AUTOMATED HUMAN INFERENCE IN DIGITAL FORENSIC INVESTIGATIONS USING HYPOTHESIS REDUCTION

Joshua Isaac James

The thesis is submitted to University College Dublin for the degree of PhD in the College of Engineering, Mathematical & Physical Sciences

June 3, 2013

School of Computer Science and Informatics

Head of School: Dr. John Dunnion
Supervisor: Dr. Pavel Gladyshev
Secondary Supervisor: Dr. Joe Carthy

This work is dedicated to Norbert Pugh and Wayne Stinson.

Table of Contents

List of Figures

List of Tables

List of Appendices

Abstract

Digital forensics is a relatively young, and rapidly growing area of forensic science. Both practitioners and academia have made much progress in the field since the late 1960's, when computer evidence was first being considered for use in trials; however, there are still many challenges that have yet to be resolved. This work focuses on challenges with analysis in digital forensic investigations. Specifically, how more information about happened actions in a system might be inferred, as compared to a human investigator, based on the observation of low-level traces in a system. This research aimed to solve this problem using hypothesis reduction based on the observation of trace evidence to reconstruct sequences of happened events.

This research contributes to the field of digital forensics in the following ways:

- Provides a method for automated action hypothesis encoding, testing and reduction using a form of signature-based matching that is extended beyond simple matching to include relational consistency between objects and events. This relational consistency is then used for hypothesis detection through elimination.
- Provides a more comprehensive method of detecting the instance(s) of past actions based on the observation of the ensemble of traces altered in the system. Actions may be modeled as trace update patterns allowing for automated, signature-based detection of the actions during a post-mortem analysis.
- Provides logical action-sequence checking over detected actions to infer other actions that must have happened, where there are no longer observable traces of these actions. This method uses the same signature-based methods to detect action sequences, and allow the time bounding of newly inferred actions.
- Defines categories of trace update behaviors that allow for information about an action to be detected. This includes the reliable detection of the most recent as well as prior instances of an action, and allows for consistency checking that helps to detect anti-forensic techniques.
- Submits a method for approximating the execution time-span of an action based on the observation of associated traces. When an action occurs, traces are not created instantaneously, but over a given period of time. Instead of

assuming the action must have happened at the time the trace was updated (possibly minutes from when the action originally executed), a more precise time-span in which the action must have occurred may be found.

- Gives a methodology for measuring investigation beyond false positive and false negative error rating that is currently standard in digital forensic tool testing. This methodology allows for the measurement and comparison of tools as well as processes, and may be used in conjunction with traditional error-rating to help determine what the cause of errors are over time.
- Provides a method for detecting the occurrence and generic association of actions when no prior information is known about the action based on the naïve clustering of objects (files) within a certain time-span, where the collection of objects will mostly be related to, and contain information about, a generic action, such as browsing the Internet.

In this work, a formal model for event reconstruction using hypothesis reduction is given, followed by practical application of the model for hypothesis encoding, detection and reduction purposes. A case study is then given comparing the proposed model to a machine learning categorization method for the detection of happened actions in a system.

Finally, achievements and implications of this research are described and considerations for future research are presented.

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at this, or any other, University or institute of tertiary education.

Joshua Isaac James
June 3, 2013

Acknowledgements

This dissertation would not have been possible without the help of so many amazing people. I am especially grateful to Debbie and Dan Stinson for their love, trust and constant support that can be felt from half a world away.

To my supervisor Dr. Pavel Gladyshev it's impossible to express how thankful I am. It's a rare occasion that someone has such a drastic positive impact on your life, and even more rare when that impact is on your mind. Thank you.

To my reviewers Dr. Félix Freiling and Dr. Neil Hurley, thank you for your time and valuable comments.

To all the amazing researchers I have had the pleasure of speaking with, and especially to my friends Dr. Yuandong Zhu, (soon to be Dr.) Ahmed Shosha and Alejandra Lopez Fernandez.

To the Law Enforcement officers I had the honor of working with, especially my friends at INTERPOL, Jaime Ansieta, An Garda Síochána, Martin Koopmans, Jake Jang and the Korean National Police.

To the countless people I've met and learned from over the years, especially for the cultural insights from Boyoung, Dawei, Kathi and Afrah.

And finally to Vic: my motivation to change.

Research partially funded by the 2Centre Project under the European Commission (EC)

Research partially funded by the Science Foundation Ireland (SFI) under Research Frontiers Programme 2007 grant CMSF575.

Research partially funded by the Science Foundation Ireland (SFI) under Short Term Travel Fellowship 07/RFP/CMSF575 STTF 09

Research partially funded by the Science Foundation Ireland (SFI) under Short Term Travel Fellowship 11/STTF/I2203

This Research was conducted using equipment funded by the Higher Education Authority of Ireland under the Research Equipment Renewal Grant Scheme.

Chapter 1

Introduction

This chapter outlines the motivation for this work, and gives the research objectives to be achieved. The contribution of this work is examined, and a summary of related achievements is given. The chapter ends by giving an overview of the structure of this dissertation.

1.1 Motivation

Digital forensics[1] is a branch of the forensic sciences that deals with the analysis of digital evidence from digital sources (Palmer 2001). Unlike traditional forensic sciences, a digital forensic analysis attempts to analyze non-physical evidence, or evidence that cannot be directly observed by humans without interpretation. It is because digital evidence cannot be directly observed that the admissibility of such evidence in court is under constant scrutiny (Casey 2004). To help establish digital forensics as a credible forensic science, digital forensic science was defined at the first Digital Forensics Research Workshop (DFRWS) in 2001 as:

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

Since the time of this definition, the areas of preservation, collection, validation and documentation have been considerably developed, and various models and standards for each phase of digital investigations have been proposed. Although the courts do expect some general information assurances (see section 2.3), no one standard has seen global acceptance (Reith, Carr et al. 2002; Beebe 2009).

In terms of research and development, the areas of analysis and interpretation have seen much less development due to higher complexity. The current state of analysis and interpretation of digital artifacts has been developed in a largely ad-hoc fashion,

[1] Also known as *computer forensics* and *cyber forensics*

producing methods that are highly manual, time-consuming and prone to human bias and error (Ogawa, Yamazaki et al. 2010).

Many analysis methods currently proposed and used in digital forensic laboratories are highly manual and demand much time from digital investigators. With the continuing rise in digital crimes, as shown by Gogolin (2010), and the unsustainable consumption of data storage (Casey, Ferraro et al. 2009; Garfinkel 2010), it is no longer feasible for investigation techniques, and specifically analysis, to continue to be highly manual and time consuming. It is for these reasons that future methods of digital investigation must focus on increased automation and validation at each phase. The research proposed here is a step to such highly automatic and formalized techniques.

1.2 Research Objectives

The aim of this research is to contribute to digital forensic science by providing a faster, more accurate and more efficient method in which digital investigators may conduct digital forensic investigations. The focus of this work will be on the analysis and interpretation phases of an investigation, and specifically on how digital forensic investigators make decisions and derive information from suspect data.

The objective of this research is to develop a solution that can effectively automate reasoning about artifacts and events derived from a suspect system.

More specifically, this research is expected to find solutions for the problems of:

1. Accurate event reconstruction from the observation of the final state of file time stamps in a post-mortem analysis
2. Automated interpretation of information to present the meaning of the state of the system to an investigator

that allow for:

3. Verification of human inferences during a digital forensic investigation
4. Automatic identification of suspect chains of events

One approach towards practical methods of utilizing all relevant sources of information to automatically produce knowledge for a digital forensic investigation is

to consider the implications of actions on the system, and what this collection of actions means in the context of a given fact to be proved.

The analysis method proposed in this work is based on the theory that both the direct observation and inference phases of an investigation of actions can be automated using signature-based detection methods. By determining the action instance traces that normally appear in a system after the execution of an action, it is possible to automatically 'infer' the occurrence of the action based on the observable traces.

1.3 Contribution

This research contributes to the field of digital forensics in the following ways:

- Provides a method for automated action hypothesis encoding, testing and reduction using a form of signature-based matching that is extended beyond simple matching to include relational consistency between objects and events. This relational consistency is then used for hypothesis detection through elimination.
- Provides a more comprehensive method of detecting the instance(s) of past actions based on the observation of the ensemble of traces altered in the system. Actions may be modeled as trace update patterns allowing for automated, signature-based detection of the actions during a post-mortem analysis.
- Provides logical action-sequence checking over detected action instances to infer other actions that must have happened, where there are no longer observable traces of these actions. This method uses the same signature-based methods to detect action sequences, and allow the time bounding of newly inferred actions.
- Defines categories of trace update behaviors that allow for information about an action to be detected. This includes the reliable detection of the most recent as well as prior instances of an action, and allows for consistency checking that helps to detect anti-forensic techniques.
- Submits a method for approximating the instance time-span of an action based on the observation of associated traces. When an action occurs, traces are not created instantaneously, but over a given period of time. Instead of assuming the action must have happened at the time the trace was updated (possibly

minutes from when the action originally executed), a more precise time-span in which the action must have occurred may be found.

- Gives a methodology for measuring investigation beyond false positive and false negative error rating that is currently standard in digital forensic tool testing. This methodology allows for the measurement and comparison of tools as well as processes, and may be used in conjunction with traditional error-rating to help determine what the cause of errors are over time.
- Provides a method for detecting the occurrence and generic association of actions when no prior information is known about the action based on the naïve clustering of objects (files) within a certain time-span, where the collection of objects will mostly be related to, and contain information about, a generic action, such as browsing the Internet.

1.4 Summary of Achievements

This work has had a number of academic achievements. Initial work using formal event reconstruction methods was published extending state machine analysis techniques (James and Gladyshev 2010). Post-mortem analysis of user activities, and specifically reconstruction of user activities through Windows Restore point analysis resulted in a number of joint publications with Yuandong Zhu as lead researcher (Zhu, Gladyshev et al. 2009; Zhu, Gladyshev et al. 2009; Zhu, Gladyshev et al. 2009; Zhu, James et al. 2009; Zhu, James et al. 2010). Work then shifted specifically to signature-based detection of past user activities, which has thus far resulted in one publication with another currently under review (James, Gladyshev et al. 2010). To support this work, information on various aspects of digital crime investigation was also collected through surveys with law enforcement (James 2010; James and Gladyshev 2010).

Through this work, research partnerships in other areas of digital forensics were created that have lead to publication. Most notably, in the area of digital investigation in the Cloud which led to the publication of one paper (Ruan, James et al. 2012) and a book chapter (James, Shosha et al. 2012). Other areas include work in malware analysis using event reconstruction techniques (Shosha, James et al. 2011; Shosha, James et al. 2012; Shosha, James et al. 2012).

In an attempt to demonstrate the practical aspects of this research, a software project was developed that focused on automated digital forensic investigation, and specifically automated analysis. This project led presentations in industry and law enforcement conferences (James, Koopmans et al. 2011), as well as collaboration on projects with the An Garda Síochána, INTERPOL, Europol, the Korean National Police University and the Korea University Digital Forensics Research Center.

1.5 Dissertation Structure

This dissertation is divided into 11 chapters.

Chapter 2 gives a brief introduction into the history of digital forensics and digital devices in the legal context. A comparison is made between traditional forensic investigation and evidence, and the relatively new concept of digital forensic investigation and digital evidence. Legal concepts and terminology relevant to the admissibility of evidence in court will be introduced, and an introduction into the concepts of digital evidence will be given. The chapter concludes with an introduction into the process required of representing media as data; data as information; and how the investigator may use information as evidence that supports or denies a hypothesis.

Chapter 3 begins by establishing terminology and concepts of digital forensic investigations that will be used throughout this work. General phases of a digital forensic investigation are explained, and commonly used digital forensic investigation process models are briefly discussed. This work then begins to limit its focus to the analysis, or knowledge acquisition, phase of an investigation, and specifically the process of event reconstruction. Finally, issues with traditional event reconstruction are examined.

The use of automation in digital forensic investigations is not only a technological issue, but also has political and social implications. Chapters 4 and 5 are provided to give further context about the state of the digital forensic community's acceptance and attitudes towards automation in digital forensic investigations.

Chapter 4 discusses some issues with the implementation and acceptance of automation in digital investigation, and the implication for human investigators. Attitudes towards the use of automation in digital forensic investigations are examined, as well as the issue of digital investigators' knowledge acquisition and

retention. The argument is made for a well-planned, careful use of automation going forward that allows for a more efficient and effective use of automation in digital forensic investigations while at the same time attempting to improve the overall quality of expert investigators.

Chapter 5 explores the identified issue of how to objectively compare the accuracy of highly automated analysis tools to real digital forensic experts. This chapter introduces a method based on information retrieval techniques to measure and compare the accuracy of tools, investigators and investigation processes. Related work is explored, and an argument is given why objective measurement of the accuracy of digital forensic investigations is necessary. A simple method of accuracy measurement is proposed as a starting point, and a very brief case study is given to illustrate the proposed method.

Chapter 6 examines the need for automatic event reconstruction in digital forensic investigations, and explores previously proposed methods. Current issues with automatic event reconstruction are given, as well as practical needs that must be met before automatic event reconstruction can be of practical value. This chapter concludes by introducing the objective and basic ideas of this work.

Chapter 7 begins by giving a brief introduction into human inference in the context of digital investigations. Next, the concept of causation is discussed at a high level. After, causal relations in a computer system are discussed that allow for back tracing of causal chains from effect to cause. Mathematical notation used throughout the remainder of this work is then given, followed by a formal definition of system and action models that are the base from which event reconstruction of actions in the system can take place.

Chapter 8 describes the derivation of action instance object update patterns. A practical method for determining the relation between actions and object update patterns is given. Analysis of object update patterns allows traces to be categorized, and rules of consistency to be determined. The need for object update thresholds is discussed, and a method for determining the object update threshold for an action is given. Next, a brief discussion and method for generalizing signatures for portability across suspect systems is given. The chapter ends by giving an overview of the object trace update experimentation used to derive general categories.

Chapter 9 begins by discussing the inferences of single and multiple occurrences of a given action instance based on the previously discussed signature matching methods. Consistency between object updates as well as higher-level consistency between actions is discussed. A probabilistic method for associating traces to actions when uncertainty exists is discussed, and the issues with such a method are discussed. Next, generic matching of action instances based on no prior information is then briefly described, with case examples explored and weaknesses given. After, high-level event reconstruction using the proposed signature-matching model is discussed that enables more actions to be inferred based on the presence of previously inferred action instances. Finally, the use of the proposed signature-based hypothesis reduction methods for human inference verification in digital forensic investigations is discussed.

Chapter 10 gives an evaluation of the proposed signature-based action instance detection method. First, the case scenario is given that forms the base case data for the remainder of this chapter. Next, an analysis of the extraction and categorization of traces and update thresholds needed to create signatures of action instances is given. The results of the signature-based action instance detection method is compared to similar work involving action "footprint" generation and matching via machine learning techniques conducted by Khan (2008). Weakness of the proposed signature-based action instance detection method is then examined and discussed.

Finally, Chapter 11 will give conclusions for this work and explore possible future extensions of this research.

1.6 Summary

This chapter outlined the motivation for this work, and gave the research objectives to be achieved. The contribution of this work was stated, and a summary of related achievements outlined. Finally, an overview of the structure of this dissertation was also given.

Chapter 2

Digital Forensic Science

This chapter gives a brief introduction into the history of digital forensics and digital devices in the legal context. A comparison is made between traditional forensic investigation and evidence, and the relatively new concept of digital forensic investigation and digital evidence. Legal concepts and terminology relevant to the admissibility of evidence in court will be introduced, and an introduction into the concepts of digital evidence will be given. The chapter concludes with an introduction into the process required of representing media as data; data as information; and how the investigator may use information as evidence that supports or denies a hypothesis.

2.1 A Brief History

Although one of the most cited definitions of digital forensic science, as previously stated, was defined at the 2001 DFRWS, digital forensic investigation predates this academic definition. Several notable but less developed definitions were previously proposed, such as those submitted by McKemmish (1999) and Civie and Civie (1998). Likewise, beyond academic definitions, research and digital investigations were already taking place prior to 2001. For example, Pollitt (1995) claimed that "[f]or a number of years now, law enforcement agencies have been seizing computers and other electronic devices". A growing interest in digital forensic investigation is confirmed by looking at other works of the early 1990's (Collier and Spaul 1992a; Collier and Spaul 1992b; Clede 1993; Spafford and Weeber 1993). Hannan (2004) claims "forensic computing origins lay in the late 1980s…", which is when computer-based evidence was encountered more often by police (Jones 2004), and is perhaps true for forensic computing as a field or separate science (Garfinkel 2010), but from a legal perspective computers and computer evidence were topics of concern before then. For example, the U.S. Computer Fraud and Abuse Act was first enacted in 1984 (USDoJ 2002), and also in the early 1980s *Computer in Court - A Guide to Computer Evidence for Lawyers and Computing Professionals* (Kelman and Sizer 1982) was published that considers fundamental legal issues in relation to computer evidence. Even prior to this work, the admissibility of computer evidence from digital devices has been discussed as early as 1974 (Tapper), with references to computer technology in trials appearing as early as 1962. In the mid-1970's computer evidence became more of a focus (Roberts 1974; DeHetre 1975; Jenkins 1975) rapidly evolving

through the late 1970s when the computer was compared to an expert witness (Teubner 1978), and the basic tenants for admissibility of computer evidence were explored (Connery and Levy 1979); most of which is still relevant today. This increased interest in the mid-1970's correlates to the increasing reliance on computer systems in business, and an increasing availability and usage among the public (Polsson 2011). Even with a considerable amount of evolution and previous work, Wilsdon and Slay (2005) claim that there are issues when it comes to implementation of past research and development in digital forensic investigations; a claim reiterated by Pollitt (2007) and Garfinkel (2010).

Despite a multitude of technological, and in some ways philosophical, changes to the field of digital investigation, the DFRWS definition is still widely accepted. Alternatives, or amendments, have been proposed (Hannan 2004; Kent, Chaevalier et al. 2006), but the DFRWS definition has remained popular. Since the time of this definition, the areas of evidence preservation, collection, validation, identification, analysis, interpretation, documentation and presentation have been continually developed, and various process models and standards for each phase of digital investigations have been proposed. However, no one standard has yet to see global acceptance (Wilsdon and Slay 2005; Hunton 2012; Lim, Savoldi et al. 2012), and according to James and Gladyshev (2010) the majority of organizations are creating their own standard operating procedures (SOP), which are not always directly based on a common standard. But regardless of the process used, when considering digital forensic investigations, the resulting evidence must be admissible in court (Carrier 2006a).

2.1.1 Current State of Digital Crime

According to Internet World Stats (2011), from late 2000 to late 2011 there has been an estimated 528.1% worldwide growth in Internet users, numbering 361 million in 2000 to 2.267 billion at the end of 2011, a trend which is expected to continue (IBTimes 2010; Meeker 2012). Cisco (2012) estimates that there will be over 10 billion mobile-connected devices by 2016. Further, traditional and non-traditional digital devices - such as TVs and kitchen appliances - are expected to be increasingly connected to the Internet, contributing to a forecasted 50 billion devices connected by 2020 (Higginbotham 2010). This growth in users, devices and associated services has given rise new possibilities for business and communications, as well as digital crime.

Statistics from the Internet Crime Complaint Center (IC3) (IC3 2011), a US-based organization, show that there was a 3.4% increase in complaints filed from 2010 to 2011, with 303,809 and 314,246 complains per year, respectively. McAfee (2010) gives an overview of the growth of digital crime from 2000 to 2010, claiming that the cybercriminal community evolved from hackers simply looking for a challenge, to organized gangs looking to profit off the rapidly increasing use of connected devices and services. Some gangs are well established, and in 2011 a single highly-organized gang was suspected of being responsible for up to a third of all data thefts (Williams 2011). Digital forensic investigators, however, not only need to deal with booming online criminal activity, but also the use of digital devices related to more traditional crimes. For example, Gogolin (2010) claims that most investigations today involve some sort of digital component. Cisco (2012) estimates that "[b]y the end of 2012, the number of mobile-connected devices will exceed the number of people on earth". In more saturated regions when a crime happens, both criminals and bystanders are likely to be producing digital information which may be of use in the investigation of a non-digital crime. Amateur pictures and videos taken by cell phones have captured abuse, and even murder (Barnard 2009; CNN 2009), and have been used as evidence in court (Nguyen 2012). Further, text messages, cell phone logs and geo-location services are also commonly analyzed by law enforcement in digital and non-digital crime investigation. This presents a difficult situation for digital forensic investigators.

The field of digital forensic science is relatively new, technologies are rapidly advancing, and the scope of the digital investigator's job continually expands. Yet funding for digital investigators in law enforcement is slow to increase (Gogolin 2010). This may change with the perceived[2] growing threat of a full-scale "cyber war"[3], and some countries may begin to focus budgets on digital investigation and

[2] Some researchers believe various groups have exaggerated the threat of a cyber war. Schneier, B. (2010). "The Threat of Cyberwar Has Been Grossly Exaggerated." Schneier on Security http://www.schneier.com/blog/archives/2010/07/the_threat_of_c.html 2011.

[3] "Cyberwar is a form of war which takes places on computers and the Internet, through electronic means rather than physical ones." Smith, S. E. (2011). "What is Cyberwar?" Retrieved 28 Jan., 2011, from http://www.wisegeek.com/what-is-cyberwar.htm.

security (Paul 2012). Regardless, digital forensic investigators must attempt to keep up with rapidly advancing digital, as well as traditional, criminals while ensuring the integrity of the science of digital forensics. Rapid changes in digital forensic science, however, cannot come from digital forensic investigators alone. Improved collaboration and communication is needed between the currently siloed digital crime investigation players – military, law enforcement, corporate and academia – and must be focused at a global, not national, level (Jones 2012).

2.2 Digital Forensics and the Forensic Sciences

Traditional forensic sciences such as forensic pathology (Dolinak, Matshes et al. 2005) and forensic dactyloscopy (fingerprinting) (Galton 1892) rely purely on physical evidence such as wound and disease identification, ballistics, bloody knives, fingerprints, etc. Investigators use "tangible, physical items found on, in, or around… the crime scene" (Palmer 2002) to determine the circumstances surrounding the event that is being investigated. Because all evidence in regards to traditional forensic sciences is physical, it can be physically manipulated. Fingerprints at a crime scene, for example, may be extracted by dusting the print with a powder, and "lifting" it with adhesive tape (Fingerprinting 2010). The same is not possible with digital evidence. "The digital realm transcends physical space" (Herrera 2006). Digital forensics is as concerned with the representation of information as it is with the tangible physical object on which the information is stored.

Like other forensics diciplines, utmost care must be taken to preserve the physcial aspect of evidence, but unlike these areas, care must also be taken to ensure the integrity of the represented information. Generally, the hardware associated with an incident will by itself provide no information as to the incident in question. For example, examining the physical components of a computer will give no insight into if someone created a blackmail letter with the machine. To determine if a letter has been created, an investigator must look to see what information is being represented within the system. This is where digital forensics greatly differs from traditional areas. Information about what has happened is not tangible and not directly observable (Pollitt 1995). This non-tangible evidence provides a whole new set of challenges with regards to evidence for use in court. Where before a jury could see a bloody knife and the fingerprints of the suspect, now they must be able to understand

that a letter that exists only as a series of magnetized areas on a hard disk drive, and hence not 'real' in a traditional sense, still has an impact on reality.

2.3 Legal Concepts

As stated earlier, when considering digital forensic investigations, the resulting evidence must be admissible in court (Carrier 2006a). This is because "[t]he ultimate purpose of digital forensic analysis is to assist in finding and convicting perpetrators of crime" (Gladyshev 2004). This section will not go in-depth into the various legal systems, but instead scope will be limited a Common Law system – a law system based on legal precedence, such as that used in Ireland and the United States. The purpose of which is simply to familiarize the reader with evidential concepts and vocabulary relevant to digital forensic investigations.

A digital forensic investigator, in assisting to find and convict perpetrators of crime, analyzes digital media in an attempt to find evidence that supports a probandum, or proposition to be proved. For example, that the suspect knowingly downloaded illegal content. In an investigation, multiple hypotheses are proposed that put forward a possible explanation for known facts, or points that are indisputably true. Hypotheses should be created both for and against the guilt of the suspect. One of the possible hypotheses is the probandum to be proved. A digital forensic investigator derives facts from data stored on digital devices that are then used as evidence to support or negate a hypothesis. Digital evidence is defined as data that persuades a tribunal to reach a reasoned belief on a probandum. This evidence can be either inculpatory or exculpatory; that is, supporting guilt or innocence of the accused, respectively. Both types of evidence are important to consider since a fact can be dependent on the context of the information from which it is derived. For example, in many countries child exploitation images are illegal to knowingly possess. From this, it is not sufficient for a digital forensic investigator to propose the guilt of a suspect as fact based solely on the presence of exploitation images on a suspect's computer, but he or she must provide evidence that the suspect was aware of the images. If a computer virus was found which is known to automatically download illegal content, this fact could be used as exculpatory evidence to support the innocence of the suspect; whereas the fact of not finding such a virus could be used as inculpatory evidence against the suspect.

Since the possible explanations are infinite, found evidence does not prove a given hypothesis, but merely increases or decreases the probability of a given hypothesis. According to Anderson and Twinning (1998) proof is the persuasiveness of the total mass of evidentiary facts. The standard of proof generally required in Common Law systems for criminal cases is "proof beyond a reasonable doubt". *In re Winship*, 397 U.S. 358 (1970) found that proof beyond a reasonable doubt has existed as a semi-formal standard of poof in courts at least "as late as 1798". The same proceeding determined that the "reasonable-doubt standard" is a constitutional requirement in the U.S. in accordance with Due Process. Due process is defined as "a judicial requirement that enacted laws may not contain provisions to result in the unfair, arbitrary, or unreasonable treatment of an individual" ("Due Process" 2012). This means that in criminal proceedings, the total body of evidence must prove the guilt of the suspect beyond a reasonable doubt, or that "no other logical explanation can be derived from the facts except that the defendant committed the crime" ("Beyond a Reasonable Doubt" n.d.).

2.3.1 Legal Proof

When discussing proof in the legal context, the presumption of innocence, or the concept of 'innocent until proven guilty', is generally recognized. Because of this, it is the responsibility of the prosecution to provide proof supporting a claim against the suspect in order to convince the 'finder of fact' – e.g. a judge and/or jury – that a certain hypothesis is true. This responsibility is referred to as the 'burden of proof'. It is the responsibility of the defense to disprove or otherwise discredit the evidence given by the prosecution in order to weaken the persuasiveness of the argument against the suspect. "The standard of proof required to discharge the legal burden depends on whether the proceedings are criminal or civil" (Keane 2008). The standard of proof required in criminal proceedings is defined as proof 'beyond a reasonable doubt' ("Beyond a Reasonable Doubt" n.d.); while in civil proceedings it is generally defined as 'the balance of probabilities' ("Balance of Probabilities" n.d.).

The job of police officers, or digital forensic investigators in this case, is to conduct an impartial investigation, and provide an unbiased report on the discovered facts – both inculpatory and exculpatory – that are relevant to the probandum. Discovered facts can be submitted as evidence, which are then tested by the courts to determine if the evidence is admissible. Evidence from digital forensic investigations is considered

scientific, like other forensic sciences, and therefore can be tested against the requirements for the admittance of scientific evidence. Assuming the evidence is admitted in court, a jury then compares the observed chain of events to the probandum, and guilt or innocence is determined.

2.3.2 Requirements of Scientific Evidence

As stated by Cohen (2010), "[o]n a global level, the most commonly applied standards are similar to the U.S. Federal Rules of Evidence and the Daubert decision". In the United States, the admissibility of scientific evidence was primarily examined using the Frye standard - from *Frye v. United States*, 293 F. 1013(1923) - where accepted scientific evidence based on the scientific community's general acceptance of the employed technique. The Frye standard was superseded in 1993 by the Daubert standard [*Daubert v. Merrell Dow*, 509 U.S. 579 (1993); *General Electric Co. v. Joiner*, 522 U.S. 136 (1997); *Kumho Tire Co. v. Carmichael*, 526 U.S. 137 (1999)], although not without criticism (Gutheil and Bursztajn 2005; Giannelli 2006). The Daubert standard allows a judge to determine the reliability of scientific evidence during what is called a "Daubert Hearing", usually before the trial. Four general categories are used as guidelines when assessing a scientific technique: testing, error rate, publication of the technique, and acceptance from the scientific community (Carrier 2003). The Daubert standard was introduced to reduce the misuse of scientific evidence, but still not all states in America have adopted the Daubert standard; either keeping with the Frye standard or opting for their own testing methods (O'Connor 2010).

2.4 Digital Evidence

The Scientific Working Group on Digital Evidence (SWGDE) define digital evidence as "[i]nformation of probative value that is stored or transmitted in binary form" (SWGDE 2009). There are two states of data that digital forensic investigators must work with: live data and persistent (post-mortem) data. Live data is data in a system that is powered on, and is more prone to change. Persistent data is data available when the system has been shut down. It must be said that all data has a degree of volatility, or susceptibility to change, and the speed in which data is likely to change, ordered from fastest to slowest, is known as the order of volatility (OoV). Farmer and Venema (2005) give an example of a "rough guide" for the order of volatility for different locations of data storage, assuming a live system (Table 2.1):

Table 2.1 Approximation of the lifespan of data by Order of Volatility

Registers, peripheral memory, caches, etc.	nanoseconds
[Random Access Memory]	nanoseconds
Network state	milliseconds
Disk	minutes
Floppies, backup media, etc.	years
CD-ROMs, printouts, etc.	tens of years

Digital investigators must consider the implications of collecting each type of data, and be able to prioritize the data by the order of volatility. The order of volatility, and even availability of data, differs between a live system and an offline system.

2.4.1 Post-Mortem Data Forensic Acquisition and Verification

Many digital investigators have been taught to physically disconnect the power, or "pull the plug", on a suspect computer that is powered on at a crime scene (NIJ 2008). Disconnecting the power would ensure that evidence would not be modified by processes running on the suspect system. Pulling the plug became standard practice until relatively recently when investigators, and the legal system in general, started considering volatile data, such as data in Random Access Memory, that was being lost. Once the power had been removed, a digital investigator was able to copy the persistent data, such as data written to the hard drive before powering down. Because persistent data is static, it is relatively easy to verify that the data has not changed over time. Verification of static data for digital forensic investigation purposes is normally done using a cryptographic hash.

"A hash function (H) is a transformation that takes an input m and returns a fixed-size string, which is called the hash value h (that is, h = H(m))" ("Public-Key Cryptography Standards" 2009). Hash values are used in digital forensic investigations to verify the integrity of data. Hash value in the context of digital forensics are defined as "numerical values, generated by hashing functions, used to substantiate the integrity of digital evidence…" (SWGDE 2009).

Once a suspect's device is shut down, an examiner can make a bit-by-bit copy of the storage media, known as a 'forensic image' (Shipley and Door 2012). A common way

to acquire the suspect's storage media is by removing the suspect hard drive, and attaching it to a workstation using a hardware or software write-blocker to ensure no data can be written. Once connected, many tools exist to make a bit-by-bit copy of either the whole physical disk, known as a physical disk image, or to make a copy of specific partitions on the disk, known as a logical disk image.

By hashing the suspect bit-level data on the media before acquisition, the resulting hash value becomes the standard with which to compare to ensure 1) the data on the suspect disk has not changed through actions of the examiner, 2) the created forensic image is an exact copy of the original suspect media, and 3) the suspect data can be verified by a third-party. Hashing static data at the time of acquisition allows investigators throughout the chain of custody to verify that any previous, or current, processes have not altered the data they are working with. Persistent data may be verified by utilizing the fact that the data should not change over time. Because of this, a hash of the data can be compared with the original, even years later. In the case of a post-mortem data analysis, the hard drive of a suspect computer can be removed and hashed, and a forensic disk image, or exact copy of the data, could be created. The disk image could then be hashed. If both hash values are exactly the same, then the data on the hard drive, and within the disk image, can be said to be the same. Hashing and imaging is normally done after powering down the suspect computer. The hashing process is not instantaneous, and shutting down ensures that no data changes while hashing process is taking place. In many situations, however, powering down a critical server may not be feasible, or data that is relevant to the case may not be persistent. In these cases, live data forensics may be the only alternative.

2.4.2 Live Data Forensic Acquisition and Verification

Live data forensics has been proposed to "provide additional information that is not available in a disk-only forensic analysis" (Adelstein 2006). Live data forensics is conducted on a running suspect system, and is used to collect volatile data – such as the contents of RAM – that may contain encryption keys, chat fragments, active network connections, active processes, cache, etc.

Live forensic imaging of a suspect drive may also be done on critical systems that cannot be shut down. There are a number of benefits and drawbacks to live data forensics. Sometimes live data forensics is necessary because the system may be

critical to business continuity, shutting down may create legal liability for examiners, or the system may have encrypted media that will be inaccessible when dismounted (Bilby 2006). In these situations, live data forensics may be the best way to collect evidence, although not ideal. Several challenges with live data forensics must be considered.

First, the suspect system is still running, and data is likely to be changing as it is being collected. Because of this, it is impossible for a third-party to verify that the data collected is correct. This is similar to previously discussed fingerprint extraction methods. Acquiring a copy of the fingerprint alters the original, making it impossible to be verified by a third-party. In the case of digital forensics, to acquire RAM (from the suspect machine) a program must first be loaded into the suspect RAM. Meaning that to collect the data, some data must be altered. Further, acquiring RAM takes time, and during that time processes may be starting or stopping and users may be connecting and disconnecting from the suspect system. By the time the entire contents of RAM have been acquired, the state of the RAM has changed from when collection began. Hashing methods for verification of live data are only useful for the data that has been collected and is no longer changing. Data that has been collected cannot be verified after acquisition since the original data is constantly changing, and is no longer the same as at the time of acquisition. Also, there is a possibility of crashing the suspect system when attempting live data forensics, which has potential to disrupt, or even corrupt, critical business data.

Second, when conducting live data forensics, the investigator will make changes to the system. However, in order to collect volatile evidence, the suspect's computer must remain on, and the suspect's operating system must be used to access the needed data[4]. When retrieving information from RAM, for example, a program must be loaded into the running memory, changing its contents. Even just inserting a USB key into a running suspect system will alter the system. However, Principle 2 of the ACPO guideline states:

[4] Some data, such as imaging Random Access Memory, can be acquired directly from an external FireWire connection: Gladyshev, P. and A. Almansoori (2010). Reliable Acquisition of RAM dumps from Intel-based Apple Mac computers over FireWire. Second International Conference on Digital Forensics and Cyber Crime (ICDF2C). Abu Dhabi, UAE, ICST.

> *In circumstances where a person finds it necessary to access original data held on a computer or on storage media, that person must be competent to do so and be able to give evidence explaining the relevance and the implications of their actions.*

Principle 2 essentially allows the use of live data forensics in extraordinary situations, as previously mentioned, as long as the investigator is both competent, and knows the full impact of his or her actions.

Finally, live data forensics usually relies on the suspect system. Alternative methods of RAM acquisition, such as the so-called 'cold boot attack' (Halderman, Schoen et al. 2009) and FireWire acquisition methods (Martin 2007; Gladyshev and Almansoori 2010) have been proposed, but these methods also have associated risks. Carrier (2006c) claims that the suspect system cannot be trusted. Rootkits or other malware in the suspect system can provide various anti-forensic functions, resulting in unreliable evidence (Bilby 2006). To minimize the reliance on the suspect system, Carrier suggests that analysis applications should use their own file system code rather than relying on the kernel and system calls, and to use trusted binaries on write-protected media; a technique most live data forensic tools, such as Microsoft's Computer Online Forensic Evidence Extractor (COFEE) (Mansfield-Devine 2010; Microsoft 2010), are using. These techniques do not completely remove the risk of malicious code on the suspect system affecting a forensic investigation, but, assuming Principle 2 of the ACPO guideline has been met, they do reduce the risk to a level that tends to be accepted in court.

Regardless of the drawbacks, live data forensics is sometimes the only option for collecting evidence. Live forensic data has been accepted in court (Adelstein 2006), and since computer systems are becoming increasingly more distributed, data center oriented, and services are becoming more 'Cloud' based, traditional post-mortem data forensics becomes less of an option.

2.4.3 Sources of Digital Evidence

Sources of digital evidence are dependent on the current state of the suspect system. For example, if a suspect's system is live, data from RAM may be accessible. This data could possibly hold information that could be used as inculpatory or exculpatory evidence, but would not be available if the system was powered down. While

computer systems are common sources of digital evidence, other digital devices, such as cellular phones, are becoming just, if not more, common (Kanable 2007; Mislan, Casey et al. 2010). Generally, digital forensic science can be separated into four generic categories:

- Computer Forensics
- Network Forensics
- Mobile Device Forensics
- Database Forensics

From these, different types of data and information can be found. For example, computer forensics may focus more on the actions of the user on the system, while network forensics would focus on the connections between systems, and cell phone forensics may be concerned with the communication between people. All of theses types of information, if relevant, may possibly be used as evidence in court to prove or disprove a claim if acquired in a forensically sound manner.

2.4.4 Deriving Digital Evidence

The previous sections gave a brief background into sources and types of data, but some further aspects of evidence derivation from digital devices must be considered. The National Institute of Standards and Technology (NIST) give a digital forensic investigation process model and describe the conversion of data to evidence at each step in the process (Figure 2.1) (Kent, Chaevalier et al. 2006). In this model, each layer builds on the one prior. This has several implications regarding the derivation of evidence from digital sources. For example, changing even one magnetic charge on the physical media has the potential to alter the data that the magnetic charge represents.

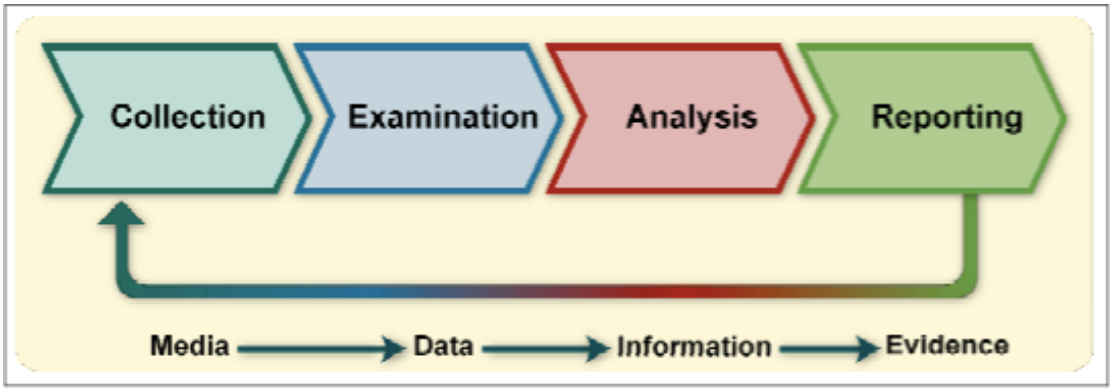


Figure 2.1 National Institute of Standards and Technology, four-phase digital investigation model proposed in SP 800-86: Kent, K., S. Chaevalier, et al. (2006). Guide to Integrating Forensic Techniques into Incident Response, National Institute of Standards and Technology: 121.

Data is information in digital form, and this altered data may then represent different information. Incorrect information may lead investigators to false conclusions, where the incorrect information is used as evidence to support the false conclusion. It is for this reason that the integrity of the data is of the utmost importance. As mentioned before, currently the most accepted way to test whether data has not been altered is to create a cryptographic hash the data upon initial acquisition. Since the bit level is the lowest level of electronic data representation, verifying that the bit level has not been altered also ensures that any higher layers, such as the information layer, have also not been altered. However, to derive evidence of criminal activity, data alone is not sufficient. In the previously used example of possession of illegal content, context of the data is needed to determine whether the possession was known by the suspect, and thus illegal. It is because of this that the analysis process in a digital forensic investigation involves transforming the raw data (magnetic or bit-level) into a human-readable (information) form. The first layer involves the interpretation of the data into the information it represents. For the purposes of a digital forensic investigation, it is imperative that the data is interpreted correctly. Verification of the methods used to interpret data has been briefly discussed previously in the context of court. Further discussion, however, is beyond the scope of this work. Interpretation of data into the information it represents is generally tedious, time consuming, prone to human error and sometimes impossible. Because of this, computer software has been, and continues to be, developed to attempt to automate the interpretation of data to extract the information it represents.

Once the data has been transformed into human-readable form, an investigator must use his or her training and experience to determine what the information means in the context of the investigation. For example, when the interpreted data represents a file timestamp, the timestamp itself is information that the investigator may use to support or deny a hypothesis. A single piece of information, however, may not provide sufficient support for a claim. In this case, multiple pieces of information may be necessary to provide enough support or context for a claim. Sufficient[5] support for a claim produces new facts or conclusions (Giarrtano and Riley 2005). In forensic investigations, facts are normally reported in terms of the events that led up to the incident, effectively answering how and why an incident took place, if at all. The reconstruction of events that have happened are supported with a combination of the current state of the data, and knowledge of the way the system functions. The digital artifacts that support this reconstruction of events are considered evidence for the derived most likely chain of events.

2.5 Summary

This chapter gave a brief introduction into the history of digital forensics and digital devices in the legal context. A comparison has been made between traditional forensic investigation and evidence, and the relatively new concept of digital forensic investigation and digital evidence. Legal concepts and terminology relevant to the admissibility of evidence in court has been introduced, and an introduction into the concepts of digital evidence has been given. The chapter concluded with an introduction into the process required of representing media as data; data as information; and how the investigator may use information as evidence that supports or denies a hypothesis.

[5] Investigators normally attempt to find support they believe will be sufficient to demonstrate the likelihood of one claim compared over another in court.

Chapter 3

Digital Forensic Investigation and Analysis

This chapter begins by establishing terminology and concepts of digital forensic investigations that will be used throughout this work. General phases of a digital forensic investigation are explained, and commonly used digital forensic investigation process models are briefly discussed. This work then begins to limit its focus to the analysis, or knowledge acquisition, phase of an investigation, and specifically the process of event reconstruction. Finally, issues with traditional event reconstruction are examined.

3.1 Digital Forensic Investigations

Carrier (2006a) differentiates between 'digital investigations' and 'digital forensic investigations', claiming that "[a] digital investigation is a process to answer questions about digital states and events", while "[a] digital forensic investigation is a special case… where procedures and techniques that are used will allow the results to be entered into the court of law". Under this definition, when an investigator is conducting a digital forensic investigation, the focus is on ensuring that the overall investigation process is able to produce evidence that is admissible in court. While this is a requirement for law enforcement (LE), non-LE entities do not always consider the admissibility of evidence when beginning an internal investigation. This has led to administrators or technicians unknowingly overwriting data of evidential value that was later unusable by law enforcement once it was found that the case needed to be escalated to a criminal investigation. It is for this reason that all digital investigators should be competent, and able to consider how their actions could affect future legal recourse. "[A]ny case involving computer forensics should always be treated as though it were going to court, and that any documentation and evidence will eventually be turned over to a prosecuting attorney" (Shinder and Cross 2008).

Digital forensic investigations are comprised of many different sub-fields. The general sub-fields include computer forensics, network forensics and cellular phone forensics: each of which is comprised of more specific forensic studies, e.g. file system, memory and software analysis. New forensic combination or sub-fields are being created with the advancement of technology, such as critical infrastructure (Purdy 2010) and Cloud forensics (Ruan, Carthy et al. 2011). While the technical

aspects of an investigation may be specific to the suspect device, the examination and investigation process models can be generically applied.

