Cascade on a Simple System

Consider an extremely simple computing device, a *feature phone*. A feature phone is a cellular phone that is not a smart phone. It does not run applications the way a smart phone does, rather it has a simple embedded operating system that is capable of making phone calls and sending SMS messages but not much more. A feature phone has many characteristics and components in common with a computer system as we described earlier, namely: a CPU, memory, network connectivity (cellular), a display, and a keypad.

Consider the following cascade of actions and state changes that occur when a person powers on their feature phone and makes a phone call.

Table 9

Example Activity Cascade on a Feature Cellular Phone

User Action	Phone Action(s)	Data State Changes
User powers on phone, waits for signal.	Bootstrap process reads storage and boots.	Phone stores boot log
User waits	Phone searches for cell tower, connects to network	Phone updates internal time from cell network
User waits	Phone identifies itself to cell network, registers with carrier	Cell company logs phone location via tower
User waits	Phone receives stored SMS from cell network, displays to user.	Phone records SMS in storage. Cell company logs SMS as delivered.
User dials phone number	Phone communicates with cell tower	Call attempt logged by cell provider
Receiving user answers call	Voice call begins	Call logged by calling phone, receiving phone and receiving provider

Even if a user deletes the logs on their phone, overwrites the records with new activity, or destroys it beyond recovery, records of the actions may persist on other systems that the calling phone interacted with. In this example that would be the receiving phone and at least one cellular service provider.

In this example many of the user’s actions can be inferred solely from data stored with the cell provider, without analyzing the phone itself. This data does need to be recovered in a timely manner, as third parties cannot be expected to store data indefinitely. Although a feature phone is itself a relatively simple computer system, it still has extensive interactions with other systems when performing its core functionality.

Example Activity Cascade on a Single Computer

Desktop computers are more complex devices than feature phones, therefore more significant internal state changes can be expected as a result of user actions. On a typical home computer, changes from a single user action quickly cascade throughout the data state of the system, across memory and storage, including multiple files and TAD collections. The table below describes an example activity cascade that occurs when a user double clicks an icon on their desktop to execute a program. In this example all of the activities are performed in the background by the computer after the user has triggered the initial action.

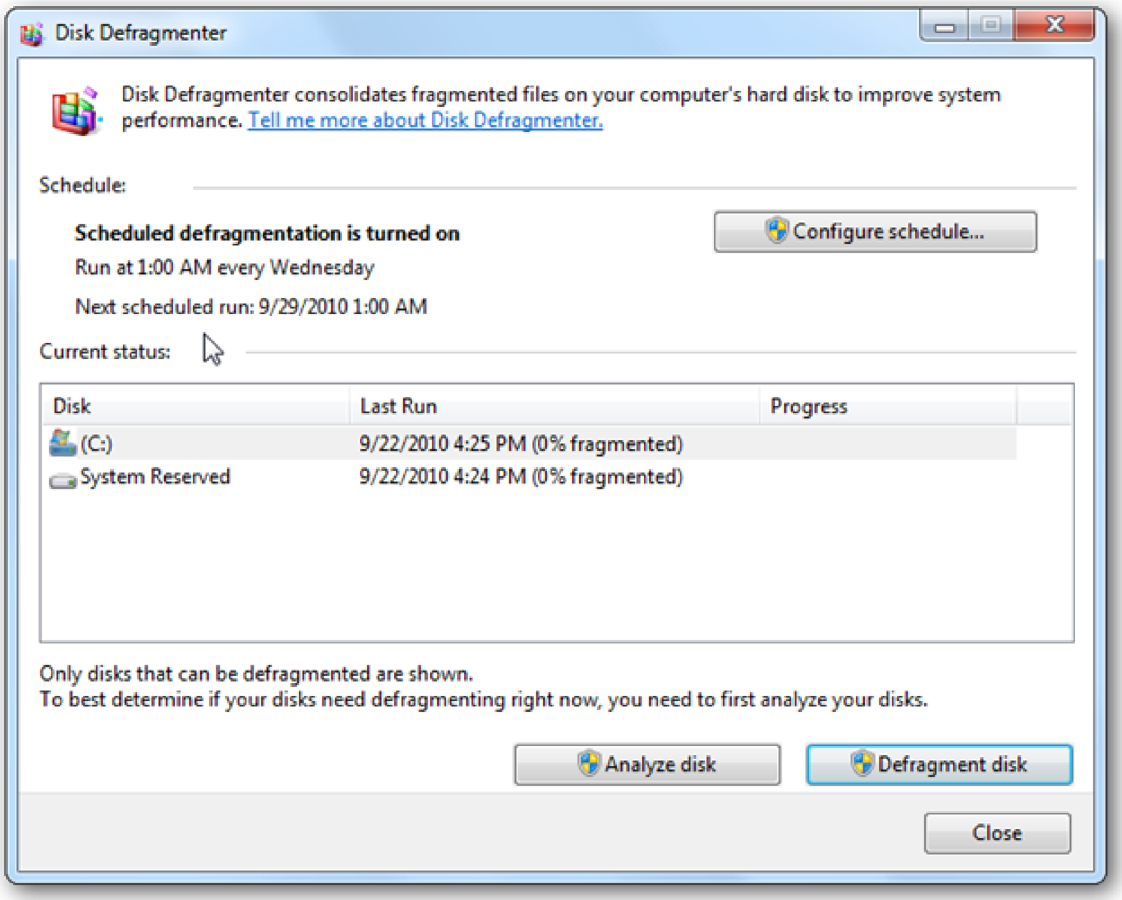
Table 10

Example Activity Cascade from Program Execution on a Desktop Computer

Computer Action(s)	Data State Changes
Operating system (OS) reads program into memory	Memory state updated, old data overwritten.
Filesystem updates from read action	Last accessed time on program file updated
OS reads prefetch, reads additional data into memory	Prefetch file read from disk, updated or created; Data blocks read into memory
OS executes program in memory	Program reads configuration files from disk into memory
Program displays splash screen to user	Video memory updated with graphics, program continues reading files into memory
Program completes loading, waits for user input	Operating system and program continues to execute background tasks

Temporal Separation Between Action and Change

It is possible for a user to cause an action to occur at some future time even without directly interacting with the system at that time. This could be achieved by explicitly configuring a scheduled task to occur at some future date. It is also possible for these future unattended actions to occur indirectly. A user could install a program with some particular default behavior specified by the developer which will trigger future actions without further user interaction. Some older versions of Microsoft Windows were pre-configured run a disk defragmentation process every 1 a.m. each Wednesday by default, as shown in the following figure.



Disk Defragmenter consolidates fragmented files on your computer's hard disk to improve system performance. Tell me more about Disk Defragmenter.

Schedule:

Scheduled defragmentation is turned on

Run at 1:00 AM every Wednesday

Next scheduled run: 9/29/2010 1:00 AM

Current status:

Disk	Last Run	Progress
(C:)	9/22/2010 4:25 PM (0% fragmented)	
System Reserved	9/22/2010 4:24 PM (0% fragmented)	

Only disks that can be defragmented are shown.
To best determine if your disks need defragmenting right now, you need to first analyze your disks.

Figure 3

Preconfigured Disk Defragmentation in Windows 7

This defragmentation process can cause significant changes to the storage state of the system. The placement of data blocks on the defragmented storage will change, although the underlying logical file content remains the same. Defragmentation on some operating systems can also update the last accessed MAC time for each file, overwriting the prior value. Although the user may not have specifically configured this process themselves, they still caused it to occur as a result of their action of installing the software. The state changes that now occur to the storage every Wednesday at 1 a.m. is attributable to the action of the user. Automated backup programs and virus scans can also cause widespread system changes, depending on the operating system configuration and specific implementation.

Intent or awareness that the user caused the change is not required. Critical investigative information is often obtained from system artifacts that the user was unaware that they created through their actions and corresponding activity cascades.

Activity cascade across multiple systems

A single user action can also set off activity cascades across multiple computers with a single action. Consider a user Alice that composes and sends an email to Bob using client software on their workstations, described in the table below.

Table 11

Example Activity Cascade Involving Multiple Systems

User Action	System Action and Data State Changes
Alice composes an email on her computer	Alice's mail client is run, updating prefetch and access times for files. Email saved in 'draft' folder within client
Alice addresses draft email to Bob and sends it	Mail client copies email to outbox folder, deletes draft
	Mail client contacts configured mail server, passes credentials and sends message to server
	Mail client moves email from outbox to sent folder, updates time stamps
	Mail server parses email, adds headers, determines destination mail server, and sends email
	Destination mail server receives email, adds headers, copies email to destination client mailbox
Bob opens email client on his computer	Mail client is run, updating prefetch and accesses
	Bob's mail client contacts configured mail server, passes credentials and receives message from server
	Bob's mail client copies message to inbox folder, displays "new mail" notification
Bob opens new mail in his email client	Bob's mail client displays message on screen, marks email as read.

In this instance the action of Alice sending the email triggered the activity cascade across multiple computer systems even though the direct changes she made were only to her system. It is worth noting that although Alice caused this specific activity cascade, many other people contributed to the effects. A system administrator in Alice's IT department installed and configured their mail server to work properly. Bob's IT department did the same. Bob provided his credentials to log into his email so that the message could be retrieved from the server and delivered onto his workstation.

Chapter Seven - Activity Cascades and Data States

As discussed previously, a computer in state n is composed of corresponding physical$_n$ and data$_n$ states. The data$_n$ state is composed of the corresponding states memory$_n$ and storage$_n$. For most of this work I will focus on the storage portion and infer the other portions as necessary.

In IDF, investigators usually only have the most recent storage state available for analysis. At the most basic level the storage state consists of all the data from all the persistent storage in the computer system, represented as a series of binary streams, one from each data source. For both convenient handling and to preserve data integrity, this data is usually acquired and handled as a raw bitstream such as DD or RAW, or placed into a containerized format such as E01 or AFF. The data is also sometimes acquired as a collection of logical files instead, owing to technical, logistical, or legal limitations.

Numerous forensic tools can take these raw bitstreams and parse them, interpret the partitions and filesystems into files and other artifacts, extract metadata, create text indexes, hash files, algorithmically recover file fragments from free space, and so on. Alternately, if logical files or a subset of files are collected directly from the system they can be analyzed or reviewed directly. Care must be taken to ensure that the collected files are not inadvertently modified during the analysis process. A subset of these extracted items is the time-activity data corpus (defined in chapter four) which consists of all the extracted time-activity data collections.

Suppose Alice sends a threatening email to a government official. Prior to this action her computer was in $state_0$. The activity cascade from her action “sending a threatening email” causes her computer to change from $state_0$ to $state_1$. At some later time, the computer is seized and forensically imaged, which is the last state where Alice’s actions could have affected it, $state_{collected}$. The next section will examine what this means from an IDF and activity analysis perspective.