3.2 Digital Forensic Investigation Phases

When considering digital forensic process models, there are essentially two layers of abstraction that are discussed: the depth of digital forensic examination, and the process phases of each type of examination. The depth of forensic examinations was not always considered, and each piece of media normally received an in-depth analysis. Since the capacity of common data storage media has, and still is, growing rapidly – combined with a growing number of cases involving digital media – some investigators are finding it impractical to conduct in-depth analysis on each piece of media. "[F]ew [Digital Forensic Laboratories] can still afford to create a forensic duplicate of every piece of media and perform an in-depth forensic examination of all data on those media… It makes little sense to wait for the review of each piece of media if only a handful of them will provide data of evidentiary significance" (Casey, Ferraro et al. 2009). This philosophy has led to the growing acceptance of digital forensic triage (Rogers, Goldman et al. 2006; Koopmans 2010; Mislan, Casey et al. 2010), as well as the concept of a preliminary analysis; both of which will be discussed further.

Casey, Ferraro et al. (2009) described a three-tiered model of forensic examination to enable the "tailoring [of] forensic examination[s] of digital evidence to the type of crime or case under investigation". This model includes a survey/triage forensic inspection, a preliminary forensic examination, and finally an in-depth forensic examination. Many law enforcement agencies are currently considering, and even implementing this model (Goss 2010). Though, corporate and contract-sector digital forensic analysts are currently more likely than law enforcement to use triage and preliminary analysis techniques (James and Gladyshev 2010).

3.2.1 Triage Forensic Inspection

Koopmans (2010) proposed a definition of digital forensic triage that is derived from the definition of triage in the medical context:

> *A process for sorting injured people into groups based on their need for or likely benefit from immediate medical treatment. Triage is used in hospital emergency rooms, on battlefields, and at disaster sites when limited medical resources must be allocated.*

Koopmans then applies the medical definition to computer forensics, resulting in computer triage being defined as: "A process of sorting computer systems into groups, based on the amount of relevant information or evidence found on these computer systems". Casey, Ferraro et al. (2009) defines triage as a "[t]argeted review of all available media to determine which items contain the most useful evidence and require additional processing". In Rogers, Goldman et al. (2006), and later expanded by Mislan, Casey et al. (2010), triage is a process, normally performed on-scene, that is used to:

1. assess the severity of a crime and prioritizing it accordingly;
2. assess the offender's possible danger to society;
3. obtain actionable intelligence in exigent circumstances (e. g., missing person, military operations, risk of evidence destruction);
4. identify the richest sources of digital evidence pertaining to an investigation;
5. identify victims that are or may be at acute risk;
6. identify potential charges related to the current situation; and
7. determine whether a certain item requires deeper inspection, such as recovery of deleted information or decoding of encrypted data.

Generally, the key point with digital forensic triage is that triage can be used to identify the 'richest sources' of digital evidence at the scene, allowing the detected media's in-depth examination to be focused and expedited. Triage is an extremely high-level examination, designed to be fast, not thorough. Under this philosophy, and Koopmans' definition, triage is not designed to normally remove media from needing further examination, and only allows for a shallow, focused view of found evidence – aka low hanging fruit – that helps in the prioritization of all the media. However, some organizations, such as ADF Solutions (2011), claim that "triage can identify and eliminate negative computers with the same degree of confidence as can full forensic examinations". Differences in definition have caused some confusion within the digital forensic community. This claim, however, is very different technically and

philosophically from triage as previously defined, and should instead be considered as a preliminary forensic examination.

3.2.2 Preliminary Forensic Examination

A more in-depth examination than triage, preliminary forensic examinations have "the goal of quickly providing investigators with information that will aid them in conducting interviews and developing leads" (Casey, Ferraro et al. 2009). Depending on the results found during triage, the type of case, and the policy of the department, a preliminary analysis might be deep enough to justify discontinuing analysis on that particular media if nothing was found. Because preliminary analysis is slightly more in-depth, it usually takes longer than triage tasks, and as such may or may not be conducted on-scene. Several tools with various preliminary analysis workflows exist, such as SPEKTOR[6], ADF Triage-Examiner[7], and the Rapid Evidence Acquisition Project[8]. Most are designed to be easy to use, and are highly automated. This allows first responders with minimal training to be able to conduct preliminary analysis tasks. Again, depending on the organization's policy, if evidentiary artifacts are found during a preliminary analysis, then the media will continue to receive an in-depth analysis with a focus on the already discovered artifacts. If no artifacts were found, then further analysis may not be necessary.

A common concern with making decisions to discontinue analysis on a piece of suspect media based on the results of a preliminary analysis is that the analysis is not comprehensive enough to make informed decisions, and possible evidence could be missed. Studies have shown that decisions to continue or discontinue an analysis based on the results of a pre-planned, standardized preliminary analysis could be accurately made that did not exclude suspect media that contained evidence (Goss 2010; James 2012). However, preliminary analysis tasks must be tailored specifically for the particular case type, and some tasks, such as file hash comparison, are easier to automate than others.

---

[6] For more information about the SPEKTOR project, see: http://www.evidencetalks.com/

[7] For more information about ADF Triage Examiner, see: http://www.adfsolutions.com/

[8] For more information about the Rapid Evidence Acquisition Project, see: http://cybercrimetech.com/

3.2.3 In-Depth Forensic Examination

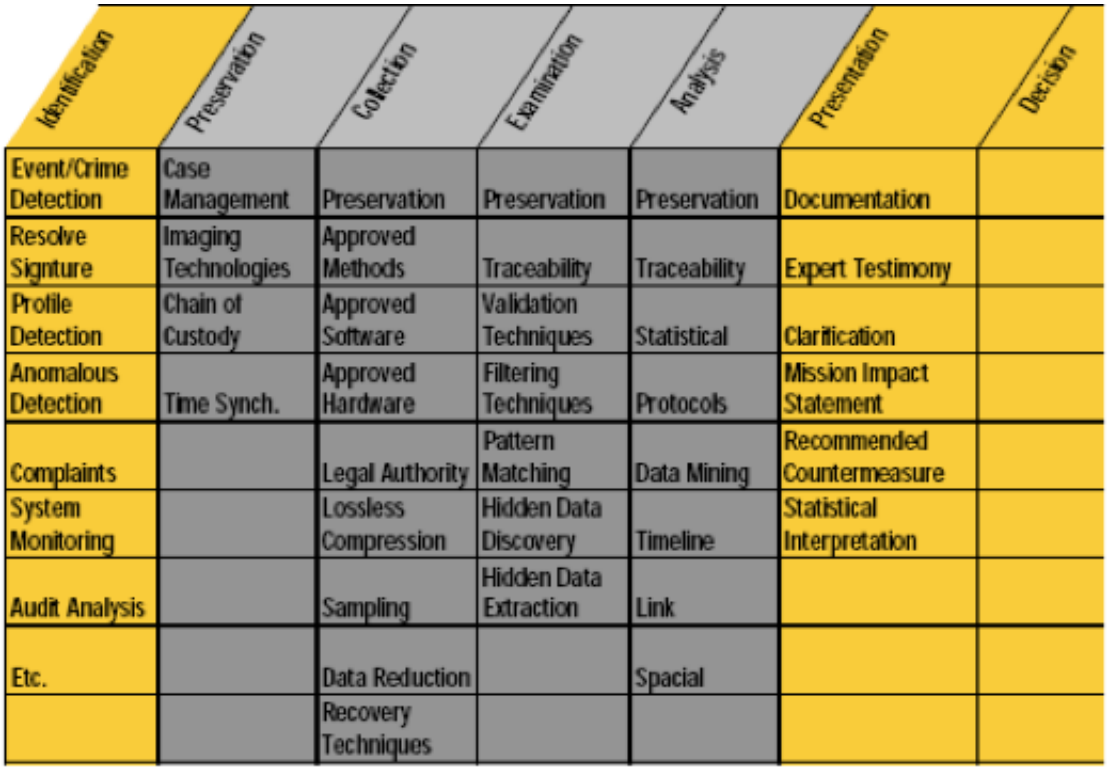
An in-depth forensic examination is usually an analysis conducted in a digital forensic laboratory with suspect media that has been seized, and is much more thorough than the previous tiers of examination. Because of this, in-depth forensic examination also takes a considerable amount of time longer (Goss 2010). Also, because of complexity of high-level tasks, this level of examination is more manual than previously discussed levels. Casey, Ferraro et al (2009) defines an in-depth forensic examination as a "[c]omprehensive forensic examination of items that require more extensive investigation to gain a more complete understanding of the offense and address specific questions". This can be considered the standard depth of analysis. The type of case and the used digital forensic investigation process model dictates if the previous two tiers of examination will be used. There will rarely, if ever, be a case where an in-depth analysis is not necessary to answer questions about some media.

3.3 Digital Forensic Investigation Process Models

"[T]he reality is that there is no single process for digital forensics" (Carrier 2008). Various process models have been proposed, and commonly used models will be briefly discussed; however, as stated before, there is no one accepted standard, and the majority of organizations are creating their own standard operating procedure, which may or may not be based on an existing process model.

3.3.1 Digital Forensic Research Workshop 2001

In 2001 the Digital Forensic Research Workshop identified an investigative process for Digital Forensic Science (Palmer 2001). This process included identification, preservation, collection, examination, analysis, presentation, and decision phases (Figure 3.1). The items in gray were the most agreed upon, and "were identified by the attendees as core processes [of digital forensic investigations]" (Pollitt 2007).



Identification	Preservation	Collection	Examination	Analysis	Presentation	Decision
Event/Crime Detection	Case Management	Preservation	Preservation	Preservation	Documentation	
Resolve Signature	Imaging Technologies	Approved Methods	Traceability	Traceability	Expert Testimony	
Profile Detection	Chain of Custody	Approved Software	Validation Techniques	Statistical	Clarification	
Anomalous Detection	Time Synch.	Approved Hardware	Filtering Techniques	Protocols	Mission Impact Statement	
Complaints		Legal Authority	Pattern Matching	Data Mining	Recommended Countermeasure	
System Monitoring		Lossless Compression	Hidden Data Discovery	Timeline	Statistical Interpretation	
Audit Analysis		Sampling	Hidden Data Extraction	Link		
Etc.		Data Reduction		Spacial		
		Recovery Techniques				

Figure 3.1 Digital Forensic Research Workshop 2001 diagram of the digital investigation process where the gray area is the area of focus for the workgroup: Palmer, G. (2001). DFRWS Technical Report: A Road Map for Digital Forensic Research. Digital Forensic Research Workshop. G. Palmer. Utica, New York.

3.3.2 National Institute of Justice

The U.S. Department of Justice (DoJ) National Institute of Justice (NIJ), created an electronic crime scene investigation guide for law enforcement in 2001. In this work a four-phase process model was proposed. This model, however, was later expanded in the second edition of the guide (NIJ 2008). The overall model consists of preparation, preservation, documentation, collection, examination, analysis, and reporting.

- Preparation – knowledge about the types of devices commonly encountered, potential evidence sources, investigative tools, and equipment for collection, packaging and transportation of electronic evidence
- Preservation – securing and evaluating the crime scene; ensuring the safety of persons and protecting the integrity of all evidence
- Documentation – documentation of the scene, and electronic evidence
- Collection – "the search for, recognition of, [and] collection of… electronic evidence"

- Examination – "helps to make the evidence visible and explain its origin and significance"
- Analysis – "looks at the product of the examination for its significance and probative value to the case"
- Reporting – "A written report that outlines the examination process and the pertinent data recovered completes an examination"

3.3.3 National Institute of Standards and Technology

The National Institute of Standards and Technology proposed a four-phase model in the special publication 800-86 (Kent, Chaevalier et al. 2006). This model includes collection, examination, analysis and reporting phases (Figure 3.2).

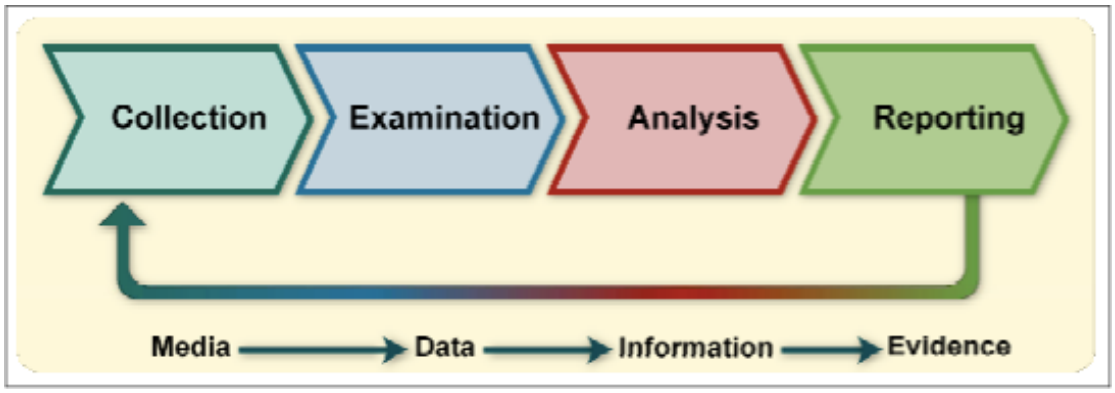


Figure 3.2 National Institute of Standards and Technology, four-phase digital investigation model proposed in SP 800-86: Kent, K., S. Chaevalier, et al. (2006). Guide to Integrating Forensic Techniques into Incident Response, National Institute of Standards and Technology: 121.

In this model each phase is defined as such:

- Collection - "identify potential sources of data and acquire data from them"
- Examination – "assessing and extracting the relevant pieces of information from the collected data"
- Analysis – "study and analyze the data to draw conclusions from it"
- Reporting – "preparing and presenting the information resulting from the analysis phase"

3.3.4 Common Process for Incident Response and Computer Forensics

Freiling and Schwittay (2007) proposed a 3 phase investigation model, named "The Common Model", that seeks to combine incident response and digital forensic tasks when an incident occurs to ensure that the incident may be investigated after incident response and mitigation has taken place. During incident response, data relevant to the investigation of the incident may be deleted or otherwise destroyed, intentionally or not, by the suspect or incident responder. By integrating digital forensic tasks into the incident response workflow, data relevant to the investigation may have been collected before being destroyed.

This model consists of pre-analysis, analysis and post-analysis phases. Each phase has several sub-phases. An analysis in the context of this phase is defined as when "compromised hosts or data are reviewed in detail with the intention to reconstruct the reason for the security incident in question".

The pre-analysis consists of a pre-incident preparation sub-phase, which includes development of an incident response plan. An incident response plan should detail organization policies on what is acceptable or not. These policies will be the standard from which behavior can be tested as a security incident. Next, the incident detection sub-phase takes all events as an input (from digital and physical sources), of which suspicious events raise an alert. If an alert is raised, the initial response phase confirms the incident, or discards the suspicion based on collected information. As information is collected, a response strategy is developed to attempt to approach, mitigate and investigate the incident.

Once a response strategy has been developed, the analysis phase begins. The analysis phase includes live response, forensic duplication, data recovery, (meta-data) harvesting, and data reduction and organization tasks as sub-processes. Once data has been collected and organized, the analysis sub-phase attempts to determine the reasoning behind the incident. This phase "…tries to answer the questions of what happened, when, how did it happen, and who is responsible".

The final phase is the post-analysis phase. Post-analysis includes reporting the details of the incident, and resolution (and further prevention) of the incident.

3.3.5 Integrated Digital Investigation Process

Carrier and Spafford (2003) proposed a investigation process model that considered the physical investigation along with the digital investigation (Figure 3.3). This model, named the Integrated Digital Investigation Process (IDIP), takes a holistic approach to crime scene investigation, where the physical crime scene reconstruction (Figure 3.4) affects, and is affected by, the digital crime scene reconstruction (Figure 3.5) to form a complete theory of happened events. Unlike the previous models, the IDIP feeds back between the physical and digital investigation phases. However, the overall model can be described as: Readiness, Deployment, Preservation, Survey, Documentation, Search, Reconstruction, Presentation, and Review.

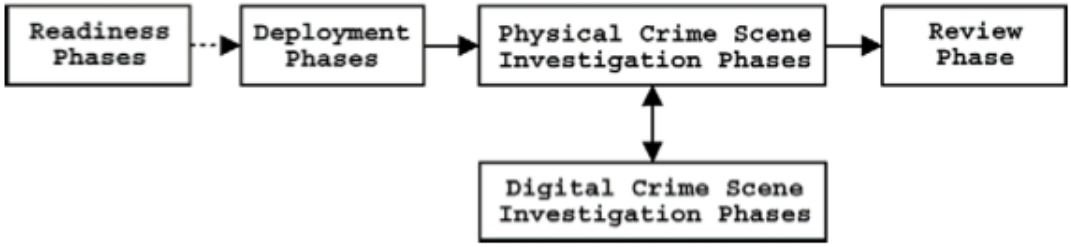


*Figure 3.3 Integrated Digital Investigation Process Phases: Carrier, B. D. and E. H. Spafford (2003). "Getting physical with the digital investigation process." International Journal of Digital Evidence **2**(2): 1-20.*

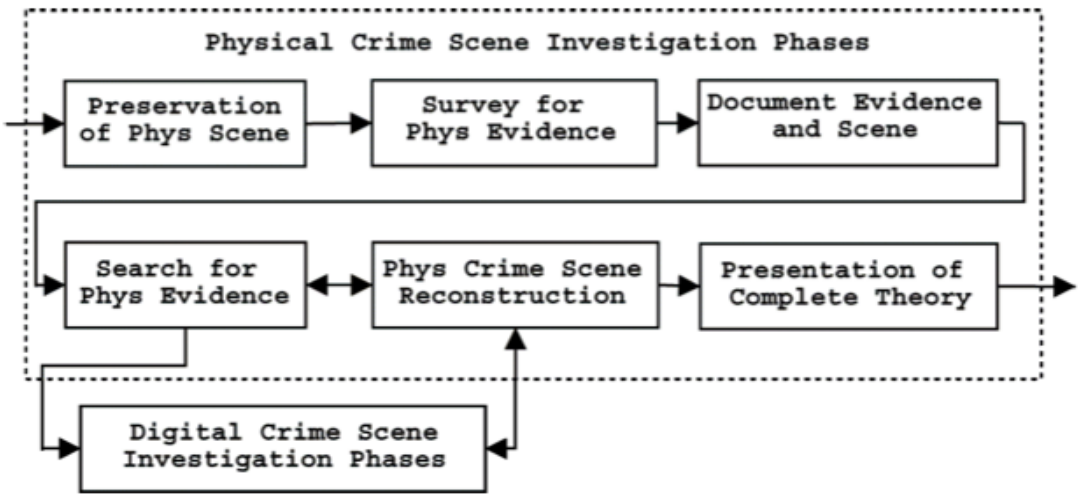


Figure 3.4 Integrated Digital Investigation Process breakdown of physical crime scene investigation phase

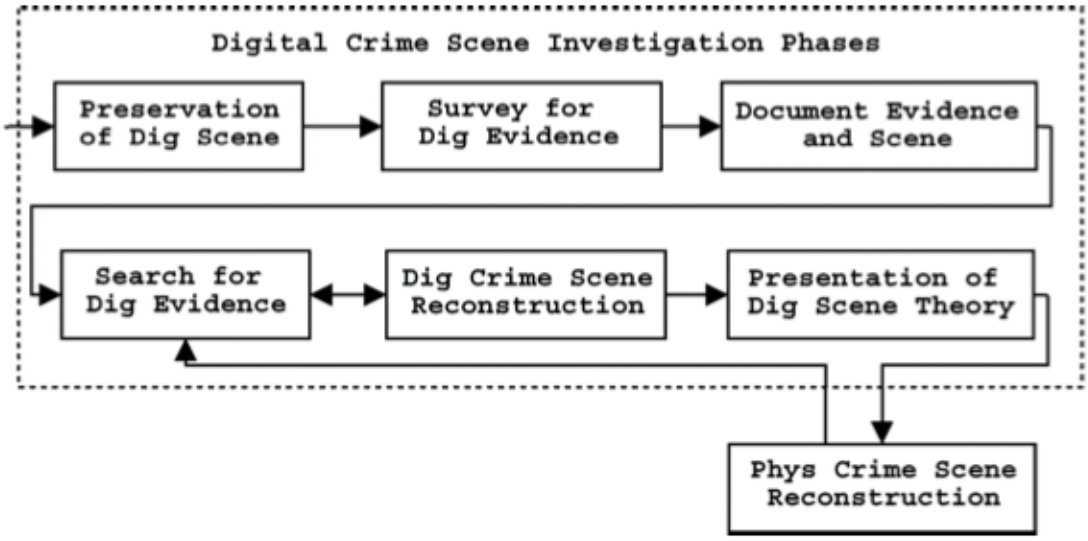


Figure 3.5 Integrated Digital Investigation Process breakdown of digital crime scene investigation phase

All of the previously discussed process models have a common core. While each model has components that will continue to be debated, each model generally attempts to secure suspect data, analyze the secured data, and report on the findings in a way that is admissible in court. To accomplish this, preparation beforehand is necessary, as is meticulous documentation throughout the entire process. A particular model may be better suited to a certain organization, but generally each model will accomplish these main goals. One issue is in the standardization and verification of models used. Ensuring that the investigation and examination processes are valid is a topic of concern that can only begin to be alleviated when rigor is introduced into each phase of a digital forensic investigation. In each of the models an analysis or reconstruction phase exists. For the purposes of this work, the scope will be limited to the examination and analysis phases of an investigation.

3.4 Current Digital Evidence Analysis

When conducting a digital forensic investigation, analysis is possibly the most complex phase in any investigation model. This is because during a digital forensic analysis the meaning of the state of the suspect data must be considered. Determining meaning entails reasoning about the data. When an investigator observes a system, and the information that is represented, they are able to use their knowledge of the system – which may be incorrect or incomplete – to reason about what the information means in terms of the hypothesis to be proved. Reasoning about the

represented information is a complex, knowledge-intensive task. Further, some investigators claim that intuition plays a part in their investigation. The complexity of interpreting what observed information actually means (or doesn't mean) in the context of an investigation, and the challenge of defining and emulating an investigator's intuition, make fully comprehensive automatic analysis software impractical. For this reason, current digital evidence analysis is highly manual, using software to quickly interpret data and display information that digital forensic investigators can then manually reason about.

3.4.1 Inference Processes

During an investigation, a digital forensic investigator attempts to reconstruct happened events. As previously mentioned, a digital investigator uses observed information to support new facts. This is done by observing the sate of the interpreted data (information), and reasoning about what the observed information means based on their prior knowledge of the system. This process is defined as the process of inference.

"Inference signifies merely the process of thinking about a piece of evidence, not the result" (Anderson and Twining 1998). To clarify, inference is not the derived fact itself, but rather the reasoning process from known facts that produces new facts or conclusions. This work will focus on two common modes of human reasoning used to infer new information: deductive and inductive reasoning.

Deductive reasoning is a process of reasoning from a general case or rule to a specific case. Grinnell (1993) states that "…deductive logic consists of forming a theory, making deductions from the theory, and testing those deductions, or hypotheses, against reality". "A deduction… proceeds from a generalization to a specific case, and that is generally what happens in forensic practice" (Thornton and Peterson 1997). Inductive reasoning, on the other hand, begins with a specific observation, and builds on these observations to make more general statements (Grinnell 1993). "An inductive argument… is where the conclusion is made likely, a matter of some probability, by offering supporting conclusions. It is at best a prediction about what might be true" (Turvey 2008).

If a digital forensic investigator has in-depth knowledge of how the system functions, inferences can be made to derive facts from observations that allow reconstruction of

the events in the system that lead up to the final observed state. The truth of these derived facts, however, may not be certain due to incomplete knowledge or due to a limitation of observable information.

3.5 Event Reconstruction

Crime scene reconstruction is necessary because an investigator may have no knowledge about what happened at the scene of the crime. By reconstructing the crime scene with artifacts found at the scene, an investigator can potentially determine the happened events and give context to those events. Physical crime scene reconstruction is defined as "[t]he use of scientific methods, physical evidence, deductive reasoning, and their interrelationships to gain explicit knowledge of the series of events that surround the commission of a crime" (ACSR 2006). This definition can be directly applied to digital crime scene reconstruction by substituting physical evidence for digital evidence. The resulting definition is:

> *Digital crime scene reconstruction is the use of scientific methods, digital evidence, deductive reasoning, and their interrelationships to gain explicit knowledge of the series of events that surround the commission of a crime.*

When reconstructing a crime scene, the reconstruction is divided up into events. An event in this work is defined as an occurrence that causes a change in the state of an object or system. Event reconstruction is done in both physical and digital investigations. For example, when investigating a fire, the progress of a fire can be determined by using a technique termed "arc mapping" (NFPA 2005). Arc mapping is the process of reconstructing particular events; in this case, arcs in electrical circuits caused by the heat of the fire. By determining where particular events have occurred, the progression and origin of the fire can be determined. This is similar to crime scene reconstruction. In order to determine what has happened, clues from the scene, as well as reasoning about observed evidence, are used to reconstruct the happened events. When more events have been identified, a more comprehensive picture of the overall crime scene reconstruction is possible. Since event reconstruction deals with partially known facts, the investigator must use reasoning about observed information to help fill in the gaps of unknown information.

3.5.1 Challenges with Traditional Event Reconstruction

In law enforcement, investigators rarely, if ever, have a complete picture of exactly what happened at a crime scene. When attempting to reconstruct the events that led up to an incident, investigators have to contend with a number of factors that make event reconstruction difficult, and sometimes impossible. The first of which is the issue of incompleteness. Usually artifacts that provide information about what has taken place are available at the crime scene, or on the suspect or victim. These artifacts – either through malicious activities or simple evidence dynamics[9] – do not explicitly give a complete picture about the crime that has taken place; this knowledge must be inferred from the collection of artifacts and conclusions that can be made. For example, finding a human hair on a murder victim does not allow for the assumption that the owner of the hair is the murderer. It does, however, allow an investigator to infer that the owner of the hair and the victim has come in contact some time in the past. The investigation then can continue using this new, but still incomplete, knowledge.

The next issue investigators face when attempting to conduct event reconstruction is the use of incorrect facts on which his or her inferences are based. For example, assume a witness, who was thought to be reliable, was mistaken about seeing Mr. X at the scene of the crime at the time of the murder. Based on this information, it could be inferred that if Mr. X was at the scene of the crime at the time of the murder, then Mr. X has knowledge of the murder. However, this may be a false conclusion. The initial fact, that Mr. X was at the scene of the crime, may be false, meaning that the conclusion that follows could also be false.

Similarly, facts from which inferences are based are very dependent on the way humans perceive the world. "… [P]eople are not very good at [reasoning] because we tend to mix up the semantics with the reasoning process itself, and so do not always reach a valid conclusion" (Giarrtano and Riley 2005). Likewise, cognitive bias very much depends on cultural and environmental factors that naturally make humans more or less biased in a particular situation. "… [M]any questionable and erroneous

[9] Evidence dynamics is an influence that changes evidence regardless of intent, such as rain washing away blood. Chisum, W. and B. Turvey (2000). "Evidence dynamics: Locard's exchange principle & crime reconstruction." Journal of Behavioral Profiling **1**(1): 1-15.

beliefs have purely cognitive origins, and can be traced to imperfections in our capacities to process information and draw conclusions" (Gilovich 1993). For example, bias against race, creed or gender can stem from stereotypes about these groups being accepted as fact, and can cause a perceived increase in the likelihood of a certain act when there is no statistical proof to back up this conclusion. Even training and experience in law enforcement can allow an investigator to have a perceived increase in capability to identify when a suspect is lying, when studies would suggest that there is actually little to no increase in deceit judgment accuracy (Meissner and Kassin 2002).

3.6 Summary

This chapter has established terminology and concepts of digital forensic investigations that will be used throughout this work. General phases of a digital forensic investigation are explained, and commonly used digital forensic investigation process models were briefly discussed. Focus was then shifted to the analysis, or knowledge acquisition, phase of an investigation, and specifically the process of human event reconstruction. Finally, issues with traditional event reconstruction were examined.

Chapter 4

Challenges with Automation in Digital Forensic Investigation

The use of automation in digital forensic investigations is not only a technological issue, but also has political and social implications. Chapters 4 and 5 are provided to give further context about the state of the digital forensic community's acceptance and attitudes towards automation in digital forensic investigations. This chapter discusses some challenges with the implementation and acceptance of automation in digital investigation, and possible implications for human investigators. Attitudes towards the use of automation in digital forensic investigations are examined, as well as the issue of digital investigators' knowledge acquisition and retention. The argument is made for a well-planned, careful use of automation going forward that allows for a more efficient and effective use of automation in digital forensic investigations while at the same time attempting to improve the overall quality of expert investigators.

4.1 Attitudes Towards Automation

Highly automated digital forensics – sometimes referred to as "push-button forensics" (PBF) – receives much criticism from the digital investigation community. Criticisms generally appear to focus on two aspects of digital investigations: a deterioration of expert knowledge by an overreliance on PBF, and a perceived less thorough, or lower quality, investigation when relying on a high level of automation. However, critics also currently accept a certain level of automation to help them in their daily tasks. Manually comparing each hash in a hash database, for example, would be impractical without some level of automation. The challenge comes when higher-level processes, such as analysis, are being automated, and also when the investigator begins to loose understanding of the underling concepts of the investigation. Both of these scenarios are currently happening to some extent. Digital investigation software suits such as EnCase (Guidance 2010), Forensic Tool Kit (AccessData 2010), Autopsy Forensic Browser (Carrier 2010), and others allow an investigator to conduct preliminary, and even some complex investigation tasks simply by knowing which button to press. These popular tools endeavor to make the job of the investigator easier, or even remove the expert altogether, as seen in a claim from Access Data (2009): "Digital investigations are no longer the exclusive domain of highly trained experts". Full-

featured forensic software suites are not the only potential source of issues. Simple programs such as TraceHunter (Zhu, James et al. 2010) provide correlation, interpretation, and some analysis of the Windows Registry regardless of the practitioner's knowledge, and the same can be said for scripts written by experienced investigators that are then distributed to others who may have little to no knowledge of the underlying processes and data sources. Despite criticisms, the reality today is that investigators are currently using a high level of automation in investigations, and newer, or uninterested, investigators who are trained on a specific tool may be unable or unsure how to do investigations without the use of automation. This is contrary to the view that every investigator should have a solid, non-tool-centric knowledge of the investigation process. Whether this lack of knowledge comes from a lack of training, time or funding, it has real implications on the quality of investigations being conducted.

Kovar (2009a; 2009b), considers the value of push button forensics, and cites three main reasons for the acceptance of increased automation: The tool vendors' need for the expanded non-expert market will evolve push-button interfaces, speed-related financial interest from consumers of computer forensics services (law enforcement, private sector, etc.), and the growing volume of digital evidence resulting in case backlogs. The general reply to this claim was that practitioners must understand "…the science, logic, and art behind the PBF tools" (Kovar 2009b). The admissibility of evidence found purely from a technician running a tool was also questioned. Kovar claims that many technicians are already using push button tools as part of the computer forensic process. He ultimately concludes, "There clearly are risks to using PBF and inexperienced examiners inappropriately, but through sound business practices they can safely contribute to our projects and improve our efficiency in the process". However, no mention is given to how PBF tools should be implemented in the investigation process to ensure quality. Kovar is also focusing only on inexperienced examiners, whereas improperly implemented automation can become an issue for experienced and inexperienced alike, if not managed properly.

Similarly, Casey (2006) states that "too little knowledge is a dangerous thing" in regards to digital investigations. He claims that inexperienced internal or outsourced investigators may have an "…over reliance on user-friendly or automated forensic software", and may "…apply a form of pseudo-automation by rigidly following

predefined protocols". Casey also states that, "Inexperienced individuals who do not critically review the results of a tool will inevitably misinterpret or completely miss digital evidence". He cites the limitations of self-assessment, and calls for set standards in digital investigation competencies and processes.

While each point may be valid, the reason for the use of less-experienced investigators, reliance on automation, and a lack of certification and thorough checks of labs and contractors is simply an issue of time and cost. Some organizations may have the resources to be able to use certified labs and highly trained and experienced investigators, but law enforcement is a tradeoff between the cost and results, known as a 'balance scorecard' (Ashworth 2010). "[Hiring specialists] would force law enforcement agencies to incur great expense, perhaps a crushing expense for smaller police departments that already face tremendous budget pressures" (US v. Comprehensive Drug Testing, 2009). If Law Enforcement (LE) gets satisfactory, admissible results from a non-certified lab or automated software in half the time and cost, then management will have to judge whether the cost savings outweigh the perceived cost to justice, which also raises the challenge of precision measurement in digital investigations. This is an unfortunate reality that may be helped with improved performance measurements and standardization, such as the standards proposed by Lee (2008) and the UK Forensic Science Regulator (2011), but will never be completely solved; potentially resulting in missed digital evidence.

Another challenge is the definition of an experienced investigator. Experience is not the same as competence. Irons, Stephens, et al. (2009) specifically differentiate between practice and theory as well as skills and knowledge when dealing with the training of competent digital investigators, claiming that each area needs to be developed to ensure competency. However, neither Irons nor the UK Forensic Science Regulator focus on the investigator's retention of learned theory and knowledge; only present performance and undefined technical skill maintenance. Similarly, most studies assume that all investigators or technicians are also interested in their job, and are free of possibly undiagnosed psychiatric and learning disorders that can inhibit or prevent learning and retention beyond the basics required for the

job[10] (Goldstein 1997; Wender, Wolf et al. 2001). This is a bold assumption considering "only 51 percent [of Americans] now find their jobs interesting" (CBS 2010), and approximately 1 in 4 adults in America suffer from a mental disorder, some of which may affect learning and retention (Kessler, Chiu et al. 2005). In these situations, potential evidence may be missed due to disinterest or lack of retention of knowledge.

Despite skepticism, automation is already being used in digital investigations. Digital forensic triage, for example, is a rapidly growing and highly automated area. Many works (Rogers, Goldman et al. 2006; Casey, Ferraro et al. 2009; Goss 2010; Koopmans 2010; Mislan, Casey et al. 2010) have called for advances in computer and mobile phone triage because they realize the benefits of fast, automated, on-scene intelligence gathering. These benefits have been examined within a UK high-tech crime unit in Goss (2010), which showed a reduction in the amount of seized computers and suspect data needing an in-depth analysis. Goss also compared automated triage performance with manual investigation finding that triage gave comparable examination results in a fraction of the time for specific case types where in-depth knowledge is not required, such as child exploitation image detection. In more complicated cases, where knowledge of the system is necessary for analysis, triage was less reliable, and often missed potential evidence.

While triage has been shown to have definite benefits in some specific cases, there are still challenges, such as investigator training, potential missed evidence and verification challenges that need to be addressed.

4.2 Knowledge

Ianuzzi (2007) claims that one challenge with automating digital investigations is that automation may "dumb down the profession". This parallels the idealistic view, shared by some investigators and researchers, that most investigators have already achieved an undefined "good" standard in forensic knowledge. This seems to be a dangerous assumption, but is commonly cited as an argument against automation. There are currently unqualified, unknowledgeable forensic practitioners that are

[10] For more information on workplace disability anti-discrimination legislation in the US and UK, see the Americans with Disabilities Act of 1990 (ADA) and the Disability Discrimination Act 1995 (DDA)

conducting digital investigations (Jones 2004; Everett 2005). This may be true for practitioners who were chosen for the job because they had basic computer skills, to people with 10+ years experience that generally have no interest in learning anything but the minimum required to keep their job.

Just like any profession, expansive expert knowledge cannot come only from the field; it must also come from continued formal and informal study. As stated by Gogolin (2010), "digital skills are perishable if not kept current", and 75% of investigators receive between zero (30%) and 5 days annual training. Casey (2009) affirms that "there are some fundamental principles, concepts, and skills that everyone in this field must abide by and know". Likewise, Irons, Stephens et al. (2009) claim, "the implicit expectation is that digital investigators should be competent before undertaking any digital investigation duties". But this is not always the reality. Many investigators have little to no digital investigation training before starting, and even after, "[o]nly 34% of [digital forensic] investigators [surveyed in Michigan, USA] received formal training in laboratory forensics, with the majority being trained 2 weeks or less" (Gogolin 2010).

Likewise, EURIM (2004) claimed that in 2004 "…barely 1,000 [police officers in the UK] have been trained to handle digital evidence at the basic level and fewer than 250 are currently with Computer Crime Units or have higher level forensic skills". Neil Hare-Brown claimed that as much as 20% of digital investigation consultants in the UK are "unfit" (Everett 2005); however, "fitness" in this case is undefined, and many initiatives since 2005 may have improved this estimation. This lack of definition for investigator fitness is industry-wide since "there is an absence of standards and competencies in the field of cybercrime" (Jones 2004). However, efforts for standardization are being made (Lee 2008; HomeOffice 2011). Also, consider that these statistics are from the US and Europe; areas that generally have money to invest in standards and training. This may mean that countries with less funding receive even less quality training, and may have less access to education, equipment and skilled investigators in general. The result of this global, and even national, imbalance in funding, training, aptitude, interest, intelligence and skill results in few experts, many technicians and a huge variation in the quality of digital investigations world wide.

A lack of knowledge can lead to missed evidence and incorrect conclusions, and based on the given statistics this is likely happening without the use of automation. A "dumbing down" of the profession may happen if investigators with expert level knowledge over-rely on automation for their analysis, but for untrained investigators with little-to-no knowledge to begin with, automation may enable them to find and evaluate evidence that may have otherwise been overlooked.

4.2.1 Digital Investigation Training

In a report by the High Technology Crime Investigation Association (HTCIA) (2010), training for practitioners was a primary concern, claiming that 73% of respondents believe they do not receive enough training, especially in digital forensics, online investigations, and computer and network security. "Many respondents did not indicate that more personnel were necessarily needed, but rather believed more training at all levels was important." This parallels a study done by Gogolin (2010) that showed the majority of digital investigators interviewed were not receiving formal training before or during their employment. The result is that "…it typically takes between one and two years of on-the-job training before a newly minted forensics examiner is proficient enough to lead an investigation" (Garfinkel 2010). These studies indicate that more training is desired, but there are some challenges, especially funding, time and training quality, that are hindrances to investigator development.

4.2.2 Training Quality and Retention

One challenge is simply a lack of quality investigator training, partially caused by the absence of a "…common agreement on the sets of skills for which training would be most beneficial" (EURIM-ippr 2010). Without a standard digital investigation common body of knowledge (CBK), such as the work proposed by (Lee 2008), colleges and other training organizations are able to design digital investigator training based on marketing rather than industry-driven quality. Many organizations have started programs specifically to capitalize on the current popularity of crime investigation TV shows, and the increased interest in the field based on the show, known as the "CSI effect". "The CSI effect is leading many high school and college students to take forensic science or criminalistics courses and prepare for careers in the field" (Dempsey and Forst 2009). When education programs have no standard to which they must adhere, it becomes difficult for new students, and possibly

employers, to ensure they are getting a quality education. A standard body of knowledge or standard of quality, however, should not be confused with an educational content limitation. The aim of education is to prepare independent thinkers, but those thinkers should also be well versed in their respective areas. This is not always the case. As stated by Jones (2004), "we all know of training courses where there is no independent assessment of the student knowledge, yet the certificates issued to the students (after payment of the course fee) lend them credibility".

The same can also be said for vendor training with no third-party assessment where the certification of an unfit customer and what is best for the vendor's business could be a conflict of interest. Also, in other fields it has been shown that short, intensive training sessions, such as those offered by many certification bodies, produce gains in knowledge that are acceptable to immediately certify, usually at the end of the training session (McGguire, Hurley et al. 1964; Wik, Myklebust et al. 2002), but the majority of the knowledge is not retained after 6 months, as illustrated in Figure 4.1. "The data strongly suggest that in the absence of opportunity to practice the recently enhanced skill under conditions where critical evaluation of performance is available, individual gains are not maintained over a period of several months" (McGguire, Hurley et al. 1964). Since some basic and many advanced forensic concepts are not used on a daily or even weekly basis, intensively trained and certified investigators could eventually have credentials that certify them beyond the scope of their understanding or skill. This situation is made worse when a trained investigator is able to rely on highly automated tools, rather than utilizing their own knowledge. Automation in this case may allow unknowledgeable investigators to appear acceptable, and may cause a deterioration of knowledge in knowledgeable experts.

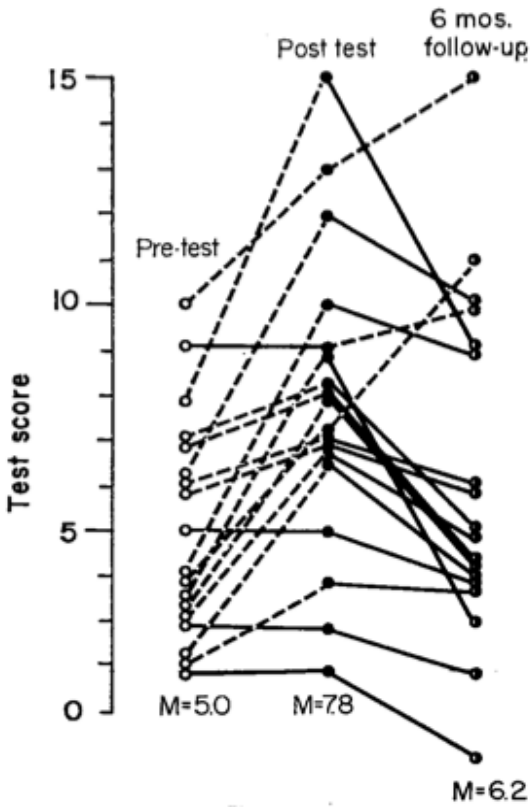


*Figure 4.1 A diagram of scores before, immediately after and 6 months after a short period intensive learning, showing immediate gains and long-term retention. McGguire, C., R. E. Hurley, et al. (1964). "Auscultatory Skill: Gain and Retention after Intensive Instruction." Journal of Medical Education **39**(2): 120-131.*

4.2.3 Time

Along the lines of intensive training courses, time for training is a factor. Consider that in 2004 the United Kingdom digital crime investigation backlog was 6 to 12 months (EURIM-ippr 2004), and rose to 18 to 24 months in 2009 (2009) before being improved through a number of policy, case prioritization and evidence outsourcing initiatives (Kohtz 2011). Currently in the United States there are reports of backlogs from 12 to 18 months (Raasch 2010), and in some cases “approaching or exceeding 2 years” (Gogolin 2010). With a growing backlog, investigators are pressured to work faster, which could mean less time to study underlying concepts, even informally. Likewise, during discussions few investigators have reported being up to date on research topics that could improve their processes. If a technician can quickly be trained to extract information without receiving in-depth training about the underlying concepts, and the information they extract is still admitted into court, then, as previously stated, management will have to judge whether the time and cost savings outweigh the perceived cost to justice.

4.2.4 Funding

Funding is yet another rival to knowledge. “Most cybercrime training funding comes out of budgets rather than assistance [grants]” (HTCIA 2010), which means needed training is competing with department maintenance, new equipment, software, budget cuts, increasing amounts of data as well as expanding job scope. “No agency in the study reported an increase in funding for digital crime investigation over the previous year” (Gogolin 2010). Casey, Ferraro et al. (2009) observes “few [Digital Forensic Laboratories] can still afford to create a forensic duplicate of every piece of media and perform an in-depth forensic examination of all data on those media”. This may force departments to choose purchasing a software suite and rely on its automated features, rather than have trained experts but not enough software licenses. A common training model appears to be purchasing a certain tool, sending one person to receive training, and then having this person train others when they return. This method could be effective, but as seen before, a lack of understanding or knowledge retention could lead to partial or incorrect knowledge being disseminated. Again, this is considering countries that have a budget to work with. In some countries the total annual budget for cybercrime investigation may be less than the cost of a single three-day training course from a popular forensic software vendors.

Ultimately, in the current investigation model tools with more features and a greater amount of automation will save time, but will consistently need to be upgraded and renewed. More tools will be needed to cover the expanding scope, and new equipment will be needed to support the tools. The technological needs of a department contend with the training needs, and products that require little to no training to use will be considered a cost saver.

4.2.5 Interest and Attitudes

Interest in the job and belief in its mission, as mentioned before, is also often taken for granted. This has a number of implications: First, if people put their personal desires before the good of the company, funding can be easily wasted, as seen in the US Congress (2004) where the cost of excessive flights outweighed the benefit of attending a conference. Second, with a mere 51% job satisfaction rate in the US (CBS 2010), and a +34% net job satisfaction rate in the UK (CIPD 2010), it would be naïve to think that law enforcement, and digital investigators specifically, are immune to uninterested employees. This means some investigators may be doing the minimum

required for the job, not for justice, but simply as a means to an end. Finally, overexposure to particular types of investigations could lead to categorization bias (Anderson 1991). Each of these implications could have a profound negative impact on the quality of investigations being conducted. By utilizing automation in digital investigations as it is currently implemented, uninterested persons may be able to conduct investigations with minimal effort that appear satisfactory but in fact lack in-depth expert analysis.

4.3 Certification and Licensing

Certification and licensing is a much-debated topic in information security and digital investigation. In the US, for example, some states require digital forensic examiners to be licensed as private investigators, while others do not. And "licensing requirements for forensic examiners have yet to be standardized on a national level…" (Manes and Downing 2009), which can lead to examiner licensing issues in multi-state cases. The American Bar Association (2008) is opposed to requiring private investigator licenses for forensic examiners, but "supports efforts to establish professional certification or competency requirements for such activities…". Other countries have yet to adopt licensing requirements for digital forensic examiners, and currently rely on the plethora of digital examiner certification programs. But again, training and certification will only be done in countries with the budgets to do so. And in those countries, without a defined standard, investigators and employers may find it difficult to differentiate between the levels of quality of various certifications.

4.3.1 Investigator Certification

The Digital Forensics Certification Board (DFCB), SANS, $(ISC)^2$, International Association of Computer Investigative Specialists (IACIS), The International Society of Forensic Computer Examiners (ISFCE), EC-Council, forensic software vendors, and many others offer certification for digital investigators. Some with little or no work experience requirements. In *Codes of Practice and Conduct for forensic science providers and practitioners in the Criminal Justice System* (2011) the term "competence" is often used, but nowhere is competence or competence testing explicitly defined, save references to National Occupational Standards (NOS) from Skills for Justice (2010), which claim to "describe competent performance in terms of outcomes [and] …allow a clear assessment of competence against nationally agreed standards of performance". However, NOS appear to be limited to overall general

forensic procedures, allowing for a standard procedure to ensure the proper consideration and handling of evidence, but with no digital examination-specific performance criteria. Beyond NOS, the Forensic Science Regulator (2011) has proposed to rely on "appropriate records of education, training, skills and experience in sufficient detail to provide evidence of proper training and formal assessment", which again leaves the definition of "appropriate" to each individual department. This lack of concrete definition, combined with an overwhelming amount of available certifications, can cause confusion among practitioners and employers, and allow incorrect evaluation of competency. "It's important to understand that certification does not mean mastery… In fact, certification doesn't necessarily even mean professional *competency*" (Huber 2010).

4.4 International Collaboration

Forensic practitioners and government officials often talk about LE knowledge, standards and competency from a national or regional perspective. However, Huber (2010) explained that "digital forensics is very much an interstate and international issue". The IT security sector is beginning to accept the need for global cooperation for incident response (Sambandaraksa 2010), but the forensics community has been slower in this regard. Some attempts at international cooperation involving investigator-sharing for digital forensic investigations have taken place (INTERPOL 2008), which have led to many questions of jurisdiction and acceptable procedure. However, investigations with cross-jurisdiction components happen often, with varying degrees of success. According to Brenner (2006) "countries' ability to assert jurisdiction over those who perpetrate cybercrime, both locally and transnationally, does not seem to be particularly problematic. …[The] issue is how to prioritize conflicting claims to assert jurisdiction over transnational cybercriminals". When considering standardization, unless there is an accepted global standard for digital investigations, the way evidence is handled in another country may not meet the legal or time requirements of requesting countries. Even if standards were put in place, practitioners in some countries may not always be able to meet these standards because of a lack of training, funding, etc. For this reason international collaboration is imperative to create non-conflicting standards. Global participation and investment will help build legal and technical infrastructures worldwide, and unified standards encapsulated in formally proven automated tools will help to streamline international

investigations, for trained and untrained alike, which may ultimately have a positive impact locally.

4.5 More Intelligent Automation

Since the definition of Digital Forensic Science at the Digital Forensics Research Workshop in 2001, the field has grown almost as dramatically as technology itself. As described by Casey (2009), digital forensic science is "coming of age", which not only brings about a maturation in the principal concepts of the field, but also an increased scrutiny against these principles and their lack of rigorous scientific backing. Investigators are now finding themselves overwhelmed with the scale and quantity of cases, along with the pressure of increasingly restrictive standards. This combination simply translates into a consistently increasing number of delayed or even neglected cases. To combat an "impending crisis in digital forensics", Garfinkel (2010) claims "advanced systems should be able to reason with and about forensic information in much the same way that analysts do today". This means automated processes that can interpret what observed information means rather than simply interpreting bits to human-readable information, as is currently being done.

4.5.1 Current State of Automation in Digital Forensic Investigations

As mentioned before, automation is already being used in digital investigations, and research is being done in this area. A high level of automation can already be achieved with evidence collection, processing, and to some extent, documentation, and a growing amount research is being done that attempts to automate analysis (Khan, Chatwin et al. 2007; Zhu, James et al. 2009; James, Gladyshev et al. 2010). This work tends to move from simply representing data as human-readable information to representing what the information means, and even reasoning about this meaning. Despite this, current tools still focus on converting data to information that an investigator can then use to manually draw conclusions. No software is perfect, and the ability for non-professional programmers to add on to this software allows even more room for error. James and Gladyshev (2010) found that only 23.3% of respondents verified the accuracy of examinations. From this and some of the points mentioned previously, it is likely that verification of the software is not always properly done, and sometimes not at all. Furthermore, verification of the non-professional add-ons could be regularly overlooked.

Regardless, no mention is usually given to what evidence is currently being missed during investigations. Because of budget restrictions, some departments only have and know how to use one piece of "full-function" software. This attitude shows an over-reliance on a specific piece of software. And this is where the one challenge for justice comes in; when investigators are previewing a system, they must get to a point where they decide if the case needs further analysis or not. Since time, money and justice are always competing, many technicians are taught how to run automated tools and then analyze the results themselves. If it can be assumed that the automation built into these tools is reliable – keyword list search, hash search, etc. – then there are really two challenges that remain: a reliance on the investigator to correctly run the automated tool and a reliance on an investigator, with possibly no knowledge of the data, to interpret the presented information.

Currently there are investigators that are not running all of the automated tools available to them. For example, during discussions, some examiners who make the decision to continue or stop the examination claimed they never use known hash databases in exploitation cases because hash comparisons "never find anything", and is therefore not worth the additional time. Both of these are challenges of knowledge, and perhaps policy, which have been discussed previously, but it is also about investigator attitudes. These challenges appear to be partially caused by the over-use of automation itself, but the issue is really what is automated, where automation is placed, and who is using it.

4.5.2 Opportunities with Automation in Digital Forensic Investigation

There are a number of opportunities to a higher level of automation at the preliminary analysis level. These include:

- **Standardized knowledge and investigations**

  Automation can be used for arduous manual tasks, such as hash matching, but also to encode the knowledge of trained investigators in a repeatable, verifiable way. "[C]omputer forensic software should ideally provide an objective and automated search and data restoration process that facilitate consistency and accuracy" (Guidance 2009). If automated first-responder processes were based on expert knowledge and standardized processes, higher quality first-level

investigations could take place than normally would with a less trained technician.

Automation is also an accumulation of expert knowledge. This bank of knowledge applied at the first step in the investigation may lead to less missed evidence by encoding knowledge of obscure evidence that a normal investigator would not necessarily know about. Not only does automation allow for standardized processes in a department, but the same standard of first-level investigation could take place everywhere in the world regardless of investigator or department knowledge and budgets. This could allow for first-level digital investigations in countries that would otherwise not have the capability. Ultimately, an automated first-level analysis applied at the first-responder stage of the investigation may allow for a faster, higher quality, standardized preview with little associated training investment.

- **Department load reduction and knowledge retention**

  Tasking the first responder with the automated first-level analysis benefits expert forensic investigators in a number of ways. First, since the preliminary analysis is being done in the field, less suspect machines will need to be analyzed by experts, as shown with current preliminary analysis techniques in (Goss 2010). This reduction in suspect machines will reduce the workload on the forensic lab. Along the same lines, every suspect machine given to the expert will definitely require a thorough analysis. This means that valuable expert investigators will no longer need to simply run automated tools, but will normally be conducting investigations that will require in-depth knowledge. “Field triage, evidence previews and even rudimentary evidence collection can free investigators and forensic examiners to focus on investigative and analysis activities” (HTCIA 2010). Not only could there be a reduction of suspect devices, but potentially a reduction in cases that require a laboratory examination at all. Many digital forensic investigators as well as the Mac Forensics Lab (2010) agree that since the first responder is obtaining information and potential evidence, it could allow the first responder to immediately secure a confession based on results found on-scene. This is, however, only if the officer understands that what he or she is looking at is, in fact, relevant to the case. For this, there needs to be an understanding about how crime is committed.

- **Increased Training and Education**

  Further, with the reduction in workload and the need for in-depth expert knowledge for all incoming suspect material, investigators will, potentially, have more time and much more need for quality digital forensic investigation education. Since the experts will be using high-level expert knowledge more than before, more information will be retained providing a higher return on investment and more knowledgeable, competent experts.

4.5.3 Challenges with Automation in Digital Forensic Investigation

The main challenge to automated tools is that they have neither a complete knowledge nor capacity to process this knowledge in a particular case. Because of this, if they do not show anything it does not mean that the evidence is not there. Ianuzzi (2007) and Goss (2010) believe that completely automated process cannot work in all cases; automated tools will miss some evidence. Missed evidence, however, is not only a problem of automation, but instead a problem of investigation in general. Investigators also cannot know how much evidence they are missing.

To approach this challenge, it must be known how well automated tools currently work, and in what situations they are best suited. A method for measuring a tool's performance is proposed in Chapter 5. Once a performance measure has been established based on the objectives of an investigation, weaknesses in automation may be objectively evaluated and improved upon. This must be able to include factors such as weight of the evidence in relation to the case, and accuracy of interpretation.

The second challenge is of the admissibility of automatically derived evidence by a non-expert. The UK Forensic Regulator (2011) has outlined that forensic software must be validated using specified, rigorous validation methods. Validation generally establishes that "the validation work is adequate and has fully demonstrated compliance of the method with the acceptance criteria for the agreed specification; and the method is fit for its intended use". Once an expert validates the tool, any user should then follow the tool's defined operating procedure. A non-expert investigator that followed the procedure, however, may be called to give expert testimonial about their findings. If they cannot establish that they have complete knowledge of how they arrived at their conclusions, this may introduce doubt and reduce the credibility of evidence derived using automated processes.

4.6 Implementing Advanced Automation in Digital Investigations

Automation may "dumb down" the profession when experts are able to rely solely on automated tools for all data interpretation and analysis tasks, and derived evidence is admitted in court without challenge. However, the technologies to be able to fully automate complicated evidential analysis and reasoning tasks without human intervention are not yet available, and courts admitting automatically extracted evidence without some sort of human expert verification appears unlikely.

Automation can potentially increase the speed of the investigation process, and reduce the number of suspect devices in the lab, which will ultimately reduce case backlog while avoiding bias and prejudice. Because speed is one of the biggest concerns during an examination, digital investigators need to re-evaluate the examination process, and consider faster ways to determine if an in-depth analysis is necessary, such as profiling (Marrington, Mohay et al. 2010) or automatic event reconstruction (James, Gladyshev et al. 2010).

Along with speed, the validity of automated processes must be formally proven. "Despite having a variety of practical techniques and tools, [Digital Forensics] provides little theoretical basis to support correctness of investigation findings" (Gladyshev 2004). Until the forensic community embraces formal methods, time consuming ad-hoc verification will continue, and the quality of the tools and investigations will suffer. Formal verification will also allow for the use and design of more efficient and intelligent forensic tools as called for by Garfinkel (2010).

Finally, improved measurements in regards to the identification of digital evidence are necessary. Until baseline performance measurements are created for digital investigations that account for evidence identification in the context of the case, performance of highly automated tools will unverifiable, and perhaps hinder development.

4.7 Summary

This chapter discussed some issues with the implementation and acceptance of automation in digital investigation, and the implications for human investigators. Attitudes towards the use of automation in digital forensic investigations have been examined, as well as the issue of digital investigators' knowledge acquisition and retention. The argument was made for a well-planned, careful use of automation

going forward that allows for a more efficient and effective use of automation in digital forensic investigations while at the same time attempting to improve the overall quality of expert investigators.

Chapter 5

Measuring the Accuracy of Investigations

Following from the previous chapter, one identified challenge is how to objectively compare the accuracy of highly automated analysis tools to real digital forensic experts. This chapter introduces a method based on information retrieval techniques to measure and compare the accuracy of tools, investigators and investigation processes. Related work is explored, and an argument is given why objective measurement of the accuracy of digital forensic investigations is necessary. A simple method of accuracy measurement is proposed as a starting point, and a brief case study is given to illustrate the proposed method.

5.1 Measuring the Accuracy of Analysis

In digital forensics, the verification and error rates of forensic processes are a common topic. This is mostly due to the evidence admissibility considerations brought on as a result of *Daubert v. Merrell Dow Pharmaceuticals*, 509 US 579 (1993). “The Daubert process identifies four general categories that are used as guidelines when assessing a procedure” (Carrier 2003). These are Testing, Error Rate, Publication and Acceptance.

Tools are commonly tested and organizations such as the National Institute of Standards and Technology (NIST) have created test methodologies for various types of tools which are outlined in their Computer Forensic Tool Testing (CFTT) project ("Computer Forensics Tool Testing Program" 2011). But beyond testing, error rates for tools are not often calculated (James and Gladyshev 2010; Lyle 2010). The argument has been made that a tested tool with a high number of users must have a low error rate because if there was a high rate of error, users would not use the tool (Guidance 2009). So far this argument appears to be widely accepted, however Carrier (2003) submits that “At a minimum this may be true, but a more scientific approach should be taken as the field matures”. Furthermore, Lyle (2010) states that “[a] general error rate [for digital forensic tools] may not be meaningful”, claiming that an error rate should be defined for each function. Because of this, and the lack of Law Enforcement’s (LE) time and resources, verification of a tool rarely passes beyond the testing phase of the Daubert process. The same can also be said for the investigator’s overall examination process. A Standard Operating Procedure (SOP)

should dictate the overall examination process (Jones and Valli 2008). Validation of the overall process is commonly done by peer review, but according to James and Gladyshev (2010) peer review does not always take place. None of the survey respondents mentioned any form of objective measurement of accuracy for the examination process. There has also been little research in the area of overall examination accuracy measurement.

Forensic examinations are a procedure for which performance measurement, specifically the measurement of accuracy, is not being conducted, due to concerns about the subjectivity, practicality and feasibility. Error rates are created for procedures, tools and functions to determine their probability of failure, and also as a measure for which other methods can be compared against. "…[E]rror rates in analysis are facts. They should not be feared, but they must be measured" (Palmer 2002). This chapter is a brief introduction to the problem of accuracy measurement in subjective digital forensic analysis, why it is needed, and how it may improve not only current analysis techniques, but also allow for a baseline with which to compare new techniques.

5.2 Related Work

Many areas attempt to measure the accuracy of their processes. In *Crawford v. Commonwealth,* 33 Va. App. 431 (2000) - in regards to DNA evidence - the jury was instructed that they "…may consider any evidence offered bearing upon the accuracy and reliability of the procedures employed in the collection and analysis…" and that "DNA testing is deemed to be a reliable scientific technique…". Although the technique may be reliable "occasional errors arising from accidental switching and mislabeling of samples or misinterpretation of results have come to light…" (Thompson 2002). Furthermore, the relatively recent "Phantom of Heilbronn" incident has led to questions of not just internal, but also the external processes that may ultimately effect evidence (Obasogie 2009; Yeoman 2009). While the DNA testing technique itself has been deemed to be reliable, erroneous results are still possible due to human error. Digital examinations are not much different in this regard. While a tool may be able to accurately display data, that data is not evidence until an investigator, or a human, interprets it as such. No amount of tool testing can ensure that a human interprets the meaning of the returned results correctly. The law in a region being measured may be used to attempt to objectively define the

correctness of an investigation; however, correctness in an investigation is somewhat vulnerable to the subjective conclusions of the investigator and their biases.

Information Retrieval (IR) is one area where accuracy measurement is paramount. Much work has been done in the area of IR, and IR accuracy measurement techniques have been applied to forensic text sting searching (Beebe and Clark 2007), document classification (de Vel 2004), and even fragmented document analysis in digital forensics (Li, Wang et al. 2006). The focus, however, has been on the accuracy measurement of particular techniques or tools within the digital examination process, and not for the examination process itself.

5.3 Objective Measures of Analysis Performance

At present, the efficacy of digital forensic analysis is, in effect, a function of the duration of an examination and of the evidence it produces. These factors force investigators to increase their use of automated tools, and explore autonomous systems for analysis (Kim, Kim et al. 2004). As previously identified, there are several weaknesses with automated tools in digital investigations today. Many automated digital forensic tools focus on inculpatory evidence, such as the presence of images, leaving the search for exculpatory evidence to the investigator. Also, many investigators are not comparing their automated tools to a baseline performance measure, such as other similar tools or the results of a manual investigation, which could lead to missed evidence and incomplete investigations. Tools are also not the only component in a digital forensic analysis. Even if all data is displayed correctly, the investigator must then interpret the data correctly. As such, a system of accuracy measurement capable of considering both tools and investigators is needed.

Two simple but informative metrics used in information retrieval systems are precision and recall (Russell and Norvig 2009). We propose that the IR performance measures of precision and recall can be applied to the analysis of digital forensic examinations. An overall performance measure relative to both the precision and recall, called an F-measure, may be used as the score for overall accuracy of the examination. This measurement will help to identify problem areas over time, that may lead to more focused training, smarter budgeting, better tool or technique selection and ultimately higher-quality investigations.

The use of precision and recall is suggested rather than current percentage error methods normally employed in digital forensic tool testing. Normally, percentage error is used to determine the error of a particular function of a tool. While percentage error could be used to evaluate the overall error of an investigator conducting an investigation process with various tools, there is no clear indication where error is being introduced. Precision can be thought of as the investigator's (or automated tool's) ability to properly classify a retrieved object. Recall can be thought of as the investigator's (or automated tool's) ability to discover and retrieve relevant objects. These scores can then be used to calculate overall accuracy, which can allow not only identification of weaknesses over time but also whether problems are arising from classification or recall challenges.

By measuring tools, investigators and laboratories, investigators can get feedback on performance, lab managers can objectively evaluate investigators' usage of tools, trainers can tailor courses to the needs of specific departments, trainees can more effectively choose courses to suit their weaknesses, and customers can have an objective measure with which to compare digital forensic laboratories.

5.3.1 Digital Analysis

Evidence, as defined by Anderson and Twinning (1998), is "any fact considered by the tribunal as data to persuade them to reach a reasoned belief [of a theory]". Digital forensic analysis attempts to identify evidence that supports a theory, contradicts a theory, as well as evidence of tampering (Carrier 2003). If an investigator focuses only on inculpatory evidence, it is possible that they could miss a piece of evidence that may prove the innocence of the suspect, and vice versa. Current digital forensic tools help an investigator to view objects that may have possible evidential value, but what that value is – inculpatory, exculpatory, tampering, or nothing – is determined manually by the investigator. The investigator must take the type of case, context of the object and any other evidence into account. This means that the identification of evidential objects strongly relates to the knowledge of the investigator. For example, in a survey, 67% of investigators claimed only a basic familiarity with the Microsoft Windows Registry (James 2010). If an investigator has little or no knowledge of the Microsoft Windows Registry, he or she may not consider it as a source of evidence. In this case the accuracy of the tool may not be in question, but instead the accuracy of the process or investigator. In digital examinations, to get true error rates for the

investigation process, the accuracy of both the tool and investigator must be measured. "Although increased rigor can be viewed initially as troublesome, its benefit to the collection analysis and presentation of scientific evidence will result in much higher confidence levels associated with the information presented to all decision-makers, including judges and juries" (Palmer 2002).

5.3.2 Precision and Recall

Some areas of computer science, such as information retrieval, use methods to measure the accuracy of the information that is retrieved. Two commonly used metrics are precision and recall. As defined by Russell and Norvig (2009), "precision measures the proportion of documents in the result set that are actually relevant… [and] recall measures the proportion of all the relevant documents in the collection that are in the result set". Manning, Raghavan et al. (2008) define the calculation of precision and recall mathematically using the following formulas:

$$\text{Precision} = \frac{\#\text{ relevant items retrieved}}{\#\text{ retrieved items}} = P(\text{relevant}|\text{retrieved})$$

OR

$$\text{Precision} = \frac{|\{\text{relevant items}\} \cap \{\text{retrieved items}\}|}{|\{\text{retrieved items}\}|}$$

$$\text{Recall} = \frac{\#\text{ relevant items retrieved}}{\#\text{ relevant items}} = P(\text{retrieved}|\text{relevant})$$

OR

$$\text{Recall} = \frac{|\{\text{relevant items}\} \cap \{\text{retrieved items}\}|}{|\{\text{relevant items}\}|}$$

Consider a search engine, for example. When a user enters a query, given enough time, a document containing exactly what the user was looking for could be returned from the set. But if the search had a high level of precision, then the number of documents returned (recalled) would be low and would take more time. Search engines, however, attempt to return results as quickly as possible. Because of this,

precision is reduced and a higher number of relevant, but possibly less exact, documents are returned.

An accuracy measure relative to both the precision and recall, called an F-measure (F), may be used as the score for overall accuracy of the measured query. The equation for calculating the F-measure is defined by Russell and Norvig (2009) as:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

5.3.3 Accuracy of Analysis

Precision and recall may also be applied to digital forensic analysis. This will allow departments to determine baseline accuracy figures for their forensic analysis processes. A digital examination can be considered like a query. Digital investigators are asking a question, and are returning the previously defined types of evidence in accordance with that question, which are then verified by a court. The evidence found (recalled) can be used to calculate the accuracy of the examination as compared to a baseline standard. For comparison, a baseline of correctness, or 'gold standard', must be established. In digital forensics, peer reviewed in-depth examination of a suspect's system by an expert is the level of examination that is accepted for use in court. This level of examination may not accurately identify all potential evidential artifact, but is presently the most accurate and comprehensive examination method that is normally accepted. An artifact is defined as information that supports or denies a hypothesis. The results of an examination (a collection evidential artifacts) are evaluated for admissibility by the court, resulting in a possible subset of artifacts accepted as evidence. From this, the 'gold standard' will be defined as *the resulting set of evidentiary artifacts returned during a peer-reviewed examination that are accepted as admissible evidence in court*. With this definition, the gold standard is set at the level of a peer-reviewed human investigation. Using this standard, the results of an examination from another human investigator using particular tools or processes may be objectively compared. Likewise, autonomous digital forensic analysis systems may also be measured against the gold standard, and compared to other processes.

Accuracy of analysis for a certain process, investigator or autonomous system can also averaged over time to evaluate trends. For example, as software used for analysis becomes out of date, new evidential data sources may exist that the software cannot

analyze. By measuring the performance of the process over time, the accuracy may decrease, signaling either an issue with the software or the investigator's knowledge. Since the accuracy of tools using precision and recall has been discussed in other works, this paper will focus on a method for investigator and analysis phase accuracy calculation.

5.3.3.1 Grading the Investigation Process

In digital forensic analysis, the ideal investigator performance is a high precision (no false positives), and a high recall (no false negatives); all found as fast as possible. Essentially, requirements for an investigator are similar to the requirements for an analysis tool, as described by Carrier (2002). An investigator that is comprehensive, accurate and whose work is deterministic and verifiable could be considered competent. This means that both high precision and high recall – high accuracy – is equivalent to high performance. This work does not take the weight of artifacts into account. That is, no one artifact is considered any more important than any other. By calculating the investigation process's precision and recall for an analysis, compared to the results that were admitted in court, the resulting accuracy measure may be calculated.

Consider an example where the results of a particular process discovered 4 inculpatory artifacts and 3 exculpatory artifacts for a total of 7 artifacts. During a peer-reviewed examination, the gold standard found the same 4 inculpatory artifacts, along with 5 others, for a total of 9 inculpatory artifacts. The gold standard found just one exculpatory artifact, which corresponded to one of the three identified by the process. It did not conclude that the other 2 objects identified by the process were in fact exculpatory artifacts. From this, the process' ability to discover evidential artifacts can be measured against the gold standard using precision, recall and the F-measure, where these performance measures are based on the presumption that the gold standard classification corresponds to the ground truth:

$$P = \frac{\#\ relevant\ items\ retrieved}{\#\ items\ retrieved} = \frac{5}{7} = 0.71$$

Recall (R) is found to be:

$$R = \frac{\#\ relevant\ items\ retrieved}{\#\ relevant\ items} = \frac{5}{10} = 0.5$$

Finally, the F-measure (F) is found to be:

$$F = 2 \cdot \frac{P \cdot R}{P + R} = 2 \cdot \frac{0.71 \cdot 0.5}{0.71 + 0.5} = 0.59$$

In this case the process's precision is 0.71 or 71%. However, if the process led to the discovery of only one artifact, and that artifact was of evidentiary value, then the process's precision would be 100%. In digital investigations, it may be possible that one piece of evidence is all that is necessary, but in many cases supporting information may need to be provided. This is why recall is important. A high precision with a low recall means that the process is missing evidence. In the current example the recall is 0.5 or 50%. This means that the process missed half of the possible artifacts. The F-measure is the relative combination of the precision and recall. In this case, the process scored 0.59 or 59%. This is the process's accuracy score for this analysis.

By measuring Precision, Recall and F-measure over time, departments can observe trends in the process, as well as calculate overall performance. Consider the imaginary example shown in Table 5.1. By examining the F-measure, it can be seen that the process's accuracy is decreasing (Figure 5.2). It can also be seen that the process is consistently missing almost half of the relevant artifacts. By using this method, it becomes easy to see if there are problem areas, and if there are issues that need to be addressed.

Table 5.1 Fictional example calculating Precision, Recall and F-measure for an investigator over time

	Precision	Recall	F-measure
Analysis 1	0.71	0.5	0.59
Analysis 2	1	0.6	0.75
Analysis 3	1	0.5	0.67
Analysis 4	0.7	0.3	0.42
Average	**0.85**	**0.48**	**0.61**

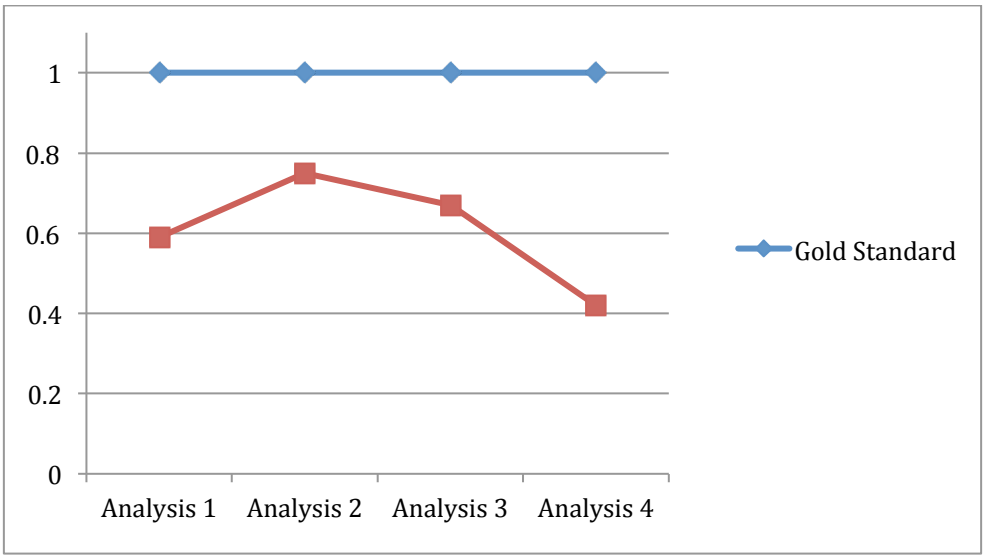


Figure 5.1 Analysis accuracy over time compared to the gold standard

5.3.3.2 Other Levels of Forensic Examination

Casey, Ferraro et al (2009) describe multiple layers of digital forensic examination to help handle an ever-increasing amount of data needing to be analyzed. The use of a multiple layer investigation model has been confirmed by James and Gladyshev (2010), where 78% of respondents claimed to use some sort of preliminary analysis. Most forms of preliminary analysis involve some form of automation, and much of the time if a preliminary analysis is done, the decision to continue or stop the examination will be made based on what is found – or not – with these less in-depth processes. It also appears that in all cases if anything suspicious is found during a preliminary examination, then an in-depth analysis will normally take place. Current processes, such as triage, have been shown to help reduce the number of suspect machines needing an in-depth examination (Goss 2010); however, triage and highly

automated preview examinations must not be as effective as an in-depth analysis, or automated processes would likely be used to conduct a full analysis. The issue then is that decisions to not continue analysis are being made based on a minimum amount of information. Also, investigators conducting preliminary analyses do not know what is being missed since they are not conducting a full examination.

"To reduce the incidence of incorrect conclusions based on unreliable or inaccurate data it is necessary to quantify uncertainty and correct for it whenever possible" (Casey 2002). The proposed method to measure accuracy may be applied to all layers of examination. If a highly automated tool, such as a triage solution, is being used to make decisions about a system, an F-measure can be calculated for the solution or process in the same way as the described, and compared to the gold standard. By doing this comparison departments can determine the limitations and benefits of their preliminary analysis techniques and particular tools, resulting in more informed decisions about their overall analysis process.

5.4 Case Study

In this section, two cases are given where the proposed accuracy measurement method is used. The first case will use data where an investigator was testing a triage tool against a full human investigation. The second case involves five investigators separately testing a different preliminary analysis tool. Comparisons between the investigators, as well as the tools are then evaluated.

5.4.1 Case 1

The following example case has been adapted from the work of Goss (2010), where the accuracy of a newly implemented triage process is being compared to a human investigator conducting a full analysis on the given media. In this case the accuracy of automated triage analysis will be compared to the gold standard set by an in-depth manual investigation based on the analysis of a data set with an unknown ground truth. In other words, the automatic classification of objects as having evidential value is being compared to the human gold standard. Five automated triage examinations (5 different media) are given in Appendix A, with their precision, recall and F-measure calculated. In this case, the human investigator validated the gold standard. For this reason, only false positives, as compared to the gold standard, with no further validation, are given. Table 5.2 gives a summary of the examination accuracy results.

Table 5.2 Summary of examination accuracy results using precision and recall to calculate the overall F-measure

	Precision	Recall	F-measure
Analysis 1	0.67	0.33	0.44
Analysis 2	0.00	0.00	0.00
Analysis 3	1.00	1.00	1.00
Analysis 4	0.07	0.53	0.12
Analysis 5	0.15	0.12	0.13
Average	**0.38**	**0.40**	**0.34**

From Table 5.2, the accuracy of the triage analysis conducted varies greatly. By observing these fluctuations, their cause may possibly be determined. Analysis 2, for example, had poor results because triage is unable to take the context of the case into consideration, and out of context the results returned by a quick triage examination might be suspicious. Alternatively, analysis 3 was extremely accurate because all discovered evidence was found using a known-bad hash database, and only previously known artifacts (artifacts that were in the hash database) were on the suspect system. Overall it can now be said that triage is good for finding known, or "low hanging", evidence but it is not nearly as effective as an in-depth examination by the investigator.

Using this method, it is shown that the overall precision of the implemented triage solution in this particular case study is 38%, and that it is missing 60% of the possible evidentiary artifacts as compared to the gold standard. The overall accuracy 'grade' for the implemented triage analysis is 34%. From here, this measurement can be used as a baseline for improvement, or to focus when triage should and shouldn't be used. Also, when using this method it becomes clear in which situations triage is missing evidence. With this knowledge, the triage process could possibly be changed to be more comprehensive. The triage process can now also be accurately compared to other examination processes, such as an investigator's performance, or other triage solutions.

5.4.2 Case 2

The second case involves five pieces of suspect media that each received a full expert digital forensic analysis, and had reports written as to the findings of all evidential artifacts. Each case was an investigation into possession of suspected child exploitation material. After the media received a full analysis, five preliminary examiners conducted a blind analysis on each piece of media using a locally developed preliminary analysis tool. One preliminary examiner (examiner 1) had experience conducting in-depth digital forensic investigations, while the remaining investigators had no experience with in-depth digital forensic analysis. The goal was to determine if decisions to discard media that did not contain illegal material could accurately be made without a time-consuming full examination. To test this method, the decision error rate was examined as well as the preliminary analysis precision rate using the described method to attempt measure the precision of both the tool and the examiner. The results of each preliminary analysis are given in Appendix B.

In the context of this case study, false positives are defined as objects identified as suspicious, but are in fact not illegal according to the gold standard. False negatives are defined as objects that are illegal that were not identified according to the gold standard. It is important to note that in a preliminary analysis it is acceptable – and likely – to have false positives in both the object identification and decision for further analysis. This process, however, must have a false negative rate of 0 for the decision for further analysis, meaning that exhibits with illegal content are always sent for further analysis. This process does not necessarily need a false negative rate of 0 for illegal object identification, since full illegal object identification is the purpose of the full analysis.

Five test cases were given where the suspect media with unknown ground truth received a full manual analysis, from which a report of findings was created. This report is considered the gold standard for classification of objects as illegal or unrelated. All cases were based on charges of possession of child exploitation material. Out of the five suspect media, three (60%) were found to not contain illegal content. Two exhibits (40%) were found to contain illegal content, most of which were illegal images.

A preliminary examiner then used an automated tool for object extraction purposes, and manually classified objects as illegal or unrelated. Table 5.3 gives the overall results of the preliminary examiner's further analysis decision and accuracy rates, Table 5.4 shows the average artifact identification error rate per preliminary examiner compared to the gold standard, and Table 5.5 displays the average accuracy rate based on artifact identification per investigator compared to the gold standard.

Table 5.3 Further analysis decision false positive and false negative error rates per preliminary examiner

Media Further Analysis Decision Error Rate				
Examiner	**Num. of False Positives**	**False Positive Error**	**Num. of False Negatives**	**False Negative Error**
Examiner 5	2	.4	0	0
Examiner 4	2	.4	0	0
Examiner 3	2	.4	0	0
Examiner 1	1	.2	0	0
Examiner 2	2	.4	0	0

Table 5.4 Average artifact identification error rate per preliminary examiner

Average Object Identification Error Rate		
Examiner	**Ave. False Positive Error**	**Ave. False Negative Error**
Examiner 5	.4	.26
Examiner 4	.31	.13
Examiner 3	.35	.02
Examiner 1	.21	.24
Examiner 2	.31	.09

Table 5.5 Average accuracy rate based on artifact identification per preliminary examiner

Average Accuracy Rate	
Examiner	**F-measure**
Examiner 5	.35
Examiner 4	.57
Examiner 3	.80
Examiner 1	.64
Examiner 2	.55
Unit Ave.	**.58**

From the Table 5.3, it is shown that no preliminary examiner falsely excluded suspect media containing illegal material. This means that all exhibits containing illegal material would have received an in-depth analysis. Also, Table 5.3 shows that the preliminary examiner with more experience – Examiner 1 – had a lower false positive rate in the decision making process. This is presumably due to a better ability to categorize and differentiate between illegal and borderline content.

From Table 5.4, it can be seen that false positive rates for object identification were relatively high. This was an expected outcome since the preliminary examiners are not capable of definitely categorizing borderline illegal content. A higher false positive rate may also indicate the preliminary examiners being overly cautious.

Also from Table 5.4, the false negative rate for object identification is somewhat high. This is also expected since preliminary examiners are not conducting a full analysis. Object identification false negatives must be compared with the results in Table 5.3. When comparing object identification to the decision process, missing some of the illegal material did not have an effect on the decision process. This is because if there are suspect objects, there are likely multiple sources that are suspicious. However, this correlation should be continuously monitored.

5.4.3 Evaluation

Table 5.5 is the calculated average accuracy rate based on automatic object identification and manual classification. This is a metric that may be used for

measurement and comparison in the future to ensure continued quality, where recall correlates to the ability of the tool to return related objects, and precision correlates to a preliminary examiner's ability to correctly categorize returned objects. If each preliminary examiner dropped in accuracy, it may indicate an issue with tools not extracting the required objects, or possibly an issue with the training of the preliminary examiner.

The calculated average accuracy rates may also be used to compare two analysis methods. As an example, consider Table 5.2, where the average accuracy of the Case 1 triage solution compared to the gold standard (full analysis) was .34 (34%). If this is compared to the average calculated accuracy – .58 (58%) – of the (mostly untrained) preliminary examiners in Case 2, it can be seen that the preliminary examiners in Case 2 are approximately .24 (24%) more accurate than the Case 1 triage solution for making similar decisions. Other metrics, however, should also be considered, such as the time for processing and analysis. For example, the Case 1 triage solution is meant to run on-scene in approximately 2 hours or less, not including analysis. The preliminary analysis solution in Case 1 is designed to be ran in a laboratory from 24 to 48 hours, depending on the size of the suspect media. Because of this, improved accuracy may be expected, but at the cost of time.

5.4.3.1 Limitations

There are two main limitations to the proposed method, the greatest being the definition of the gold standard. The gold standard, as defined in this paper, requires an expert to verify the findings of a given analysis. While such verification is sometimes performed as a matter of course, not all organizations can afford to duplicate efforts, even on a random sample. Furthermore, it should be noted that a gold standard is only as good as the experts creating it. If a sub-par examiner is setting the standard, the results of measurement may look very good even for poor examinations.

The second limitation is that the accuracy measurement cannot be used when no illegal artifacts were found in the full analysis. This method is only useful in measuring when some objects – either inculpatory or exculpatory – are discovered by the gold standard.

5.5 Summary

In this chapter a previously identified issue is how to objectively compare the accuracy of highly automated analysis tools to real digital forensic experts has been explored. A method based on information retrieval techniques to measure and compare the accuracy of tools, investigators and investigation processes has been discussed. Related work was explored, and an argument was given why objective measurement of the accuracy of digital forensic investigations is necessary. A simple method of accuracy measurement has been proposed as a starting point, and a very brief case study was given to illustrate the proposed method.

Chapter 6

Automatic Event Reconstruction

This chapter examines the need for automatic event reconstruction (AER) in digital forensic investigations, and explores previously proposed methods. Current issues with automatic event reconstruction are given, as well as practical needs that must be met before automatic event reconstruction can be of practical value. This chapter concludes by introducing the objective and basic ideas of this work.

6.1 State of the Art

Several methods have been developed which attempt to derive the happened events from available data. The following is a review of various previously proposed methods that directly and indirectly relate to the ideas presented in this work.

6.1.1 Traditional Timestamp Analysis

In traditional digital forensic investigations, timestamp information is often used in the analysis phase. Timestamps associated with logs, files, and even Windows Registry entries give investigators clues about when events took place. Various tools exist to arrange detected timestamps in chronological order, creating a timeline of available modification, access and creation times (Buchholz 2004; Gudjonsson 2010). Timelines allow for a quick overview of the creation, access and modification of data on a system that can be easily restricted to a certain period of time. The investigator may then focus in on data specific to the user to determine items that may need further investigation.

Koen and Oliver (2008) submitted a method of file timestamp analysis, stating that there is a relation between an 'action' – such as opening an application – and the update of related file timestamps. This relation may allow an investigator to determine which application must have been executed.

> *Event data is generated when a significant digital event occurs. Although the generated event data is of little value when viewed independently, collectively event data can produce information that can help investigators to deduce relationships between events to produce abstract views of the evidence at hand.*

While a relationship between an action and the update of file timestamps was loosely defined, no method was given for determining this relationship. Also, how the

collective data produces helpful information is not immediately clear beyond reverting to the previously described timeline-analysis techniques. Overall, this work showed that a relation exists between executing an application and the subsequent updating of related file timestamps.

This relation, however, may be inaccurate. For example, Willassen (2008b) states that the "use of timestamps as evidence can be questionable due to the reference to a clock with unknown adjustment". Further, timestamps available in a suspect system could be altered by a malicious user (Hilley 2007). Several methods to verify the consistency and validity of timestamp information have been suggested (Willassen 2008b; Parsonage 2009; Zhu, James et al. 2010). Willassen (2008b), for example, proposed a method based on clock hypothesis testing. This method seeks to create models of system timestamp update patterns that can be tested against hypothesis about the state of the system clock. An 'action sequence' is defined where one or more actions is related in time to the next action in the sequence. A 'timestamping order' is the order in which timestamps must have been changed in relation to each other. Timestamping orders must have been created by at least one action sequence. If no action sequence exists for the particular timestamping order, then the timestamping order is inconsistent with the system's timestamp update model. Figure 6.1 shows possible timestamping orders of modified (t_m), accessed (t_a) and created (t_c) timestamps, and the associated action sequence that could have created such a timestamping order. Hypotheses about the state of the system clock, such as an alteration of the time zone, can then be tested against the derived model to determine if observed timestamp information is consistent with the model; either supporting or refuting the hypothesis.

No	Timestamping order	Action Sequence
1	($t_c < t_m < t_a$)	(Create / CopyDest < Write < ReadGroup)
2	($t_c < t_a < t_m$)	None
3	($t_m < t_c < t_a$)	(Create / Write < CopyDest < ReadGroup)
4	($t_m < t_a < t_c$)	None
5	($t_a < t_c < t_m$)	None
6	($t_a < t_m < t_c$)	None
7	($t_c = t_m < t_a$)	(Create / CopyDest=Write < ReadGroup)
8	($t_a < t_c = t_m$)	None
9	($t_m = t_a < t_c$)	None
10	($t_c < t_m = t_a$)	(Create / CopyDest, Write)
11	($t_c = t_a < t_m$)	None
12	($t_m < t_c = t_a$)	(Create / Write, CopyDest)
13	($t_c = t_m = t_a$)	(Create / CopyDest=Write)

Figure 6.1 Derived timestamping order logic in Windows XP/NTFS. Willassen, S. Y. (2008b). Timestamp evidence correlation by model based clock hypothesis testing, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

While the clock hypothesis method attempts to determine the consistency of timestamp updates in a system, timestamps could be maliciously altered[11] in a way that is still consistent with the system. This still leaves questions about the reliability of timestamps that have yet to be resolved.

Along with reliability issues, extracting knowledge from collections of timestamps still requires a knowledgeable expert to manually reconstruct the events from the given information. To this end, pre-incident system monitoring methods have been proposed to attempt to automate the reconstruction of events.

[11] Programs exist that allow a user to change any or all file timestamps. For more information see: Foster, J. C. and V. Liu (2005). Catch Me If You Can: Exploiting Encase, Microsoft, Computer Associates, and the rest of the bunch... Black Hat USA 2005. Las Vegas, USA.

6.1.2 Pre-Incident System Monitoring

Some methods of in-depth automatic event reconstruction have been proposed, such as process coloring (Jiang, Buchholz et al. 2007) and backtracking (King and Chen 2005). The method proposed by King and Chen (2005) records "objects and dependency-causing events" before, during and after an incident. For example, if a process is created that updates a file, a dependency between the process and the update of the file is recorded. The logged dependency relationships between objects are then presented in a dependency graph (Figure 6.2).