The analysis in this work will focus primarily on the TAD corpus contained within each storage state. It is beyond the scope of IDF to analyze the user’s actual state pre-and post-action. Instead the focus is on the storage post-action to see what can be inferred about the pre-action storage state as well as the particular user actions which caused that transition.

Occasionally changes to the physical state of a system will come up during this work, but the analysis will focus primarily on the data state subcomponents. A physical state change could the result of adding or removing a component, such as an add-in card, inserting removable media, or connecting a peripheral. A physical change could also be as simple as powering on the computer or turning it off. Physical state changes can often be inferred from effects captured within the data state.

Component State Changes Caused by User Actions

Let there be a user|computer system in state n, depicted as $computer_n$. $Computer_n$ consists of components CPU_n, RAM_n, HDD_n (hard disk drive), and so on. The user performs an action on computer$_n$ which transforms it into computer$_{n+1}$ as shown in equation (1).

$$Computer_n \rightarrow (User\ Action) \rightarrow Computer_{n+1} \tag{1}$$

Alternately a time interval passes, with one or more user actions potentially taking place as shown in equation 2.

$$Computer_n \rightarrow (Time\ Interval) \rightarrow Computer_{n+1} \tag{2}$$

Computer$_{n+1}$ consists of physical components CPU$_{n+1}$, RAM$_{n+1}$, Hard Drive$_{n+1}$, and so on. In a particular computer system RAM$_n$ could consist of 16 Gigabytes of data reflecting the contents at 12:01am and RAM$_{n+1}$ could be the content of the same 16 gigabytes of memory at 1:01am the same day, with a one-hour time interval between n and n+1. Rather than continuing to specify each distinct component of the computer in this decomposition, they will be logically combined into the data state. The data state is then subdivided into the memory state and the storage state. When describing the data state of a system or subcomponent of that system it is actually the data stored within that is being discussed. The physical characteristics of that subcomponent are part of the physical state.

Definitions and Decomposition

The following definitions and labels will be used for the system activity model decomposition and analysis:

- Given a user-computer system Y
- Y consists of computer C and user U

- U may perform actions on C
- C consists of a physical state P and a data state D
- D consists of a memory state M and a storage state S
- Subscripts are used for C, P, D, M, and S to indicate a specific instance of a state

When decomposing components from a particular state, the sub-components retain the state designation subscript from the main item. D_1 consists of M_1 and S_1, and so on.

Mechanism of State Changes

There are two ways of describing the mechanism through which the C_n state becomes C_{n+1}: action change and temporal change. The amount of time that elapses between C_n and C_{n+1} is called the *time interval*.

An *action change* occurs when a user U interacts with a computer C in the same user|computer system Y and causes the state to change from C_{pre} to C_{post}. There is no fixed time interval for action changes, merely the requirement that all actions in the cascade complete prior to entering the new state. The user's interaction may cause one or more actions (or activity cascades), but an action change requires some interaction on the part of the user. The technique of analyzing action changes to computer states is commonly used in debugging, system troubleshooting, malware analysis, and related activities, although it is not described using the same terms used here.

A *temporal change* occurs when a period of time passes on a computer C that was previously in an identified state n. After that period of time C_n is said to be in a new state C_{n+1}. It is not necessary for any user actions to have occurred during this interval. Even if no changes were made to the physical or data states of a system during the time, they are still designated as two distinct states. As a result, it is possible for the contents of D_n and D_{n+1} to be identical.

An *adjacent state* is the state that immediately proceeds or follows a particular state, with no intervening state. Although integers are often used for simplicity to indicate adjacent states, descriptive terms such as $S_{initial}$ and $S_{collected}$ can also be used as long as their meaning and order are well documented. When analyzing action changes the two states can be thought of as the *pre-action state* and *post-action state*, or C_{pre} and C_{post}, as depicted in equation 3.

$$C_{pre} \rightarrow (User\ Action) \rightarrow C_{post} \tag{3}$$

Theoretically, there are a nearly infinite number of intermediate states that can be located between any two states, down to the granularity of CPU clock cycles. In practice, there is no issue in choosing any arbitrary interval that works with the available data. The intervals do not need to be regular; the only requirement is that they are ordered in strictly increasing sequence according to time.

The granularity of the intervals in temporal changes will be a function of the storage states available for analysis or the experimenter's ability to capture those states. In practice this will depend on the ability to collect the data, either based on the inherent characteristics of the system under analysis or the design of the experiment. The ordering and adjacency of the available individual states should be readily apparent. In some circumstances this may require a review of the contents of the states if the collection process was not sufficiently well documented.

Connected States and TAD Records

As mentioned earlier this work will focus on components where the contents are both available and relevant. In IDF this is most commonly the storage subcomponent. S_{initial} is the data contained on the storage components of a system in its initial state. $S_{\text{collected}}$ is the data contained on the storage components of a system when it is collected. The designation of "initial" may seem arbitrary in some cases, but it can be thought of as "the state prior to any user actions that are interesting or which may affect the investigation". Initial may be the earliest available state, or it may be a notional placeholder for a state prior to all the available data.

Collected in most instances is when storage is forensically imaged by appropriately trained personnel, or otherwise collected by the investigator. Working backwards from $S_{\text{collected,}}$ a series of prior states can be constructed connecting back to the initial state of the storage. This is shown in equation 4.

$$S_{initial} \longrightarrow S_1 \longrightarrow \cdots \longrightarrow S_n \longrightarrow \cdots \longrightarrow S_{Collected-1} \longrightarrow S_{Collected} \tag{4}$$

$S_{\text{Collected}}$ represents the all the binary data on the storage capable components in the computer system at the time it was collected by the investigator. It is well understood that "file systems evolve significantly with use" (Woods et al., 2011) and this a more explicit and granular depiction of that effect.

To perform activity analysis the TAD records contained within the storage are then extracted, analyzed, and reviewed using various tools. A set of TAD records is considered "stored" on a device or component when that the data is present in the storage component and can be extracted for analysis using one or more appropriate tools and techniques.

Let T be the collection of TAD records that are stored on or can otherwise be extracted from storage S. T_n is the collection of TAD records from S_n. $T_{initial}$ is transformed into $T_{Collected}$ by a series of user actions and / or time intervals and through intermediate states, shown in equation 5.

$$T_{initial} \longrightarrow T_1 \longrightarrow \cdots \longrightarrow T_n \longrightarrow \cdots \longrightarrow T_{Collected-1} \longrightarrow T_{Collected} \tag{5}$$

It is worth nothing that although T_n represents the collection of TAD records extracted from S_n, it is possible for changes to be made to S_n (transforming it into S_{n+1}) that do not directly change the TAD records T_{n+1}. TAD records consist of data extracted and parsed from metadata and logs, so if a change is made to file or disk contents without changing any metadata or TAD records on the entire system then the TAD records in the T_{n+1} state will be the same as the prior T_n state.

TAD records consist of extracted and parsed metadata, as well as data contained in logs and other similar sources. If a change is made to a file's contents without changing any TAD records on the storage, then the TAD records in the new state will be unchanged. This is unlikely in most typical user interactions with a system, but it is technically possible, and does get more likely as the time interval (or amount of user actions) between S_n and S_{n+1} decreases.

Analysis of Sequential Collections

Returning to the storage component, suppose there are two adjacent storage states S_n and S_{n+1}, separated by a temporal interval. When S_n and S_{n+1} are compared in their entirety, only two results are possible:

- $S_n = S_{n+1}$
- $S_n \neq S_{n+1}$

If two states are *identical* then all decomposable components (memory, storage, etc.) from that state are also identical. The reverse is not necessarily true, as it is possible for one component to be identical between states while other components are different.

It may seem initially simplistic, but it is important to deconstruct this. If S_n and S_{n+1} are identical, then either no changes were made to the storage during the temporal interval, or changes were made that we then changed back to their original value during the temporal interval. Although it is possible to change storage contents without affecting TAD records, it is not possible to change TAD records without also changing storage contents, as long as those records are contained within the storage component.

If S_n and S_{n+1} are different, then all possible cases can be enumerated. Since multiple actions and changes can occur during a particular temporal interval these cases are non-exclusive, i.e. one or many of these may have happened if changes are detected between the two adjacent states:

- new time records were added during the interval between S_n and S_{n+1}
- existing records in S_n were removed during the interval between S_n and S_{n+1}
- the storage contents were changed in a way that did not affect time records

At least one of these cases must be true if there are differences between two adjacent states S_n and S_{n+1}. It is also possible for multiple cases to be true. Data could be changed that created new time records, while also changing unrelated data that did not affect the time records. The storage state is represented as a fixed length string of binary 1s and 0s. Data is not added to or deleted from the storage state, rather some of the 1s were changed to 0s and vice versa as TAD records are added or removed. The binary data stream is changed so that some TAD records are no longer retrievable or new records are found. A record is considered to have been "removed" from a set if the data that represented that record is no longer present or able to be parsed from the data.

When collecting data states sequentially, such as in an application characterization experiment, it is important to ensure that the collection method itself does not introduce changes to the data. If this is not practicable then those changes must be identified and not incorrectly attributed to the user or to the activity under analysis. Care must also be taken to ensure that similar logic errors are not introduced during the analysis portion of the investigation.

When comparing two records from separate TAD collections it is essential to only compare primary metadata, which is metadata that is contained within the item being investigated. Descriptors such as date collected or image file name are secondary metadata which are applied as part of the acquisition and analysis and would cause otherwise identical records or states to be incorrectly flagged as "different" if they are considered as part of the comparison. This could occur if the "data source container" is part of the normalized TAD data record as shown in chapter three and included in the fields being compared.

TAD States and Records

A *TAD state*, T_n, is the collection of TAD records that can be extracted and parsed from the corresponding storage state S_n. T_n and T_{n+1} are the adjacent TAD states from S_n and S_{n+1}, respectively.

Consider two adjacent TAD states, T_n and T_{n+1}. For each record in either T_n and T_{n+1} one and only one of the following conditions is true:

- the record is present in both T_n and T_{n+1} (unchanged)
- the record is present in T_{n+1} but not in T_n (new)
- the record is present in T_n but not in T_{n+1} (missing)

These three subsets completely describe the interaction between T_n and T_{n+1}. To refer to these specific subsets later we will designate them as $\text{Unchanged}_{n,n+1}$, $\text{New}_{n,n+1}$, and $\text{Missing}_{n,n+1}$.

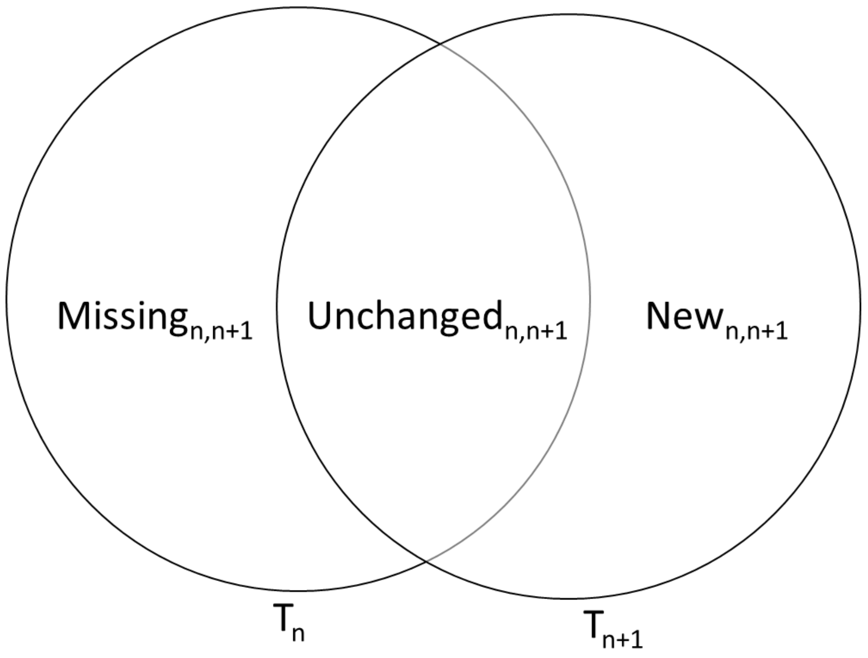


Figure 4

Diagram of Subsets when T_n and T_{n+1} Interact

It has been previously established that only user actions can change the state of the storage component. Unless both the Missing $_{n,n+1}$ and New $_{n,n+1}$ sets are empty, user actions must have occurred or resolved in the temporal interval between S_n and S_{n+1}. The inverse is not necessarily true; even if T_n and T_{n+1} have identical sets of records it is possible for an action to occur that did not change any records, or for an action to have created new records and then a subsequent action deleted those new records before the collection of the T_{n+1} state could occur.

It is possible in practice for an action to be initially triggered in an earlier time interval than the one where the causal changes are made to the storage and TAD states.

This can happen if states are captured while actions are still occurring, or if there is a significant delay between when an action is triggered and when it resolves. This can be handled by assessing the action to have occurred when the activity cascade resolved, rather than at the moment the user performed the triggering action. In practice this serves as a reminder to carefully assess all available data before drawing conclusions about the actual incidence time for user activities. This also means that a single user action can cause changes across multiple future states as the activity cascade continues to resolve.

Decomposing TAD States into Subsets

Suppose a researcher has three sequential storage states, $S_{1,}$ $S_{2,}$ and S_3. Each was collected from a single computer as part of a forensic experiment at the end of the first, second, and third days, respectively. During each temporal interval a student assistant performed a series of actions on the computer according to an assigned script from the lead researcher. At the end of each day the computer was shut down and forensically imaged. The disk images were processed and three sequential TAD states were extracted for analysis, $T_{1,}$ $T_{2,}$ and T_3.

Beginning with T_1, each TAD state is decomposed into three subsets by comparing the record in that state (T_1) with the records in the next adjacent state (T_2). This subdivides the records into distinct sets as unchanged, new, or missing for each comparison of adjacent states. After comparing T_1 and T_2, the same comparison is performed between T_2 and T_3. Referring back to the Venn diagram and the three subsets, the three adjacent states are decomposed as shown in the table below.

Table 12

Decomposition of TAD states into shared subsets

	Contribution from TAD State		
Subset Type	T_1	T_2	T_3
Base Set	T_1	T_1	T_2
New Set	n/a	New_{12}	New_{23}
Missing Set	n/a	$Missing_{12}$	$Missing_{23}$

Each successive TAD collection contains the records that are unchanged from the prior collection, adding the records that were newly created during the interval, and removing the records that were missing when compared to the prior set. The first state (T_1) contains all the records from the initial configuration of the computer prior to the experiment, plus any records created during the first day, minus any records from the initial configuration that are missing due to activity on the first day. Without information from a prior state T_1 cannot be decomposed any further.

When T_2 is decomposed with respect to T_1, the three subsets are the unchanged records from T_1 plus the new records from T_2 that are not in T_1 (New_{12}), and subtracting records that are present in T_1 that are not found in T_2 ($Missing_{12}$). Continuing this decomposition into T_3 and beyond reveals a method to represent any future TAD state as a combination of the earliest available state and all of the changes that took place during each temporal interval. The T_2 base component of T_3 can now be further decomposed, showing that T_3 can be represented as T_1 with the New_{12}, New_{23}, $Missing_{12}$, and $Missing_{23}$ sets representing the changes that happened during each temporal interval.

If an existing TAD record is modified by user actions, it would be considered both “deleted” as the prior record is missing and “created” as a new distinct record exists. Changes to a particular file’s metadata could be the result of editing the file during the interval, or the prior file could have been deleted and a new file with the same name could have been created and stored in the same location.   A modified item cannot be positively identified solely through set analysis, rather it requires further content analysis and inference.

Order of Operations when Combining TAD Sets

The previous example demonstrates that a large number of subsets are identified from the decomposition of just a few adjacent TAD state comparisons. In a slight departure from basic set operations, the $Missing_{n,n+1}$ sets can be thought of as a negative set in that it is removed from the previous set. In traditional set theory that is accomplished by the intersection of the complement of the set, but with multiple sets being removed it quickly gets messy and awkward. Fortunately for the field of IDF, these comparisons are fairly straight forward to execute using modern database software as demonstrated in chapter eight. The next example populates these sets with example records and repeats the previous decomposition to validate the logic and identify any pitfalls regarding the order of operations If the set operations are not ordered properly there may be spurious records included in the sets after decomposition that should not be present or elements excluded where they should not be.

The previous decomposition of T_3 offers several possible orders of operations when the re-composed later states are generated from the first state and the new and missing subsets. Consider the following example of adjacent TAD states which contain the following elements (TAD records):

- T_1 contains elements A,B,C, and D
- T_2 contains elements A,B,E, and F
- T_3 contains elements A,C,E, and G

The adjacent state decomposition logic yields:

- T_2 is composed of $T_1 + New_{12} – Missing_{12}$
- T_3 is composed of $T_2 + New_{23} – Missing_{23}$

The full expansion of T_3 yields:

- T_3 is composed of $T_1 + New_{12} – Missing_{12} + New_{23} – Missing_{23}$

Directly performing adjacency comparisons between T_1, T_2, and T_3 shows that:

- New_{12} contains elements E and F
- New_{23} contains elements C and G
- $Missing_{12}$ contains elements C and D
- $Missing_{23}$ contains elements B and F

Decomposing T_3 and appending the contents of each subset yields equation 6.

$$T_3 = T_1(ABCD) + New_{12}(EF) + New_{23}(CG) - Missing_{12}(CD) – Missing_{23}(BF) \qquad (6)$$

The order of operations becomes very important, as element C is part of T_1, New_{23}, and $Missing_{23}$. Element F is part of New_{12} and $Missing_{23}$. Normally addition and subtraction are transitive but they behave somewhat differently in set theory. To identify the proper application, we will enumerate several related sets and observe the results of follow four different ordering options:

1. Additions first, lowest to highest by set number
2. Subtractions first, lowest to highest by set number
3. Addition then subtraction by paired set number, from lowest to highest
4. Subtraction then addition by paired set number, from lowest to highest

When the sets from the example are combined according to the different ordering options to generate T_3 from the decomposed sets, the problem becomes obvious:

1. Performing addition first yields T_3 = A, E, G.
2. Performing subtraction first yields T_3 = A, C, E, F, G.
3. Addition first by temporal pairs yields T_3 = A, C, E, G.
4. Subtraction first by temporal pairs yields T_3 = A, C, E, G.

Note that options 3 and 4 yield the correct result for T_3. Option 1 is missing records, and option 2 has extra records that should not be included. This shows that addition and subtraction of sets are transitive in this case, provided we process them in pairs from oldest to newest. Written in the order to be executed from left to right, the correct equation to reconstruct or deconstruct T_3 is shown in equation 7.

$$T_3 = T_1 + New_{12} - Missing_{12} + New_{23} - Missing_{23} \tag{7}$$

Any particular $New_{n,n+1}$ can be exchanged in execution order with the corresponding $Missining_{n,n+1}$ only. The corresponding $New_{n,n+1}$ and $Missing_{n,n+1}$ are transitive because they are each mutually exclusive subsets of the union of T_n and T_{n+1}, because a single record cannot both be missing and new within the same set T_n. These sets can be described as *temporal pairs* in that they are the change sets that derive from a single state transition.

Recurrence of TAD Records

On a typical system it would be unusual for a missing TAD record to re-appear in a later state if that record was from an actual or functional log, if for no other reason than a normally occurring event would be recorded with the current time. In an assembled log there are several common actions that would cause an otherwise identical TAD record to recur in a later state. A file could be removed from the system and then re-introduced. Many file operations preserve some if not all of the existing file metadata (Garfinkel, Nelson & Young, 2012). Alternately a different file with the same name in the same location could appear to be a recurring TAD record with a changed time. Default filenames and locations for many programs as well as temporary working files make this occurrence more common than it would otherwise initially appear.

For a granular example of how basic file system actions cause records to be added and deleted, consider the following example. Suppose in T_1 there are TAD records for to the MAC times for a Microsoft Word document. These are three distinct records, even though they describe the same file. During the T_1-T_2 temporal interval the user edits the document and saves the changes to storage. This leaves the created timestamp unchanged, but updates the accessed and modified timestamps to the current system time. When the changes that occurred in the T_1-T_2 interval are analyzed, the original created record will still be present, the original modified and accessed records from T_1 will be missing, and the updated modified and accessed times will show up as new records.

Consider an alternate scenario where the file is not edited at all, but simply renamed through the operating system. On a default Windows 10 configuration this does not update any of the time stamps, so the three MAC times of the original file will be missing, and all three MAC times for the same file with the new name will be new.

This is not a bad or incorrect result. It indicates that something happened to that particular file during the T_1-T_2 interval. It may not be known exactly when the file was renamed, only that it occurred within that particular temporal interval, and the fact that the new records have a timestamp from outside the interval may also be a useful indicator, as the created record for that "new" file will be prior to the T_1-T_2 interval.