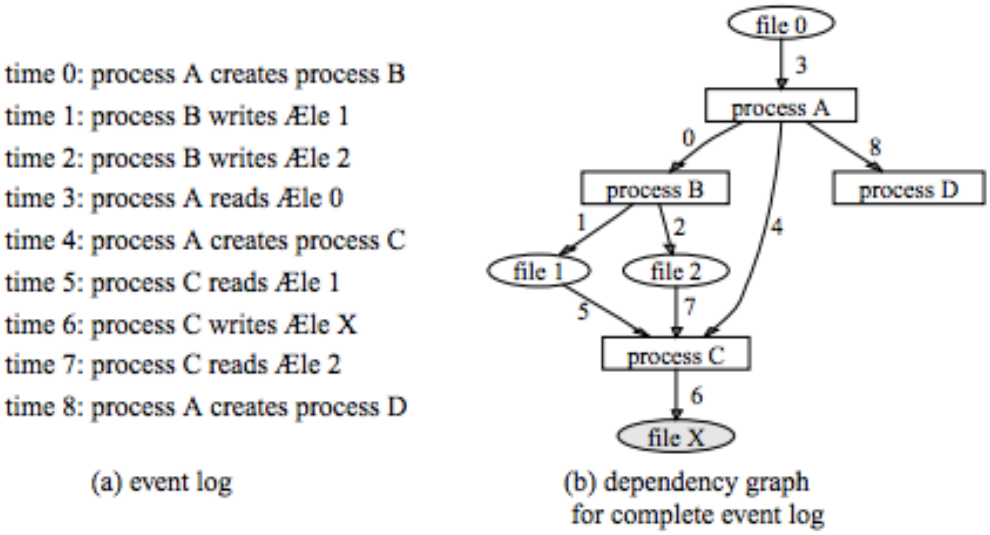


*Figure 6.2 Logged object and dependency-causing events (a) graphed as a dependency graph (b). King, S. T. and P. M. Chen (2005). "Backtracking intrusions." ACM Transactions on Computer Systems (TOCS) **23**(1): 51-76.*

Once suspicious activities have been detected, the created dependency graph can then be back-traced from the identified trace (such as a process or altered file) to determine what operations have acted upon this object. Since all operations have been recorded, suspicious operations that interacted with the objects in question can be easily identified, reconstructed and acted upon.

There are several issues with pre-incident monitoring analysis. First, this method can only be used in controlled environments. A system administrator who is security conscious would need to implement logging before an incident occurred. Home users generally do not host critical services or complex configurations, so comprehensive

pre-incident monitoring would not normally be found in a home or small business network. Along the same lines, criminals are unlikely to install a monitoring program on their own computers without also knowing how to hide their own activities.

The second issue with pre-incident monitoring is the resources consumed. Although storage space has dramatically decreased in cost, rigorous logging of many machines can consume prohibitively large amounts of storage per day, depending on the verbosity. On servers hosting critical services, security measures that do not increase processing and system load will be preferred over resource intensive monitors. While some organizations may be able and willing to provide complete monitoring for event reconstruction purposes, this level of pre-incident monitoring will generally be too cost and resource intensive for most organizations.

Given the issues with pre-incident monitoring, rigorous logging of the operation of the system will likely be unavailable. Instead, investigators normally have only information relating to the final state of the system before it was shut down. Because of this, methods are needed for the post-mortem analysis of a system that derives knowledge of the operation of the system from a limited amount of data.

6.1.3 Finite State Machine Analysis

Finite State Machine Analysis is a method to formally represent a system and analyze certain scenarios based on the formal model. Several works have proposed modeling a computer system formally as a finite state machine (FSM) (Gladyshev and Patel 2004; Gladyshev 2005; Carrier 2006b; Arasteh, Debbabi et al. 2007), which allows event reconstruction to be reduced to a state-space exploration problem.

For example, in Gladyshev and Patel (2004) a system is modeled as an FSM where all possible states of a system correlate to the states of the FSM, and all possible events are modeled as transitions from one state to another. In this work, Gladyshev and Patel define a computation as a non-empty sequence of state transitions (Figure 6.3). A ‘run’ of a computation is defined as a “possibly empty sequence of finite ‘computations’ in which the next computation is obtained from the previous computation by discarding its first element”.

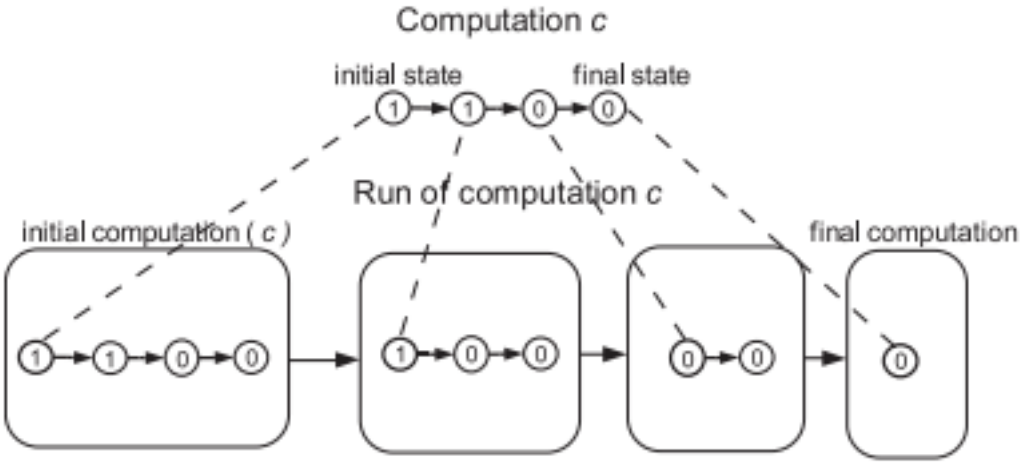


*Figure 6.3 Run of a computation defined as a Finite State Machine. Gladyshev, P. and A. Patel (2004). "Finite state machine approach to digital event reconstruction." Digital Investigation **1**(2): 130-149.*

Next, the concept of witness statements is presented, where 'observations' of some property of the system are explained as a run restricted in length. An 'observation sequence' is a chronological sequence of observations. These observation sequences are combined to form 'evidential statements', which are used to restrict the possible computations of the system model in the past.

Finally, investigative hypotheses can also be modeled as observation sequences. If there are explanations that agree with both the evidence and the hypothesis, then the hypothesis may be true. If there are no explanations that agree with both the evidence and the hypothesis, then the hypothesis must be false.

This method then accounts for all possible state transitions that could have occurred to reach the final observed state. This is beneficial from a defensive point of view, where alternative explanations of the final state may also be found.

This work was extended in James, Gladyshev et al. (2010) where a system is modeled as a deterministic finite automaton (DFA) that accepts symbolic encodings of computations of the finite state machine model of the system. Witness statements, again, are restrictions on the computations of the system, and can be modeled as DFA representing the patterns over the sequences of transitions. Automata intersection can then be used to construct an automation that represents the intersection of the two sets. In other words, a restriction on the possible computations of the system.

When a witness – or suspect – puts forward a statement, this statement may also be modeled as an automaton that restircts the possible computations. When this and other witness statements are interescted with the system model, if intersection does not result in the final observed state, then the statement must be false. If the final observed state is present then the witness statement is possible.

State machine analysis provides objective reasoning based on the generated formal model, does not need to be installed before an incident occurs, and allows for the use of more types of information than just logs or timestamps. One major benefit of the proposed methods is that they account for all possible events. This, however, is also a weakness.

The issue with the described FSM and DFA methods is that the formal representations of real-world models are extremely complicated with an extremely large state-space. By allowing for all possibilities, the state space becomes too large for practical computation. The resources used to model even the simplest real-world systems would be intensive, making these methods currently impractical for real-world use.

6.1.4 Inconsistency Checking

Stallard and Levitt (2003) proposed a method of event reconstruction using semantic integrity checking. The method seeks to determine invariant relationships "between objects that holds true for [a] system operating in an authorized state". After invariant relationships are found, contradictions to the invariant relationships can be flagged as suspicious and in need of further examination as to the cause of the inconsistency. Using this method, data that conforms to known invariant relationships may be discarded, leaving only suspicious exceptions. Figure 6.4 is an example of such a decision tree "with the goal of determining the set of users who may have changed the contents of a file". This process was automated by creating an expert system whose knowledge base is comprised of invariant relationships in terms of observable objects in a system, hypotheses of possible reasons for contradiction, and the "mode" to collect evidence to support a given hypothesis. Hypotheses for the reason of the contradiction may then be tested. "For a given set of facts, the hypothesis with the most supporting evidence could be pursued until evidence is found that refutes its assertion, or the investigator determines the purposes have been met."

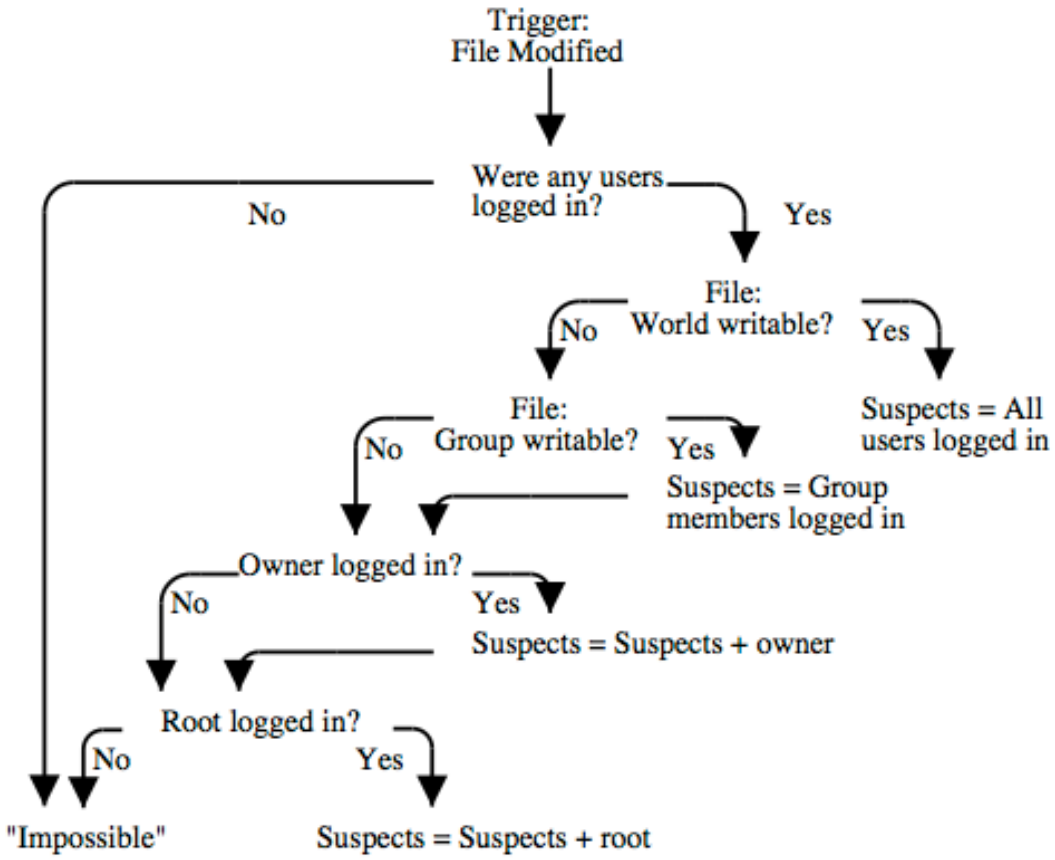


Figure 6.4 Decision tree for a consistency-checking expert system. Stallard, T. and K. Levitt (2003). "Automated analysis for digital forensic science: Semantic integrity checking."

Similarly, the use of the logic of invariant relationships used in the reconstruction of events in the event time bounding method proposed by Gladyshev and Patel (2005). A method for checking the consistency of events from data stored in the Microsoft Windows Registry based on event time bounding was proposed in Zhu, James, et al. (2010). This method defines an event as a modification to a Registry key. Windows Registry keys have an associated 'modified' timestamp. The method seeks to extract events and their associated timestamp information from the Registry, and check the consistency of these events based on the invariant relation between the events. Events are split into two categories; extracted and inferred events. An extracted event is one whose evidence of the event happening is directly observable. For example, if a Registry key and its associated timestamp information exist, then an event may be extracted directly from the observable data. An inferred event is an event that may be extracted based on observed data, but whose timestamp information is unknown or otherwise unavailable.

Once events have been extracted, events are ordered in time. Inferred events with no associated timestamp information are time-bound by events with a known relationship. This method relies on the redundancy of data stored within the Windows Registry. Multiple data sources may allow for the extraction of information about the same event. If the same event information may be extracted from different data sources, or if an invariant ordering of events must occur, then inconsistencies may be found. Finding inconsistencies may lead to the reconstruction of more events, or at least hypothetical events, which can help an investigator to understand what events were likely to have happened to cause the inconsistency.

A formal method of inconsistency checking for digital evidence has been proposed by Gladyshev and Enbacka (2007), where a system is modeled as a state-machine. Models of consistency for this state-machine are formally defined and verified using form B Method, which was originally a method of formally developing programming language code from specifications. Consistency checking within a system relies on invariant relations that are specified by a B model. The B model is validated to prove relations are invariant. This work defined inconsistency in two ways:

1. Evidential data may contradict the way [a] digital device works, and/or
2. It may contradict the hypothesis of the incident

If the evidence contradicts the given B model, then it specifies a state that cannot be reached from the initial state of the model, and is therefore inconsistent. Alternatively, the authors make no assertion that the evidence is consistent if it does satisfy the invariant relationship, but instead that nothing can be inferred.

Inconsistency checking allows an investigator to determine if the system has been manipulated in an unusual, possibly suspicious, way. Inconsistency checking may also help in the reconstruction of events that have happened in the past. The concept of inconsistencies should not be limited to the malfunctioning of a system, but could also be considered a deviation from the way the system would normally function. For example, a computer will generally not log in without a user interacting with it. A login event could be considered inconsistent with the normal routine of the system, if it was unexpected. This interaction could be considered an inconsistency in the normal running of the system if no user interaction took place, and is the basis of baseline analysis in host-based intrusion detection systems (Wagner and Soto 2002).

The identified known-good baseline or system model is used as the standard of consistency. The nature of the inconsistency, such as the absence of expected data, can give insight into the behavior that caused the inconsistency. By detecting an inconsistency, it may be possible to restrict the possible events to a smaller subset of possible events, and maybe even to one specific event that must have happened.

One issue with inconsistency detection is that is it difficult for humans to detect small inconsistencies in large amounts of data (Gladyshev and Enbacka 2007). For this reason, automation of inconsistency detection, and event reconstruction, is much more reliable. The issue with automated inconsistency detection is that defining what is consistent in a system is a labor-intensive task. To be able to determine if an action is truly inconsistent, an investigator would need to know every possible consistent result of the action. When considering high-level logic, such as in semantic integrity checking, inconsistencies are defined as a break in logical order, which humans can usually reason about relatively easily. The issue is that a human must define each and every action that is acceptable, as well as the order of those actions when attempting to automate the inconsistency detection process. When determining the consistency of a data structure, such as in Zhu and Gladyshev, et al. (2009), the consistent pattern is no longer necessarily based on reason, but on how the system is programmed to function. If the system is not well documented, then determining all consistent functions becomes difficult. Finally, inconsistency detection is able to reconstruct events when there is an inconsistency in the first place, and there is enough information to infer events that must have happened. Rigorous automatic event reconstruction must be able to determine consistent, as well as inconsistent events through time.

6.1.5 Comparative Analysis

As described in Kahvedzic and Kechadi (2008) and Zhu, James et al. (2009) the Windows Registry contains much information about user activities. By default, Windows systems backup the Registry when creating Windows Restore Points. Traditional digital investigation approaches look at a system as a single state, with only the most recent state being observable. Comparative analysis seeks redundant data sources, such as Windows Restore Points, and uses this data to determine system changes through time. If each Restore Point, and the Registry hives within, is considered as a state snapshot, then changes between states may be enumerated. As

shown in Figure 6.5, each Registry hive snapshot is a state snapshot that can be ordered by time. Each state snapshot is compared to the next consecutive state, and all differences – changes in the state over time – are enumerated.

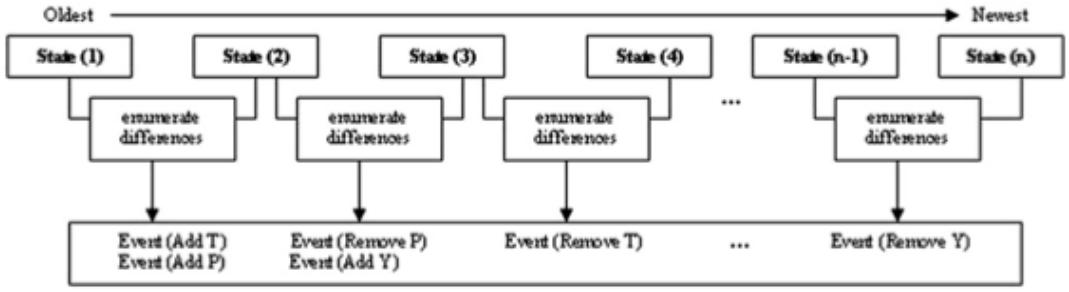


*Figure 6.5 Enumerating differences between state snapshots, and extracting events that must have happened based on these differences. Zhu, Y., J. James, et al. (2009). "A comparative methodology for the reconstruction of digital events using Windows Restore Points." Digital Investigation **6**(1-2): 8-15.*

Based on these changes, and how the system is known to work, user and system actions that must have happened between the two state snapshots can be inferred. Comparing saved states produces a list of changes to the data spanning the length of time, at the very least, between the first and last saved state. Further, extracted events with no associated timestamp information may be time-bound as previously described. In this case, changes must have happened between the creation times of the ten consecutive snapshots. Time bounding of an event between snapshot creation times is shown in Figure 6.6.

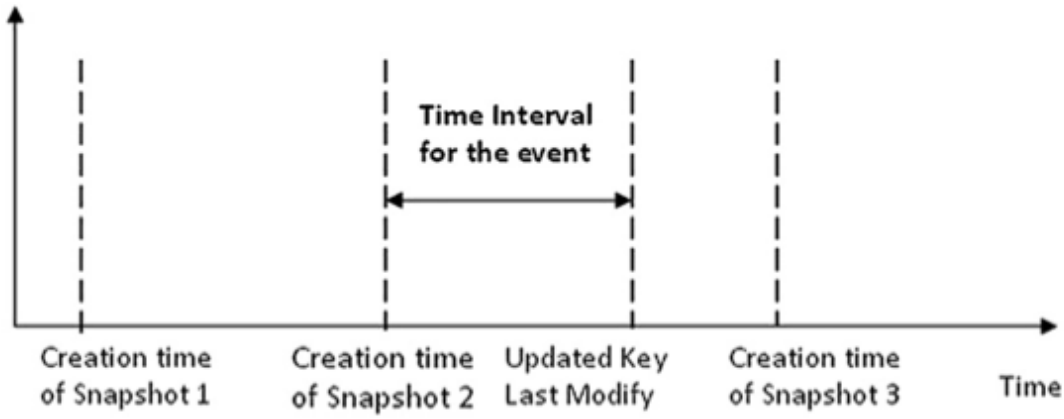


Figure 6.6 Event with no timestamp information may be time-bound between a prior state snapshot and a known limiter

As previously described in Zhu, James et al. (2010) the consistency of such types of data must be intact, and if not then deliberate alterations to the system or data must have been made. Comparative analysis has a number of benefits: First, much more information about past events can be found that would otherwise be unavailable. Second, it is possible to detect anti-forensics methods if the known consistency of the system has been altered along the timeline. Third, comparative analysis can be used after an incident has occurred without the need for pre-installed software. And finally, comparative analysis has proven to be useful for many types of redundant data across different operating systems (Mac OS time machine backups, iPhone backups, tape backups, etc.).

However, comparative analysis is only useful if enough redundant data exists to compare. If only the most current state of suspect data is available, this method is not useful. Since Windows Restore Points can be turned off, and savvy users can attempt to remove any redundant data, this method will not always be effective.

6.1.6 Probabilistic Methods

Probabilistic methods, such as Bayesian belief networks, have been applied to forensic investigations in a number of ways in an attempt to objectively measure the level of probability about a given hypothesis. In the works of Keppens, Shen et al. (2005) and Keppens (2007), Bayesian networks were used for statistical hypothesis testing, were “a piece of evidence [may] be evaluated by determining its likelihood under alternative hypotheses…”. As shown in Figure 6.7, multiple hypotheses are probabilistically related, and given observed evidence the probability of hypotheses may increase or decrease, also having an effect on the probability of related hypotheses. Bayesian networks in this case facilitate the computation of joint probability distributions over a large set of variables. This means that for a given piece of evidence, the joint probability distribution of all hypotheses that explain the evidence may be calculated. “Bayesian networks simplify these calculations by considering the independencies between variables” (Keppens 2007). Keppens attempted to apply Bayesian networks for any type of evidential reasoning. However, probabilistic analysis methods have recently been applied to attempt to reason specifically about artifacts in digital forensic investigations.

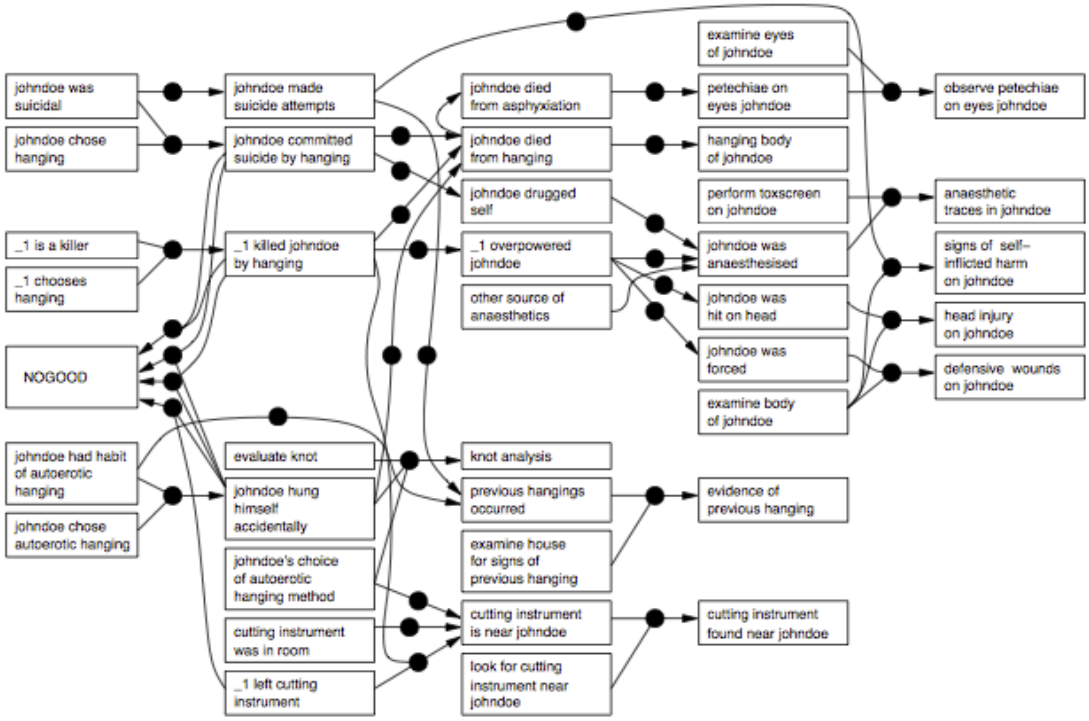


Figure 6.7 Scenario structure where each square is a hypothesis that has a probabilistic relation denoted by an arch. Reducing the probability of one hypothesis based on observed evidence would have an effect on the probability of related hypotheses. Keppens, J. (2007). Towards qualitative approaches to Bayesian evidential reasoning, ACM.

Probabilistic methods in digital investigation thus far generally look at the state of the system such as in Carney and Rogers (2004) and Xia, Fairbanks et al. (2008), where updating the attributes of files were considered in the context of a given hypothesis. In these works, probabilistic models of update patterns of artifacts are created. For example, Xia, et al. created lists of all touched artifacts, and attempted to extract specific artifacts that are unique to the specific action of interest as well as the artifact's probabilistic relation to other artifacts in the system. Similarly, Carney and Rogers attempted to model the relations between artifact updates that were then compared to the current state of the artifact. Matching the created model on pre-determined variables allowed for a probabilistic analysis of the likelihood of the hypothesis. In this case, the likelihood of a virus downloading images on a computer versus a human user based on the time-span between the image creation times.

Khan, Chatwin et al. (2007) models artifact update patterns happening during program execution in a system with the previously mentioned Bayesian network analysis. In this work, activity in a system is modeled as related artifact update

patterns that are observable on a hard drive. Update patterns were observed, and the probability of an artifact updating given a particular action was determined through experimentation. Once probabilities for multiple actions updating a particular artifact are created, a Bayesian network may be used to calculate the joint probabilities of each action updating the artifact. The result the action that is most likely to have updated the artifact if the artifact is updated.

Kwan, Chow et al. (2008) also submitted a method of reasoning about evidence using Bayesian networks. Again, in the proposed method the probability of three states of some hypothesis – yes, no and uncertain – are assigned probability values. Their method, in effect, attempts to determine the likelihood of a hypothesis being true, false or inconclusive rather than directly evaluating the likelihood of one hypothesis compared to another. If prior probabilities for hypotheses are unknown, then the probabilities of these hypotheses are equal. Once a root hypothesis is established, if other hypotheses exist that are conditional on the root hypothesis, then the state of the root hypothesis has a probabilistic relation to any other hypotheses with a causal relation. This method allows an investigator to statistically relate multiple hypotheses that may otherwise appear to be unrelated. It also allows an investigator to direct the investigation based on the most likely hypotheses if little information is initially known. As evidence is observed, hypotheses may be updated, allowing the investigator to easily determine which hypotheses are more relevant based on the observed evidence.

Probabilistic reasoning about the current state of artifacts on a suspect system does have benefits in reasoning with uncertainty, but there are currently some challenges with these methods. The first challenge is an often-used decreasing probability of a hypothesis when an artifact is missing, incorrect or inconsistent. Overill, Silomon et al (2010) argue that “a lack of evidence is not grounds to decrease probability of the event happening”. This is because the event may of happened, and the evidence was either not created or altered by other events that happened later in time. This is made even more relevant since in digital forensics, anti-forensics practices are becoming more common (Schlicher 2008). Also, if we assume an alternative hypothesis where the absence of all, or most, artifacts denotes anti-forensics, this also gets us no closer to the truth since it is just as likely that the event did not happen.

Another issue is that probabilistic methods cannot account for an updated model without testing and updating. Investigators can look for installed software and traces, and do research or ask others (knowledge updating) in relation to their findings, then continue the investigation with this updated knowledge. Probabilistic methods, however, are limited to the scope of artifacts that were known when the model was derived. If new relations cannot be created, e.g. knowledge of a previously unknown artifact-cleaning program, then reduction of the probability of the hypothesis based on the lack of artifacts is incorrect. Finally, the identification of accurate prior probabilities for actions or hypotheses is difficult. For example, in Kwan Chow, et al. (2008), prior probability is determined by interviewing expert investigators from a single region in a single country. The defined probability, however, is conjecture. Not only is this probability subjectively defined, but also what is true in one region or country may be completely false in another. While probabilistic methods may help in recommending the possible next step or direction of an investigation, as discussed in Carrier and Spafford (2005), the usage of probabilistic methods may lead to an overconfidence in false conclusions based on false prior assumptions.

6.1.7 File System Activity Analysis

Continuing from probabilistic methods, one method of file system activity analysis has been presented by Khan and Wakeman (2006). In their work, a two layer back-propagation Elman neural network (Elman 1990) was used to learn and detect application “footprints”. Traces that were created on a disk (usually file creation and manipulation) by a particular application of interest were fed into the neural network in the order in which they were accessed, which is illustrated by Khan and Wakemen in Figure 6.8. Extracted features, including log updates; Windows Registry updates; created, access and modified files; and free blocks, were used to learn the update time-span relationships and file-system manipulation patterns of the application.

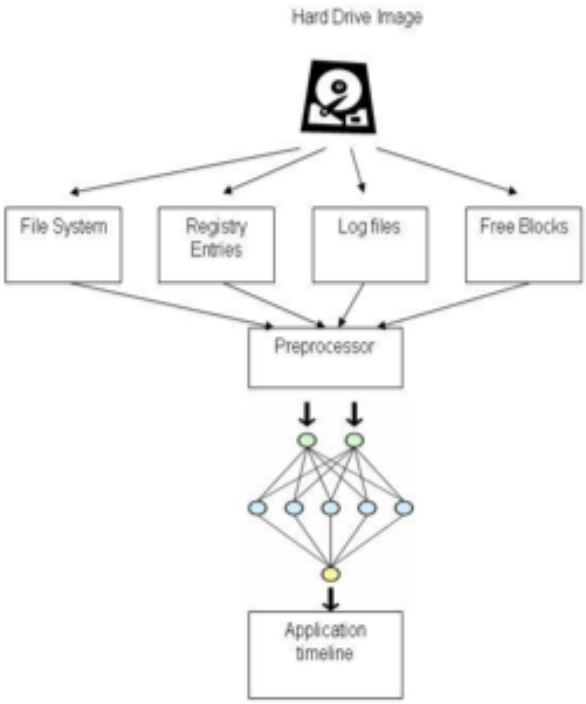


Figure 6.8 Input diagram for training a neural network on the artifacts updated by the application's execution, where modification to the file system, Windows Registry, log files and low-level storage are aggregated to learn overall update patterns. Khan, M. and I. Wakeman (2006). Machine Learning for Post-Event Timeline Reconstruction.

This method is also highly probabilistic, where the neural network is able to calculate the likelihood of an observation matching a previously derived model (footprint). Khan, Chatwin et al. (2007) showed that neural networks trained on specific application trace creation variables show relatively good results in determining and differentiating one application's footprint from another. One issue – that was also discussed by the authors – is that these systems need a very large amount of training data to be reliable. The training data needed, manual variable selection and separate neural networks per application make this method highly manual. Also, when comparing the application footprint of this method to the multi-layered application signatures described in James, Gladyshev et al. (2010), the differences in models suggest that learned signatures lack some specificity that could provide more event information for use in reconstruction. This issue is discussed further later in this work.

6.1.8 Computer Profiling

The work on computer profiling presented in Marrington, Mohay et al. (2007) and Marrington, Mohay et al. (2010) attempts to generate a computer usage profile that "… allows a human examiner to make an informed decision regarding the likely value of the computer system to an investigation before undertaking a detailed manual forensic examination". In this work an abstracted object model is used to classify objects observed in a suspect system. Observed objects are categorized as particular object types, such as system, principal (people/groups), application or content data. Relationships between these objects are expressed using predicate expressions, for example, a principal (p) updating content data (c) could be expressed as "updated(c, p)". The set of these relationships provide insight into the logic of the system, and allow for the identification of indirect relationships between objects that were otherwise thought to be unrelated. As shown in Figure 6.9, by examining an object of interest, object x in this case, the relation of other objects (a, b, c and the principal "Wally") may be found. Times and events – defined as recorded, inferred and unknown types – are found, and are associated with their corresponding objects, where possible. The overall computer profile is then represented by these object, relationship and event connections. Based on this representation, investigation theories about a computer system and its history can then be formally described in terms of the model and tested based on the derived computer profile.

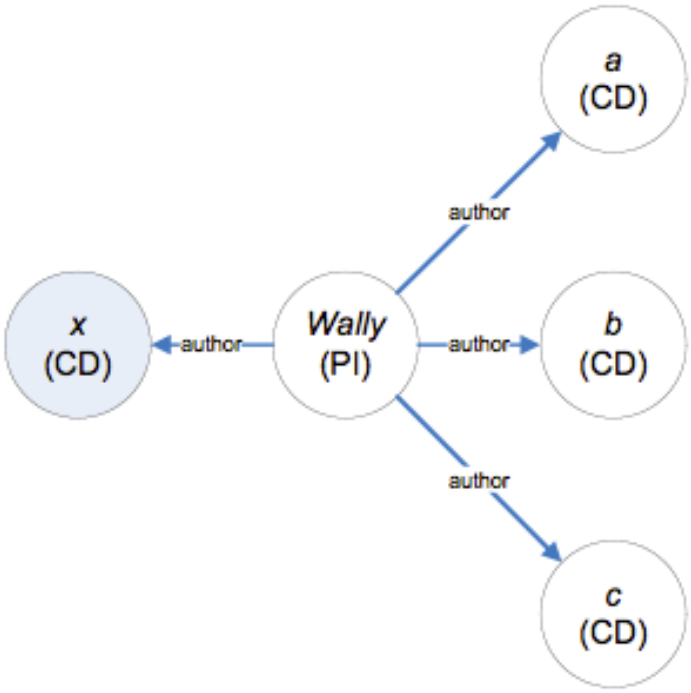


Figure 6.9 A diagram of found objects and their relations determined by investigating object x. The diagram is used to create a computer profile where the object (user) "Wally" is the author of objects a, b, c, and x. Marrington, A., G. Mohay, et al. (2010). A Model for Computer Profiling, IEEE.

By attempting to concentrate on an informational rather than a computational level, this method does increase practicality compared to methods described in section 6.1.3. In general, states are defined at a more abstract (object) level, and are based on observed evidence. Because of this, creation of this informational model is less computationally intensive, and also potentially less comprehensive, as not all possible combinations of past states are considered. The drawback is this model represents suspect objects and their relations, but makes no conclusion about what exactly these relations mean. The investigator is still left to manual hypothesis generation and testing, where the method proposed by Gladyshev and Patel (2004) attempts to automatically present possible hypothesis as well as test the hypotheses within the created models.

6.1.9 Signature Matching

The method of using signatures to detect certain types of actions or events is commonplace in many information security related products such as antivirus and intrusion detection systems (IDS)(Sy 2005; Scarfone and Mell 2007). Pouzol and Ducassé (2002) gave a formal specification of intrusion signatures and detection rules. In their work, an event is defined as a collection of data values, such as IP address and port number, identified by a name. The concept of a 'trail' is defined as an ordered sequence of events. The concept of a 'filter', which blocks or passes events, is also defined as "a set of constraints between event names, constant values and variable names". "An instance of a signature is a collection of events that (a) fulfill the constraints in filters (b) with respect to the correlation specified by the logical variables (c) and are in a correct order according to the temporal constraints." In other words, signatures are essentially encoded sequences of events and constraints. Once a signature is defined, an algorithm is given to find instances of multi-event signatures. From a given input trail, a search starts from the first part of the trail. Defined filters reject non-matching events and move to the next in the sequence, or if an event instance is matched, the filter outputs the preliminary match and the position of the match on the input trail. Once the first event has been matched, the current event in the signature is replaced with the next event in the signature's event sequence. If multiple instances of the second event in the sequence are detected, separate instances are created from the original sequence, each correlating to a difference instance of the second event; essentially a separate trail of investigation. When a full signature sequence is detected, if all event constraints are compatible, then a new notification is issued that an instance of the sequence has been found. In the case of multiple possible matches that meet all the constraints, multiple notifications may be issued.

This work attempted to formalize the signature detection process for intrusion detection systems, but many of the concepts such as the formalization of signatures and matching may be able to be applied to other areas utilizing signature-based matching methods. Signature based methods have been used in security products for many years, and have proven to be effective when a known pattern can be tested for. The downside, however, is that "traditional signature-based antivirus and antispyware fail to detect zero-day exploits or targeted, custom-tailored attacks" (Roiter 2007).

With signature-based methods if a signature does not exist, the event cannot be detected, unlike probabilistic methods that may be able to present each possibility in terms of the likelihood that a malicious event took place without explicitly knowing about the specific malicious event.

The hypothesis of this work is that, similar to detection of events in intrusion detection systems, events that have happened in a computer system – such as a user executing a program – may also be detected using signature-based methods. Further, unlike IDS and pre-incident monitoring, signature based methods may be used to detect a limited history of events during a post-mortem analysis.

6.2 Research Problem Statement

Generally, the previously mentioned work must either be implemented before an event takes place, such as pre-incident monitoring, are too computationally complex to be practical for real world digital forensic investigations, such as the described state-machine analysis, or simply produces more information without producing more knowledge for the investigator, such as the described probabilistic and computer profiling methods. This means that expert-level manual human inference is still necessary. However, manual inference is rarely verified, and if so, is manually verified by another expert, requiring much time and duplication of efforts (James and Gladyshev 2010).

To address these issues, the objective of this research is ***to develop a method that automates the inference of actions from the observation of low-level traces in a suspect system that results in automatic reconstruction of a collection of happened actions***.

More specifically, this research will address the previously mentioned issues in the following ways:

- Develop a practical and effective method to reconstruct events based on the automation of logical inferences an investigator manually draws about observed traces. This solution, at least, must be able to be conducted during a post-mortem analysis per standard digital investigations.
- Investigators encounter many different types of suspect operating systems, each with their own unique features. This research will aim to use human

thought processes as a model, allowing the process of inferring information from observations to be applied regardless of the operating system.

- Formal proof for digital forensic methods has been called for by (Gladyshev and Patel 2004; Taylor, Endicott-Popovsky et al. 2007) and as the field of digital forensics matures, rigorous mathematical analysis of proposed methods will be of increasing importance for the method to be accepted by investigators and courts of law. As such, a formal analysis of the proposed methods will be given.

6.3 Research Idea

Current investigative tools do well in providing more *information* to an investigator. For example, many forensic applications display Windows Registry information, presenting the results of parsing to an investigator. When this information is presented to the investigator, the investigator uses his or her knowledge of the system combined with the information presented to derive further knowledge about what has occurred. For example, if an investigator examines the Windows prefetch folder, the objects contained in the folder have little information in their content, but with knowledge of the system, the investigator can know that each object corresponds to a program that has been run in the system, and the object's timestamps may correlate to the last time the actual application was ran. This inferred knowledge is also assumed correct unless further information, such as a user manually creating the MRU entry, is introduced, and the inference is reevaluated.

This work analyzes the inferences an investigator makes during an investigation. With knowledge of the system, investigators must normally observe the state of the system, and attempt to infer what actions have happened in the past. This inference process is manual event reconstruction. When analyzing evidence, investigators normally gather knowledge in at least two ways: by direct observation and by the inference of one fact from the observation of others. This research is based on the theory that both the direct observation and inference phases of an investigation of user actions can be automated. This work seeks to prove that by determining the user action traces that normally appear in a system after a user action, it is possible to automatically infer the occurrence of the event based on the observable traces.

6.4 Summary

This chapter examined the need for automatic event reconstruction in digital forensic investigations, and explored previously proposed methods. Current issues with automatic event reconstruction were given, as well as practical needs that must be met before automatic event reconstruction can be of practical value. This chapter concluded by introducing the objective and basic ideas of this work.

Chapter 7

Theoretical Background

This chapter begins by giving a brief introduction into human inference in the context of digital investigations. Next, the concept of causation is discussed at a high level. After, causal relations in a computer system are discussed that allow for back tracing of causal chains from effect to cause. Mathematical notation used throughout the remainder of this work is then given, followed by a formal definition of system and action models that are the base from which event reconstruction of actions in the system can take place.

7.1 Inference in Investigations

When analyzing evidence, investigators normally gather information in two ways: by direct observation and by the inference of information from the observation of facts (Coopman 2006). As legal professionals put it, "The object of inferences is to reach a valid conclusion based on the facts and form of the argument" (Giarrtano and Riley 2005). Human inference, however, is prone to assumption and error, examples of which are discussed by Gilovich (1993) and Ogawa, Yamazaki et al. (2010). To accurately infer information from given facts an investigator must understand the underlying relation between the observed facts and the inferred conclusion (Senge 2006).

The problem of human investigation was studied in Anderson, Schum et al (2005). Somewhat similar to empirical verification in the philosophy of science, they claim that to define a relationship between observed evidence and a hypothesis requires "relevance [to the hypothesis], credibility [or believability of the evidence] and probative [inferential] force or weight". An argument is then defined as "a chain of reasoning from evidence to hypothesis", where each link in the chain of reasoning must be justified either via deductive or inductive reasoning. "Each link exposes a source of possible doubt or uncertainty." Because an investigator is observing evidence of an action and not the action itself, any inferred information may not definitely be true. Instead, inferred information is based on a level of belief held by the investigator based on their observations combined with their knowledge and experiences (and biases). Several methods, such as Bayes' theorem and especially the

related Dempster-Shafer theory of evidence, attempt to assign probabilistic degrees of belief to a hypothesis based on evidence (Shafer 1976).

An example of a common inference in digital forensic investigations utilizes the Microsoft Windows Registry. The Windows Registry is a database for storing operating system configuration data (Hillier 1996). The "TypedURLs" key "stores all [web addresses] that the user has typed into [Internet Explorer]" (Farmer 2007). Because entries in the TypedURLs key are normally only added after a user manually types the address into the Internet Explorer address bar, an investigator may normally infer that the user must have knowingly typed the specific website address. It is possible that alternate explanations may exist, such as the computer being infected with a virus that writes address to the TypedURLs key. As alternate hypothesis are developed, they must be tested to determine which explanation is more probable.

7.2 Causation

The concept of causation has often been discussed in philosophy. Spinoza (1677) stated that "[g]iven a determinate cause, the effect follows of necessity, and without its cause, no effect follows". Later, Hume (1886) postulated that "…whatever begins to exist, must have a cause of existence". In the same work Hume went on to cast doubt on the relation between cause and effect, essentially stating that effects follow causes based on observation and experience, and since human observation is incomplete there is no reason to believe effects will always follow causes (Anscombe 1999). Philosophers such as Kant and Russell challenged this view, but much controversy still exists concerning the philosophy of causation. Despite unresolved philosophical conflict concerning causation, science continues to rely on empirical study, and so shall this work. Hume defined rules by which to judge cause and effects. The two most relevant to this work are as follows:

- If multiple causes produce the same effect, it is because they share a common quality.
- Different causes must produce different effects. If a different or unexpected effect is found, it is from a different cause.

Causation can generally be defined as the relationship between one event and a second event, where the second event is a consequence of the first. Cumulative causal

relations in a system have previously been applied to digital forensic investigations in the form of happened-before relations (Gladyshev and Patel 2005; Willassen 2008a).

7.3 Causal Relation Between Actions and Objects

Throughout the remainder of this work the following definitions will be used:

Object – an item in a system (e.g. a file) that has associated information in the form of content and/or meta-data.

System – a collection of objects.

System state – the collective state of all objects in a system at a point in time.

Process – an occurrence in the system that changes the state of one or more objects over time. Processes will be represented in graphs by a diamond symbol.

Trace – a change to an object's associated information on the occurrence of a process. Traces will be represented in graphs by an oval symbol.

Action – any event external to the system that is the direct cause of a process. An action is the farthest point at which a happened event can be inferred. Actions will be represented in graphs by a rectangle symbol.

A causal relation between actions and object updates (trace creation) via process execution can be determined because of causal chaining, where an action causes the process and the process causes the update of corresponding objects (Figure 7.1).

Figure 7.1 Causal chain of an action causing a process that causes trace creation

In a computer system, an action can be the cause of one or more processes, which can in turn be the cause of the production or modification of one or more traces in the system. The relation of actions to processes to traces, then, is one-to-many. In this case, the observation of one or more resulting traces may be related back to the process that caused it, which may in turn be related back to the action that caused the process (Figure 7.2).