Decomposition of TAD States using an Initial State

In most investigations where activity analysis is important, the initial system state contains a numerous TAD records that is not particularly pertinent, such as dates from system files and default configuration settings. Forensic programs will generally attempt to exclude known operating system files from further processing based on hash values and a comparison with National Institute of Standards and Technology (NIST) or developer-specific hash lists, but there is not currently a tool-level equivalent capability specifically for TAD records.

Suppose that the in the previous example the student assistant who set up the computer for the experiment kept a copy of the initial image so that the experiment could be repeated. This image will be designated S_0 with corresponding T_0 and is deconstructed into subsets as shown in the table below.

Table 13

Decomposition of TAD ***S****tates into* ***S****hared* ***S****ubsets with Initial State*

	TAD Subsets			
	T_0	T_1	T_2	T_3
Unchanged Records	T_0	T_0 - $Missing_{01}$	T_0 - $Missing_{01}$ - $Missing_{02}$	T_0 - $Missing_{01}$ - $Missing_{02}$ - $Missing_{03}$
New Records		New_{01}	New_{01} New_{12}	New_{01} New_{12} New_{23}
Missing Records		$Missing_{01}$	$Missing_{01}$ $Missing_{12}$	$Missing_{01}$ $Missing_{12}$ $Missing_{23}$

The T_0 (initial state) component is a contributor, at least partially, to all the successive TAD states shown above. It is only partially contributing because some elements of the T_0 state might have been deleted in later states. These deleted records would be identified in one of the $Missing_{n,n+1}$ sets. As the focus of this work is the user actions and their effects on TAD records, T_0 data can be safely excluded from the activity analysis. The initial state typically includes the initial operating system installation, program configuration, and the like. The contents of the files in the corresponding S_0 state are still available for review during the analysis if necessary, and any changes to these initial TAD records will be identified in the $New_{n,n+1}$ subset.

Logically, ignoring the initial TAD records should make sense, as items that pre-date the period of investigative interest and do not correspond to any activity during the lifespan of the acquired data are unlikely to shed any insight into actions on the system during that time. In most investigations the initial set (T_0) can be removed from all the subsequent TAD states prior to further activity analysis. This can greatly decrease the amount of records that need more detailed analysis and review. These set operations can be accomplished with basic database commands.

Chapter Eight – Processing the M57-Patents Dataset

This chapter details the M57-Patents dataset and the steps needed to process it and prepare the data for analysis. The characterization of the activity data contained within the dataset as the qualitative analysis of the record lifespan and data degradation follows in chapter nine.

The Digital Corpora Project and M57-Patents Dataset

When conducting forensic research, one of the recurring challenges is obtaining realistic and representative data. There are privacy and human studies restrictions on actual user data in the classroom or for research, and actual investigative data or unknown sources may have issues with contraband and other dangerous materials. Self-created datasets run the risk of not being representative of real-world data and have their own problems with sharing and reproducibility.

Substantial headway in this area has been made through the Digital Corpora project. The project contains representative sets of particular file types such as documents and pictures, as well as exemplar forensic disk images and complete investigative scenarios. (Garfinkel, Farrella, Roussev & Dinolt, 2009). These datasets are commonly used in forensic education, training, and tool testing.

One robust Digital Corpora scenario is M57-Patents, which involves a notional patent search company "m57.biz" with four employee personas. The role for each employee persona was either performed directly by a researcher or by scripts that performed actions for that persona. User actions on the systems were both nefarious and innocent. The computer data and user actions in the scenario “were designed to closely replicate many of the properties of real-world data” (Woods et al., 2011). The timeline of the creation is also relevant:

> The M57-Patents scenario took place within a 17-day period between November 16th and December 11th 2009. Within the scenario, a workday started at 9:00am and ended at 4:00pm. Each day was marked in the timeline by a small number of primary objectives to be completed by research assistants playing the company personas. In addition to these objectives, each persona performed some normal background activity–web browsing, emailing friends and co-workers, patent searches, and writing word-processing documents.

The complete M57 Patents dataset contains 79 images from five different computer systems, four USB drive images, 67 memory images, 49 network captures, fictious investigative materials, and other classroom aids. This chapter and the next contains the application of differential activity analysis techniques described previously to three sets of computer images from the M57-Patents dataset. The images from computer systems are captured as a sequence of daily images from each scenario computer system, with an initial and final image as well. The images incorporated realistic “wear patterns and depth” and contain “realistic background data” (Woods et al., 2011).

There is an overarching investigative scenario for the M-57 Patents dataset with multiple distinct areas of investigative interest. Criminal and otherwise suspicious behavior by the personas is captured in the data. In order to create a more realistic scenario and challenging analysis for students the noteworthy activities and files are comingled with a significant amount of innocuous or otherwise normal activity and data. By design, the disk images are free from personally identifiable information (PII), copyrighted materials, are unencumbered by human studies restrictions, and can be widely distributed for further in-depth analysis, which is rarely the case for real world forensic data (Garfinkel, Farrella, Roussev & Dinolt, 2009). By creating a realistic corpus sufficient for education and tool validation, it can also be used for forensic research (Woods et al., 2011).

As discussed in chapter two, it is functionally impossible to separate evidence related to nefarious activity from innocent activity without analyzing the content and fully understanding its context within the larger investigation. It is also not unreasonable to assume that stored TADs will be naturally changed through routine user activities at a comparable rate regardless of whether the records are innocent or incriminating. User actions that attempt to obfuscate incriminating actions are likely to trigger their own activity cascade, leaving additional incriminating information behind even though the targeted records may be erased.

To focus primarily on the effects of user actions on TAD record degradation and decrease any variation caused by different operating system implementations, the three systems selected for analysis from the M57-Patents scenario are all Windows XP based.

There will still be variation caused by the "choices" the user personas made and variation from the applications used. Additionally, although it will not be specifically analyzed here, I expect that TAD records relating to nefarious actions will degrade at roughly the same likelihood as their innocent counterparts of a similar characteristics. This remains an area for further research.

Differences from Traditional IDF Processing and Analysis

In a traditional forensic examination for M-57 Patents scenario, after processing the raw data with one or more forensic tools the investigative team (or students) would search, filter, and review the content to find the relevant investigative information among the chaff of irrelevant or innocuous data. This is distinct from my analysis in this work. For the purposes of characterizing TAD sources and determining the effective lifespan of activity data on a representative system, the relevance of any particular item to a criminal investigation is irrelevant. When processing the items from the M57-Patents dataset I will generally ignore any relevance to the investigative scenario and treat all extracted TAD records equally.

Restrictions in Publishing M57 Answers

I have obtained and reviewed the instructor solutions for the scenario which goes into significant detail on the ground truth for the scenario and subject actions, as well as detailing how the images were created. As part of the agreement to access the solutions I cannot publish these details but have reviewed them. I am confident that the forensic images from the scenario provide a representative baseline to validate the analysis techniques and provide meaningful information regarding the degradation of time-activity data as a system continues to be used, and to investigate differences in these effects based on the type of TAD record source. As described in the scenario materials, the disk images "are big enough to be realistic" and small enough to be processed in a reasonable amount of time.

To preserve their utility as educational tools, I strictly limit any discussion of items related to the actual investigative scenario, nor will I be publishing any of the actual content from the processed images. The solutions were reviewed after processing was complete to validate the relative ordering of the disk images and to verify that the activities in the drive images were reasonable and representative of user activities.

Quantitative Analysis of TAD Records

The techniques that I developed to identify changes in the TAD collections extracted from sequential storage states generally do not require a manual review of the content of the TAD records, nor a review of the content of the underlying files they describe. This is both fortunate and necessary, as the smallest TAD collection extracted from a single image in the set contains nearly one million TAD records, and many contain two to three million TADs. Most of the new and missing subsets generated from differential activity analysis contain over one hundred thousand TAD records. I will use quantitative data analysis techniques to reach conclusions from the processed data.

One note regarding nomenclature before proceeding. It should be clear from the system activity model that a user is not their computer. When a system is described as "Jo's computer", it is meant as shorthand for "the computer that is owned, primarily used by, or is otherwise associated with Jo." For the remainder of this work, whenever I refer to Jo, Charlie, Pat, or any of their characteristics, I am referring to their computers, the corresponding acquired data states, or the extracted collections of TAD records.

Data to be Analyzed

From the full M57-Patents corpus, I selected three image series corresponding to three different users and at least three different computers. The filenames are listed in the following table. All of the files listed are E01 files, which is a compressed image format common to digital forensics.

Table 14

List of M57-Patents Images to be Processed

Charlie	Pat	Jo
charlie-2009-11-12start	pat-2009-11-12start	jo-2009-11-12start
charlie-2009-11-12	pat-2009-11-12	jo-2009-11-12
charlie-2009-11-16	pat-2009-11-16	jo-2009-11-16
charlie-2009-11-17	pat-2009-11-17	jo-2009-11-17
charlie-2009-11-18	pat-2009-11-18	jo-2009-11-18
charlie-2009-11-19	pat-2009-11-19	jo-2009-11-19
charlie-2009-11-20	pat-2009-11-20	jo-2009-11-20-oldComputer
charlie-2009-11-23	pat-2009-11-23	jo-2009-11-20-newComputer
charlie-2009-11-24	pat-2009-11-24	jo-2009-11-23
charlie-2009-11-30	pat-2009-11-30	jo-2009-11-24
charlie-2009-12-01	pat-2009-12-01	jo-2009-11-30
charlie-2009-12-02	pat-2009-12-02	jo-2009-12-01
charlie-2009-12-03	pat-2009-12-03	jo-2009-12-02
charlie-2009-12-04	pat-2009-12-04	jo-2009-12-03
charlie-2009-12-07	pat-2009-12-07	jo-2009-12-04
charlie-2009-12-08	pat-2009-12-08	jo-2009-12-07
charlie-2009-12-09	pat-2009-12-09	jo-2009-12-08
charlie-2009-12-10	pat-2009-12-10	jo-2009-12-09
charlie-2009-12-11	pat-2009-12-11	jo-2009-12-10
		jo-2009-12-11-001
		jo-2009-12-11-002

From the public scenario information, it is known that:

> Friday, 13 November has no images, because the scenario did not officially start until the following Monday (16 November). Your data may contain drive images from Thursday, 12 November. These are for reference (e.g. prior to any employee activity). Friday, 20 December has images for two separate drives for Jo…

Based on this information, observing the file naming conventions, and looking at the Charlie image listing, we can identify that the Charlie-2009-11-12start image corresponds to $\text{Charlie}_{\text{Initial}}$. The other image from November 12th, prior to the start of the scenario, is Charlie_1. This convention continues according to the recorded date until the last image in the series, Charlie_{18}. The same process is applied to the Pat image series.

The Jo image series presents some potential problems. There are two images from November 20th, one designated “newComputer” and the other “oldComputer”. Based solely on this secondary metadata (see chapter three) it is reasonable to infer from that the computer that user Jo began using a new computer that day, either as part of the scenario or due to equipment failure. The published scenario information confirms this (Woods et al., 2011). In many situations an investigator could interview a system administrator or examine the content on the images before and after this transition to confirm this theory.

As discussed in chapter six, it is up to the investigator’s discretion whether two distinct systems used by the same person should be analyzed together or independently. If there was not a hint in the file names there would be no reason to separate these image series into two sets. As this work is attempting to draw conclusions based solely on the TAD corpus, the Jo series of images will be processed two theoretical arrangements.

One scenario will assume that there are two distinct computers for Jo. Jo-2009-11-12start will be the $Jo1_{Initial}$ state and six following daily images including the "old computer" image will be the $Jo1_1$ through $Jo1_6$ adjacent states. The remaining fourteen images will be processed separately as a distinct set Jo2. The "newComputer" image will be $Jo2_{Initial}$, followed by $Jo2_1$ through $Jo2_{13}$. There are two separate images for Jo on the last day of the scenario, so both will be included as $Jo2_{12}$ and $Jo2_{13}$, respectively.

The second scenario ignores the possibility of a replacement computer for Jo and treats the entire series as a coming from a single computer system, designated as JoC for "Jo combined". I suspect that the correct interpretation of the Jo images will be apparent solely from looking at the pattern of surviving TAD records in this series.

Starting with the adjacent state deconstruction detailed in the previous chapters, I conducted a census of the TAD records in each state, characterized them, calculated the lifespan for each record, and analyzed the overall survival rates of TAD records. This analysis was done both across the entire TAD corpus and also across subsets based on source type and other TAD collection characteristics.

Inventory of Data

The scenario began on November 16th, which corresponds to the $Charlie_2$, Pat_2, $Jo1_2$, and JoC_2 states. The adjacent state comparisons to identify the new and missing subsets began by comparing $Charlie_1$ and $Charlie_2$ to determine $Charlie\text{-}New_{1,2}$, $Charlie\text{-}Missing_{1,2}$, and so on for each series. For a plain English description of these subsets, remember that Charlie-New$_{1,2}$ contains the records that are present in Charlie$_2$ but were not present in Charlie$_1$, and Charlie-Missing$_{1,2}$ contains the records that were present in Charlie$_1$ but were missing in Charlie$_2$.

The processed data consists of 59 individual disk images totaling approximately 243 GB compressed, 697 GB uncompressed. Although images from the Jo series are present in multiple logical collections, for the purpose of this inventory each image is only counted once here.

Data Processing Steps

The following steps were used to process the data:

1. TAD data was extracted from each source disk image.
2. The name of each output file was normalized to facilitate automated processing and clearly identify the temporal adjacency within the set.
3. Additional copies of the Jo output were made to establish distinct sets of output files for both the Jo1 / Jo2 (two computers) and JoC (combined) scenarios.
4. Each series of database files was combined into a single file with individual tables for each TAD state.
5. Database fields that were blank across the entire corpus were removed.
6. A unique hash value was generated for each record in each database and table.
7. The records in each series' initial image were removed from the successive TAD collections, leaving only the records associated with user actions.
8. For each adjacent pair of TAD states, the items for the corresponding New and Missing subsets were identified.

9. Using the New and Missing subsets, the lifespan for each record was calculated
10. The results were collated and analyzed (see chapter 9).

Each step was automated as much as practicable using Python scripting. Most steps were well suited to batch processing using SQLite to provide database functionality. Although the overall volume of data and records was large, the batch processing methodology was able to handle all the data in a reasonable amount of time on a robust modern workstation. This eliminated the need to build, license, and maintain a separate database server, as well as avoiding the challenges of parallel processing. The initial data extraction step was primarily CPU-bound, while the remaining steps were primarily IO bound.

TAD Data Extraction

As described in chapter two, Plaso and its associated sub-programs are an effective and widely used open source tool for timeline data extraction and processing. Plaso also has robust capabilities to search and filter the output interactively, which can be useful in a traditional IDF scenario or incident response. In normal forensic analysis the volume of TADs extracted from a source would be impossible to manually review without extensive filtering. For the intended analysis in this work, all records need to be exported in order to perform the differential activity analysis and qualitative analysis of the TADs.

Each image was processed using Plaso / Psteal version 20190708 using default (maximal) options and output was saved in the “4n6time SQLite” format. 4n6time is a now-deprecated timeline analysis tool, but this was the only option to directly generate a

SQLite output database without converting through an intermediate format. This output format did add several fields to each database which would have been utilized for by the 4n6time tool to store secondary metadata applied by the reviewer. These fields were identified and were removed during the data reduction step.

Output File Normalization and Merging

Before proceeding to the next step, the SQLite files were manually renamed to the format [name]-initial and [name]-Snap## to facilitate the scripted batch processing. As discussed earlier the exact temporal spacing between each image is not critical as long as they are placed in strictly increasing order. Most of the images have an interval of one workday between them, as the longer intervals correspond to weekends during the scenario period. The Jo image set has two images from the last day of the scenario, which will be placed in numeric order.

After processing and renaming the files to the standardized format there was 269GB of SQLite output across 79 files, plus and additional 25GB of duplicated data across 21 files in the Jo-Combined set. A master SQLite file was created for each set, with the data from each TAD collection in a separate table. Across the entire corpus there were over 129 million TAD records to be analyzed.

Table 15

Collation and Renaming of Jo TAD Collections for Processing

Source File	Jo1 Set	Jo2 Set	JoC Set
jo-2009-11-12start	jo1-initial		joc-initial
jo-2009-11-12	jo1-snap01		joc-snap01
jo-2009-11-16	jo1-snap02		joc-snap02
jo-2009-11-17	jo1-snap03		joc-snap03
jo-2009-11-18	jo1-snap04		joc-snap04
jo-2009-11-19	jo1-snap05		joc-snap05
jo-2009-11-20-oldComputer	jo1-snap06		joc-snap06
jo-2009-11-20-newComputer		jo2-initial	joc-snap07
jo-2009-11-23		jo2-snap01	joc-snap08
jo-2009-11-24		jo2-snap02	joc-snap09
jo-2009-11-30		jo2-snap03	joc-snap10
jo-2009-12-01		jo2-snap04	joc-snap11
jo-2009-12-02		jo2-snap05	joc-snap12
jo-2009-12-03		jo2-snap06	joc-snap13
jo-2009-12-04		jo2-snap07	joc-snap14
jo-2009-12-07		jo2-snap08	joc-snap15
jo-2009-12-08		jo2-snap09	joc-snap16
jo-2009-12-09		jo2-snap10	joc-snap17
jo-2009-12-10		jo2-snap11	joc-snap18
jo-2009-12-11-001		jo2-snap12	joc-snap19
jo-2009-12-11-002		jo2-snap13	joc-snap20

Data Reduction and Hashing

Using a script that determined the number of unique values for each field in each table and across the full corpus, I identified several fields that were entirely blank as well as other that had a single value for all fields. The notes, reportnotes, inreport, tag, and evidence fields were artifacts of the 4n6time output format and were removed. These reduced datasets were stored in a single file for each image set, using the same naming convention previously used, adding the master reduced (MR) designation.

One particular challenge to matching records individual between different states is that there is not a common index or other built-in method of identifying the same record across multiple sets without comparing each individual field. When repeated across millions of records per set this lookup process took too long to be practicable in scale. Borrowing the idea of a hash table from computer science, I combined each field in a record into a single monolithic bitstream and then calculated the SHA1 value for that bitstream, which was then added as a new field for that record. As long as the same fields were combined in the same order for each record across different sources, the hash values will match for identical records. After making the hash field an index in the database, performing the record set comparison between two adjacent TAD sets could be completed in a reasonable amount of time.

Even with automation and batch processing, efficiencies in the logical analysis process and data management were necessary in order to complete the analysis in a reasonable amount of time.

Removal of Initial State

As shown in chapter six, the initial state in a linked series of adjacent TAD states can be removed from later states, leaving only the TAD records caused by user actions and the associated activity cascades. A second advantage of removing the initial state from later states is that it leaves many fewer records that need to be analyzed. The initial state for each TAD series was removed from each TAD state using SQL “Select Except” commands automated with scripting. Removing the initial state from the later states removed up to 90% of the records in some states. These reduced datasets were stored in a single file for each image set, using the same naming convention previously used, adding the less initial (LI) designation.

Decomposition of Remaining Records into New and Missing Subsets

For each image series other than Jo2, the first day of scenario activity is represented by the Snap02 set. The TAD record changes caused by user actions on the first day will be found in the $New_{1,2}$ subset for new records and the $Missing_{1,2}$ subset for missing records. A useful memory aid for the contents of a particular subset $New_{A,A+1}$ is “$New_{A,A+1}$ contains all the records in set A+1 that were not present in set A.” The plain English description of $Missing_{A,A+1}$ is “$Missing_{A,A+1}$ contains all the records from set A that were not present in set A+1.” Since the Jo2 series began activities immediately without an intervening idle day that we know about before the scenario began, the first state after the initial one is the first scenario day with activities for that system.

Leveraging a somewhat more complex SQL “Select Except” construction, the previously generated “master reduced less initial” (MR-LI) daily sets were decomposed into new and missing subsets by comparing them with the next adjacent daily set, starting with the first snapshot. These decomposed datasets were stored in a single file for each image set, using the same naming convention previously used, adding the $Snap_{A,A+1}$ designation.

Calculation of Lifespan for Each Record

The lifespan for a given TAD record in this scenario is the number of future states that the particular record was still present in. As the analysis is limited to only the records that were created by user actions after the start of the scenario, it is possible to determine the lifespan for any record by examining the contents of each successive $\text{Missing}_{A,A+1}$ subset. The following logical construction establishes the basis for this calculation.

Given a record in a $\text{New}_{A,A+1}$ set, the lifespan of the record can be determined by incrementally searching through each adjacent $\text{Missing}_{B,B+1}$ set, going forward in time. Stop when either the record is found in a $\text{Missing}_{B,B+1}$ set or all available sets have been searched. The lifespan is the number of Missing sets between the $\text{New}_{A,A+1}$ set that contains the record and the $\text{Missing}_{B,B+1}$ set that contains the record. If the record is not found in any following $\text{Missing}_{B,B+1}$ sets, then the lifespan is the number of sets that were searched. The shortest possible lifespan for a record is zero, which happens when a record in $\text{New}_{A,A+1}$ set is also in the adjacent $\text{Missing}_{A+1,A+2}$ set. The maximum possible lifespan in this experiment is equal to the number of states after the first state containing the record. A Boolean field was used to track whether a record survived to the end of the available data in addition to the numeric lifespan value.