Figure 7.2 Back tracing the causal chain from the observed trace to determine the corresponding action that caused the trace

7.3.1 Object Updates in Time

In a computer system, actions cause processes to execute over an unknown period of time that modify or create objects as they are being executed. Multiple actions can start multiple concurrent processes, and multiple processes can modify the same objects.

As stated by Gladyshev (2005), “many digital systems, such as digital circuits, computer programs, and communication protocols can be described mathematically as finite state machines. A finite state machine (FSM) can be viewed as a graph whose nodes represent possible system states, and whose arrows represent possible transitions from state to state”. Likewise, Carrier (2006b) claims, “modern computers are FSMs with a large number of states and complex transition functions”. By utilizing this fact, finite state machine models that represent the computations of a computer system may be formally defined.

From this, processes in a system can be modeled as finite state automata. User actions can be viewed as inputs into these automata whose outputs are updates to objects on the disk over time. In principle, these automata can be analyzed using traditional techniques, such as state space exploration, etc. However, the real-world complexity of computer systems makes this infeasible in practice due to the problem of exponential state-space growth, known as state-space explosion (Heimdahl and Leveson 1996). In order to find a tractable solution, this research proposes a simplified model. The key idea behind simplification is that most systems investigated in digital forensics are interactive, which means that user actions have immediate effect on the system state. As a result we can define actions as having an immediate effect on the system, where all effects on the system happen in a short period of time from each other. This means that when an action happens, related object meta-data is updated within a short, nearly instantaneous, period of time after the action occurs.

One of the major issues of event reconstruction in a post-mortem forensic analysis is the lack of data. Much like the crime scene for a physical investigation, only the final state is immediately observable, and other related or non-related actions could have happened after the suspected action, effectively destroying some evidence.

As shown in Figure 7.3 each process has a collection of associated objects, and multiple processes can modify the same object. This means that as more processes

with overlapping associated objects are executed traces associated with processes that happened earlier in time are overwritten, creating a collection of traces that were partially updated by an earlier process and partially updated by a newer process. The final overall state of a system is comprised of these full and fragmented collections of traces. Consider Figure 7.3, where the X axis is time, the Y axis represents objects in a system (objects 1 – 10), and three processes (P1 – 3) denote processes that modify an object a at particular time. The final observed state of all objects is at 13:15 (the straight line of object), where each object is represented by the symbol of the process that last updated it.

Only partial traces of process 1 (P1) and process 2 (P2) can be observed in the final observed state. Since process 3 (P3) was the most recently executed process, all objects associated with process 3 contain traces of P3's occurrence.

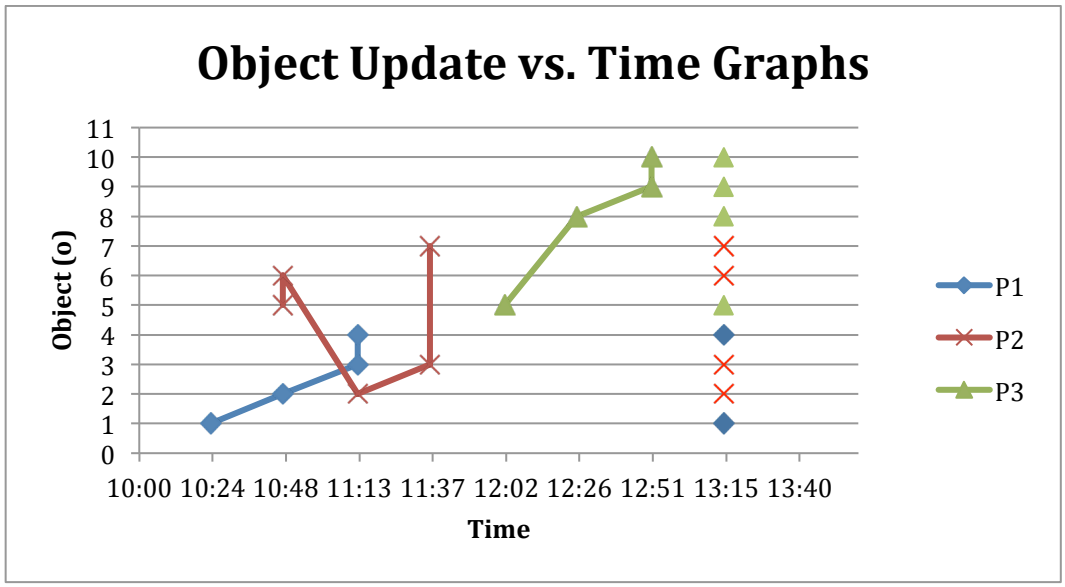


Figure 7.3 Object Modification vs. Time Graph showing that overlapping processes modify some of the same objects, producing partial traces of processes when observing the final state (the straight line at 13:15)

7.3.1.1 Functional Equivalence of Actions

If two actions, A1 and A2, both cause one process, P1, that in turn causes the creation of traces (Figure 7.4), then on the occurrence of an action the detection of the associated traces denotes the existence of either A1 or A2. However, according to Hume's rules for causation – discussed in section 7.2 – there is a quality that is shared between both A1 and A2, and the observation may be further reduced to capture objects specific to one action or the other. If the observation cannot be reduced then

A1 would be so similar to A2 that there is no observable difference to the final observed state. In this case the two actions are said to be 'functionally equal'.

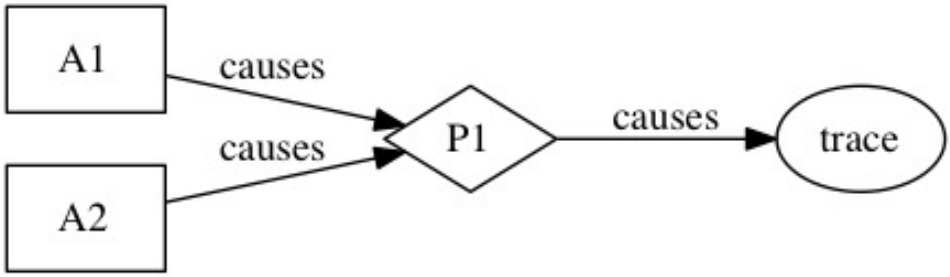


Figure 7.4 Two actions causing the same event become functionally equivalent unless they can be further reduced into unique actions in terms of resulting traces

Functional equality is the limit to what information may be inferred by observing a system. For example, detection of an action based on the observation of created traces may be able to determine that the "enter" key was pressed; however, the intention past the result of the physical action of the user cannot be directly inferred. Likewise, differentiation of multiple users, such as determining whether a human or a cat hit the enter key, is not likely to be inferred.

7.4 Mathematical Notation

For the remainder of this work, the following notation will be used:

Sets: Sets are denoted by capital Roman letters, e.g. A, B, C. Sets are defined by listing their members between { and }, e.g. $A = \{1,2\}$

Empty set: Empty set is denoted as $\emptyset$

Set Instance: A member of a set is denoted by lower-case Roman letters, e.g. a, b, c.

Membership: Statement that a is a member of set A is denoted $a \in A$. Statement that a is not a member of set A is denoted $a \notin A$

Subset: Statement that set A is a subset of set B is denoted $A \subseteq B$

Union, intersection, and set difference: Union, intersection, and set difference of two sets A and B are denoted $A \cup B$, $A \cap B$, and $A \setminus B$ respectively.

Output: An output set or element will be denoted with an apostrophe (‘). For example, a function that takes an input set and outputs a resulting set could be written as X’ = f(X), where X’ is the resulting set produced from the original set X.

Rationals: Set of rational numbers is denoted $\mathbb{R}$

List (Sequence): A list (sequence) is defined by listing its elements in round brackets, e.g. (0,1,1,0,0).

Numbering of elements in a list: Elements in a list are numbered from 0. The *i*-th element of a list *a* is denoted a_i. If $a = (1,2,3)$ then $a_0 = 1$, $a_1 = 2$, $a_2 = 3$

Functions: Usual mathematical syntax is used for functions. For example, term $f(x,y)$ denotes the application of function *f* to arguments *x* and *y*.

Dot Notation: Attributes of a structure are addressed using dot (.) notation. For example, the attribute *a* in the structure *S* is addressed as *S.a*

7.5 Formal Definition of System and Action Models

The formal models proposed are based on the observation of digital systems during experimentation that is discussed further in section 8.6. It was observed that when a user interacts with a system, the given action causes changes in the state of the system. In this work we found that real digital systems cannot be defined as completely deterministic, for example, in the case of race conditions. However, if the system is in a particular state, an action may consistently cause the same modification of state within the system each time the user action takes place while in that particular state. If the state of the system is exactly the same before the action takes place, then exactly the same effect may occur. A user clicking on a program's icon, for example, is likely to consistently execute the program. The executed program must access specific files to load, which in turn updates file content, metadata, log entries, etc. This work will focus on file meta-data, and specifically time stamp information associated with objects.

A system contains a finite set of objects, *O*, where each object in the set *O* may be defined in terms of associated access (*ta*), modified (*tm*) and created (*tc*) time stamps:

$$o = (ta, tm, tc).$$

Time stamps can be considered fixed attributes of an object that either exists or not, and a time stamp value is a variable time value contained within a time stamp attribute.

$$\tau = tsv(t)$$

where:

- *t* is a time stamp
- τ is a time stamp value
- *tsv* is a function that returns a time stamp value given a time stamp

The objects and time stamps in a system can be described as follows:

$O = \{o_1, o_2, o_3 \ldots\}$

$Tm = \{tm_1, tm_2, tm_3 \ldots\}$

$Ta = \{ta_1, ta_2, ta_3 \ldots\}$

$Tc = \{tc_1, tc_2, tc_3 \ldots\}$

$T = Tm \cup Ta \cup Tc$

An action is defined as:

$$a = (Ma, Da, Oa, af, \Delta\tau_{min}, \Delta\tau_{max})$$

where:

- *Ma* is a set of timestamps that are updated to the current time plus some random period of time ($\tau + \Delta\tau$), where $\Delta\tau_{min} \leq \Delta\tau \leq \Delta\tau_{max}$
- *Da* is a set of timestamps set to a default value
    - $Da = \{ (t, \tau) \}$, where t is the time stamp, τ is the default value
- *Oa* is a set of objects that are created if they are not already present
- *af(O, τ)* is a function which models the effect of the action. It takes in a set of objects *O* and returns another set of objects produced from the original by updating timestamps in *Ma* to the current time $\tau + \Delta\tau$, resetting timestamps in *Da* to default values, and adding *Oa* objects to O if they are not present[12]
    - $O' = af(O, \tau)$

As defined, an action may update timestamps. Since the time stamp update period is not instantaneous, the update will happen at some random interval after the action. This update takes place at a random delta after the time of the action. The action will also create an object if it does not exist. Created objects may have timestamps set to default values that may possibly be sometime before the time the action took place. For example, software installation and backup recovery actions could produce objects with time stamps that are before when the installation and backup actions occurred. Finally, an action function exists that accepts a set of objects as an input, and returns a modified set of objects produced from the original in the form of possibly updated time stamps.

The set of all actions is defined as:

$$A = \{a_1, a_2, a_3 \ldots\}$$

[12] The timestamps of members of *Oa* are members of either *Ma* or *Da* and their values are set accordingly.

Actions happen at particular points in time. An instance of an action is defined as:

$$i = (a, \tau)$$

where:

- $a \in A$ is an action
- $\tau \in \mathbb{R}$ is the time of the action instance occurring

The set of all instances occurred in the system is defined as

$$I = \{(a, \tau) \mid a \in A, \tau \in \mathbb{R}\}$$

In the proposed model, the set of actions is defined such that each timestamp is a result of some action. This simplified system model is defined as:

$$\forall t \in T, \exists i \in I, (t.\tau = X) \vee$$
$$\left(\exists d \in i.a.Da, (d.t = t) \wedge (t.\tau = d.\tau)\right)$$

where:

- $X = i.\tau + \Delta\tau + dw\tau$
  - $i.\tau$ is the real time of the action instance
  - $\Delta\tau$ a random update period
  - $dw\tau$ is the time taken to write the trace to the disk, which is negligible in terms of the timestamp, but means that two traces cannot be written at exactly the same real time

This model states that for all time stamps there either exists an action instance where the current time of the time stamp equals the time of the action instance ($i.\tau$) plus some random period of time ($\Delta\tau$) plus the time to write to disk ($dw\tau$). Otherwise the time stamp is equal to a default time stamp.

7.5.1 Action Instance Inference Function

The aim of event reconstruction is recovery of the sequence of actions that happened in the system. In terms of the proposed model, event reconstruction corresponds to determining the set of action instances *I* that happened in the system based on information contained in *O'*. This ordered set of action instances is defined as an Instance Sequence (*IS*).

Suppose there was an Ideal Instance Sequence Reconstruction function (*IISR*), such that

$$IS = IISR(O', A) \qquad (*)$$

Such a function, however, is not possible due to the destructive nature of the assignment. Suppose that *IS* consists of several instances of the same action $a \in A$:

$$IS = \{ (a, \tau_1), (a, \tau_2), \dots, (a, \tau_n)\} \text{ , where } \tau_1 < \tau_2 < \dots < \tau_n$$

The effect of *IS* on the system state *O* is

$$O' = a.af(a.af(\dots a.af(O, \tau_1), \dots, \tau_{n-1}, \tau_n)$$

Due to the destructive nature of the assignment, the final state of the system *O'* depends entirely on the last action instance, which means that any subset of *IS* that includes (a, τ_n) will result in the same *O'* and function *IISR*() as defined in (*) is impossible.

A different event reconstruction function *ISRS*() can be defined to return the set of all possible *IS* that result in given *O'*:

$$ISRS\ (O', I) \rightarrow II = \{IS_1, IS_2, IS_3, \dots\}$$

However, due to the destructive nature of the assignment, any such set *II* will be infinite. Under assumptions of discrete time, zero $\Delta\tau$, and finite duration of the incident, *II* becomes finite. That case was explored in Gladyshev and Patel (2004), which described an algorithm for computing *II*. Although computing *II* is useful for automated testing of formalized investigative hypotheses, it is not helpful from a human investigator's perspective because it obstructs investigative reasoning with a large number of possible action instance sequences, only one of which actually happened.

In order to assist a human investigator, it is more useful to determine only those action instances that definitely happened according to the available evidence. The corresponding Instance Sequence Reconstruction function *ISR*() can be defined as

$$I' = ISR(O', A)$$

whose output is:

$$I' \subseteq I$$

Unfortunately, due to randomness of $\Delta\tau$, the precise determination of the time of action instance is also impossible in general, because two action instances (a, τ_1) and (a, τ_2), such that $|\tau_1 - \tau_2| \leq |a.\Delta\tau_{max} - a.\Delta\tau_{min}|$ may result in the same timestamp: $t = \tau_1 + \Delta\tau_1 = \tau_2 + \Delta\tau_2$

Thus, for the purposes of this dissertation we define event reconstruction as an action Instance Estimate Reconstruction function *IER*(), such that

$$IE = IER(O', A)$$

where:

- $IE = \{(a, \tau_{min}, \tau_{max}) \mid a \in A, \tau_{min} \in \mathbb{R}, \tau_{max} \in \mathbb{R}, \tau_{min} \leq \tau_{max}\}$
- $\forall e \in IE, \exists i \in I, \ (i.a \in e.a) \wedge (e.\tau_{min} \leq i.\tau \leq e.\tau_{max})$

An implementation of the *IER*() function is given in Chapters 8 and 9.

7.6 Summary

This chapter began by giving a brief introduction into human inference in the context of digital investigations. Next, the concept of causation was discussed at a high level. After, causal relations in a computer system were discussed that allowed for back tracing of causal chains from effect to cause. Mathematical notation used throughout the remainder of this work were then given, followed by a formal definition of system and action models that are the base from which event reconstruction of actions in the system can take place.

Chapter 8

Action Instance Object Update Patterns

This chapter describes the derivation of action instance object update patterns. A practical method for determining the relation between actions and object update patterns is given. Analysis of object update patterns allows traces to be categorized, and rules of consistency to be determined. The need for object update thresholds is discussed, and a method for determining the object update threshold for an action is given. Next, a brief discussion and method for generalizing signatures for portability across suspect systems is given. The chapter ends by giving an overview of the object trace update experimentation used to derive general categories.

8.1 Action Instances and Trace Evidence

This chapter proposes the theory that the observation phase of an investigation can be encoded as signatures that represent knowledge about the state of the system. By determining the traces that normally appear in a system after an action instance, it is possible to automatically 'infer' the occurrence of the action based on the observable traces. In this section the focus will be limited to timestamps associated with files and Windows Registry entries. The hypothesis for signature generation is that when an action occurs, associated traces are updated within a short period of time. As a result, the occurrence of the action may be inferred by observing that the corresponding ensemble of traces have been updated within short time of each other.

When an investigator conducts an investigation, they observe suspect data and attempt to infer what this collection of data means in relation to happened actions. This work submits that trace creation signatures associated with specific action instances may be used in the same way to represent inferences about the action instance.

For example, an investigator may look at the standard Windows pre-fetch file, and from his or her experience and knowledge of Windows operating systems know that a pre-fetch file for a program is created every time, and only when, the program is launched. He or she can then infer that the creation time of the pre-fetch file in question is the last time the associated program was launched, since it must have been updated. Other traces, such as files and Windows Registry entries, may also have

properties updated in a specific pattern when a given action instance occurs. By identifying the known patterns of updated traces, the same types of inferences could be made when related artifacts are observed. Signatures encoding these unique object update patterns can be created that represents this inference. The inference in this case being that a certain program was executed at a certain – approximated – time. By finding the ensemble of traces relating to an action of interest, a signature for the action may be derived that takes advantage of unique update patterns of a group of traces. A system's current state can be checked for a pattern of traces that are known to have certain properties when a specific action instance has occurred. The signature itself then becomes an inference hypothesis with the observations being the detection or absence of the associated trace pattern.

Signatures may be created for any action that makes a change in the system; however, as described by Khan (2008) and Xia, Fairbanks et al. (2008), differentiation between multiple actions implies a uniquely observable pattern. Signatures can be created for actions, and quickly tested by scanning the system much like an antivirus scans files for patterns of code that may be malicious. The result would be either a positive or negative match of each tested signature against the state of the system, allowing for action instances to be automatically inferred from the observation of lower-level traces. In cases where timestamp information is available it may be possible to also approximate the times in which the detected action instance must have occurred, as well as infer and approximate previous executions of action instances.

8.1.1 File System Information

The majority of operating systems relate timestamp information to objects within the system, such as log entries and files. The availability of timestamps differs between different versions of Microsoft Windows. For example, the 'Last Access Time' has been disabled by default for performance reasons in Visa, 2008 and Windows 7. However, "disabling last access update does not mean that the Accessed Date on files does not get updated *at all*; it means that it does not get updated on directory listing or file opening, but last accessed can sometimes be updated when a file is modified and is updated when a file is moved between volumes" (Parsonage 2009). Pre-Vista versions of Windows using the NTFS file system, including Windows 2003, do have last accessed timestamps enabled by default. In all Windows versions, modified and created timestamps are unable to be disabled. Likewise, the Windows Registry

provides "[Key Cells] that contain the timestamp of the most recent update to the [Registry] key" (Russinovich n.d.). These keys' time stamps also cannot be disabled, providing a valuable resource to investigators. While the detection of the existence of suspect data can lead to the inference that a particular action must have occurred, this work will focus on object timestamp information as traces of interest. Existence of time stamp information allows an investigator to infer that the action has occurred, and also approximate when – within what time range – the action must have been executed. Temporal knowledge is also valuable when attempting to determine rules of consistency related to trace update patterns.

8.2 General Object Update Categories

To understand how objects are updated given a particular action instance, a series of experiments were conducted that are given in section 8.6. These experiments were conducted on a Windows XP system. Tested action instances are opening Internet Explorer and FireFox Internet browsers. In summary, the experimentation process executed the action instances while monitoring changes in the system. Objects that were updated during the action instance were recorded. Next, the state of the meta-data of the recorded objects was collected, the action instance was again executed, and the objects' meta-data was collected. This process was conducted a number of times, after which all meta-data (timestamp) state snapshots were analyzed. Objects and their associated timestamp of interest were then manually categorized in terms of the observed update patterns. For more specific details about the experimentation process, please see section 8.6.

From these experiments, three major, generalized categories of object meta-data update patterns are defined.

8.2.1 Core Update Category

The first category defined is the category most commonly used in manual digital forensic analysis, as well as previously discussed 'fingerprinting' techniques. This work defines this category as "Core" object update patterns. *Core traces are traces that are updated by exactly one action, and are always updated whenever an instance of the action occurs*. To define Core traces in terms of the model, the following supporting concepts need to be defined:

1) Let function $U: A \rightarrow 2^T$ denote the set of all timestamps *updated* by an action:

$$\forall a \in A, \forall t \in T \mid t \in U(a) \Leftrightarrow (\exists a \mid (t \in a.Ma) \bigvee (t \in a.Da))$$

2) Let function $AU: A \rightarrow 2^T$ denote the set of all timestamps *always updated* by an action:

$$\forall a \in A, \forall t \in T \mid t \in AU(a) \Leftrightarrow (\forall a \mid (t \in a.Ma) \bigvee (t \in a.Da))$$

3) Let function $UU: A \rightarrow 2^T$ denote the set of all timestamps *uniquely updated* by an action:

$$\forall a \in A, \forall t \in T \mid$$
$$t \in UU(a) \Leftrightarrow \left(t \in U(a) \bigwedge (\forall a' \in A \mid a' \neq a \implies t \notin U(a'))\right)$$

The Core traces of an action is defined as the set of all timestamps that are always and uniquely updated by the action:

$$CORE(a) = AU(a) \bigcap UU(a)$$

8.2.2 Supporting Update Category

The next update category is defined in this work as "Supporting" object update patterns. *Supporting traces are traces that are updated by exactly one action, but may not be updated with every instance of the action*. The most common reason why a trace may not be updated is because if an instance of the action occurred, and then some time later occurred again, the operating system may access a copy of the object that is in memory, or cached elsewhere. In this case, the original object may not be accessed directly on disk, and its corresponding timestamp may not be updated. This property, however, appears to be controlled by the operating system, and no discernable pattern of when an in-memory copy would be used was found. Supporting update categories can be defined in terms of the model as follows:

$$SUPP(a) = UU(a) \backslash AU(a)$$

Supporting traces are the set of all timestamps uniquely, but not always updated by the action.

8.2.3 Shared Update Category

The third update category is defined in this work as "Shared" object update patterns. *Shared traces are traces that are updated by more than one action, and are either always updated or not always updated respective to the action that is acting upon the object*. For example, if two actions updated the same object, one action may update the object on every instance of the action, while the second action may sometimes access a cached version of the object. Shared object updates can be described in terms of the given model as follows:

$$SHARED(a_1, a_2, \ldots, a_n) = U(a_1) \cap U(a_2) \cap \ldots \cap U(a_n)$$

8.3 Object Time Stamp Update Threshold

The object time stamp update process is not instantaneous. In order to accurately differentiate between multiple action instances, trace update duration must be defined for the particular action. The trace update times, in seconds, of the action instances "Open Internet Explorer 8" and "Open Firefox 3.6" were surveyed on 25 computer systems running Windows XP or Windows 7, with results shown in Figures 8.1 and 8.2 from survey data given in Appendix C. The results show that action instances' update duration will be different depending on the hardware of the system, as well as the state of the software. Because the time for a trace to be updated is variable, it must be described as a range. From experimentation, it was determined that the object update times may be modeled as a normal distribution. A standard deviation (σ) of 2σ was chosen as the standard threshold limiter to attempt to reduce unlikely outliers. This decision was made based on the fact that if the threshold is too large, then multiple instances of an action may be considered as one instance. A 2σ limit will cover approximately 95% of the distribution, effectively allowing outliers to be detected as multiple instances of the same action.

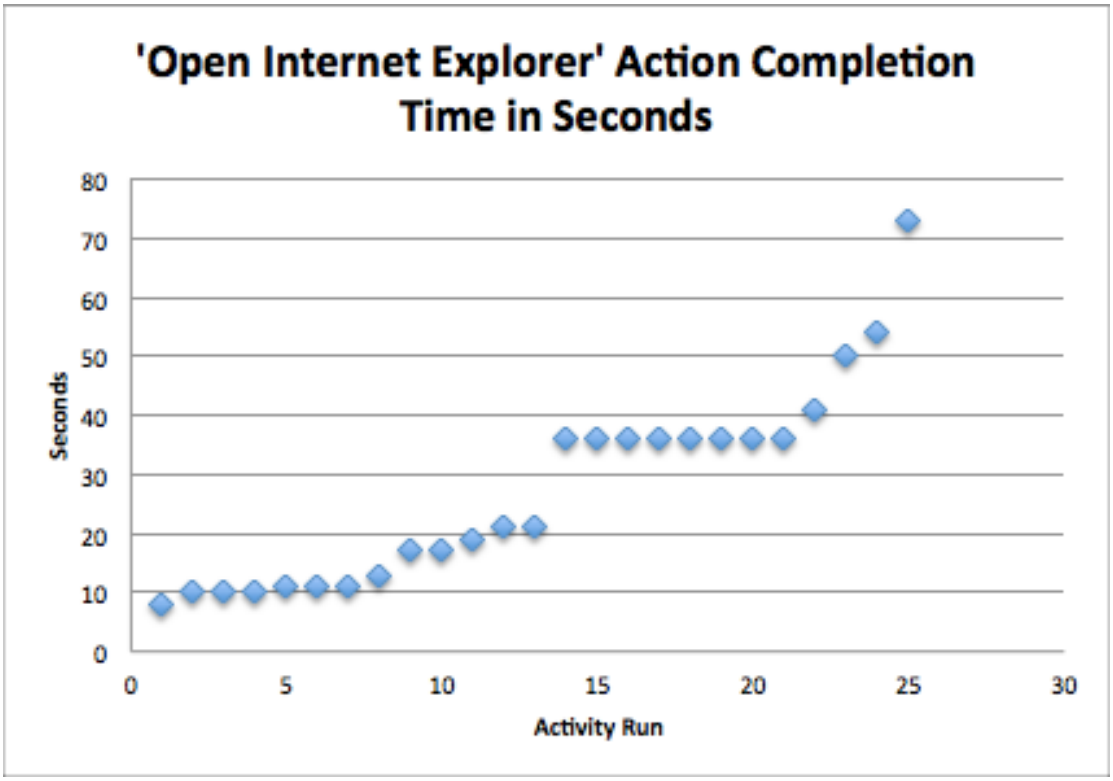


Figure 8.1 Graph of the time in seconds it took for the action 'Open Internet Explorer' to complete on the tested system ordered from shortest to longest run

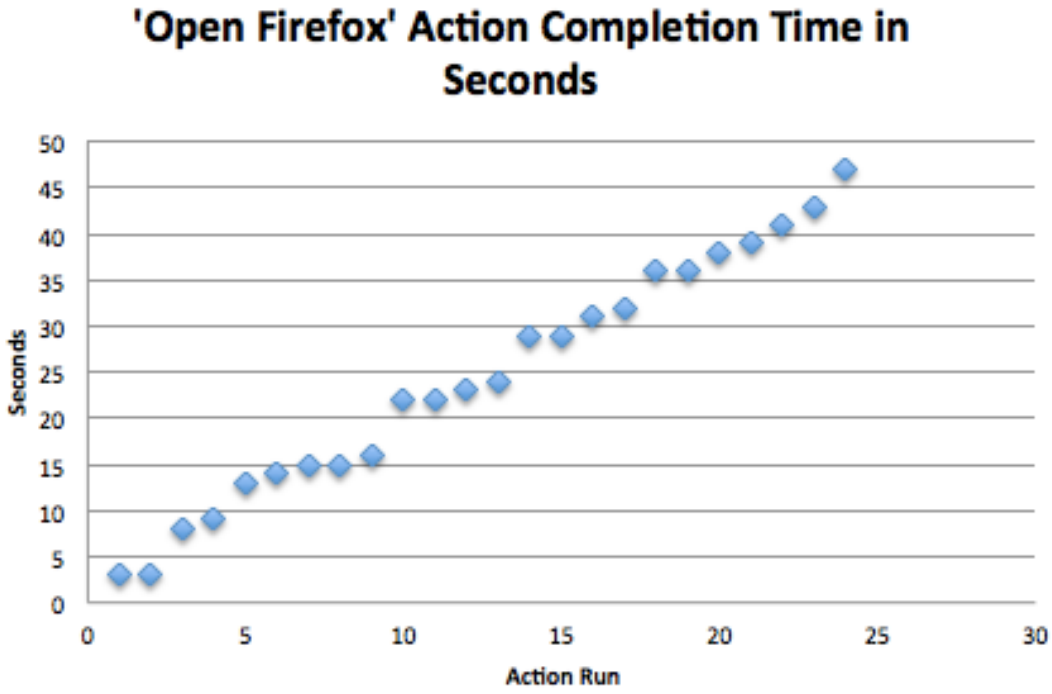


Figure 8.2 Graph of the time in seconds it took for the action 'Open Firefox' to complete on the tested system ordered from shortest to longest run

For the action “Open Internet Explorer 8”, the average trace update duration was 27.4 seconds, with a standard deviation of 16.76 seconds. The update threshold with a 2σ limiter is from 0 to 61 seconds. Figure 8.3 shows a histogram of then given data specifically for the action ‘Opening Internet Explorer’. From Figure 8.3 it can be seen that update durations become fewer as time increases. In this case, the majority of update durations took place between 8 and 42 seconds after the action instance. After which there was a decline in the number of update durations per interval, with no update duration that lasted longer than 76 seconds.

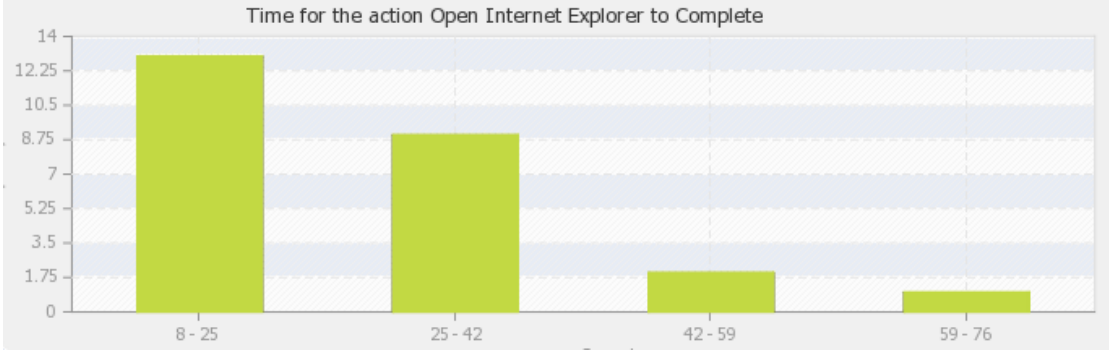


Figure 8.3 Histogram of Internet Explorer update interval times in seconds where the X axis is time in seconds and the Y axis is the number of occurrences within the update duration

By modeling the data as a normal distribution, a standard threshold limiter (θ) can be calculated, which, in the case of Opening Internet Explorer, limits the maximum update threshold to 61 seconds.

For the action “Open Firefox”, the average execution update duration was 24.5 seconds, with a standard deviation of 12.96 seconds. The threshold with a 2σ limiter is from 0 to 50 seconds. Figure 8.4 shows a histogram of then given data specifically for the action ‘Open Firefox’. From Figure 8.4 it can be seen that update durations become fewer as time increases. In this case, the majority of update durations took place between 12 and 39 seconds after the action instance. After which there was a decline in the number of update durations per interval, with no update duration that lasted longer than 48 seconds.

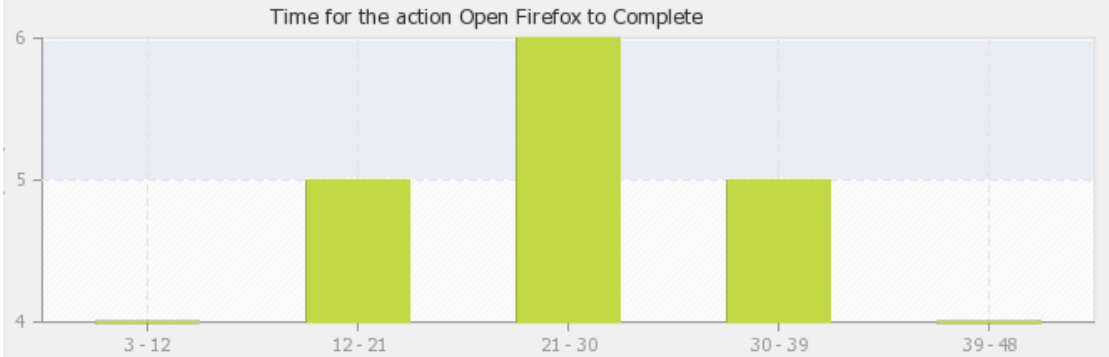


Figure 8.4 Histogram of Firefox update interval times in seconds where the X axis is time in seconds and the Y axis is the number of occurrences within the update duration

By modeling the data as a normal distribution, a standard threshold limiter (θ) can be calculated, which, in the case of Opening Firefox, limits the maximum update threshold to 50 seconds.

8.3.1 Action Instance Time Span Approximation

With knowledge of the object update threshold associated with a particular action, the time of the action instance may be approximated based on the associated object time stamp values. Objects are associated with action instances through observation, as shown in Section 8.6. Once objects are associated with a particular action instance, and have been categorized by their update patterns, the time span in which the action instance must have happened can be approximated.

First, each time stamp value in the set of returned time stamps is sorted from oldest to newest. For all objects where difference in time starting from the oldest to newest returned time stamp value is less than or equal to the action instance update threshold, these objects are grouped. The approximate time span of the action instance that updated each object is greater than or equal to the most recently updated (newest) time stamp in the set of grouped objects minus the action instance update threshold, and is less than or equal to the least recently (oldest) time stamp in the set of grouped objects.

For example, an action instance associated with time stamps t_1 and t_2 may be approximated based on the maximum action instance threshold where the instance must have occurred in the timespan before the least recent timestamp t_1 and most recent timestamp t_2 minus θ. The time-span in which the action instance may be

bound is denoted as $(t_2 - \theta) \leq i.\tau \leq t_1$. This time-bounding method to approximate the time of the action instance is shown in Figure 8.5.

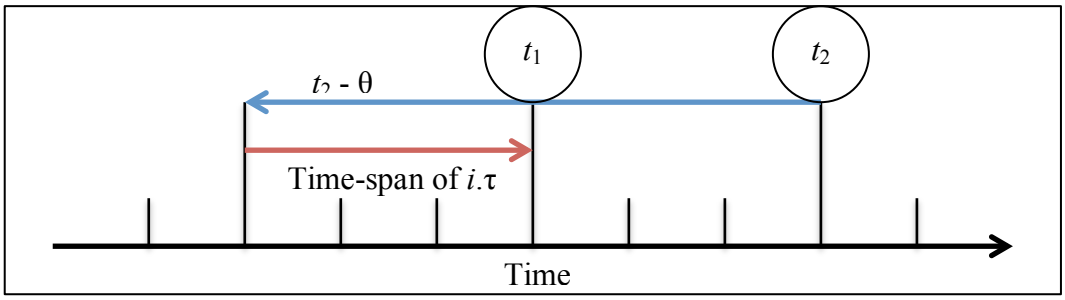


Figure 8.5 Action instance time-span approximation based on time bounding before the least recently updated timestamp (t_1) and the most recently updated timestamp (t_2) minus the action's associated update threshold (θ).

Using this method, approximation of the time-span in which each action instance must have occurred may be determined. However, if the trace update time lies within the object update threshold of multiple actions, then determination of which specific instance updated the timestamp is impossible.

8.4 Signatures of Action Instances

This work submits the hypothesis that signature based methods may be used to automate the trace observation, action inference and action instance approximation tasks. For the task of observation, the list of associated object time stamps (*Ti*) and their relation to the action instance must be known. For action instance execution approximation, the action instance execution threshold (θ) is required, where an unknown (*null*) value equals any time in the past. And for the inference task, understanding of the underlying relation between the observed facts and the inferred conclusion is required. Knowledge of the system may be encoded as a trace update consistency-checking function (*cm*). From this, a signature is defined as:

$$S = \{ Ti, \theta, cm \}$$

where

- *Ti* is the set of object-timestamps pairs, where objects and their timestamps are updated by the action *aa*
  - $Ti = \{[o, t] \mid t \in o, t \in U(aa)\}$
- $\theta = \Phi(\Delta\tau_1, \Delta\tau_2, \Delta\tau_3, \Delta\tau_4 \ldots)$

    - Φ is a threshold calculation function that accepts the set of experimentally determined Δτ for a number of *aa* executions and calculates θ as discussed above.
- *cm* is the update consistency checking function specific to the category of object update patterns that tests some property of objects and timestamps comprising Ti

This section, however, will focus on the update consistency checking function (*cm*), and the definition of three main time stamp related update patterns.

8.4.1 Core Object Time Stamp Consistency

Core object time stamps are defined as *a subset of time stamps S_{core} in T that are updated to the current value of the system clock on the occurrence of each execution of a single, specific action*.

All of the time stamps in a Core set are said to be in the 'always updated' time stamp category. Using this definition, if any trace in a Core set is observed then it can be inferred that the action instance must have happened since the artifact relates to one, and only one, action.

Also, since Core time stamps are 'always updated', it is expected that each time stamp will be within a certain time range of each other depending on the particular object update threshold of the action.

An example of a Core trace would be a configuration file that is always modified when its related program, Program X, is closed. If the configuration file were only modified when Program X is closed, the modification time stamp of the configuration file would be a Core trace for the action "Close Program X".

From this definition, an object time stamp update consistency function (*CoreTest*) can be derived to test whether each object update conforms to the Core signature category. In the case of Core, if each trace has been updated within θ, then the execution time for the action instance can be time-bound before the oldest time in the array. Action instance approximation will be further discussed in section 8.5.

First, a function *getTraceStates* is defined (Table 8.1) to return the state of time stamps of all objects defined in *S*. For each object specified in the signature, add the object, time stamp and time stamp value to the array *TraceStates*.

Table 8.1 Algorithm getTraceStates that returns an array of doubles with the object and timestamp identifier, and the current observed value of the timestamp

ALGORITHM *getTraceStates*(*O'*, *S*)

//Input: All objects in the final state of the system (*O*'), a signature (*S*) containing a list of objects and timestamps; an update threshold; and a consistency-checking function
 	
//Output: An array of doubles with the object and time stamp identifier, and the current observed value of the timestamp

foreach object in *S* that are a subset of *O'*   **add** the [[*object*, *timestamp*], *value*] double to the array *TraceStates* **done** **return** the list of doubles in *TraceStates*

Next, the function *CoreTest* (Table 8.2) defined that accepts an object update threshold and the *TraceStates* array. First the *TraceStates* array is sorted based on the time stamp values, where element 0 is the oldest and n-1 is the newest (most recent) time stamp value. If the oldest time stamp value in *TraceStates* plus the object update threshold is less than the most recent time stamp value in *TraceStates*, then the Core traces are not consistent. If the oldest time stamp value plus the object update threshold is greater than the most recent object update value, then the Core traces are considered consistent. The function *CoreTest* returns the array *Detected*, which is a single element array containing a double with the oldest and most recent time stamp values in *TraceStates*.

Table 8.2 The algorithm CoreTest that returns an array with the oldest and most recent time stamp values from a given array

ALGORITHM *CoreTest*(θ, *TraceStates*)

//Input: The object update threshold, an array of doubles with the object and timestamp identifier, and the current observed value of the timestamp
 	
//Output: A single element array containing a double with the oldest and most recent time stamp values in the *TraceStates* array

sort *TraceStates* from oldest to most recent **if** each element in *TraceStates* is < θ from the oldest element

```
    return the oldest and most recent time stamp values
else
   return null
```

8.4.2 Supporting Object Time Stamp Consistency

Supporting object time stamps are defined as *a subset of time stamps $S_{support}$ in T that may or may not be updated to the current value of the system clock on the occurrence of each execution of a single, particular action instance, but that will only be updated by a single, particular action.*

Supporting object time stamps are in the 'irregularly updated' time stamp category. However, similar to Core signatures, if any trace in a supporting signature is detected, then it can be inferred that the action instance must have happened since the trace also relates to one, and only one, action.

A time stamp can be irregularly updated if, for example, a file is cached in memory after the first execution of an action. If the file data cached in memory, rather than the file on disk, is accessed on the next execution of the action instance then the trace update will not be observable on the disk. In this case the original file's meta-data on disk would not be updated on the execution of the second action instance.

From this definition, an object time stamp update consistency function (*SupportTest*) can be derived to test whether each trace conforms to the supporting signature category. In the case of supporting, if each trace has been updated within θ, then the execution time for the action instance can be approximated to be at, or shortly before the oldest time in the array; however, objects may not always be updated. If any object time stamp is updated outside of θ from another related object time stamp, then it can be inferred that a separate instance of the same action must have happened.

The function *SupportTest* is defined (Table 8.3) that accepts an object update threshold and the *TraceStates* array. First, the *TraceStates* array is sorted based on the time stamp values, where element 0 is the oldest and n-1 is the newest (most recent) time stamp value. Each object time stamp value is compared to the oldest time stamp value in the *TraceStates* array. The comparison takes place until the time stamp value plus the object update threshold is less than the newest compared time stamp value in the array. When this happens, all time stamp values are assigned to the action instance

that must have occurred between the oldest time stamp value and the most recent time stamp that is still less than the threshold. The oldest time stamp is then replaced with the most recent time stamp that is greater than the threshold, and the process starts again.

Table 8.3 The algorithm SupportTest that returns an array with the oldest and most recent time stamp values from a given array grouped by update threshold

ALGORITHM *SupportTest*(θ, *TraceStates*)

//Input: The object update threshold, an array of doubles with the object and timestamp identifier, and the current observed value of the timestamp
 	
//Output: An array of doubles with the oldest and most recent time stamp values in the *TraceStates* array grouped by the object update threshold

sort *TraceStates* from oldest to most recent
set *timeValue to* the oldest time in *TraceStates*
foreach element in *TraceStates*
  if *timeValue* + θ > the current element
  get the **next** element
  else
  add [*timeValue*, previous element] to array *Detected*
  set *timeValue* to the current element value
done
return the list of doubles in *Detected*

8.4.3 Shared Object Time Stamp Consistency

Shared object time stamps are defined as *a subset of time stamps S_{shared} in T that may or may not be updated to the current value of the system clock on the occurrence of each execution of multiple actions*.

Shared object time stamps may be either 'always updated' or 'irregularly updated' category types depending on the action. Since the particular trace may be associated with more than one action, it is possible that any associated action instance could have updated the trace. Thus, without additional information, the only information that can be inferred from the detection of a shared object time stamps is that at least one of the associated actions must have happened.

An example of a shared object time stamp would be the access time stamp of a .dll file. Multiple actions can cause the .dll file to be accessed, so when examining the

system in a post-mortem environment with no additional information, each action that causes the .dll file to be accessed has the same probability to have updated the accessed time stamp.

From this definition, an object time stamp update consistency function – *SharedTest* – is derived (Table 8.4) to test whether each trace conforms to the shared object update category, and determine what action the trace is associated with. In the case of the shared category, if each trace has been updated within θ, then the execution time for the action can be approximated to be at, or shortly before the oldest time in the array; however, objects may not always be updated. If any object is updated outside of θ from another object, then a separate execution of the same action may be inferred. Traces may also be associated with multiple actions. Because of this, additional context is needed to determine which action caused the trace. Action-to-trace association methods, and their weaknesses, will be discussed further in Chapter 9. Keeping with the currently defined signature creation model, at best what can be said when observing a shared trace is that all actions associated with the trace could have happened within their respective update thresholds. For this reason, the consistency checking of a group of shared objects is much like consistency checking of supporting objects. Each object time stamp value is compared from oldest to most recent. Each time stamp is considered to be associated with one action instance if the value is within the update threshold. The result is an array with multiple instances of the action. In the case of a shared signature, the objects may be tested more than once, since they will be present in multiple signatures. This means that a trace may be associated with multiple actions, and all actions associated with the shared trace are assumed to have happened.