If a record was contained in a $\text{New}_{A,A+1}$ set, that means the record was contained within the snapshot for A+1 but not in A. By first removing unchanged records from the search space, searching through the New and Missing subsets for a particular record becomes roughly analogous to looking through the transaction logs for that record. If the record has not been removed, then it can be inferred that the record is still present in that state. Stopping the search as soon as a particular record is found in a Missing subset effectively short circuits the search process, preventing the algorithm from continuing to search through up to a dozen additional subsets with hundreds of thousands of records each when it is already known that the particular record was no longer be present on the system. In the edge case where that record reappears unchanged in a later state, there will be a corresponding entry in the $\text{New}_{C,C+1}$ set. Searching for an individual record is also accelerated by the presence of the unique hash for each record in the database. After the lifespan for each record was calculated, the record was copied to a corresponding output table in a single database file for each image set, using the same naming convention as before and adding the survival analysis designation. Each record had the calculated lifespan recorded as an additional integer field, and a Boolean field recorded if the particular record survived through the last image in that particular series.

When using complex select statements to perform set logic with multiple tables, the inherent optimizations of modern database systems and indexed fields were able to compute subsets of tables in a reasonable amount of time without needing to leverage specialized acceleration techniques. It was only when searching for each individual record across a dozen different TAD collections with hundreds of thousands of records each that it became essential to use the unique hash for each record as the identifier to search for. This optimization was necessary in order to compute the lifespan for each record in a reasonable amount of time.

Conclusion of Processing Phase

At this point a series of sequential disk images from three (or four) separate computers had been processed. All of the TAD records contained within have been extracted, including metadata fields that characterize the source and type of each record. The lifespan of each record within the dataset has been calculated and individually recorded. The next chapter contains the quantitative analysis, characterization of specific TAD sources, and conclusions about the degradation of time-activity data in the M57-Patents dataset. The mystery of how to properly group the Jo images will also be resolved.

Chapter Nine – Data Degradation and Lifespan Analysis of M57-Patents

In the context of investigative digital forensics, data does not “degrade” the way that physical items decay over time or radioactive isotopes decay into other elements (Garfinkel, 2012). Rather data from a previous storage state S_n is changed so that the data is not present or otherwise recoverable on S_{n+1}. Data from the previous state is replaced by new data. As time goes on and the system continues to be used, more data from previous states will continue to be overwritten by new data.

Data destruction is distinct from data degradation. *Data destruction* is the intentional act of rendering some or all of the data on a system unavailable to the investigator, whether through physical destruction, logical erasure, or the like. *Data degradation* occurs when a user or system, performing regular activities and processes, causes previously existing records to be changed, erased, or otherwise becomes unrecoverable. In the context of TAD records, data degradation occurs when existing records are overwritten with new data or when the file containing the records is removed from the system. These records could be log entries, log-like content, or pieces of file metadata within the storage state.

Analysis of data degradation in the IDF context attempts to understand how long potentially relevant information from the past can be expected to survive under normal system activity. Fragility is a TAD source characteristic that describes how easily a particular TAD record can be changed through regular system usage. Analyzing the TAD record loss over time is a first attempt at characterizing and quantifying this loss.

As described in chapter five, a computer is a deterministic device. In order for the data on a computer to change it must be affected by an external actor. A single action can cause multiple distinct state changes through an *activity cascade*, described in chapter six. In most IDF circumstances the event(s) of investigative interest occurred at some point in the past, prior to the collection of data. Any changes that are introduced during this interval can affect the ability to recover the relevant data from that system.

Fragility Revisited

File content placed on a system through user actions generally does not change without additional direct and intentional user interaction. A recorded video or downloaded file will remain where the user placed it until they interact with it again. System files, configuration data, and similar items will tend to change with more frequency as applications make records for user convenience or performance optimization and operating system patches are downloaded and applied in the background. These actions are recorded in prefetch files and lists of recently opened documents, among other sources. These records are metadata, and thus will have varying fragility based on how, where, and why the record is stored.

For an example of actual differences in fragility (defined in chapter three), consider the MAC (modified, accessed, and created) times and other metadata for a Microsoft Word document on a Windows 10 system. The MAC times are stored in the filesystem structures, external to the file, while other metadata is stored within the file itself. The created time is normally set when the file is first created, although a file downloaded from the Internet will typically have the created time set to when the copy of the file was received on the local computer. Making a new copy of the document under a different file name through either the operating system or the application “File | Save As” menu option will result in an updated creation date while preserving the modified date from the original file. This results in a file that appears to have been modified before it was created, at least under a naive interpretation. Both methods to make a new copy of the file will preserve the internal “last printed” metadata record. Interestingly the “total editing time” value stored in the document metadata is cleared if the application menu option was used to create the new file. Using the operating system commands to make a new copy of the file preserves the “total editing time” metadata as it is internal metadata stored within the file content itself. This was likely an intentional design choice by the application authors. This demonstrates how a file could appear to have been modified or even printed before it was created (Rowe & Garfinkel, 2010).

This microcosm of forensic analysis shows that actions that seem similar from an end user perspective can have different results based on the specific actions performed and the implementation within the application and operating system. It is a good practice whenever a key investigative finding depends on the interpretation of a few pieces of metadata to experimentally verify the investigative hypothesis on a controlled exemplar. This can be used to both experimentally confirm analytic findings as well as exclude alternate hypotheses, as appropriate.

Analysis of M57-Patents User Activity Records

The vast majority of the TAD records across the processed M57-Patents dataset were file times (hereafter referred to as file records) and registry record times. The Windows registry is a database-like structure that stores configuration information for the operating system and installed applications as well as considerable information about user activities (Carvey, 2009, pp158). One relevant feature of the registry is that two versions of many configuration items are stored across two “control sets”. One control set is “current” while the other control set acts as a backup in case there are system errors and the previous configuration needs to be restored.

Portions of the processed data is presented in tables in this chapter. Full tables for the Jo1 series are presented in the chapter to demonstrate the structure of the data, followed by summary tables for the larger datasets.

Jo1 Analysis Results

The Jo1 series contains six sequential TAD states. The following summary of the Jo1 analysis shows the per-interval lifespan for TADs from each incremental data state. After removing the initial state records from each image, the count of records still present in each state is shown in the total column in the table below. Each table shows the total count of records from a particular source type, as well as the count of those records that survived through each successive interval. The following three pairs of tables show the total record count from each state in the Jo1 series as well as the number of records from each that persisted into each successive interval.

Table 16

Jo1 Total Count and Percentage Record Lifespan by Successive Intervals

Record Origin State	Total Records	Count of Surviving Records by State – All Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	277135	277135	115695	115049	114259	114204	114120
Jo1_02	365568		365568	203232	202461	202432	202268
Jo1_03	241161			241161	67821	67763	67736
Jo1_04	234858				234858	143483	74428
Jo1_05	160433					160433	65900

Record Origin State	Total Records	Percentage of Surviving Records by State – All Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	277135	100	41.75	41.51	41.23	41.21	41.18
Jo1_02	365568		100	55.59	55.38	55.37	55.33
Jo1_03	241161			100	28.13	28.10	28.09
Jo1_04	234858				100	61.09	31.69
Jo1_05	160433					100	41.08

Table 17

Jo1 File Count and Percentage Record Lifespan by Successive Intervals

Record Origin State	File Records	Count of Surviving Records by State – File Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	151730	151730	53843	53204	52416	52361	52279
Jo1_02	107031		107031	17619	16854	16825	16661
Jo1_03	105613			105613	5792	5736	5709
Jo1_04	96500				96500	80356	11317
Jo1_05	22502					22502	3637

Record Origin State	File Records	Percentage of Surviving Records by State – File Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	151730	100	35.49	35.06	34.55	34.51	34.46
Jo1_02	107031		100	16.46	15.75	15.72	15.57
Jo1_03	105613			100	5.48	5.43	5.41
Jo1_04	96500				100	83.27	11.73
Jo1_05	22502					100	16.16

Table 18

Jo1 Registry Count and Percentage Record Lifespan by Successive Intervals

Record Origin State	Registry Records	Surviving Records by State – Registry Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	121171	121171	60970	60970	60970	60970	60970
Jo1_02	244318		244318	183976	183976	183976	183976
Jo1_03	121957			121957	61473	61473	61473
Jo1_04	122255				122255	61656	61656
Jo1_05	122376					122376	61728

Record Origin State	Registry Records	Percentage of Surviving Records by State – Registry Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	121171	100	50.32	50.32	50.32	50.32	50.32
Jo1_02	244318		100	75.30	75.30	75.30	75.30
Jo1_03	121957			100	50.41	50.41	50.41
Jo1_04	122255				100	50.43	50.43
Jo1_05	122376					100	50.44

Over this set, the registry records appear to operate and survive in a block size of just over 60,000 records. These large record changes are likely related a software or hardware installation which caused an update to the current control set backup scheme. Extensive content analysis and record similarity comparisons would be necessary to verify this, which is beyond the scope of this work. It remains an area for future research.

Table 19

Jo1 Web History Count Record Lifespan by Successive Intervals

Record Origin State	Web Records	Surviving Records by State – Web History Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	1978	1978	228	228	226	226	226
Jo1_02	10999		10999	61	61	61	61
Jo1_03	11354			11354	14	14	14
Jo1_04	12887				12887	13	13
Jo1_05	13234					13234	5

Table 20

Jo1 Event Count Record Lifespan by Successive Intervals

Record Origin State	Event Records	Surviving Records by State – Event Records					
		Jo1_01	Jo1_02	Jo1_03	Jo1_04	Jo1_05	Jo1_06
Jo1_01	1022	1022	0	0	0	0	0
Jo1_02	1112		1112	0	0	0	0
Jo1_03	1162			1162	0	0	0
Jo1_04	1210				1210	0	0
Jo1_05	1260					1260	0

Several TAD sources had unexpected lifespan results. The Windows event logs are an actual log, storing activity until overwritten by large amounts of data or deliberately cleared. The Internet history is composed of functional logs which allow a user to recall websites visited in the past, as well as cache web page components for faster load times in the future. Both of these sources showed almost no persistence of records from one interval to the next. These results were consistent across all datasets.

There were several other categories of records that contained only had a handful of records. Their numbers will be included in the total count but there were not enough records to characterize specific patterns with any confidence. The table below combines the counts of each record in the Jo1 set over its potential and actual lifespans. The potential lifespan for any record is the number of intervals for which data was available in the dataset after the record occurred.