Table 8.4 The algorithm SharedTest that returns an array with the oldest and most recent time stamp values from a given array grouped by update threshold

ALGORITHM *SharedTest*(θ, *TraceStates*)

//Input: The object update threshold, an array of doubles with the object and timestamp identifier, and the current observed value of the timestamp
 	
//Output: An array of doubles with the oldest and most recent time stamp values in the *TraceStates* array grouped by the object update threshold

sort *TraceStates* from oldest to most recent
set *timeValue to* the oldest time in *TraceStates*
foreach element in *TraceStates*
  if *timeValue* + θ > the current element
  get the **next** element
  else
  add [*timeValue*, previous element] to array *Detected*
  set *timeValue* to the current element value
done
return the list of doubles in *Detected*

8.4.4 Determination of Multiple Action Instances

From the three main consistency functions defined, traces will be associated to the same action if each has been updated within the given object update threshold. To reiterate: in the situation of an overlap of two actions instances where a trace could be associated with either instance of the action, a search for the oldest and most recent timestamp in the set will be conducted. If the update duration between the oldest time stamp value and the most recent time stamp value is greater than the defined object update threshold, then at least two instances of the action must have happened. For example, a time stamp $t2$ is observed. $t2$ is associated to an action whose signature consists of the set $\{[o1, t1], [o2, t2], [o3, t3]\}$, and whose threshold (θ) is 60 seconds. If $t1$ = 12:59:30, $t2$ = 13:00:00, and $t3$ = 13:00:58 then $t2 - t1 < \theta$ and $t3 - t2 < \theta$. In this case $t1$ and $t3$ could be associated to the same action instance since $\theta < 60$ when compared to $t2$, even though $\theta < t3 - t1 = 88$. To handle this situation, when a trace is found, all existing time stamp values associated to the action (all time stamps in the signature) are observed and sorted. The oldest time stamp value is then used as the point from where all other returned time stamps values are compared. If any time

stamp is greater than θ from the oldest time stamp, then multiple instances of the action must have happened.

8.5 Action Instance Signature Portability

To be practical, once a list of traces associated with a certain action is generated and classified, the related objects and traces must then be generalized to allow identified traces to be detected on suspect systems other than in the test environment. This must take into account system-unique usernames, program paths, etc. To do this, any user or system-specific paths would have to be generalized. Take the Windows prefetch file as an example:

C:\Windows\Prefetch\IEXPLORE.EXE-**27122324**.pf

The system-unique identifiers, in this case the system drive and the string of numbers, would need to be replaced with variables, as so:

%SystemRoot%\Prefetch\IEXPLORE.EXE-**%s**.pf

Where the variable %SystemRoot% is the location of the Windows system folder including the drive and path, and %s is a string of numbers and letters.

The generalization should include the possibility that programs may be installed in non-default locations. This means that other information sources, such as reading the installation path from the Windows Registry[13], may be required. This generalization will allow signatures generated on one system to be used in the analysis of other systems, however, it cannot be guaranteed to work in every situation, depending on how the suspect has configured his or her system. Further, generalization allows a greater possibility for false positives. For example, if the prefetch folder were copied from one system to the suspect system, this copy of the prefetch folder would also likely be detected, and considered part of the suspect system.

To be usable a generalized object must be as generic as possible while still returning the correct object from the suspect system with no false positives. In this work, we propose the use of Regular Expressions for pattern matching objects defined in signatures.

[13] One Windows Registry key containing installed program path information is: HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall

8.5.1 Regular Expression Representation for Object Detection

When attempting to generalize objects in a signature, the object entries must be able to match a range of possible file paths and names while at the same time returning only objects related to the action. These paths and names may contain random strings that differ between systems, but the overall path and file name structure may have a detectable pattern. The second generalization is the particular trace of interest associated with the object. For example, a specific time stamp of interest. These two pieces of information allow for the observation of the state of the trace in a random system.

Regular expressions[14] are a commonly used, flexible method to match patterns of characters. The pattern generalization will depend on the input and trace of interest, but for the purpose of this example the output of The Sleuth Kit's mactime[15] command will be used to match file path, name and time stamp information. Object-trace pairs in the set *Ti* can be defined as regular expressions to support portability and allow analysis of the many file system types that The Sleuth Kit supports. An example of objects and traces converted to regular expressions is shown in Table 8.5. This table shows that time stamps of interest (from the mactime format) are in the form "macb", representing modified, access, and created times for the object (Carrier 2009). If the object's modified time stamp is of interest, the time stamp of interest can be generalized in the form of a regular expression as "m[a|\.][c|\.][b|\.]". Meaning that the modified (m) time is required, and the other timestamps (acb) may or may not be set. This will return the time associated with the modified time stamp regardless of the value of the other time stamps.

Objects are characterized by their filename and path. Objects can also be generalized by determining the minimum amount of information that is needed to uniquely identify the object while still returning the expected object and no false positives or negatives. When trying to access the same object in the same operating system, just installed on different hardware, generalization should allow the object to be installed

[14] For more information on Regular Expressions, see http://www.bsd.org/regexintro.html

[15] For details on The Sleuth Kit's mactime format and how it represents mac times from different operating systems, see http://wiki.sleuthkit.org/index.php?title=Mactime_output

on a different logical partition, account for differing user names, or allowing for random strings in the file name. From the previous example:

C:\Windows\Prefetch\IEXPLORE.EXE-**27122324**.pf

Instead of using system-specific variables, regular expressions could be used. The result may be similar to:

.*/Prefetch/IEXPLORE\.EXE-**.***\.pf**$**

By using the "match any character any number of times (.*)" regular expression, the system drive and Windows system folder can be generalized. In this example, if the prefetch folder is moved to any other location it can still be detected as long as it is not renamed as well. If the folder is renamed, detection will fail. Also, the numeric string at the end of the file name correlates to the time the program was executed. Since this number is variable, it can also be replaced with a "match any character any number of times" expression. Since the remainder of the filename is very specific, this will allow the correct object to be located by file name even if the numeric string changes.

Table 8.5 Firefox trace category, trace and corresponding object of interest

Type (*cm*)	**Time stamp (trace)**	**Objects related to "Opening FF3"**
Core	**m[a|\.][c|\.][b|\.]**	.*/Firefox/Profiles/.*default/urlclassifierkey.\.txt
Core	**m[a|\.][c|\.][b|\.]**	.*/Prefetch/Firefox\.EXE-.*\.pf$
Support	**[m|\.][a|\.]c[b|\.]**	.*/Prefetch/Firefox\.EXE-.*\.pf$
Support	**[m|\.][a|\.]c[b|\.]**	.*/Firefox/Profiles/.*default/cookies.sqlite-journal
Support	**[m|\.][a|\.]c[b|\.]**	.*/Firefox/Profiles/.*\default/urlclassifierkey.\.txt
Support	**[m|\.][a|\.]c[b|\.]**	.*/Firefox/Profiles/.*default/startupCache$
Support	**[m|\.][a|\.]c[b|\.]**	.*/Firefox/Profiles/.*default/pluginreg.dat

8.6 Experimentation

To determine the action that is the cause of a particular set of objects being updated, a method of repeatedly observing the effect of an action instance on the system is proposed. A basic method has been defined by James, Gladyshev et al. (2010), where Windows Sysinternals Process Monitor[16] (procmon) is used to monitor file system activity in a Microsoft Windows system while a particular action is executed. This produces a comprehensive list of object time stamps that are updated because of the given action instance. By executing the action many times and analyzing each update result, the object time stamp update patterns can be determined. From this, action traces can be categorized by their update patterns. Along with the determination of associated traces, normal execution duration for the action must also be defined to be able to approximate the time in which the action instance occurred.

8.6.1 Identification of Action Instance Traces

This section will discuss the process of deriving objects and timestamps associations for a given action. This example will focus on the action "Opening Internet Explorer".

The experiment in this section will be conducted on computer running Windows XP service pack 3 with Internet Explorer 8.0.6001.18702, all with default settings. Windows XP was chosen because according to James (2010; 2011), at the time of this work Windows XP was still the most commonly encountered operating system by surveyed Law Enforcement in digital forensic investigations.

In the first exploratory analysis, Microsoft Process Monitor[17] (procmon) was used to record all system calls executed during the action "Opening Internet Explorer 8 (IE8)". The initial tests recorded all system activity. The action was executed 400 times per test via a macro utility that can repeatedly simulate mouse and keyboard interaction. In order to minimize noise (unrelated system calls) generated by other running processes, the entries that were not present during every run were removed. Three of these tests were conducted with 400 action executions per test. The filtering process reduced the number of traces from around 11,000 to approximately 4,000 (Figure 8.6), however noise was consistently found to be present.

[16] More information on Procmon can be found at: http://technet.microsoft.com/en-us/sysinternals/bb896645

[17] http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx

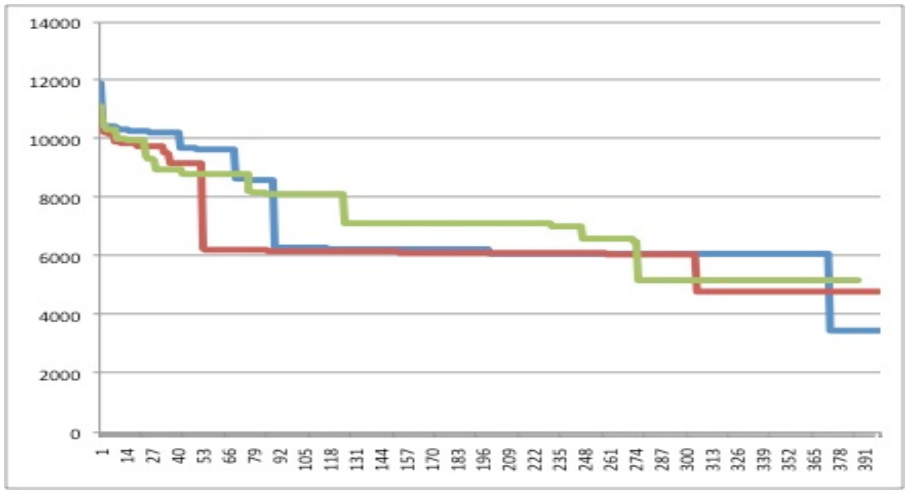


Figure 8.6 Removal of noise caused by background processes where the x-axis is the number of times Internet Explorer has been run and the y-axis is the number of traces common to each test

Using this data it was determined that by filtering the Process Monitor output with the selected program's process name (iexplore.exe), as well as the "explorer.exe" process, similar results of around 4,000 traces could be achieved without needing to repeat the action hundreds of times.

From the exploratory analysis, three observations were made. First, that when using procmon to monitor the resulting traces of an action that causes one specific process, filtering procmon for all but that specific process was able to reduce noise while still detecting the same related traces. Next, object and trace derivation and categorization is a two-phase process, which will be discussed further in section 8.6.2. And finally, objects that are not always updated may still be associated with the action, and should not be filtered.

8.6.2 Object and Trace Derivation and Categorization

Object to action relation derivation consists of two distinct phases. Phase 1 involves determining the objects associated with the particular action instance. Phase 2 involves determining which specific object time stamps are updated by the action instance as well as the time stamp update pattern needed to specify the associated consistency-checking model. These phases will be illustrated with examples for deriving signatures for Internet Explorer 8 (IE8) and Firefox 3.6 (FF3).

Phase 1: To determine the traces associated with starting IE8, Process Monitor was started and configured to filter out system calls not associated with the "iexplore.exe" and "explorer.exe" processes. The procmon capture was then cleared and IE8 was opened. After 120 seconds the procmon capture was stopped and exported as a comma separated value (.csv) file, an excerpt of which is given in Appendix D. This process was completed 10 times, after which all 10 resulting files were combined. Duplicate entries in the combined file were removed so each line is unique. This file is then renamed to the action being tested and given a .sig extension. In this case the file name is, 'IEOpen.sig'. The resulting file contains a list of paths of files and Registry keys that are somehow accessed during the opening of Internet Explorer 8. It contained 3,161 file and Registry traces. Most of which, however, were Windows Registry entries that were not Registry keys, and therefore have no associated timestamp information.

The same process was conducted for FF3. An excerpt of the resulting procmon output is given in Appendix E. In this case the resulting activity signature is named 'FFOpen.sig'. The resulting file contains a list of paths of files and Registry keys that are somehow accessed during the opening of Firefox 3.6. It contained 1,923 file and Registry traces. Most of which, however, were Windows Registry entries that were not Registry keys, and therefore have no associated timestamp information.

The result of Phase 1 for signature derivation is a list of all objects that were modified or accessed during the time that the experiments took place for the single action that was being tested.

Phase 2: Phase 2 involves observing the identified object update patterns to determine the specific traces related to the action. A Perl script 'sigtest.pl' (Appendix F) was written to retrieve the associated file and Registry timestamp value information from the objects identified in Phase 1.

To observe the behavior in which the identified object time stamps are updated when the action takes place, IE8 and FF3 were opened respectively as before. The difference from the first phase is that procmon was not used, and after each action instance, 'sigtest.pl' was executed to output the time stamp values of the file and Registry entries specified in the previously created 'IEOpen.sig' and 'FFOpen.sig' files. 'sigtest.pl' was executed between 5 and 20 minutes after the action instance.

This process was carried out 10 times over three days for each type of action, with each action instance time being recorded. An excerpt of the produced file timestamp information is given in Appendix G. By comparing object timestamps in relation to the known time of the action instance, time stamp update behaviors can be analyzed from which update patterns can be derived.

8.6.3 Object Time Stamp Update Behavior Analysis

From the data produced in the previous section, a manual analysis of timestamp update behaviors took place. This section details the analysis of update patterns for Internet Explorer 8, however, the same major update patterns were also found for Firefox 3.6. For each action instance that was recorded, a timestamp would either be before (not updated), or after (updated) at the same time as the action instance. Based on these update patterns over the 10 recorded instances of the action, timestamp update behaviors can be determined.

This section will list the observed object time stamp update patterns that will be used to define specific trace update categories. Note that in Windows XP each file has three associated timestamps, and therefore the same object may be listed multiple times.

- *Always Updated File and Registry Key Timestamps* (*AU*): It was observed that 21 file and Registry time stamps were updated each time IE8 was opened, however, other processes could have updated these time stamps as well. Of these, 9 files had updated ‘accessed’ times, 10 files had updated ‘modified’ times, and 9 Registry keys had updated ‘modified’ times. This specific category can be further subdivided into five more-specific update categories based on the uniqueness of observed update patterns. These subcategories are explained as follows and are summarized in (Table 8.6):

Table 8.6 Internet Explorer 8 Always updated sub-category update patterns showing which object time stamps are updated, unchanged, or unpredictably updated

	Modified Time	Accessed Time	Created Time
AU1	**Updated**	**Updated**	**Unchanged**
AU2	**Updated**	**Updated**	**Unpredictably**
AU3	**Unchanged**	**Updated**	**Unchanged**
AU4	**N/A**	**Updated**	**N/A**
AU5	**Updated**	**Unchanged**	**Unchanged**

- *AU1*: Files categorized in the *AU1* group were found to update their accessed and modified time stamps every time Internet Explorer was started, but also with the execution of unrelated actions. Of these it can be said that their updated timestamps must be greater-than or equal to the time of the most recent execution of IE8. It was also observed that the created time stamps of these files are less-than or equal to the installation of the operating system itself.

- *AU2*: Files categorized in the *AU2* group were found to have their accessed and modified time stamps updated with each execution of IE8. Of these files, one was the prefetch[18] file for Internet Explorer. Its created timestamp was found to correlate with the first time Internet Explorer was run on the system. Only the accessed and modified times were updated with each action instance. The other two files were Internet Explorer 'cookie' files correlating to 'administrator@live[1].txt' and 'administrator@msn[1].txt' where 'administrator' is the name of the local user account. The accessed and modified time stamps of these files were updated with each action instance, and the created timestamps were found to update often with the action instance, but not always and with no predictable pattern. Of these files it can be said that any time stamp with a value before the most recent action instance denotes the time of a previous action instance.

[18] More information about Windows prefetch files can be found at: http://www.microsoft.com/whdc/archive/XP_kernel.mspx#ECLAC

- *AU3*: Files categorized in the *AU3* group were found to have only their modified, and not their accessed, time stamps updated with each opening of Internet Explorer 8.
- *AU4*: Registry keys in the *AU4* group were found to have always had their associated time stamp information updated.
- *AU5*: Files categorized in the *AU5* group were found to have only their accessed time stamps updated: IExplore.exe and shell32.dll.

- *Timestamps Updated on the First Run Only* (*FRO*): It was observed that 1 Registry time stamp was updated only during the first opening of IE8 per user login session[19].
- *Usage-Based File Timestamp Updates* (*UB*): It was observed that 4 Windows shortcut (.lnk) files' accessed time stamps were updated often, but not always when they were used to start IE8. If they were not used to start IE8 they were never updated by the action instance.
- *Irregular Update of Timestamps* (*IU*): It was observed that 93 files had 'irregular' time stamp update patterns, and each in this category had only its accessed time stamp updated.
  - *IU1*: Although the majority of the traces categorized as *IU* are seemingly irregular, it was observed that cookies within the user's "Cookies" folder were always updated on the first run of the session, and then irregularly updated during the starting of IE8 in the same user login session, making cookie files a combination of *FRO* and *IU*.

8.6.4 Categories of Object Time Stamp Update Behavior

By design of the experiment we could observe update patterns from which four primary categories of object time stamp updates can be defined. Two important observations apply to each category. First, as previously described, in our experiments we observed that specific trace update patterns are the result of a specific action instance; for example, double-clicking an icon causes a program to execute. This

[19] A 'user login session' is defined as the time-span in which a user account is logged into the operating system until the account is logged out.

process is not instantaneous and therefore any observable traces were created or updated some time *after* the actual action instance. Second, it was observed that each trace was updated within a given period of the action instance. This means that the update process must also be defined as a *time-span*, and is not instantaneous, as previously discussed in section 8.3.

Category 1: *Always Updated Traces* – 6 files and Registry entries with associated time stamp information were consistently updated each time, and only when, Internet Explorer 8 was opened (Table 8.7). Traces that are always updated by opening IE8, as well as by other actions, have been removed. The remaining traces in this category are defined as 'Core' traces, as they are the most reliably updated.

Table 8.7 Internet Explorer 8 file and Registry traces updated each time Internet Explorer 8 is opened

C:\WINDOWS\Prefetch\IEXPLORE.EXE-27122324.pf
C:\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Feeds Cache\index.dat
HKEY_CURRENT_USER\Software\Microsoft\CTF\TIP
HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Security\AntiPhishing\2CEDBFBC-DBAB-43AA-B1FD-CC8E6316E3E2
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Ext\Stats\{E2E2DD38-DO88-4134-B2B7-F2BA38496583}\iexplore
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Ext\Stats\{FB5F1910-F110-11D2-BB9E-00C04F795683}\iexplore

Category 2: *Traces Updated on the First Run Only* – One registry entry and all cookie files were found to have their timestamp information consistently updated on the first run of Internet Explorer 8 *per user session* (Table 8.8).

Table 8.8 Internet Explorer 8 Registry trace updated during the first run of the session

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MenuOrder\Favorites\Links

Category 3: *Irregular Update of Timestamps* – 93 files (Appendix H) were found to have their time stamps updated in an irregular fashion. In this case, irregular means that the time stamps were sometimes, but not always, updated, and no specific pattern could be determined. Irregular updating timestamps could potentially be caused by caching functions of the operating system, or other such interference that would result in a non-deterministic update of a timestamp given an action instance.

Category 4: *Usage-Based Timestamp Update* – 4 Windows shortcut files were identified that were inconsistently updated when the particular link file itself was used (Table 8.9), and never during the starting of IE8 when the file itself was not used.

Table 8.9 Internet Explorer 8 traces updated only when the specific trace is used (link files)

C:\Documents and Settings\Administrator\Desktop\Internet Explorer.lnk
C:\Documents and Settings\Administrator\Start Menu\Programs\Internet Explorer.lnk
C:\Documents and Settings\Administrator\Application Data\Microsoft\Internet Explorer\Quick Launch\Launch Internet Explorer Browser.lnk
C:\Documents and Settings\Administrator\Start Menu\Programs\Accessories\System Tools\Internet Explorer (No Add-ons).lnk

Also, from the original associated object list there were a number of file and Registry entries that were never updated during the opening of IE8. By experiment design, these entries have been discarded since, during the experimentation, it is impossible to associate them with the specific action instance.

8.6.5 Experiment Conclusions

From these experiments it can be seen that a subset of updated objects related to an action may be derived, and trace update patterns relating to these objects – given the specific action – can be determined. In these experiments, specific trace update patterns were found with little consideration about exactly what action updated the trace, save the process filtering done with Process Monitor. Instead of focusing only on how traces can be updated, it is also important to consider the actions that can update a trace. For determining what actions took place in a system, both pieces of information are needed.

In order to combine the trace update patterns with knowledge about actions that can cause the trace to be updated, this work submits the general object update categories described in section 8.2. These general update categories consider both the update behavior of traces within the category, and how other actions can affect those traces. A summary of each category is as follows:

- Core Traces
  - Always updated
  - Updated only by the single action
- Supporting Traces
  - Irregularly updated
  - Updated only by the single action
- Shared Traces
  - Always or Irregularly updated – respective of the action
  - Updated by two or more actions

With knowledge of how a trace may be updated, and what actions could potentially update the trace, it becomes possible to determine when the most recent execution of an action must have, or was likely to, occur. Further, since some timestamps are not always updated, it may be possible to determine past instances beyond the most recent instance of the action.

In these experiments, the trace derivation and timestamp update analysis process were both conducted using highly manual techniques. This was intended to allow a better understanding of trace update patterns than have been demonstrated in prior works using automated techniques such as Khan (2008), however, manual analysis for all actions is impractical, and automation that can be focused at the previously derived level of specificity should be used. Automation would increase the feasibility, and potentially the accuracy, of such object update behavior analysis techniques.

8.7 Summary

This chapter described the derivation of action instance object update patterns. A practical method for determining the relation between actions and object update patterns was given. Analysis of object update patterns allows traces to be categorized, and rules of consistency to be determined. The need for object update thresholds has been discussed, and a method for determining the object update threshold for an action was given. After, a brief discussion and method for generalizing signatures for portability across suspect systems was given. Finally, an overview of the object trace update experimentation used to derive general categories was given.

Chapter 9

Automatic Event Reconstruction Using Signature Based Analysis

This chapter begins by discussing the inferences of single and multiple occurrences of a given action instance based on the previously discussed signature matching methods. Consistency between object updates as well as higher-level consistency between actions is discussed. A probabilistic method for associating traces to actions when uncertainty exists is given, and the issues with such a method are discussed. Next, generic matching of action instances based on no prior information is then briefly described, with case examples explored and weaknesses given. After, high-level event reconstruction using the proposed signature-matching model is discussed that enables more actions to be inferred based on the presence of previously inferred action instances. Finally, the use of the proposed signature-based hypothesis reduction methods for human inference verification in digital forensic investigations is discussed.

9.1 Inferring Actions from Trace Observations

As previously shown, the state of a system changes as a direct result of actions that take place. Causal relations between actions and object updates can be observed, from which object update models can be constructed. As shown, if the action, resulting traces and the causal link between the occurrence of the action and trace creation is known, then the action can be inferred from observation of the collection of traces. In Chapter 8, derivation of traces updated with a given action and system update model creation was shown. This allows for the creation of a signature that encodes the knowledge needed to infer a given action instance. This knowledge, however, requires an observation of the state of a given system before it can be utilized. The observed state combined with the depth of encoded knowledge determines what information may be inferred.

9.1.1 Formalization of Action Instance Signature Matching

Action signature matching is equal to observing the state of the system and using the knowledge encoded in a signature to infer whether the associated action has or has not occurred. Signature matching is a function that requires two inputs: a system to observe, and a signature of an action in question (as defined in section 8.4). The

signature matching function *SignatureMatch* is given below (Table 9.2). The signature matching function mimics the process of observing the state of the system by retrieving an array of the current state of traces in the set *Ti*.

The previously described function *getTraceStates* returns the state of time stamps of all objects defined in *S*. An array, *TraceStates*, is returned in the form of a double, where the first element is the object, trace double, and the second element is the trace value of interest.

$$TraceStates[] = [[o, t], \tau]$$

For example, if the trace in question were a created file time stamp associated with the document "C:\Documents\mydoc.txt", then *getTraceState* would return the object name, time stamp and time stamp value for "mydoc.txt".

TraceStates[] = [["C:\Documents\mydoc.txt", created], 16:18:00]

The signature matching function then checks the consistency of the state of the returned trace values. For this, the function *checkConsistency* is defined (Table 9.1) that executes the consistency checking function specified in the given signature with the given update threshold and trace state values as arguments.

Table 9.1 The algorithm checkConsistency that outputs detected timestamp values that pass the action's consistency-checking function

ALGORITHM *checkConsistency*(*cm*, θ, *TraceStates*)

//Input: The trace category's consistency-checking function, the action's trace update threshold, an array containing the current state of the trace
 	
//Output: an array of times that are consistent with the given consistency-checking function

call the consistency-checking function
set the array *InferredActions* to returned results
if array *InferredActions* is empty
  exit
else
  return the array *InferredActions*

Table 9.2 The algorithm SignatureMatch that gets the current state of object timestamps defined in the signature, checks the returned timestamp values for consistency, and returns detected, consistency timestamp values representing the time in which the action occurred

ALGORITHM *SignatureMatch*(O', *S*)
//Input: The current system state, a signature for a given action //Output: An array of timestamp values representing the time in which the action occurred
foreach object in *S* **get** the value of the current object in *O'* **assign** the object and value double to the array *TraceStates* **done** **call** *checkConsistency* **set** the array *InferredActions* to the returned results **return** the array *InferredActions*

checkConsistency requires the model's consistency function (*cm*), the action instance threshold (θ) and the returned state of the traces in the *TraceStates* array. *checkConsistency* executes the consistency-checking function *cm* with the threshold and returned *TraceStates* array as arguments. The check will return an array of one or more consistent action instances or null if inconsistent. The particular update model will determine what the definition of consistency between traces. Some examples of inferred knowledge *per signature* might be:

- the action occurred
- the action did not occur
- the action occurred + the time range in which the action must have occurred
- the action occurred multiple times + the time range in which the actions must have occurred

The overall hypothesis reduction algorithm is shown in Figure 9.1. In this algorithm, for each signature, *SignatureMatch* calls *getTraceStates*. Returned trace states are then checked for consistency, based on their update category's consistency-checking function. If the objects are consistent, the represented action is added to the list of inferred actions with the detected time span, if available. Otherwise the next signature is tested.

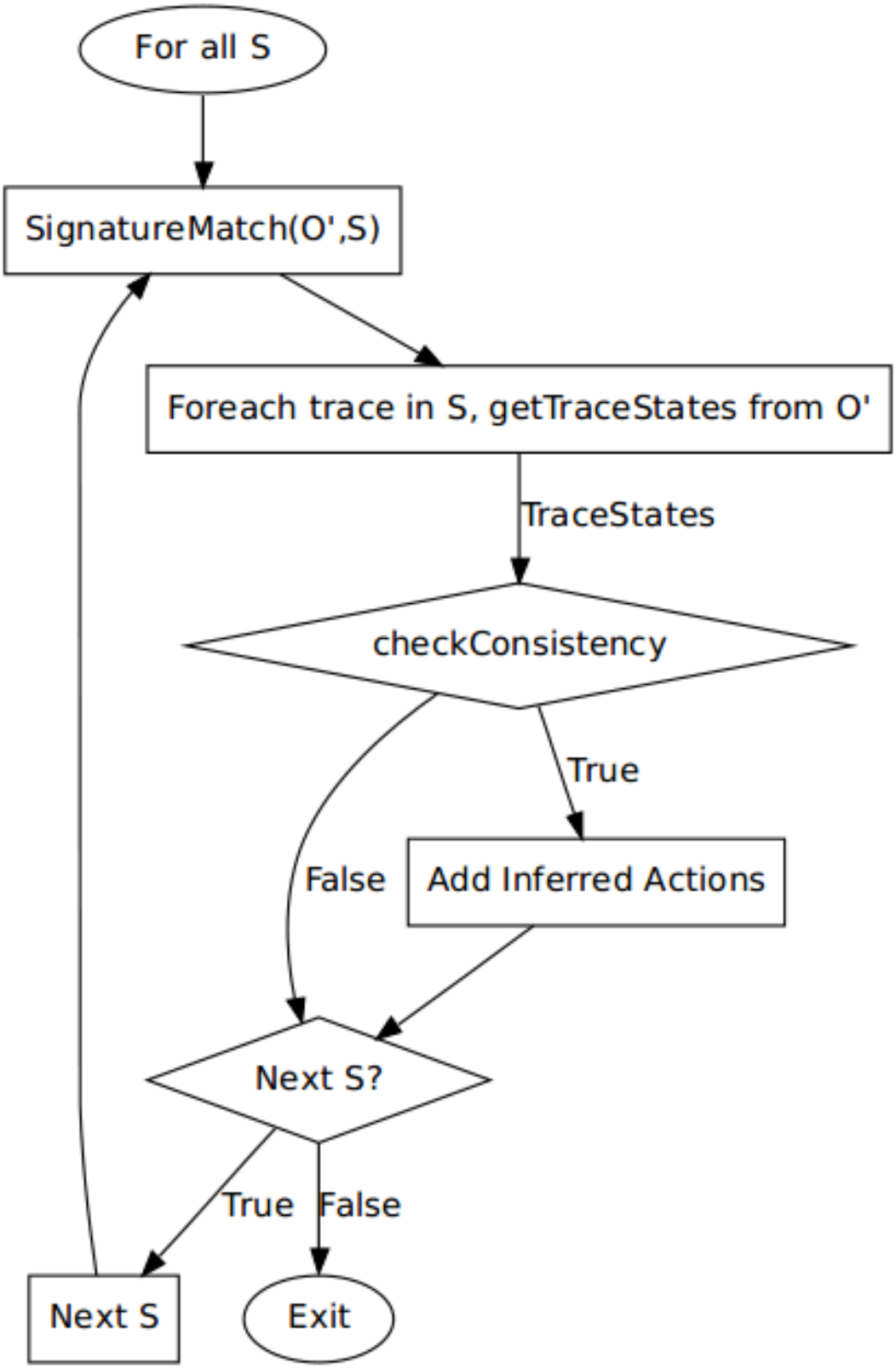


Figure 9.1 Overall hypothesis reduction algorithm that incorporates observation of the state of a system (O') and uses object update relation consistency checking from relation information encoded in signatures (S).

9.2 Inferring Single and Multiple Action Instances

With traces that are updated every time an action occurs, information about previous instances of the same action will be overwritten to the newest execution time. This means that from these traces it is only possible to determine the most recent action instance. For example, Core traces must be always updated given the associated action. Therefore, only the most recent action instance associated with the update of Core traces can be inferred regardless of how many Core traces are associated with the signature.

Some traces, however, are updated by the execution of a single action, but not on every instance of the action. As discussed in section 7.5, an action may have multiple trace update paths depending on the state of the system in which the action is taking place. For example, an action could access a file, and load the file's contents into memory. On the next execution of the action, the file contents stored in memory may be accessed instead of the original file, meaning the file's meta-data on disk may not be updated. For traces that may or may not be updated, if the trace is associated with one, and only one, action then it falls into the Supporting trace category. In the case of Supporting traces, if traces exist that are outside of the object update threshold, it can be deduced that previous instances of the action must have occurred for the traces to have been updated (or even exist). Since all traces in the Supporting signature must be associated with only one action, multiple update paths produce an update pattern over time that may reveal information about previous action instances. To illustrate, consider time stamps of four files related to the action "open a document":

Pre-fetch File: Timestamp = 0:00
Registry Entry: Timestamp = 0:00
.DLL File 1: Timestamp = 0:00
.DLL File 2: Timestamp = 0:00

With the first execution of the action, all trace time stamps are updated to the current time (6:00). The pre-fetch file is created, a Registry entry is added, and the two DLL files are loaded into memory:

Pre-fetch File: Timestamp = 6:00
Registry Entry: Timestamp = 6:00
.DLL File 1: Timestamp = 6:00
.DLL File 2: Timestamp = 6:00

The document is closed and opened again at a later time. Now assume that the Registry entry is only written the first time the program is opened to record the position of the document's window on the screen. The window position was not changed, so the Registry entry is now accessed, but not written to so the associated time stamp would stay the same while the others were updated to the new time (7:00):

Pre-fetch File: Timestamp = 7:00
Registry Entry: Timestamp = 6:00
.DLL File 1: Timestamp = 7:00
.DLL File 2: Timestamp = 7:00

The document was again closed, and opened at a later time. This time assume the window position was not changed, so the Registry entry was not updated, and the first .DLL remained in memory from the last execution of the program. The pre-fetch file is always updated, and the 2nd .DLL file was loaded into memory again, so only these two items would have updated timestamps (8:00):

Pre-fetch File: Timestamp = 8:00
Registry Entry: Timestamp = 6:00
.DLL File 1: Timestamp = 7:00
.DLL File 2: Timestamp = 8:00

This is a crude example, but it illustrates the idea that some traces are not always updated during the execution of an action, and sometimes information about previous action instances can be inferred by looking at the total collection of associated traces.

9.2.1 Consistency of Detected Actions

As shown, objects associated with the same action have a certain consistency between them. This consistency can be checked with a defined consistency model, *cm*. In the given signature-matching model, consistency-checking models are used to group sets of traces associated with a particular action instance. This model, however, only checks for the consistency between traces with the same update pattern. Multiple consistency models, such as those defined in section 8.4, can exist for a single action. Because of this, multiple consistency functions may exist per group of objects; effectively meaning that multiple signatures may exists for a single action. Further, there are also patterns of consistency across detected higher-level actions. For example, Core traces must always be updated and therefore will always be the most recently updated traces if an action occurs. In this case it would be inconsistent to find a Supporting trace for the same action that was updated *after* the core (core + θ). To

illustrate, see Figure 9.2, where action 1 (A1) Core traces are detected at time $\tau 1$. Since Core traces must be updated with each execution of A1, time $\tau 1$ must be the most recent execution of A1. Next, A1 Supporting traces are detected at $\tau 2$. Since $\tau 1$ earlier than $\tau 2$, and $\tau 1$ must have been the most recent execution of A1, then the detected Supporting traces are inconsistent.

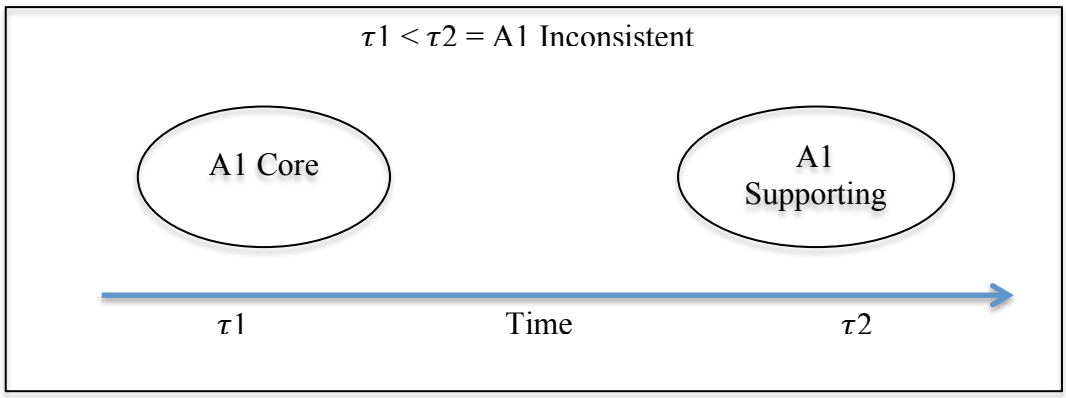


Figure 9.2 Inconsistency of traces associated with a single action detected when comparing the properties of different trace categories

Inconsistency of a signature may denote incorrect trace categorization, incorrect consistency model generation, or the use of anti-forensic methods, such as altering time stamps. However, consistency checking may also allow traces to be associated with specific actions, as will be discussed further.

9.2.2 Detection of Multiple Instances of the Same Action

The proposed signature analysis model for detecting actions uses the previously defined classes of signatures in a layered approach to build up knowledge of actions that have happened in a system. To illustrate, a fictional example of this approach is given:

An action, *ActionX*, has a Core signature (S_{XCore}) consisting of two time stamps, and a Supporting signature ($S_{XSupport}$) that has three associated time stamps. All object update thresholds are defined as 30 seconds.

S_{XCore} = { [($o1$, $t1$), ($o2$, $t2$)], 30sec., Core}

$S_{XSupport}$ = {[($o3$, $t3$), ($o4$, $t4$), ($o5$, $t5$)], 30sec., Supporting}

The Shared signature ($S_{XShared}$) for *ActionX* has two associated time stamps. Both of these time stamps are also associated with another action, *ActionY*. The Shared signature for *ActionY* is denoted as $S_{YShared}$.

$S_{XShared}$ = {[($o6$, $t6$), ($o7$, $t7$)], 30sec., Shared}

$S_{YShared}$ = {[($o6$, $t6$), ($o7$, $t7$)], 30sec., Shared}

The function *SignatureMatch*(O', S) takes the system and signature as input, and returns the value of the observed action instance update time-span. In this case, the timestamp values were chosen for the purposes of this example.

SignatureMatch(O', S_{XCore}) returns

{[“4/14/2010 19:28:25”, “4/14/2010 19:28:32”]}

SignatureMatch(O', $S_{XSupport}$) returns

{[“4/14/2010 15:13:25”], [“4/14/2010 19:28:18”, “4/14/2010 19:28:34”]}

SignatureMatch(O', $S_{XShared}$) returns

{[“4/14/2010 19:28:25”], [“5/2/2010 9:45:02”]}

SignatureMatch(O', $S_{YShared}$) returns

{[“4/14/2010 19:28:25”], [“5/2/2010 9:45:02”]}

The result of this detection process is summarized in Table 9.3.

Table 9.3 Summary of detected timestamps related to ActionX and ActionY

	ActionX	***ActionY***
Core	4/14/2010 19:28:25	
Core	4/14/2010 19:28:32	
Support	4/14/2010 15:13:25	
Support	4/14/2010 19:28:18	
Support	4/14/2010 19:28:34	
Shared	4/14/2010 19:28:25	4/14/2010 19:28:25
Shared	5/2/2010 9:45:02	5/2/2010 9:45:02

Since Core signature traces are always updated and relate only to *ActionX*, it can be inferred that *ActionX* last happened approximately at 4/14/2010 19:28:25. Both *ActionX* Core timestamps are within θ, so the traces are consistent.

With the knowledge of the last execution time of *ActionX*, the Supporting signature may now provide more information. In this case, two supporting traces confirm the last execution time (*t*3 and *t*4). Traces in the Supporting signature may not always be updated, as is shown by the supporting trace (*t*5) with a timestamp of 4/14/2010 15:13:25. This trace is consistent since the time is before the identified last execution time (*t*1). Also, since Supporting traces are associated only with one action, a previous execution of *ActionX* must have happened at this time.

Finally, Shared traces are examined. Each trace is associated with both *ActionX* and *ActionY*. The first shared trace (*t*6) has a timestamp that is within the last execution time of *ActionX*; however, *ActionY* could have also happened at this time. Calculating the probability of one trace belonging to a particular action has been discussed in Carney and Rogers (2004) and Kwan, Chow et al. (2008), and will be discussed later. Because of this, no conclusion can be made. The next trace, however, has a time that is after the detected last execution time (*t*1) of *ActionX*. Since this trace is associated only with *ActionX* or *ActionY*, it can be inferred that the trace (*t*7) must belong to *ActionY* since it is not consistent with the information known about *ActionX*. An instance of *ActionY* must have happened at approximately 5/2/2010 9:45:02, to be consistent with *ActionX*.

After this analysis and using the instance approximation method described in section 8.3.1, action instance approximations may be given, as shown in Table 9.4.

Table 9.4 Known action execution times after signature analysis

	ActionX	***ActionY***
Last Execution	4/14/2010 19:28:18 to 19:28:18	n/a
Previous Execution	4/14/2010 15:12:55 to 15:13:25	5/2/2010 09:44:32 to 09:45:02

The time stamps that are known to relate only to *ActionX* are shown in Figure 9.3. The times are grouped, where θ = 30 seconds. In the case of Core and Supporting signatures, where the traces are related only to *ActionX*, the most recent, as well as previous executions of the action can be inferred.

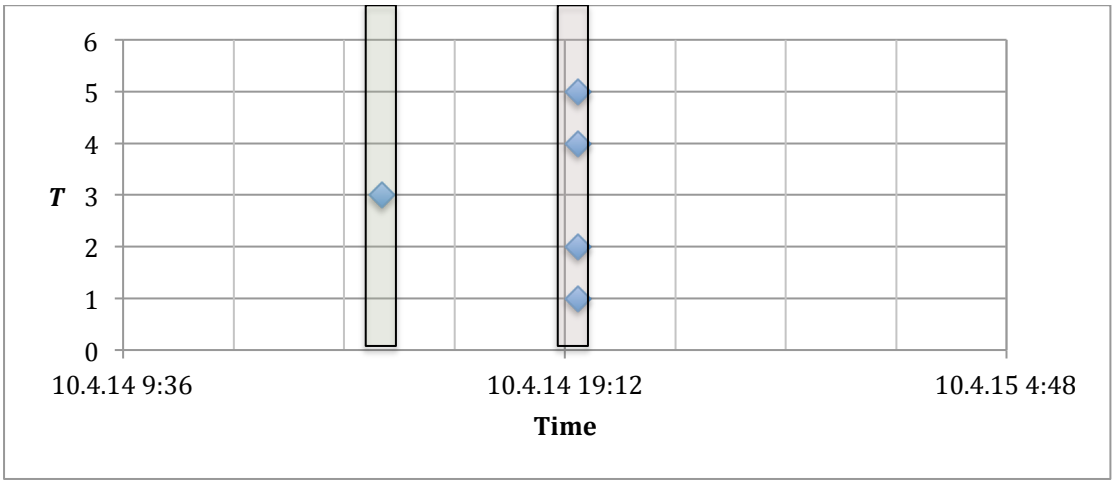


Figure 9.3 Graph of time stamps in T related to ActionX grouped by θ, that shows two distinct executions of ActionX

This example illustrates that by layering multiple observations more information about previous executions of actions can be automatically inferred. Also, by building on already detected information, inferences about other non-related actions may be made. Evaluation of this method will be presented in Chapter 10, where the process is applied to detect actions in a real environment.

9.3 Traces Updated By Multiple Actions

When a single action is associated with a particular trace, the action instance can be inferred from the observation of the trace; however, if more than one action modifies the same trace, detection of the trace without any further context can only lead to a conclusion that *at least* one associated action instance must have occurred. For example, as seen in Figure 9.4 action A1 causes process P1 that in turn creates traces T1 and T2. Likewise, action A2 causes process P2 that in turn creates traces T2 and T3. By observing only trace T2, it is impossible to associate the trace with a single action.

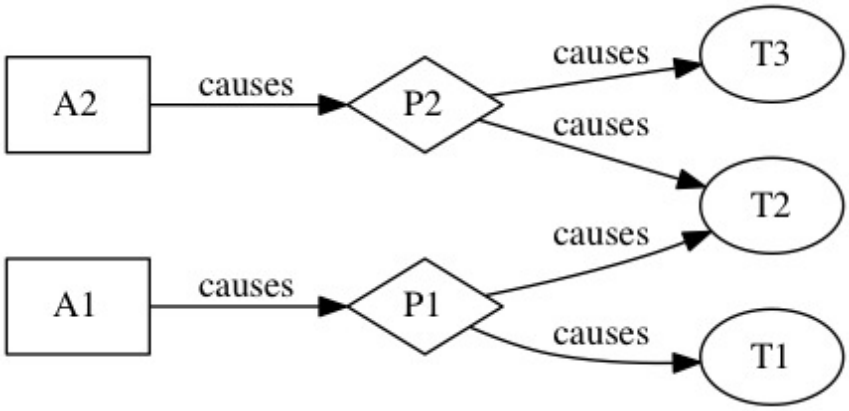


Figure 9.4 Model of multiple actions (A1 and A2) causing separate processes (P1 and P2) that create a set of traces where both processes create trace T2

In the case of traces updated by multiple actions, defined as Shared traces, rules of consistency between detected actions can help to determine which action instance the trace should be associated with. However, consistency checking does not remove all uncertainty. In these cases probabilistic methods are proposed. However, there exist some issues with currently proposed probabilistic methods, which will be discussed.

9.3.1 Consistency Checking Approach to Action-Trace Association

Consistency, as discussed earlier, exists between related objects as well as instances of an action (and multiple actions). When attempting to determine which action a trace is associated with when multiple actions could have caused the trace, additional context is needed. Additional context can be found by testing if the trace is consistent with every possibly related action. In some cases this additional context may allow for the detection of a logical inconsistency that would mean the trace has either been altered, or that the trace is not associated with the inconsistent action.

For example, see Figure 9.5, where an action A1 is associated with a particular trace O1, with a time value of $\tau 2$, and a second action A2 is also associated with the O1. In this case if the trace is associated only with A1 or A2, and A1's Core signature is determined to have occurred at time $\tau 1$ that is before $\tau 2$, then $\tau 2$ must be associated with A2. Additional context in this case comes from observing A1's Core signature to determine when the last occurrence of A1 must have been. If the value of $\tau 2$ is any time after A1's Core traces, then the trace must be associated with A2.

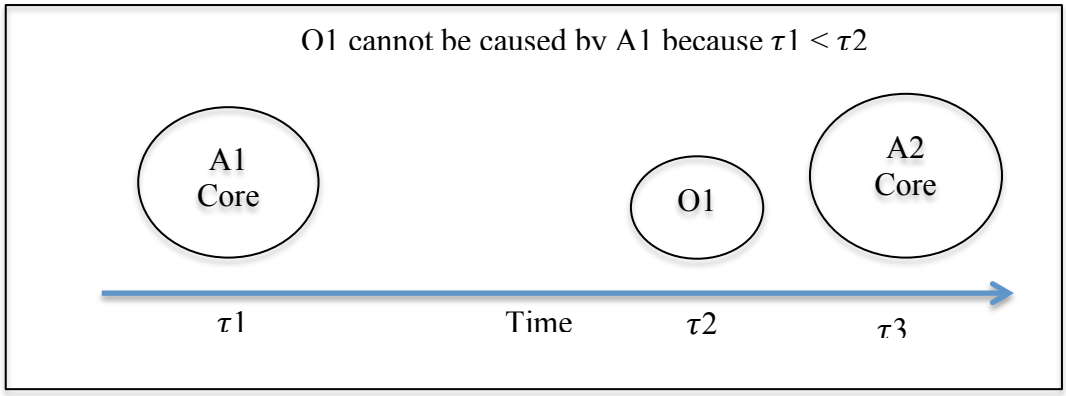


Figure 9.5 Determining action to trace association by using the known behavior of multiple actions where τ1 is the most recent instance of A1, τ2 is when O1 was last updated, and τ3 is when action A2 last occurred, showing that O1 cannot be associated with A1 since A1's Core traces were detected before time τ2

Associating Shared traces to particular actions requires deriving all applicable consistency models, and checking the consistency of traces against already extracted knowledge of related action instances. This requires a multi-phase approach, where first action instances are detected using their own internal consistency functions, then higher-level consistency and grouping is done using the extracted knowledge for more context about the system. Composite action instance reconstruction is discussed further in section 9.5.

9.3.2 Probabilistic Approach to Action-Trace Association

Consistency checking alone is only useful when knowledge of the system and each action is comprehensive, and there is an inconsistency between what is known about the actions and the observed traces. If there is no inconsistency, no new information can be inferred. For example, the object O1 is shared between A1 and A2. If A1's Core (found to be at time $\tau 2$) and A2's core (found to be at time $\tau 3$) both have times that are at or after O1, then O1 could be related to either action, as illustrated in Figure 9.6. Since the O1 is consistent with the update pattern of both A1 and A2, consistency checking cannot determine the action-trace association.

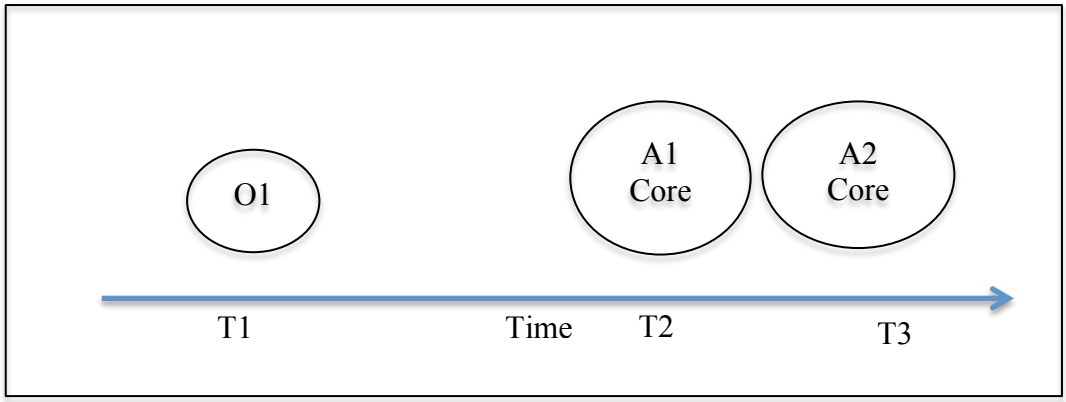


Figure 9.6 A shared trace that is earlier than both associated actions cannot be associated to a specific action using consistency-checking methods since the trace is consistent with both actions

In cases such as this, where it cannot be determined if an object is associated to either action A1 or action A2, methods that attempt to deal with uncertainty must be used. Effectively, our method generates hypotheses automatically, and probabilistic methods work where our hypothesis generation process stops. In this work, probabilistic methods will be introduced, and issues with these methods when making claims about criminal activities that have been derived under uncertain circumstances will be discussed. For example, Shen, Keppens et al. (2006) stated that "legally, assigning probabilistic values to individual pieces of evidence poses the danger of trespassing into the territory of the jury". A number of probabilistic methods have been proposed including Shen, Keppens et al. (2006) and Khan (2008). However, these methods suffer from a number of issues discussed below.

9.3.2.1 Issues with Probabilistic Methods

There are a number of issues that arise when attempting to use probabilistic methods when reasoning about evidence. This work will focus on two of the main issues encountered:

1) Decreasing probability of a hypothesis when an artifact is missing

Overill, Silomon et al. (2010) claimed that a lack of evidence is not grounds to decrease probability of the action happening. Especially since anti-forensics is becoming more common. A human investigator has the luxury of being able to apply prior knowledge; many case examples and also utilize their intuition, or "gut" feeling. Investigators also have the ability to observe the state of the system and then update their knowledge – by doing research or asking others – in relation to their findings.

After, they can continue the investigation with this updated knowledge. A probabilistic system, however, is limited to the scope of objects for which it previously knew the relation. If new knowledge cannot be created – e.g. knowledge of a previously unknown artifact-cleaning program – then reduction of the probability of the hypothesis based on the lack of artifacts is incorrect. Also, the weight of a given artifact is another issue. Assume a Core artifact, as previously defined, had a weight that was greater than a Shared artifact, but all of the artifacts defined as Core and Supporting were maliciously deleted. The lack of these removed artifacts would have a combined weight that would reduce the probability of the malicious hypothesis below other possible hypotheses where all artifacts were found. Likewise, if an alternative hypothesis is assumed where the absence of all, or most, artifacts denotes anti-forensics, this also gets an investigator no closer to the truth since it is just as likely that the event actually did not happen.

2) Prior Probability

When using probabilistic methods, knowledge of the probability of the trace creation happening is necessary when attempting to determine action-trace association. Observation of the creation of a trace given a particular action instance is important for attempting to determine the likelihood of association with an action; however, this information alone assumes that each action has an equal likelihood of happening. This likelihood, however, will depend on the user's personal habits that cannot be determined without monitoring the user. Kwan, Chow et al. (2008) attempted to get around this by determining prior probability of an action from the experience of surveyed investigators. However, the probability should be based on the particular user's action, not the probability that an investigator will encounter evidence of an action. Likewise, accurate probabilities of this type are difficult to acquire from an investigator as it is based on their subjective feeling as well as observation. In other words, determining which action is more likely to occur in a system may only be able to be generalized for a population, but the specific probabilities for the user may not be accurate, and could lead to false conclusions.

There are two main issues with prior probability generation for an action in a system. The first is that each user has a unique probability to execute a particular action. One user, for example, may be more likely to execute Internet Explorer many times in a session, while another is more likely to execute Internet Explorer only once per

session, if at all. The second is that prior probabilities are not only user-specific, but also regionally specific. For example, QQ is a popular Chinese instant messenger program, however, Europeans do not often use it. If European investigators were asked to give a probability for QQ usage in crime, the probability would likely be very low. But in China the probability would likely be much higher. The prior probabilities determined in Kwan's work then are, at best, only relevant to the country, and possibly even the specific region in which the survey was conducted. Even when confined to a specific region, the prior probability is still also generalization when considering each individual user.

Khan (2008) demonstrated an approach to determine not the prior probability of the event occurring, but of the probability of the event modifying a trace. The weakness of this method is that an action's likelihood to modify a trace when it is executed says nothing about the likelihood of the action occurring. The proposed method, then, could return an action as the most likely to have happened, even if the system is incapable of executing the action, or if the user would never execute the action.

9.4 Inferring Actions Instances with Limited Information (Generic Matching)

Thus far, the detection of known actions has been discussed. Signatures are used to describe a causal relation between a known action and the resulting traces. However, some knowledge of the occurrence of actions in a system may possibly be determined if the particular action, and therefore specific traces, are unknown beforehand. This inference of unknown actions still uses the causal relation between an action and its resulting traces, but in a highly generic way. The inference is based on the given model that states for any object to exist in a system, some event must have caused the object to exist. If the object is observed and the object's associated meta-data has been updated beyond the creation time, some action must have updated the object. Based on these properties, two methods are proposed for generically determining action instances.

One general action detection method relies on the fact that some actions create completely randomly named objects, but in specific, known locations. If the object itself cannot be specifically identified beforehand, but if every object in a location – a directory, for example – has the same relation to the action, then each object in the

given location denotes the same type of action occurring. An example of this method is given in section 9.4. Using this method, if each object in a certain directory can be associated with a certain action – "browsing the Internet", for example – then an instance of the action "browsing the Internet" may be inferred if objects exist in the directory. In this case, all objects in the directory are in the set *Ti*. Since the action's trace update timestamp is unknown, the object update threshold would need to account for any, or perhaps an average, timespan. In this case, for example, an object update threshold could be used to attempt to differentiate between "browsing the Internet" sessions. Finally, if a particular consistency between objects was known, then *cm* could be created for the generic action.

The next general action detection method is based on the fact that in a real-world system, actions manifest themselves as a collection of time stamps updated within a certain time-span of each other. For example, Farmer and Venema (2005) used trace clustering over large collections of deleted file meta-data to determine the time and date the source was packaged for distribution (Figure 9.7). Their method illustrates a basic example of using meta-data to detect a particular time a user or system action took place.

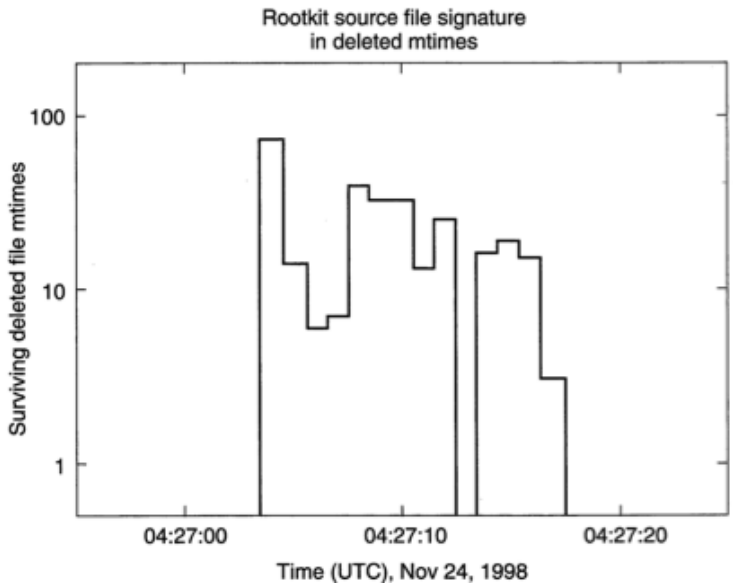


Figure 9.7 Signature analysis of file system activity associated to rootkit installation. Farmer, D. and W. Venema (2005). Forensic Discovery, Addison-Wesley Professional.

For this method, when the state of the system is observed, instead of relying on specific corresponding objects, all objects within a defined generic θ will be grouped

and defined as related to the same generic event. When a large number of objects are updated within a short period of each other, they may be associated with the same action. For example, as shown in Appendix I, a collection of artifacts related to the time "2010 10 18 Mon 17:13" are detected in a system. Of the 88 objects returned, all have either a direct or indirect relation to Internet Explorer, or Internet activity in general. In this case, the specified time could at least be flagged so an investigator may manually determine the nature of the event. This method could be augmented with case-based reasoning, or correlated with known signatures to determine what type of actions the detected event relates to. A basic analysis of the practicality of this method is given in section 9.4.1.

9.4.1 Generic Action Signature Matching

Generic signature matching, as discussed, allows for the detection of action instances when little or no information is known about the action before observing the state of the system. Generic signature matching uses the observation that many actions update traces on a system within a short period of the action instance. Based on this, an action instance may be determined to have happened when groups of like objects are updated together. Consider Figures 9.8 and 9.9. Objects in a system are updated within a short period of time of each other as the action occurs. By examining the common characteristics of objects within a set period, a general statement may be able to be made about the action that occurred.

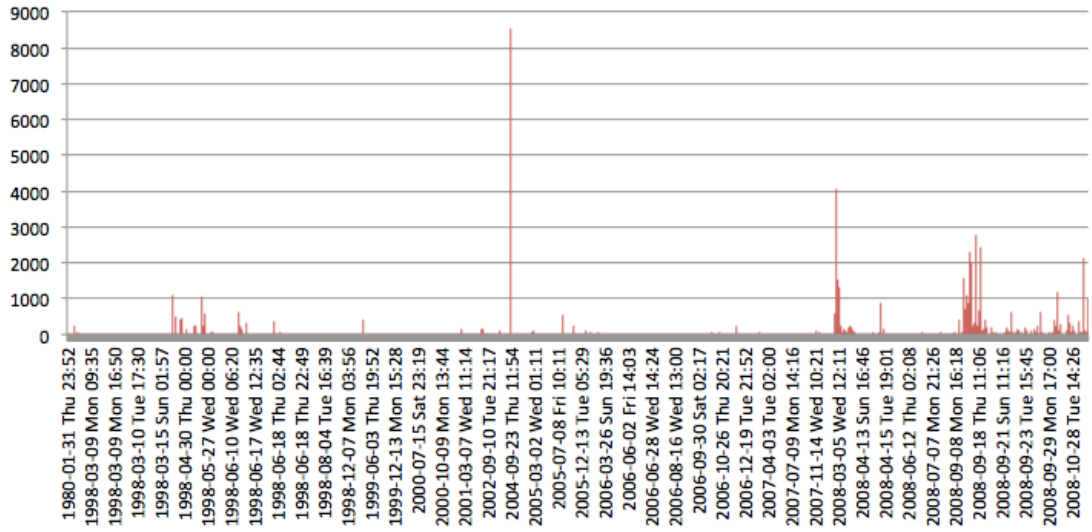


Figure 9.8 Graph of updated objects where Y is the number of objects updated and X is the time of the update

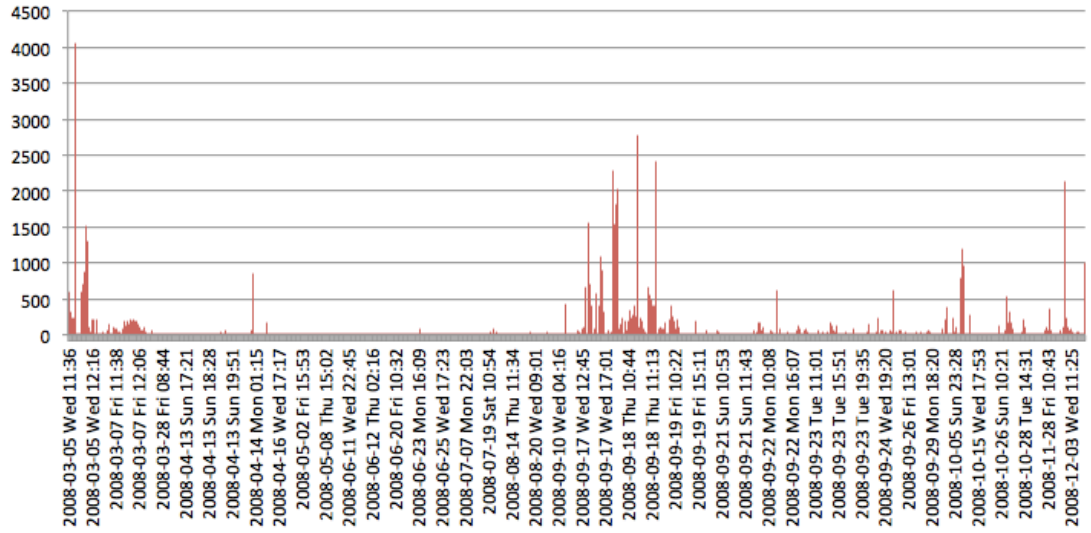


Figure 9.9 Graph of updated objects where Y is the number of objects updated and X is the time of the update starting at Operating System install time

When an object or group of objects is detected, information about what action these objects relate to may be found in at least two ways. The first method uses pre-categorized generic locations or names to check detected objects. Consider the output of a search for all objects updated at "2008 03 07 Fri 11:35" (Appendix J). The majority of these objects' locations are known to correlate to Internet activity. A match of a group of updated objects relating to the location "Temporary Internet Files" could denote the action "Internet-related Activity" at time "2008 03 07 Fri 11:35".

For example, assume all objects in the directory 'Temporary Internet Files' are found to correlate to "Internet-related Activity". In this case, when a search for only objects with that specific location is conducted and grouped by minute, 162 times are returned that correlate to "Internet-related Activity" (Appendix K). Since Internet browsing activity is usually a session over time, the groups of returned objects can be further grouped into sessions by grouping similar detected actions within a certain time-span. To illustrate, an arbitrary grouping of 1 hour between objects has been chosen as the grouping threshold. The result of such grouping is shown in Figure 9.10. In this case, 31 sessions of "Internet-related activity" over time were identified, with some showing more object modification activity than others.

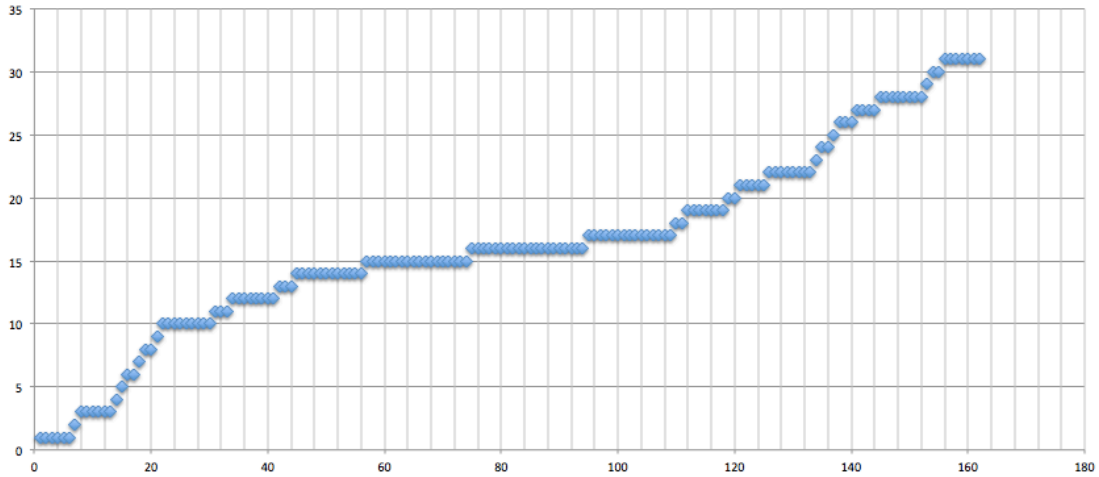


Figure 9.10 Grouping of detected Internet-related Activity objects by 1 hour sessions where the Y axis is the number of sessions and the X axis is the number of returned object times

The second generic action information derivation method attempts to extract knowledge about the group of objects when no information about the action is known beforehand. In this case, an arbitrary 1-minute threshold is used to group objects. Using the same time as the previous example, "2008 03 07 Fri 11:35", for every object that is returned, the text that is common to each object may provide some insight into what the objects relate to. One naive method involves returning the most common words in the group. In this case, the returned objects are known to correlate to Internet Explorer usage. The 10 most commonly occurring words (from file and path names) relating to the returned objects are examined below:

```
cat mactime.out | grep "2008 03 07 Fri 11:35" | awk -F, '{print $NF}'
| tr -cs "[:alnum:]" "\n"| tr "[:lower:]" "[:upper:]" | sort -S16M |
uniq -c | sort -nr | cat -n | head -n 10

     1       168 SETTINGS
     2       102 C
     3        90 YUANDONG
     4        88 DOCUMENTS
     5        88 AND
     6        80 LOCAL
     7        80 CONTENT
     8        77 TEMPORARY
     9        77 INTERNET
    10        77 IE5
```

Here it is shown that for this group of objects the most commonly occurring words reveal the user's name, and a relation to "Internet" and "IE5". This method was tested against known program executions on 'Computer 1' as described in section 10.1, with the results given in Appendix L. It is shown that out of 21 possible executions of

Mozilla Firefox, 4 execution times returned “Firefox” or “Mozilla” as one of the top 10 most frequent words. Likewise, out of 6 possible executions of Internet Explorer, 2 executions times returned “Internet” as one of the top 10 most frequent words. Using the 10 most frequent words was arbitrarily chosen only for testing purposes, and generally appeared to give enough context about actions for that time period without introducing un-related or extremely generic text. Also, it was observed that program or activity-related words were more frequent (using the 10-word threshold) at the most recent execution of the program, and less frequent further into the past.

By presenting an investigator with a summary of related words, the investigator will have a general idea of the activities of the system at that particular time. Keyword filters to reduce common but possibly irrelevant words may help improve relevance of the returned topics.

9.4.2 Weaknesses with Generic Matching

There are two main issues when attempting to generically detect actions instances. First, if pre-defined lists of what objects in a specific location (or directory) “mean” are used, in most cases it is possible that not everything in that location is related to the assumed action. For example, a user could create and store files in the ‘Temporary Internet Files’ directory. Using genetic matching based on object location would not differentiate user-created objects from standard Internet-related activity.
The second issue is with the fact that not enough information is known about what actions could have possibly happened. If any action could have happened, then categorization cannot be definite. In this case it is even impossible to use probabilistic methods to determine the action since no information about actions or objects is known beforehand.

For these reasons, generic matching methods are at best an indicator of when *some* action took place. By looking at the similarities between grouped objects the nature of the action could be guessed, but not concretely determined automatically. Because of this, generic matching should be considered as a guide for an investigator, where an action occurring is a fact, but the true nature of that action is unknown and deeper meaning of the group of objects is to be determined by the expert.

9.5 Composite Action Instance Reconstruction

The proposed action instance detection method can be used in a recursive fashion to determine more information about happened actions from the collection of detected actions. Actions as described thus far have been focused a single instance in time. However, computer usage normally consists of chains of actions that are temporally related to achieve a single purpose. For the reminder of this work, a single instance of an action will still be defined as an *action*, while a chain of actions temporally related to achieve a single purpose will be defined as a *composite action*.

The collection of detected action instances can be thought of as low-level traces for which signatures of patterns of composite action instances can be created. Detected actions have either an associated single time approximation or approximated period of time. For example, Figure 9.11 shows detected action instances and their associated times in a timeline with $\tau 1$ being the oldest and $\tau 4$ being the most recent time value.

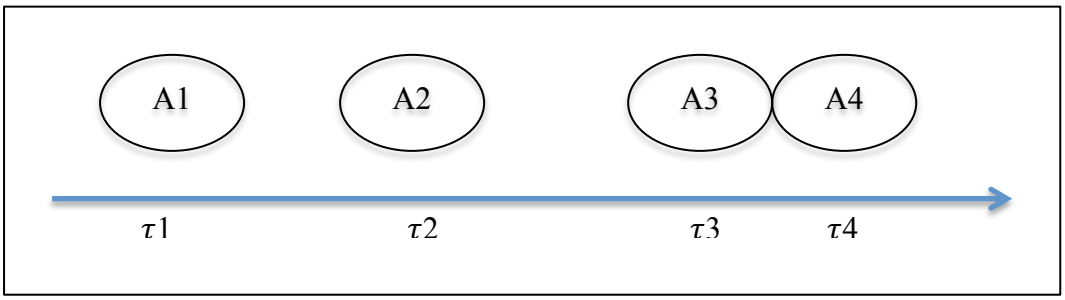


Figure 9.11 Detected Actions (A1-4) ordered in time where $\tau 1$ is the oldest detected time and $\tau 4$ is the most recently detected time

For example, a signature of a composite action such as "*Browsing the Internet with Internet Explorer*" can be created based on the detection of associated actions such as "*Opening Internet Explorer*", "*Navigating to website X*", "*Closing Internet Explorer*". Depending on the composite action, the times must allow variability. For example, an Internet browsing session could last from minutes to hours. Composite actions may also have a precise order of actions in time. For example, a program must be opened before it can be closed.

In terms of the proposed model, a composite action (*ca*) is defined as an ordered list of actions that are executed to achieve a specific objective.

$$ca = (a_1, a_2, a_3, \ldots a_n)$$

A composite action instance (*cai*) is defined as the composite action executing within a time-span.

$$cai = (ca, \boldsymbol{\lambda}\tau)$$

where

- *ca* is the composite action
- $\boldsymbol{\lambda}\tau$ is the time interval in which the composite action has taken place in the form of a double [t_{start}, t_{end}], where
    - t_{start} is the composite action's start time
    - t_{end} is the composite action's end time

Composite action instances can be approximated based on the observation of related action instance approximations, where the composite action instance approximation (*caia*) is approximated to have executed in the interval between the oldest observed action instance approximation and the most recent action instance approximation in the set of action instance approximations relating to the composite action.

Detected action instance approximations are treated as an object that can be observed in the system. Based on this, signatures of composite actions may be defined as:

$$S_{ca} = \{Ti, cm\}$$

where

- *Ti* is the set of all instance approximations associated with the composite action instance
- *cm* is the update consistency checking function specific to the category of action instance execution patterns that tests some property of $ia \in IA$ where $ia \in Ti$

Signatures of composite actions may be created where action instance approximations are in the set *Ti*, and the consistency function (*cm*) is defined by modeling the pattern of action instance approximation times in terms of the execution of the composite action instance.

Two pieces of information can be derived when using a signature of a composite action over a collection of approximated actions. First, detected action instances can be grouped to determine which composite action instance an action belongs to. For

example, in Figure 9.11 assume action A1 was "*Open Internet Explorer*", A2 was action "*Navigating to website X*", A3 was action "*Open Internet Explorer*", and A4 was action "*Navigating to website Y*". Action A2 logically cannot belong to action A3, and therefore must be related to action A1. Much like the consistency checking method discussed in section 9.3.1, action A4 could be related to either A1 or A3.

Next, additional action instances may be inferred or confirmed by using composite action signatures. Again, consider Figure 9.11. Assume A1 was action "*Windows Startup*", A2 was action "*Open Instant Messenger*", A3 was action "*Close Internet Explorer*", and A4 was action "*Windows Shutdown*". Two composite action signatures exist, "*Use Instant Messenger*" and "*Use Internet Explorer*". The composite action "*Use Instant Messenger*" consists of two actions, "*Open Instant Messenger*" (Open IM) and "*Close Instant Messenger*" (Close IM). The composite action "*Use Internet Explorer*" also consists of two actions, "*Open Internet Explorer*" (Open IE) and "*Close Internet Explorer*" (Close IE). The application of these composite action signatures is shown in Figure 9.12.

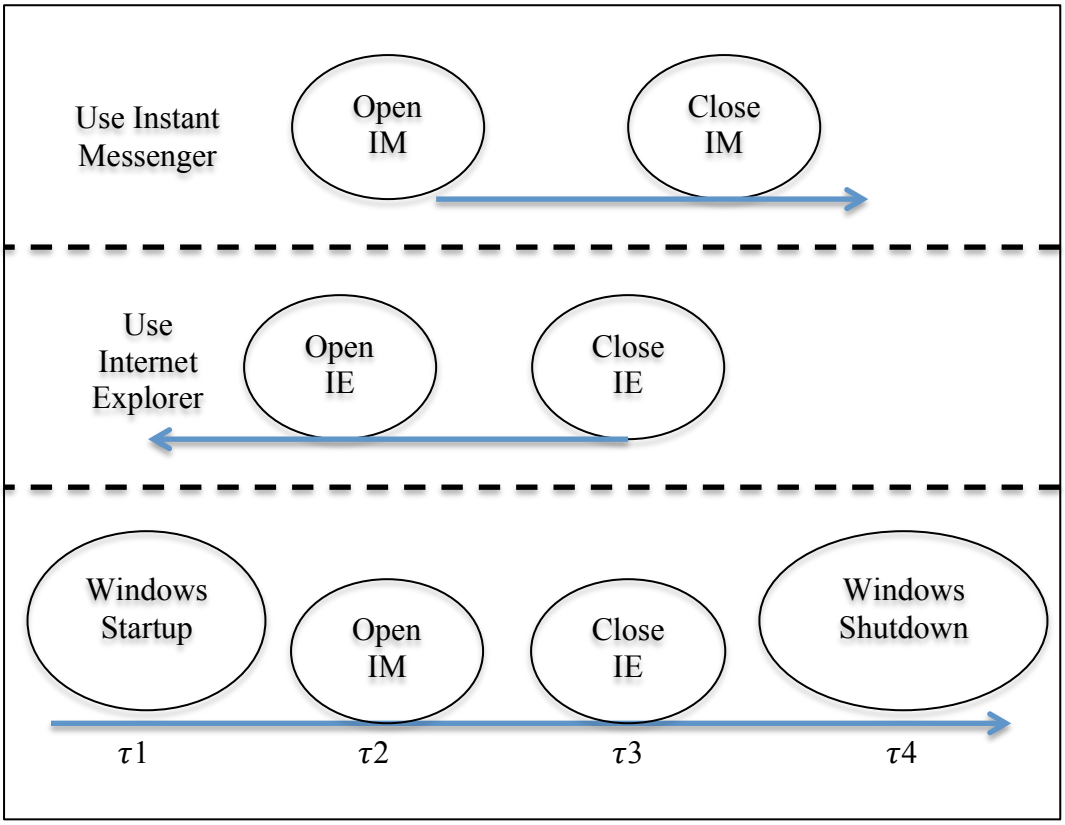


Figure 9.12 Action instances detected using composite action signatures and time bounding between already detected action instance approximations

Given the ensemble of detected action instances, composite action signatures can allow the detection of action instances that must have happened for the already detected actions to exist. However, the exact time of the action may not be able to be determined. For example, in Figure 9.12 the action *Close IE* was detected, but no corresponding open action was detected. The composite action signature *Use Internet Explorer* requires that the action *Open IE* must have happened before the action *Close IE*. Because *Close IE* was detected at time $\tau 3$, *Open IE* must have happened sometime before time $\tau 3$, but the exact time cannot be determined.

While just knowing that a particular action must have happened may be useful in some cases, usually the time in which the action has occurred is also important. Using event time bounding techniques as described by Gladyshev and Patel (2005) and Zhu (2011), a time range for newly inferred actions may be determined. In Figure 9.12 it is known that the event *Close IE* must have happened at time $\tau 3$, and therefore the action *Open IE* must have happened any time before $\tau 3$. However, utilizing other known information, the time in which Open IE must have occurred can be restricted. In this case the action *Windows Startup* is known to have occurred at time $\tau 1$. Since an open action cannot have occurred before an OS startup action, the time-span in which *Open IE* must have occurred can be reduced from infinity before $\tau 3$ to a time-span between $\tau 1$ and $\tau 3$. Like previously discussed, when attempting to determine the association of a trace to a particular action, more context can be found that allows for the inference of information that would otherwise be unavailable. In this case, the detection of the time of the OS startup action gives context to the state of the system. Some actions such as system startup or shutdown are system-wide limiters, and such actions can be used to limit the time-span in which an action could have occurred.

Using signatures of composite actions with time-bounding methods allows previously undetected action instances to be approximated between time-span limiters.

9.6 Verification of Human Inference

The proposed hypothesis reduction method could be used to verify the human inference process in digital forensic examinations. Surveyed investigators admitted that verification of the investigation process was not normally taking place, and if verification was taking place it was normally a peer review of the created report (James and Gladyshev 2010). By utilizing the hypothesis reduction method, possible

actions could be detected before a manual examination by a human was conducted. Further, signature-based hypothesis reduction methods may also help in real-time and post analysis inference verification phases.

One application could be implemented during a real-time analysis. An investigator may observe a set of objects, and infer what the state of the collection of objects mean in relation to the suspect's actions. The investigator may then submit his or her hypothesis. Signature-based methods could be used to determine the feasibility of the hypothesis, and present alternatives that the investigator may then want to investigate further.

Such methods may also be applied after an analysis has taken place. Oftentimes a report is normally created concerning the thought processes of the investigator. This report is normally checked by a supervisor, but cannot be thoroughly scrutinized without another thorough, time-consuming investigation of the same suspect media. Instead, each inference the investigator made that is encoded in the report could be tested using the proposed signature-based hypothesis reduction method to quickly determine whether hypothesis are supported by the given evidence, or whether such hypotheses don't hold, and should be investigated further.

One limitation is that the proposed hypothesis reduction method, much like a human investigator, cannot always be perfect in systems with so much uncertainty. Also, the method is attempting to automate higher-level thought processes, where the interpretation of what objects mean is being examined. Because of this, error rates normally used for traditional digital forensic tools are not adequate. The investigation accuracy measurement method proposed in Chapter 5 is offered to provide a way to objectively compare the classification of the meaning of objects between tools and investigators.

9.7 Summary

This chapter began by discussing the inferences of single and multiple occurrences of a given action instance based on the previously discussed signature matching methods. Consistency between object updates as well as higher-level consistency between actions was discussed. A probabilistic method for associating traces to actions when uncertainty exists was given, and the issues with such a method were discussed. Next, generic matching of action instances based on no prior information

was then briefly described, with case examples explored and weaknesses given. After, high-level event reconstruction using the proposed signature-matching model was discussed that enables more actions to be inferred based on the presence of previously inferred action instances. Finally, the use of the proposed signature-based hypothesis reduction methods for human inference verification in digital forensic investigations was discussed.

Chapter 10

Evaluation

This chapter gives an evaluation of the proposed signature-based action instance detection method. First, the case scenario is given that forms the base case data for the remainder of this chapter. Next, an analysis of the extraction and categorization of traces and update thresholds needed to create signatures of action instances is given. The results of the signature-based action instance detection method are compared to similar work involving action "footprint" generation and matching via machine learning techniques conducted by Khan (2008). Weaknesses of the proposed signature-based action instance detection method is then examined and discussed.

10.1 Case Study

The following case study has been designed to evaluate 1) whether generalized signatures of action instances can be derived by monitoring a similar system and 2) whether derived signatures, using the proposed action instance detection methods, are able to detect and differentiate past action instances on random real systems given only the most recent state of the suspect system. The goal is to determine whether signature-based methods are effective at automatically reconstructing past action instances during a post-mortem digital forensic analysis. Effectiveness in this case is defined as detecting past action instances with no false positives. In other words, only action instances that have occurred will be detected. As discussed, meta-data is continually overwritten. As such, observing only the final state of the system may give no indication of action instances before the most recently occurring. Because of this, false negatives – not detecting action instances that have occurred – are expected.

In this case study, Core and Supporting signatures will be derived for specific actions. Objects in these signatures will be generalized using Regular expressions as discussed. Action update thresholds will be determined through experimentation, and the consistency checking functions as previously defined will be used for Core and Supporting traces. Test 'suspect' systems will be monitored using the built-in Windows event logging, with process executions and terminations logged over time. While monitoring is taking place, the owners of the test systems will use the systems as they normally would during their daily computer usage. After the monitoring period, meta-data will be collected from each system using The SleuthKit's "fls"

program for Windows, and the generated process monitoring logs will also be collected. A collection of bash scripts named 'Forensic Investigation of Timelines using Signatures' (FITS) (Appendix M) has been created that accepts as inputs the dumped meta-data of a suspect system and signatures of actions. FITS outputs detected action instances and their associated time information. The action instance and time output will be compared to the collected process logs. Effectiveness will be evaluated based on the ability to detect action instances on differently configured systems, as well as the algorithm's ability to differentiate between action instances.

10.1.1 Testing Scenario

The test case consists of two computers running Windows XP. One is a standard install of the English version of Windows XP, and the other was a standard install of the Chinese version of Windows XP. Internet Explorer 8 (IE8) and Firefox 3 (FF3) were installed on both systems. Each computer was used daily for entertainment, work and study tasks. Both users identified that they used Firefox as their primary browser. To accurately determine when IE8 and FF3 have been opened and closed, a Windows security auditing policy was implemented on both computers to monitor process creation and executable access. By enabling Windows security auditing, detailed information about processes, such as the process name and time of starting and stopping the process, could be collected. Each computer was monitored for a number of days, after which the Windows security event log was exported and the computer's file system meta-data was collected using tools from The Sleuth Kit (Carrier 2005) version 3.2.2.

From the collected Windows event logs it was observed that on 'Computer 1' 12 instances of opening Firefox 3 from the 19th to the 24th were identified (Table 10.1), and 6 instances of opening IE8 were identified (Table 10.2).

Table 10.1 Computer 1 Windows event log of "Firefox 3 Open and Close" actions where each session is grouped by process ID

Time	User Action	Process ID	Process
07/19/2011 22:04:43	Close	5096	firefox.exe
07/19/2011 23:22:51	Open	4192	firefox.exe
07/19/2011 23:41:58	Open	472	firefox.exe
07/19/2011 23:41:59	Close	472	firefox.exe
07/20/2011 01:21:42	Close	4192	firefox.exe
07/20/2011 11:55:09	Open	4368	firefox.exe
07/20/2011 13:49:57	Close	4368	firefox.exe
07/20/2011 15:10:33	Open	4928	firefox.exe
07/20/2011 16:19:40	Open	6048	firefox.exe
07/20/2011 16:19:41	Close	6048	firefox.exe
07/20/2011 20:23:52	Close	4928	firefox.exe
07/20/2011 22:07:06	Open	6096	firefox.exe
07/21/2011 01:24:09	Close	6096	firefox.exe
07/21/2011 11:29:01	Open	4112	firefox.exe
07/21/2011 11:31:39	Close	4112	firefox.exe
07/21/2011 19:29:30	Open	4076	firefox.exe
07/22/2011 02:57:05	Close	4076	firefox.exe
07/22/2011 11:35:29	Open	1632	firefox.exe
07/23/2011 03:07:32	Close	1632	firefox.exe
07/23/2011 11:36:13	Open	5972	firefox.exe
07/24/2011 02:26:40	Close	5972	firefox.exe
07/24/2011 13:23:58	Open	4284	firefox.exe
07/24/2011 15:02:30	Open	5408	firefox.exe
07/24/2011 15:02:31	Close	5408	firefox.exe

Table 10.2 Computer 1 Windows event log of "Internet Explorer 8 Open and Close" actions where each session is grouped by process ID

Time	User Action	Process ID	Process
07/20/2011 19:15:48	Open	5396	iexplore.exe
07/20/2011 19:15:50	Open	4220	iexplore.exe
07/20/2011 19:24:03	Close	4220	iexplore.exe
07/20/2011 19:24:03	Close	5396	iexplore.exe
07/21/2011 16:08:09	Open	5524	iexplore.exe
07/21/2011 16:08:10	Open	5308	iexplore.exe
07/21/2011 18:34:38	Close	5308	iexplore.exe
07/21/2011 18:34:38	Close	5524	iexplore.exe
07/23/2011 14:56:45	Open	184	iexplore.exe
07/23/2011 14:56:46	Open	452	iexplore.exe
07/23/2011 19:20:09	Close	184	iexplore.exe
07/23/2011 19:20:09	Close	452	iexplore.exe

From the collected Windows event logs it was observed that on 'Computer 2' 14 instances of opening FF3 from the 13th to the 17th were identified (Table 10.3), and two instances of opening IE8 were identified (Table 10.4).

Table 10.3 Computer 2 Windows event log of "Firefox 3 Open and Close" actions where each session is grouped by process ID

Time	User Action	Process ID	Process
07/13/2011 11:51:45	Open	1380	firefox.exe
07/13/2011 12:14:23	Close	1380	firefox.exe
07/13/2011 12:14:28	Open	5900	firefox.exe
07/13/2011 21:48:27	Close	5900	firefox.exe
07/13/2011 21:51:45	Open	2196	firefox.exe
07/13/2011 21:54:12	Close	2196	firefox.exe
07/13/2011 23:33:29	Open	1276	firefox.exe

07/13/2011 23:33:47	Close	1276	firefox.exe
07/13/2011 23:33:52	Open	4032	firefox.exe
07/13/2011 23:35:09	Close	4032	firefox.exe
07/13/2011 23:35:16	Open	1404	firefox.exe
07/14/2011 00:35:54	Close	1404	firefox.exe
07/14/2011 01:38:42	Open	3820	firefox.exe
07/14/2011 02:13:46	Close	3820	firefox.exe
07/14/2011 23:50:48	Open	4028	firefox.exe
07/15/2011 01:33:19	Close	4028	firefox.exe
07/15/2011 22:36:31	Open	4920	firefox.exe
07/15/2011 23:15:23	Close	4920	firefox.exe
07/16/2011 18:15:34	Open	1976	firefox.exe
07/16/2011 23:34:18	Close	1976	firefox.exe
07/17/2011 00:46:14	Open	3548	firefox.exe
07/17/2011 03:16:36	Close	3548	firefox.exe
07/17/2011 15:23:01	Open	4460	firefox.exe
07/17/2011 16:04:10	Close	4460	firefox.exe
07/17/2011 16:16:52	Open	4616	firefox.exe
07/17/2011 18:20:06	Close	4616	firefox.exe
07/17/2011 20:24:14	Open	2004	firefox.exe
07/17/2011 20:26:38	Close	2004	firefox.exe

Table 10.4 Computer 2 Windows event log of "Internet Explorer 8 Open and Close" actions where each session is grouped by process ID

Time	**User Action**	**Process ID**	**Process**
07/17/2011 15:15:05	Open	4832	iexplore.exe
07/17/2011 15:15:06	Open	5472	iexplore.exe
07/17/2011 15:22:37	Close	5472	iexplore.exe
07/17/2011 15:22:43	Close	4832	iexplore.exe

The logged activity from both systems will be used as the baseline to compare the effectiveness of the proposed signature-matching method.