Table 21

Jo1 Records Possible and Actual Survivor Count by Interval

	Number of Intervals				
	1	2	3	4	5
Total Possible Count	1279155	1118722	883864	642703	277135
Total Actual Count	596131	459701	384427	316472	114120
Total Percent Survived	46.603	41.092	43.494	49.241	41.178

	Number of Intervals				
	1	2	3	4	5
File Possible Count	483376	460874	364374	258761	151730
File Actual Count	161247	87111	74950	69022	52279
File Percent Survived	33.359	18.901	20.57	26.674	34.455

	Number of Intervals				
	1	2	3	4	5
Registry Possible Count	732077	609701	487446	365489	121171
Registry Actual Count	429803	368075	306419	244946	60970
Registry Percent Survived	58.71	60.37	62.862	67.019	50.317

Table 22

Jo1 Record Survival Percentage by Source per Interval

Intervals Survived	Total	File	Registry	Web History	Event
1+	46.60	33.36	58.71	0.64	0
2+	41.09	18.90	60.37	0.85	0
3+	43.49	20.57	62.86	1.24	0
4+	49.24	26.67	67.02	2.21	0
5	41.18	34.46	50.32	11.43	0

In the Jo1 dataset only one sequence extends for a full five intervals, which is not enough samples for a meaningful result. The file records from the first state also appear to be more persistent than the ones from later states, at least across this small sample. This may be a result of initial system configuration tasks and program installation performed on the first interaction with a new computer. These records appear to be more persistent while the events from the second interval onward may be more consistent with routine daily system usage. This may be a result of the initial configuration tasks performed on the first interaction with a new computer, while the records from the second state onward show a somewhat more consistent decline of surviving records over time.

Analysis of Jo-Combined Dataset

Recall that the JoC dataset is the concatenation of the seven Jo1 images (Jo1$_{\text{Initial}}$ plus Jo1$_1$ through Jo1$_6$) with 14 Jo2 images to see if there were any TAD record linkages between the "old computer" and "new computer". If two images had not been labeled “old computer” and “new computer” then there would have been no reason to separate the images into two separate collections. The portion of the lifespan table that shows the transition from “old” to “new” is shown in the table below.

Table 23

JoC Total Record Lifespan in Intervals 5 through 11

Record Origin State	Total Records	Surviving Records by State – All Records						
		JoC_05	JoC_06	JoC_07	JoC_08	JoC_09	JoC_10	JoC_11
JoC_04	234858	143483	74428	0	0	0	0	0
JoC_05	160433	160433	65900	0	0	0	0	0
JoC_06	402799		402799	0	0	0	0	0
JoC_07	920797			920797	774676	739964	717118	696992
JoC_08	565571				565571	437440	434337	434302
JoC_09	324410					324410	154318	154302
JoC_10	759781						759781	667034

Reviewing the data there is a clean break between the two systems in that no records from activities on the first computer were found on the second computer. This does not mean that there were no TAD records in common between the two systems. The initial state of the first computer (Jo1) was removed from all the JoC datasets. $JoC_{Initial}$ contained 1.3 million records. JoC_4 contained 1.75 million records. JoC_4 with the initial set removed left 619,339 records. When JoC_3 and JoC_4 were compared to determine newly created records, only 234,858 records were new in JoC_4.

This reinforces the impact in processing and analysis efficiency as a result of eliminating the initial state from the successive data states. The number of total records identified as “new” is highest on the first snapshot of the new computer. This is because there is not a specific “initial” image of this computer to remove the records prior to user activity. In the JoC dataset the Jo2 states all had the initial state of the “old” computer removed. Subtracting the $Jo1_{Initial}$ state from the first state of Jo_2 still removed some TAD records but not as many as were removed from images with the correct corresponding initial state. The full TAD state of JoC_7 contained 949,092 million records initially, reduced to 920,797 when the $JoC_{Initial}$ state was removed. All of the remaining records in JoC_7 were new, and none of those records were present in the JoC_6 state.

Although a manual review of the content of the images would also likely also show that the images were from different computers, the TAD adjacency comparison can be automated without much difficulty. If a series of images were unlabeled then this analysis could be used to identify states that came from the same or different systems.

Based on this finding, the Jo1 and Jo2 sets should be treated as separate systems, with no further analysis on the JoC set.

Analysis of Multiple Ten-Interval Sequences Across Multiple Systems

Operating under the hypothesis that the activities from the initial system configuration tasks on the first day are not representative of routine daily usage, there remain 18 distinct ten-interval sequences that do not include the first day of each set. These sequences begin with $Jo2_2$ and $Jo2_3$, $Charlie_2$ through $Charlie_8$, and Pat_2 through Pat_8. Using each of these states as the start of a ten-interval sequence the lifespan of the file and registry records were analyzed.

Table 24

File Record Percentage Lifespan Across Multiple Ten Interval Sequences

System	Number of Intervals									
	1	2	3	4	5	6	7	8	9	10
Pat	48.05	33.97	27.08	25.82	24.87	24.65	24.47	24.40	24.26	23.73
Charlie	48.24	22.79	14.05	13.28	12.58	12.37	12.07	11.80	11.09	10.53
Jo2	58.88	39.05	35.13	30.24	30.16	29.76	29.02	28.87	28.83	28.83
Overall	50.31	30.94	23.98	22.16	21.48	21.23	20.89	20.73	20.41	19.97

Registry Record Percentage Lifespan Across Multiple Ten Interval Sequences

System	Number of Intervals									
	1	2	3	4	5	6	7	8	9	10
Pat	67.15	67.05	67.05	67.05	67.05	67.05	67.05	67.05	67.05	62.22
Charlie	68.59	68.51	68.51	68.51	68.51	68.51	68.51	64.06	46.07	32.60
Jo2	83.35	83.35	83.35	83.35	83.35	83.35	83.35	83.35	83.35	83.35
Overall	71.31	71.24	71.24	71.24	71.24	71.24	71.24	69.46	62.25	55.03

Reviewing the file record lifespan across all three systems, the data degradation is relatively consistent across all three computer systems. The largest loss of file records occurred during the first interval after record creation, with smaller losses of the remaining records each interval until the end of the sequence.

Plots of the ten-interval sequences from Pat and Charlie are plotted in the figures below. Each sequence follows roughly the same pattern across, albeit with some local variations. The individual sequences are shown with shaded blue lines, while the average is in orange.

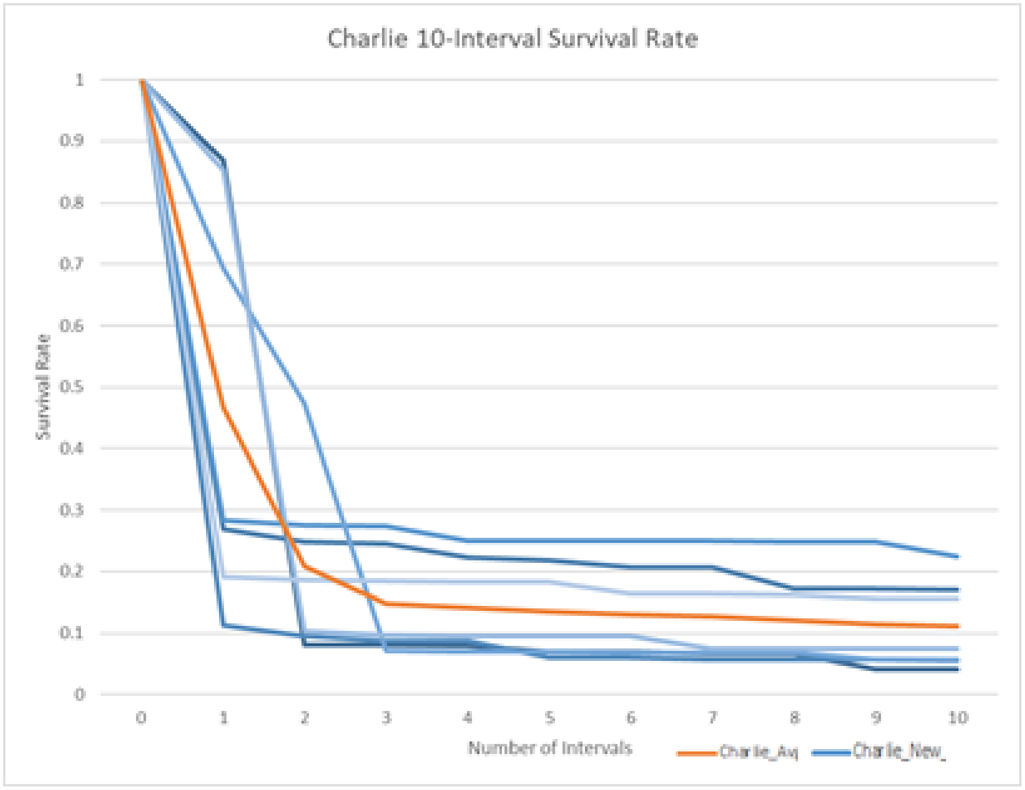


Figure 5

Charlie Ten-Interval Survival Plot

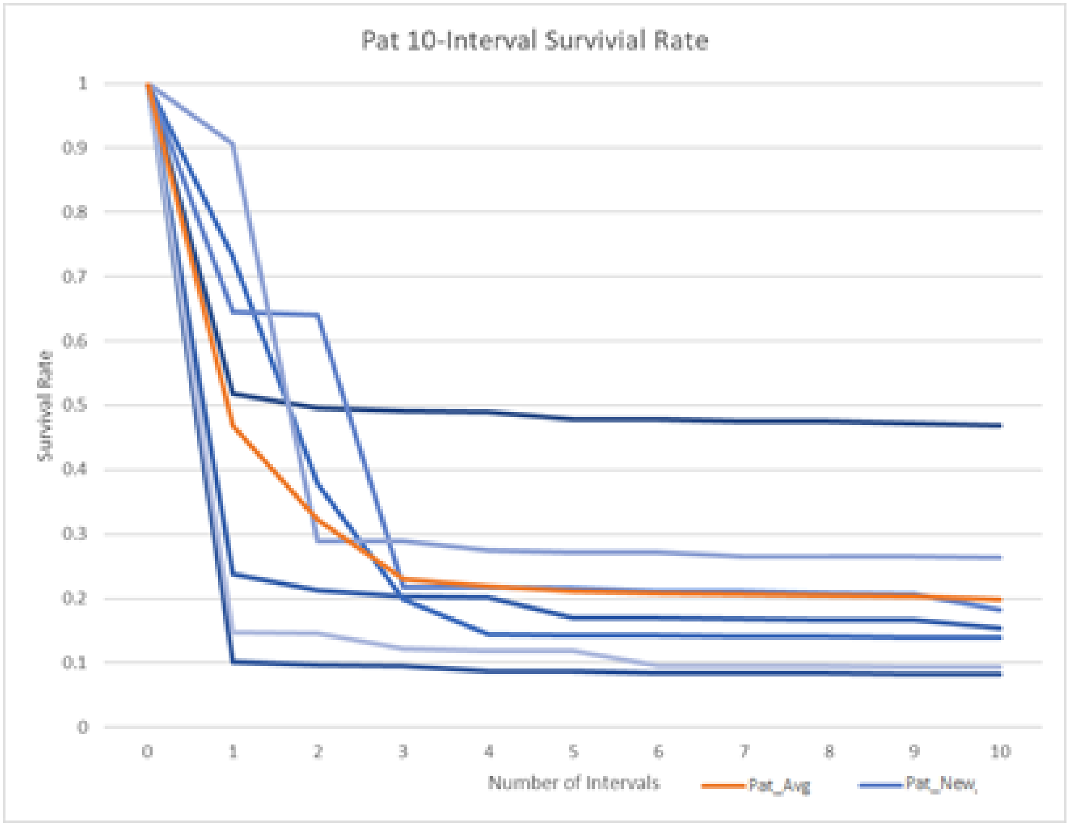


Figure 6

Pat Ten-Interval Survival Plot

The registry records show a similar block size in many of the ten interval periods as were seen did in the Jo1 series as well as a wide variation in behavior between the three different systems. Although the most of the records do remain unchanged, it is an area for future work to look at the registry records that were changing to identify any patterns, as they are the closely linked with user activities.

Chapter Ten – Conclusions and Areas for Future Work

Overview and Contributions

Activity Analysis is an increasingly common task in complex investigative digital forensic examinations. This analysis relies on extracting data from a system and projecting backwards to identify and explain events that took place in the past. A shared common language for describing temporal metadata is essential for making strong, evidence-backed conclusions and reporting the results of a forensic examination, whether for academia, an investigative team, or for a legal proceeding.

Although examples of metadata are well understood within the forensic community, there had not previously been a concerted effort to define the characteristics of temporal metadata. By identifying and classifying metadata source characteristics stronger inferential conclusions can be made especially in situations where some pertinent pieces of data are missing. The time-activity data taxonomy is a key component of this.

Some prior efforts in the forensic community have discussed high level conceptual models for system activities, while other efforts performed sector level analysis for small system changes. The system-activity model presented in this work supports the analysis of real-world data collections from successive system states represented by disk images.

Understanding a user-computer model with both physical and data state components, as well as a user that triggers cascading changes through interaction, allows investigators to consider state changes in the past that they can no longer directly observe, and project backwards through incomplete data.

The analysis of sequential images from the M57-Patents scenario validated the approach described in the activity model and system state decomposition. It also provided some initial insights into the decay of activity records on an in-use system and suggested enhancements for a more deliberate data collection and analysis effort. Similarities in the lifespan trends across the three distinct computers indicated that the findings are generally consistent, though the three sources were all created under similar artificial circumstances. Significant record degradation and distinctions in lifespan by type of record were observed and are likely transferrable to inferences about other sources and systems.

Even if a user is careful and actively attempts to hide or erase the evidence of their actions, it is likely that the corresponding activity cascades will leave some traces of the original action, or at least leave indications of obfuscating activity. These traces may be detectable through differential state analysis if they effect a large amount of records. Otherwise a careful content analysis will be required to detect the actions, although leveraging the TAD taxonomy can help as well.

Areas for Future Research and Limitations

The next logical step for the time-activity data taxonomy is to take the conceptual structure and map real-world metadata sources onto the characteristics to populate a reference library. Further analysis of record degradation to identify trends in lifespan based on record type, location, and the like would strengthen the applicability of the findings.

Automated tools could be developed to extract and present a list of TAD sources found on a system under examination and display their characteristics to the investigative team to assist their analysis efforts. Visualization tools and techniques could assist the team in understanding the pattern of life for the system in question, or for a subset of activities.

Requiring an exact match for the record lifespan analysis excluded several record sources where the older data was likely present but was not an exact match to the previous record. Using the characteristics from the TAD reference library would allow the development of techniques to link changed records from one data state to the next. A more complex survivor analysis based on sub-characteristics such as location on disk, resource described, and so on would be possible. Identifying functionally equivalent records which contain the same substantive content across multiple images would allow those records to be properly identified at unchanged as appropriate. This would be particularly helpful for Internet history, registry, and event log files.

The extraction, analysis, and differencing techniques could also be applied to sequential data sources other than full forensic disk images, such as volume shadow copies, system backups, online data repositories, and mobile devices. Future work could also look at the relative persistence of records based on the actions performed on the computer and the technical sophistication of the user. Larger datasets with more "typical" user activity such as extended web browsing over a period of months would provide expected lifespan information and allow a more robust statistical analysis.

The ability to perform lifespan analysis on extracted TAD records without needing access to the underlying content may make the acceptance of the anonymized data collection more palatable in an academic or governmental network environment.

There is also room to optimize the code and workflow used for the processing and analysis to make better use of parallel processing which would enable it to be used as a triage tool as well.

References

1. Agrawal, N., Bolosky, W., Douceur, J., & Lorch, J. (2007). A five-year study of file-system metadata. *ACM Transactions on Storage, 3*(3).
2. Altheide, C., & Carvey, H. (2011). *Digital forensics with open source tools.* Syngress Pub.
3. Carvey, H. (2009). *Windows forensic analysis: DVD toolkit 2E.* Syngress Pub.
4. Carvey, H. (2011). *Windows registry forensics: Advanced digital forensic analysis of the windows registry.* Syngress Pub.
5. Carvey, H. (2012). *Windows forensic analysis toolkit: Advanced analysis techniques for Windows 7 3E.* Syngress Pub.
6. Casey, E., Rose, C., Altheide, C., Holley, J., Luehr, P., Smith, J., Schwerha, J., Daywalt, C., Johnston, A., Pittman, R., Shaver, D., Kokocinski, A., Van, D. K., R, Maguire, T., Forte, D., De, D., & A. (2013). *Handbook of digital forensics and investigation.* Academic Press.
7. Chabot, Y., Bertaux, A., Nicolle, C., & Kechadi, M. (2014). A complete formalized knowledge representation model for advanced digital forensics timeline analysis. *Digital Investigation, 11.*
8. Farmer, D., & Venema, W. (2006). *Forensic discovery.* Addison-Wesley.
9. Friedenthal, S., Moore, A., & Steiner, R. (2009). *A practical guide to sysML: The systems modeling language.* Morgan Kaufmann Object Management Group/Elsevier.
10. Garfinkel, S. (2010). Digital forensics research: The next 10 years. *Digital Investigation, 7*, 64-73.
11. Garfinkel, S. (2012). Factors affecting data decay. *Journal of Digital Forensics, Security and Law, 7*(2), 7-10.
12. Garfinkel, S., Farrella, P., Roussev, V., & Dinolt, G. (2009). Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation, 6.*
13. Guðjónsson, K. (2010, August 1). *Mastering the Super Timeline With log2timeline*. Retrieved from SANS: https://www.sans.org/reading-room/whitepapers/logging/mastering-super-timeline-log2timeline-33438.
14. Harichandran, V. S., Walnycky, D., Baggili, I., & Breitinger, F. (2016). *CuFA: A more formal definition for digital forensic artifacts.* Presented at the DFRWS USA 2016.
15. Horsman, G. (2020). Digital evidence certainty descriptors (DECDs). *Forensic Science International: Digital Investigation, 32.*
16. Jones, J., Kahn, T., Kathryn, L., Nelson, A., Laamanen, M., & White, D. (2016). Inferring previously uninstalled applications from residual partial artifacts *Annual*

ADFSL Conference on Digital Forensics, Security and Law (pp. 113-130). https://commons.erau.edu/adfsl/2016/wednesday/3

17. Jones, K., Bejtlich, R., & Rose, C. (2015). *Real digital forensics: Computer security and incident response.* Addison-Wesley.
18. Kornblum, J. (2002). Preservation of fragile digital evidence by first responders *Digital forensics research workshop.*
19. Mandia, K., Prosise, C., & Pepe, M. (2003). *Incident response and computer forensics.* McGraw-Hill.
20. Marrington, A., Baggili, I., Mohay, G., & Clark, A. (2011). CAT detect (Computer activity timeline detection): A tool for detecting inconsistency in computer activity timelines. *Digital Investigation, 8,* 52-61
21. Neelamkavil, F. (1994). *Computer simulation and modelling.* Wiley.
22. Oh, J., Lee, S., & Lee, S. (2011). Advanced evidence collection and analysis of web browser activity. *Digital Investigation, 8,* 62-70.
23. Roussev, V., & Quates, C. (2012). Content triage with similarity digests: The m57 case study. *Digital Investigation, 9,* 60-68
24. Rowe, N. C., & Garfinkel, S. (2010, May). *Global analysis of drive file times.* Presented at the Systemic Approaches to Digital Forensic Engineering.
25. Schum, D. (2001). *The evidential foundations of probabilistic reasoning.* Northwestern University Press.
26. Shavers, B., &. (2013). *Placing the suspect behind the keyboard: Using digital forensics and investigative techniques to identify cybercrime suspects.* Syngress Pub.
27. Garfinkel, S. (2014). The prevalence of encoded digital trace evidence in the nonfile space of computer media. *Journal of Forensic Sciences, 59*(5), 1386-1393
28. Garfinkel, S., Nelson, A., & Young, J. (2012). A general strategy for differential forensic analysis. *Digital Investigation, 9,* 50-59
29. Soltani, S., & Seno, S. (2019). A formal model for event reconstruction in digital forensic investigation. *Digital Investigation, 30,* 148-160
30. Stallings, W. (2007). *Data and computer communications.* Pearson Prentice Hall.
31. Studiawan, H., Sohel, F., & Payne, C. (2019). A survey on forensic investigation of operating system logs. *Digital Investigation, 29.*
32. Woods, K., Lee, C. A., Garfinkel, S., Dittrich, D., Russell, A., & and Kearton, K. (2011, May 26). *Creating realistic corpora for security and forensic education.* Presented at the Annual ADFSL Conference on Digital Forensics, Security and Law. https://commons.erau.edu/cgi/viewcontent.cgi?article=1161&context=adfsl

Biography

Andrew C. Smallman graduated from Villa Park High School, Villa Park, California, in 1994. He received his Bachelor of Science degrees in Computer Science and Mathematics from the University of Redlands in 1998. He received his Master of Science in Network Security from Capitol College in 2005. He is currently employed by the United States Government.