10.2 Action Signature Derivation and Categorization

An action signature has been defined as the unique ensemble of traces that are ultimately caused by a specific action, which are related by a specific time-span and consistency function. The unique ensemble of traces in the context of this section will be the collection of time stamps modified by the occurrence of an action, and specifically time stamps related to files and Windows Registry keys. The signature derivation method described in Chapter 8 will be used to determine specific traces and their update categories relevant to a particular action. Using this method, objects related to an action instance can be determined, which can then be monitored for classification and update threshold determination purposes.

10.2.1 Action Signature Derivation

Microsoft Process Monitor was used to record all system calls executed during instances of the actions "*Open IE8*" and "*Open FF3*" on Windows XP service pack 3. Windows XP was chosen since, as of this writing, it is still the most frequently encountered operating system of surveyed law enforcement (James 2010; James 2011). The initial tests recorded only activities associated with the respective executed processes. Each action was executed 10 times via a macro utility that can repeatedly simulate mouse and keyboard interaction. Considering that not all traces will be updated on every instance of the action, the results of the 10 executions were combined, and duplicate entries were removed. This has potential to increase noise, or objects associated with other background activities, but all noise will be filtered in the trace categorization process. The results are lists of objects that are altered when the respective actions occur. This process produced 51537 potential traces for Internet Explorer 8, and 6416 potential traces for Firefox 3.

10.2.1.1 Trace Categorization and Object Update Threshold Approximation

Next, to determine trace update behavior for classification purposes, time stamps for each identified object were collected. Each action was again executed 10 times. Other non-related actions were executed at least 2 minutes after the action of interest, including system shutdown and startup actions to introduce noise. After each execution of the action and noise-producing session, time stamps of all previously

identified action-associated objects were collected with the previously mentioned 'sigtest.pl', given in Appendix F.

For each tested action, the results of each instance of the action were analyzed to determine trace category association. This process consisted of comparing the trace time with the known execution time. For testing, an initial object update threshold of 120 seconds was assigned based on the observation that past action instance update thresholds were normally within 60 seconds. A threshold of 120 seconds allows for initial exploratory analysis that will be made more specific in later examination.

If the trace time was within the object update range, then it was marked as Core. If the trace was before the action execution time – outside of the object update range – then the trace was marked as Supporting. If the trace was after the action execution time – outside of the object update range – then it was marked as Shared based on the fact that, to be consistent with Core or Supporting categories, the trace cannot be before the detected Core trace times, and so must be updated by some other process. Categorization considered the known action instance time for each run, and analyzed the update behavior of traces across each run. In this case, traces were allowed to be downgraded from Core to Supporting to Shared, and could not be upgraded once flagged at a lower level.

A second level of refinement was required to calculate a more accurate object update threshold, and verify traces were correctly categorized. The process was ran another 10 times, each time examining the update times compared with the known execution time. For both actions, the object update threshold was lower than the initial 120 seconds threshold. However, traces were not usually re-categorized because of the lower threshold – the time between action instances was sufficient to differentiate between executions, even with a longer threshold. Using the derived associated trace list, the object update threshold derivation process described in section 8.3 must be sampled on many different machines to attempt to get a representative update threshold. The update threshold for IE8 was found to be 61 seconds, and the update threshold for FF3 was found to be 50 seconds.

The resulting Core and Supporting objects and time stamps of interest are shown in Table 10.5 and Table 10.6 for Firefox and Internet explorer, respectively. Objects are

defined as previously described Regular Expressions to support portability to other systems.

Table 10.5 Firefox 3 objects, update categories and corresponding time stamp of interest represented as Regular Expressions

Type	Timestamp	Name	Objects associated with "FF3 Open and Close"
Core	**Modified**	T_1	.*/Firefox/Profiles/.*default/urlclassifierkey.\.txt
Core	**Modified**	T_2	.*/Prefetch/Firefox\.EXE-.*\.pf
Support	**Created**	T_3	.*/Prefetch/Firefox\.EXE-.*\.pf
Support	**Created**	T_4	.*/Firefox/Profiles/.*default/cookies.sqlite-journal
Support	**Created**	T_5	.*/Firefox/Profiles/.*\default/urlclassifierkey.\.txt
Support	**Created**	T_6	.*/Firefox/Profiles/.*default/startupCache$
Support	**Created**	T_7	.*/Firefox/Profiles/.*default/pluginreg.dat

Table 10.6 Internet Explorer 8 objects, update categories and corresponding time stamp of interest represented as Regular Expressions

Type	Timestamp	Name	Objects associated with "IE8 Open and Close"
Core	**Modified**	T_8	.*/Prefetch/IEXPLORE\.EXE-.*\.pf
Support	**Created**	T_9	.*/Prefetch/IEXPLORE\.EXE-.*\.pf
Support	**Created**	T_{10}	.*/Cookies/.*@ATDMT\[[0-9]\]\.TXT
Support	**Created**	T_{11}	.*/Cookies/.*@BING\[[0-9]\]\.TXT
Support	**Created**	T_{12}	.*/Cookies/.*@live\[[0-9]\]\.TXT

With knowledge of the object, trace of interest, object update threshold and previously given category consistency checking functions, signatures for the actions "Open IE8" and "Open FF3" may be created.

10.3 Core and Supporting Action Instance Signature Matching

This section refers to the testing scenario given in section 10.1. The meta-data from Computer 1 was scanned using the previously defined signature for "Open FF3". The identified objects and associated time stamps are shown in Table 10.7.

Table 10.7 Firefox 3 objects and associated time stamps identified using signature detection on Computer 1

Time	Name	Returned Object
07/24/2011 13:24:14	T_1	C:/Documents and Settings/User1/Application Data/Mozilla/Firefox/Profiles/94370b5u.default/urlclassifierkey3.txt
07/24/2011 15:02:31	T_2	C:/WINDOWS/Prefetch/FIREFOX.EXE-28641590.pf
12/26/2010 04:26:24	T_3	C:/WINDOWS/Prefetch/FIREFOX.EXE-28641590.pf
07/24/2011 13:24:10	T_4	C:/Documents and Settings/User1/Application Data/Mozilla/Firefox/Profiles/94370b5u.default/cookies.sqlite-journal
01/05/2011 23:15:34	T_5	C:/Documents and Settings/User1/Application Data/Mozilla/Firefox/Profiles/94370b5u.default/urlclassifierkey3.txt
N/A	T_6	.*/Firefox/Profiles/.*default/startupCache$
12/26/2010 03:04:55	T_7	C:/Documents and Settings/User1/Application Data/Mozilla/Firefox/Profiles/94370b5u.default/pluginreg.dat

All Core artifacts were discovered, however, T_1 had a time stamp that was different than T_2. This unexpected behavior can be explained by looking at the Open and Close times from the Windows Event log for Computer 1 (Table 10.1). The Firefox open event with the process ID 4284 occurred at 13:23, and was never followed by a process close event. While process 4284 was still open, another instance of Firefox was started, process 5480, at 15:02. If process 4284 had locked the object in question, then the time stamp may not be updated upon another instance of the action. However, T_2 was not locked by the first action instance, and was updated. Since both objects must be updated when the action instance happens, this must mean that two instances of the same action must be running in parallel, otherwise both objects would be updated. Consistency checking, however, would only detect an inconsistency in timestamps, and flag such an inconsistency for further manual investigation unless the particular case was built into the consistency checking function.

Figure 10.1 shows the signature-detected times of the action instance “Open FF3”. In this figure, action instances up to a year prior to the test were detected using signature-based methods, but not all of the most recent executions were detected.

Figure 10.2 shows the most recent detected executions compared to the Windows Event Log entries.

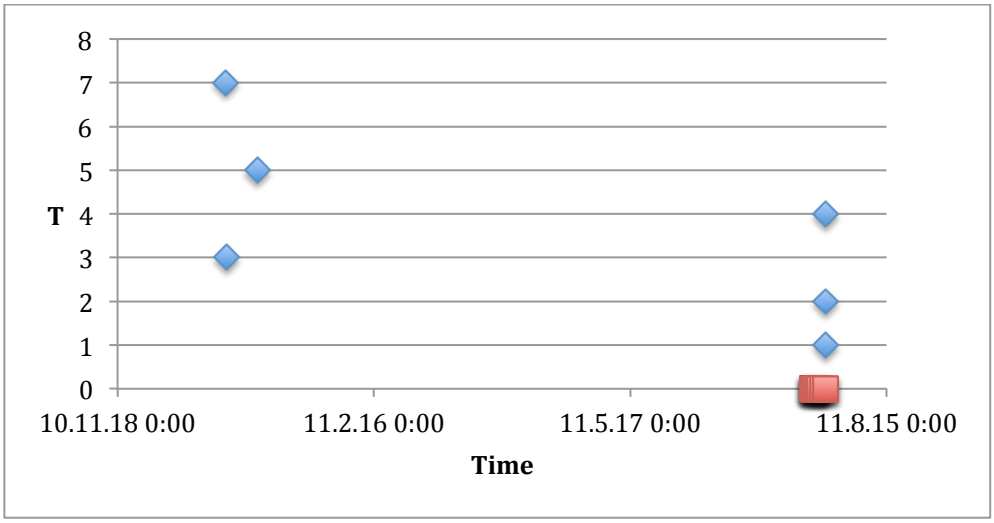


Figure 10.1 Full times of the "Open FF3" action on Computer 1, where signature-detected times are shown in blue diamonds compared to the Windows Event Log times shown in red squares

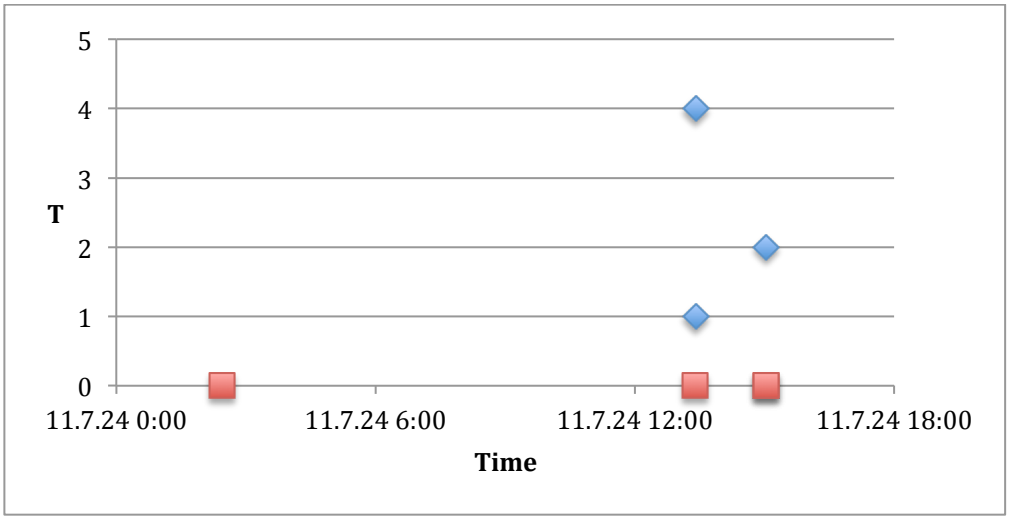


Figure 10.2 Most recent times of the "Open FF3" action on Computer 1, where signature-detected times are shown in blue diamonds compared to the Windows Event Log times shown in red squares

Next, the meta-data from Computer 1 was scanned using the previously defined signature for "Open IE8". The identified artifacts and associated time stamps are shown in Table 10.8.

Table 10.8 Internet Explorer 8 objects and associated time stamps identified using signature detection on Computer 1

Time	Name	Found Artifact
07/23/2011 14:56:53	T_8	C:/WINDOWS/Prefetch/IEXPLORE.EXE-27122324.pf
07/19/2011 00:57:22	T_9	C:/WINDOWS/Prefetch/IEXPLORE.EXE-27122324.pf
06/14/2011 10:47:26	T_{10}	C:/Documents and Settings/User1/Cookies/user1@atdmt[2].txt
01/11/2011 19:40:26	T_{11}	C:/Documents and Settings/User1/Cookies/user1@bing[2].txt
06/14/2011 10:47:26	T_{12}	C:/Documents and Settings/User1/Cookies/user1@live[1].txt

All Core artifacts were detected, identifying the most recent execution of IE8 as happening at approximately 14:56 on 07/23/2011. All other associated artifacts had time stamps before this time. Figure 10.3 shows a graph of all the signature-detected times of the event "Open IE8". Again, in this figure action instances up to 6 months prior to the test were detected using signature-based methods, but not all of the most recent executions were detected. Figure 10.4 shows the most recent detected executions of IE8 compared to the Windows Event Log entries.

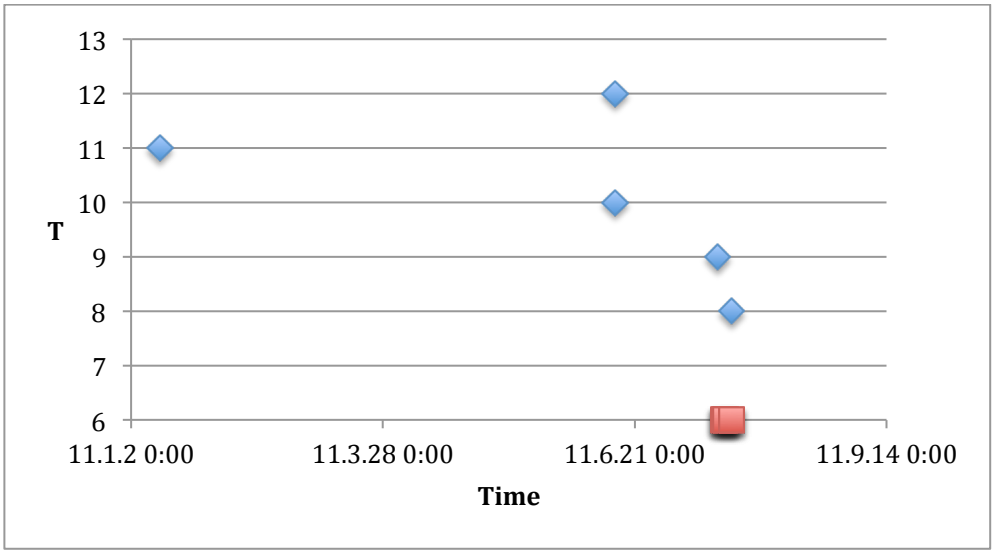


Figure 10.3 Full times of the "Open IE8" action on Computer 1, where signature-detected times are shown in blue diamonds compared to the Windows Event Log times shown in red squares

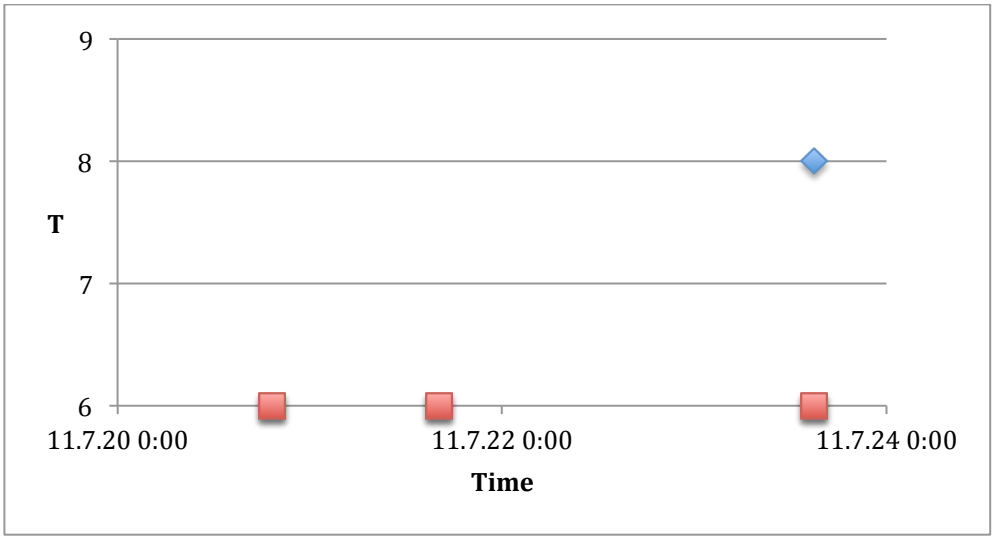


Figure 10.4 Most recent times of the "Open IE8" action on Computer 1, where signature-detected times are shown in blue diamonds compared to the Windows Event Log times shown in red squares

Next, the meta-data from Computer 2 was scanned using the previously defined signature for opening FF3. The identified objects and associated time stamps are shown in Table 10.9.

Table 10.9 Firefox 3 objects and associated time stamps identified using signature detection on Computer 2

Time	T	Found Artifact
07/17/2011 20:24:26	T_1	C:/Documents and Settings/user/Application Data/Mozilla/Firefox/Profiles/c2yzki95.default/urlclassifierkey3.txt
07/17/2011 20:24:18	T_2	C:/WINDOWS/Prefetch/FIREFOX.EXE-28641590.pf
05/21/2010 16:15:23	T_3	C:/WINDOWS/Prefetch/FIREFOX.EXE-28641590.pf
05/14/2010 16:44:22	T_4	C:/Documents and Settings/user/Application Data/Mozilla/Firefox/Profiles/c2yzki95.default/cookies.sqlite-journal (deleted)
10/23/2010 11:26:02	T_4	C:/Documents and Settings/user/Application Data/Mozilla/Firefox/Profiles/c2yzki95.default/cookies.sqlite-journal (deleted)
04/13/2011 00:33:49	T_5	C:/Documents and Settings/user/Application Data/Mozilla/Firefox/Profiles/c2yzki95.default/urlclassifierkey3.txt
07/17/2011 15:23:05	T_6	C:/Documents and Settings/user/Local Settings/Application Data/Mozilla/Firefox/Profiles/c2yzki95.default/startupCache
07/17/2011 00:46:36	T_7	C:/Documents and Settings/user/Application Data/Mozilla/Firefox/Profiles/c2yzki95.default/pluginreg.dat

All Core artifacts were detected, identifying the most recent execution of FF3 as approximately 20:24 on 07/17/2011. All other associated objects had time stamps before this time. One action of note relates to object T_4. In this case multiple instances of the object were created and deleted, and the meta-data information could be recovered for two of those instances. The update rules for the artifact in question still hold, even if the object is deleted. For this reason, multiple detected instances of T_4, or any other artifact, may contribute time stamp information per usual. This allows for the possibility of much more information to be derived about an action by utilizing all available meta-data, similar to methods used by Farmer and Venema (2005) who used deleted file information for malware analysis.

Figure 10.5 shows a graph of the signature-detected times of the event "Open FF3". In this figure, action instances up to one year prior to the test were detected using signature-based methods, but not all of the most recent executions were detected. Figure 10.6 shows the most recent detected executions compared to the Windows Event Log entries.

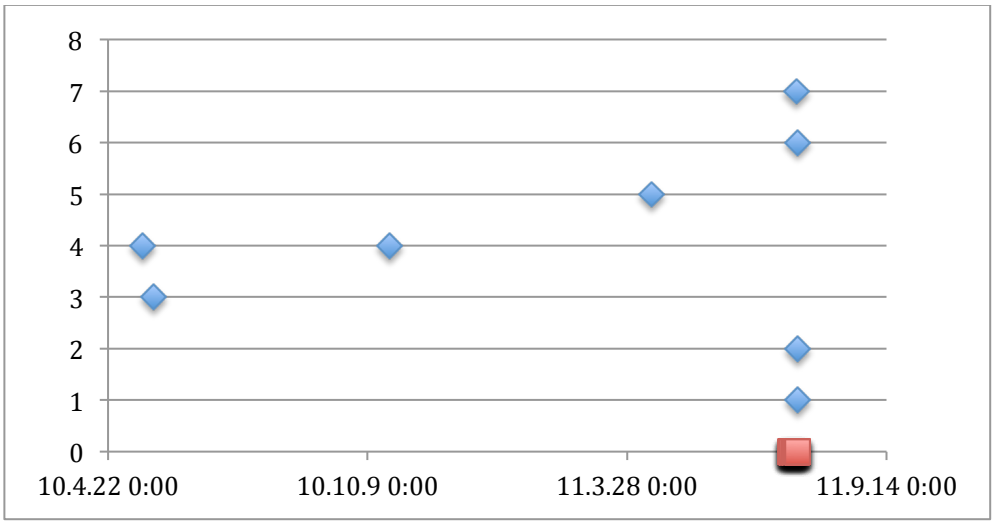


Figure 10.5 Full times of the "Open FF" action on Computer 2, where signature-detected times are shown in diamonds compared to the Windows Event Log times shown in squares

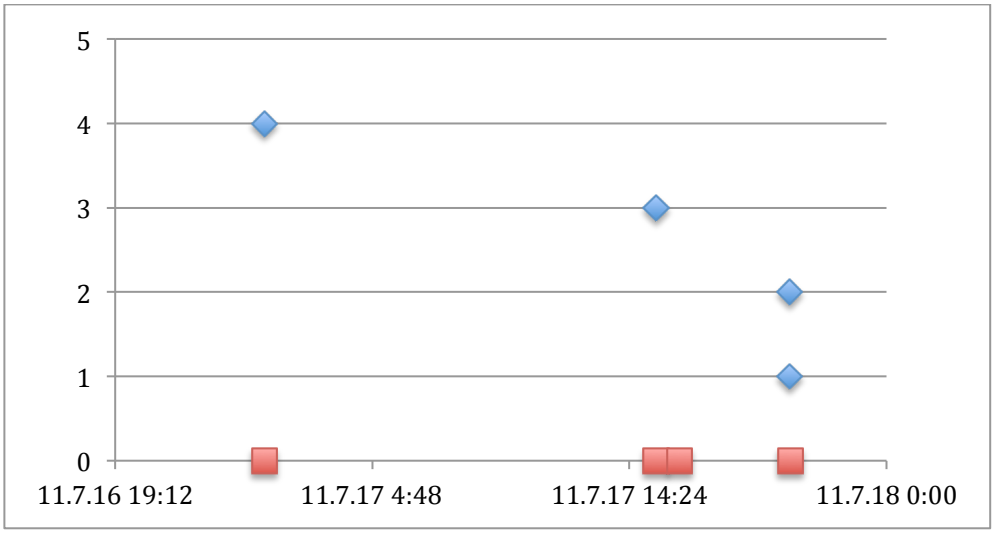


Figure 10.6 Most recent times of the "Open FF3" action on Computer 2, where signature-detected times are shown in diamonds compared to the Windows Event Log times shown in squares

Next, the meta-data from Computer 2 was scanned using the previously defined signature for opening IE8. The identified objects and associated time stamps are shown in Table 10.10.

Table 10.10 Internet Explorer 8 objects and associated time stamps identified using signature detection on Computer 2

Time	T	Found Artifact
07/17/2011 15:15:13	T_9	C:/WINDOWS/Prefetch/IEXPLORE.EXE-27122324.pf
07/17/2011 15:15:09	T_{10}	C:/WINDOWS/Prefetch/IEXPLORE.EXE-27122324.pf
03/10/2011 15:01:01	T_{11}	C:/Documents and Settings/User2/Cookies/user2@atdmt[1].txt
03/10/2011 15:38:37	T_{12}	C:/Documents and Settings/User2/Cookies/user2@bing[2].txt
03/10/2011 15:38:37	T_{13}	C:/Documents and Settings/User2/Cookies/user2@live[2].txt

All Core objects were detected, setting the most recent execution of IE8 at approximately 15:15 on 07/17/2011. All other associated objects had time stamps before this time. Figure 10.7 shows a graph of the signature-detected times of the event "Open IE8". In this figure, action instances up to four months prior to the test

were detected using signature-based methods, but not all of the most recent executions were detected. Figure 10.8 shows the most recent detected executions compared to the Windows Event Log entries.

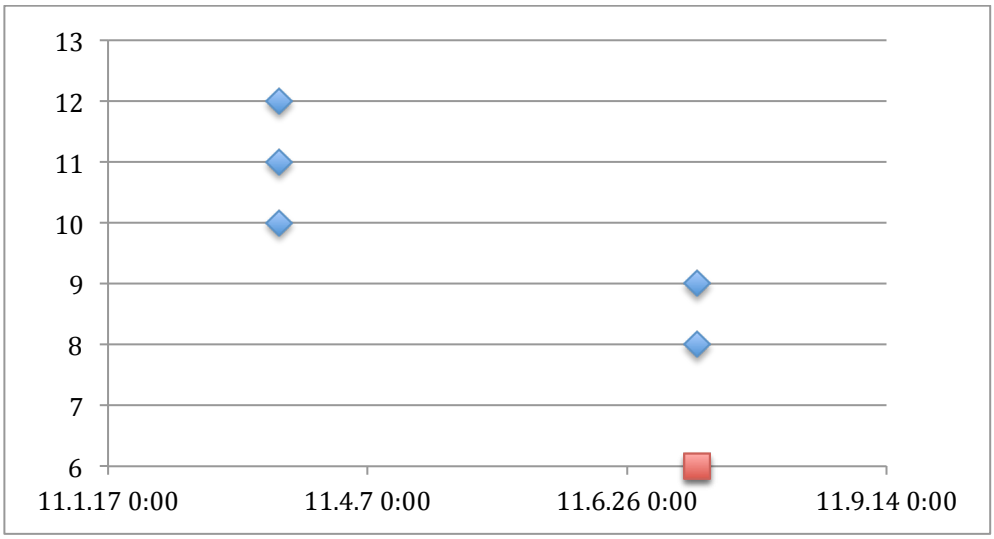


Figure 10.7 Full times of the "Open IE8" action on Computer 2, where signature-detected times are shown in diamonds compared to the Windows Event Log times shown in squares

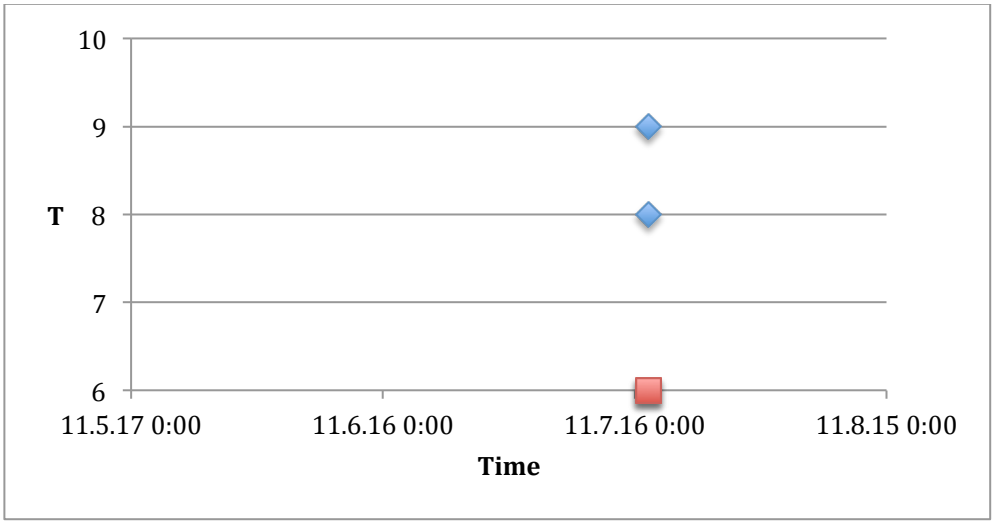


Figure 10.8 Most recent times of the "Open IE8" action on Computer 2, where signature-detected times are shown in diamonds compared to the Windows Event Log times shown in squares

In each case there were no discernable false positives, and the most recent action instance was always detected. There were a large number of false negatives, especially going further back in time, which was expected as evidence of past action instances are overwritten. Further analysis is given in section 10.5.

10.4 Signature-based Detection Error Rate and Analysis

In order to determine the rate of error for the described signature-matching method, the true positives, false positives and false negatives for each action that was detected by Core and Supporting signatures are given in Table 10.11. A true positive is the detection of an action instance that actually happened according to the extracted Windows event logs. A false positive is detection of an action instance when an instance did not happen according to the Windows event logs. A false negative is not detecting an action instance when the action instance actually happened according the Windows event logs.

Table 10.11 Matching true positive, false positive and false negative rates for each tested Core and Supporting signature-detected action instance

Computer	Action	True Positive	False Positive	False Negative
Computer 1	Open FF3	2	0	10
Computer 1	Open IE8	1	0	2
Computer 2	Open FF3	3	0	11
Computer 2	Open IE8	1	0	0
Total		**7**	**0**	**23**

When comparing the signature based detection method to the results of the collected Windows Event Logs, the most recent execution of the action was always accurately found. Past executions of actions were found with no determinable false positives. False negatives, however, were common. This is because observable trace evidence degrades over time as the system is used, and may no longer be available in a post-mortem analysis.

The results show that using signature based detection methods allows an investigator to easily gain knowledge of at least the most recently executed actions, sometimes detecting actions that happened months prior to the examination; however, just because an action was not detected does not mean it did not happen.

Two unexpected happenings were found in the given cases. First, when an action occurs, some artifacts that would normally update (Core) may be locked by the first

instance of the action. If a second instance of the same action occurs while the artifact is locked, the artifact may not be updated. By trusting only the single artifact, incorrect inferences about the most recent execution time of the action could be made. This property could also be beneficial if it is known that more than one object must be updated. In that case, two different time stamps will exist that could provide more information about, and therefore consistency checking of, the action. Second, multiple deleted instances of a single object were found. In this case, each instance of the object will have the same update characteristics as a non-deleted artifact, and would provide more overall time stamp information.

10.4.1 Comparison of Results to Other Methods

For a comparison of the given signature matching method, the action instance detection method proposed by Khan (2008) will be studied. Khan proposed machine-learning methods – specifically Bayesian networks – to learn an application's "footprint". A footprint is essentially the pattern of alteration across objects, normally files, that denotes when an application is executed. This method is effective at automatically learning the footprint of an application to determine the last time an application has been run; however, Khan found that "it is very strenuous and laborious to find shadowed application runs". The learned footprint is somewhat analogous to the Core signature proposed in this work with the exception that even Khan's Core signature is probabilistic.

Khan stated that "[t]he matter of obtaining the application program's accurate footprint is complicated by the fact that much of the application footprint is modified each time the application runs". Using the Khan's method, it is difficult to determine instances of a program's execution beyond the most recent. "Evidence of these runs are required to be derived from events within the audit log files, history files and temporary files found either within the file system entries or by searching for their traces in the free blocks of storage media." However, when information in these data sources is incomplete, Khan believes "it is more appropriate to attach a likelihood measure to the assertion that an application was running at that particular time".

The signature-based method proposed in this work did not yet consider action instance inference based on the state of the content of objects. Even without, previous executions of the action instance may be detected by utilizing categorization of traces

without assigning likelihood measures. With categorization of traces, patterns of relations between traces may be modeled and used to include or exclude hypotheses about an action instance. Khan's method looked only at the fact that traces were updated by the execution of applications, and not the behavior pattern of the update. This is the limiting factor that makes it difficult to infer action instances that occurred before the most recent execution. Also, attempting to learn signatures without considering the update behavior of the associated traces can introduce more uncertainty and error. By utilizing trace update behavior analysis, the method described in this work is able to determine whether some past action instance have occurred without using likelihood measures. However, after consistency checking, when attempting to make statements about which action was likely given two or more possible hypotheses, probabilistic methods may be used while considering their limitations.

One advantage of Khan's method lies in the fact that "footprints" are automatically learned. By combining the trace categorization given in this work with machine learning methods, performance of learned signatures may increase while reducing the complexity of highly manual signature generation.

10.5 Weaknesses of Signature-Based Action Instance Detection

There are four main weaknesses to the proposed signature-based action instance detection method.

1) Lack of Knowledge

The greatest weakness with this method is the same weakness in all signature-based detection methods. Not all possible actions can be known, and unknown actions will not be considered. This lack of complete knowledge is especially relevant in the case of action instance signature generation. If not all possible actions are known, then it is impossible definitely determine a trace's category.

The answer to this weakness comes from the investigator. Human knowledge is also incomplete, yet human investigators are able to state that actions in a system must have happened based on their knowledge of how the system works. Human investigators also update their knowledge based on new experiences. Signature-based action instance detection methods must be able to be updated when new information

that leads to changing a trace's category is found. Even if action signatures are being updated, it is still not possible to account for every piece of custom-made software. For this reason, this method should be used as a pre-analysis inference guide for the human investigator or for post-examination human inference verification rather than for completely automated investigations.

By using the proposed action hypothesis reduction methods, action instances can be derived. However, multiple action hypotheses may be associated with a single trace, and both may be consistent with trace update patterns. In this case, integrating probabilistic approaches may help reduce signature-based weaknesses while still maintaining a high level of accuracy.

2) Accurate Signature Derivation

In order to explore trace update behavior and associated categories, manual signature derivation methods were used. This causes two issues: First, object meta-data update patterns can be easily observed, but object content update patterns are much more difficult and time consuming. Object content updates, such as log file entries, can provide much more information when attempting to determine the execution and consistency of actions.

Next, signature derivation must be automated for the proposed method to be practical. The complexity of the signature generation, and the varying traces generated between different versions of the same software will make maintenance prohibitive. It is because of this that both specific and general signature types need to be explored. With multiple signature types, general signatures may allow for the detection of actions at a high level, and specific signatures will be used for actions that are specifically important to an investigator.

Alternatively, now that trace and signature categorization has been defined in this work, machine-learning techniques to determine each category type may potentially be more effectively tuned for learning signatures of actions by focusing learning algorithms on specifically on trace update consistency checking functions.

3) Object Update Threshold Derivation

The weakness when attempting to accurately determine the object update threshold for a particular action instance is that the update threshold is completely dependent on

unknown variables, such as the speed of the computer and the processing load on the computer when the action instance takes place. Even with a wide sample, it is possible that a very slow computer could produce updates outside of the threshold for the same action instance. Alternatively, if the threshold is too large, faster computers could have multiple action instances within the threshold, making it difficult to differentiate instances.

Possible solutions to this weakness include creating a specific computer update threshold derived from an analysis of the time stamps on the suspect machine being analyzed and modeled against previously known object update threshold times.

4) Use of Anti-forensics Techniques

The proposed signature-based detection of actions is somewhat vulnerable to anti-forensics techniques in at least two ways. First, signature-based methods must have objects to observe. If all objects associated with the action instance are removed, the signatures will not be able to detect an instance of the action. However, if any objects related to the action instance remain, then the instance may possibly be detected.

The second vulnerability involves altering time stamp information. While consistency checking is built into the action instance signature, it may be possible to alter time stamp information in a way that is consistent with the signature consistency checking functions. An intentional altering of time stamp information could result in an action instance being detected at a time when the instance did not occur. The altering of time stamps will be detected with Core signatures that have more than one associated object. However, for Supporting, altering the object's time stamp value to anything before the last execution (Core) time will be considered consistent and presented as the time of an action instance.

10.6 Summary

This chapter gave an evaluation of the proposed signature-based action instance detection method. First, the case scenario was given that was the base case data for the remainder of the chapter. Next, an analysis of the extraction and categorization of traces and update thresholds needed to create signatures of action instances was given. After, a case study was given to demonstrate the practicality of signature-based action instance detection. The results of the signature-based action instance detection method

was measured, and compared to similar work involving action “footprint” generation and matching via machine learning techniques conducted by Khan (2008). Weakness of the proposed signature-based action instance detection method were then examined and discussed.

Chapter 11

Conclusions and Future Work

Digital forensics is a relatively young, and rapidly growing area of forensic science. Both practitioners and academia have made much progress in the field since the 1960's, when computer evidence was first being considered for use in trials; however, there are still many issues that have yet to be resolved. Some issues specifically in regards to the automation of digital forensic investigation tasks have been discussed in this work, and a method to measure the performance of highly automated, knowledge-extracting digital forensic investigation tools compared to the performance of human investigators has been given. Previous research has been examined with a focus on event reconstruction methods, and weaknesses of these methods have been considered.

The aim of this research was to contribute to digital forensic science by providing a faster, more accurate and more efficient method in which digital investigators may conduct digital forensic investigations. The focus of this work was on the analysis and interpretation phases of an investigation, and specifically on how digital forensic investigators make decisions and derive information from suspect data. To this end, this research examined object update patterns within a system on the occurrence of an action, and what inferences may be made from the observation of these update patterns.

Causal relations between actions and a resulting ensemble of traces may be derived for the observation of trace update patterns during a given action instance. Patterns of traces in conjunction with their specific update behaviors may be modeled and encoded as signatures, allowing for the application of signature-based detection methods to be applied to the detection and temporal approximation of happened action instances in a post mortem analysis. By focusing on update behaviors of a given trace, the most recent as well as some past instances of a modeled action may be detected.

Formalization of the causal relation between the occurrence of an action and the resulting ensemble of updated traces has been given in this work. Likewise, a method for the derivation of action-trace relations and update behavior categorization of these traces in a real-world system was discussed. Once done, the object update threshold

specific to the action instance may be statistically derived. With these three pieces of information encoded in an action instance signature, the most recent and some past instances of the modeled action may be automatically approximated for a given system by returning matching objects and checking that returned objects meet consistency requirements defined by the update behavior analysis. The results of which have been described in Chapter 10.

11.1 Achievements

This research proposed a method to automate the human observation and inference process in a digital forensic investigation by examining the causal relation between action instances and resulting trace update patterns in a system for the purposes of hypothesis reduction during investigations. There are seven main contributions of this work:

1. Provided a method for automated action hypothesis encoding, testing and reduction using a form of signature-based matching that is extended beyond simple matching to include relational consistency between objects and events. This relational consistency is then used for hypothesis detection through elimination.
2. Provided a more comprehensive method of detecting the instance(s) of past actions based on the observation of the ensemble of traces altered in the system. Actions may be modeled as trace update patterns allowing for automated, signature-based detection of the actions during a post-mortem analysis.
3. Provided a logical action-sequence checking over detected action instances to infer other actions that must have happened where there are no longer observable traces of these actions. This method uses the same signature-based methods to detect action sequences, and allow the time bounding of newly inferred actions.
4. Defined categories of trace update behaviors that allow for information about an action to be detected. This includes the reliable detection of the most recent as well as prior instances of an action, and allows for consistency checking that helps to detect anti-forensic techniques.
5. Submitted a method for approximating the instance time-span of an action based on the observation of associated traces. When an action occurs, traces

are not created instantaneously, but over a given period of time. Instead of assuming the action must have happened at the time the trace was updated (possibly minutes from when the action originally executed), a more precise time-span in which the action must have occurred may be found.

6. Gave a methodology for measuring investigation beyond false positive and false negative error rating that is currently standard in digital forensic tool testing. This methodology allows for the measurement and comparison of tools as well as processes, and may be used in conjunction with traditional error-rating to help determine what the cause of errors are over time.
7. Provided a method for detecting the occurrence and generic association of actions when no prior information is known about the action based on the naïve clustering of objects (files) within a certain time-span, where the collection of objects will mostly be related to, and contain information about, a generic action, such as browsing the Internet.

11.2 Research Challenges

There are three main research challenges encountered during the course of this research. The first deals with time and resource limits related to action instance signature creation. Initial exploratory tests and analysis to derive action-trace update relationships took a considerable amount of time for generation, processing and analysis phases. Once the three main trace update categories were derived, the time-consuming process of action-trace extraction and categorization still needed to be conducted using mostly manual methods for each action. Because the process is currently highly manual, it is impractical to derive and categorize traces for many actions. Likewise, the sampling of object update thresholds for a single action is also a manual, time-consuming process that must be sampled on many systems with different hardware. To be practical, action-trace association tasks would need to be automated. Machine learning methods are suggested as long as the various trace update behaviors could be effectively learned, and traces could be categorized effectively. Similarly, the object update threshold derivation may possibly be automated using a cluster of differing hardware, if such hardware was available. Without the use of automation in the signature derivation, categorization and threshold sampling stages, signature creation for many common actions in a system would be impractical.

The second challenge comes from a lack of experimental data over long periods of time. In the cases given, volunteers allowed the monitoring and subsequent analysis of their personal computers over a one-week period; however, finding users who will let a researcher monitor their computer usage while using their computer per usual, proved to be difficult. Some data was taken from test cases created by the research group, but the usage patterns on the test cases did not appear to be representative of a normal user that uses their computer for personal and work tasks. Ideally, a larger sampling of real user computer-usage data monitored over at least 12 months would be ideal. However, this data is understandably difficult to acquire, mostly due to privacy issues. Also, if the users were aware that their actions were being monitored, they may be inclined to alter their behavior, skewing the results.

The third challenge is that many other questions, implications and possibilities were discovered during this research. This work can be expanded in many directions, and staying within the defined scope proved to be a challenge.

11.3 Future Work

From this research many areas requiring further research were identified. Directly related work that is currently in progress will be discussed, followed by a summary of some related topics that will be considered in the future.

11.3.1 Deriving Prior Probabilities for Actions Based on User Profiling

Issues with probabilistic methods were discussed in this work, and solutions were defined as beyond the scope of this work, but one possible starting point for future work will briefly be discussed. Since prior probabilities for an action are based on specific user behavior, user profiling could be used to determine whether the user was likely to execute a particular action or not. Investigators currently attempt to do user profiling at a very general level to determine the computer literacy of a user and thus assume what actions the user is likely to be capable of. Some work has been done on user profiling for digital forensic purposes (Marrington 2009; Marrington, Mohay et al. 2010), but user profiling, especially from a post-mortem analysis, always leads to an incomplete picture of the user's normal behavior. User profiling based on detectable computer activities could at least be used to generate more user-specific prior probabilities for actions by combining regionally defined prior probabilities with specific user action analysis. Regardless, user profiling and statistical methods will

always be more general than the particular user in question, and therefore should be used as a guide for what action is likely to have happened and not confused as a fact that one action did happen and another did not.

11.3.2 Further Application of Statistical Methods

Some issues with probabilistic methods have been previously described, but the use of these methods could potentially ease the signature creation process, give better trace categorization performance, and potentially lead to the inference of more action instance information. Statistical methods could be applied in at least two areas: machine learning for signature generation and matching, and probabilistic action-trace association decisions.

In regards to machine learning for signature generation and matching, the main goal is to incorporate the use of trace update behavior categories. If these categories could be learned, they would potentially be more accurate than when derived manually, while at the same time providing similar most recent and past action instance detection rates to the proposed signature-based method. Automated learning of action instance signatures would also make signature-based action instance detection more practical for real-world use, since many new action instance signatures could be learned more quickly and more accurately.

In regards to probabilistic action-trace association decisions, the main issue is how and at what level to derive accurate prior-probabilities. The probability of an action creating a trace should be considered as much as the probability that the action will occur. The existence of a trace may be based on the probabilities of many different actions both occurring and creating the trace when they occur. Further, since not all possible actions can be known, another issue is how to incorporate the probability of an unknown action. Many issues exist with probabilistic methods. To address this issue the previously mentioned user profiling method will be investigated to determine if accurate prior probabilities may be extracted based on a post-mortem analysis.

11.3.3 Analysis of Additional Data Sources

The use of Regular Expressions to locate a specific trace of interest was proposed in this work to enable portability of signatures across systems where file and path names may be variable from system to system. In all cases given, the output of the

SleuthKit's mactime tool was used as the system-state input to extract the state of a particular file's meta-data. Regular Expressions work well over this standardized format; however, Regular Expressions may also be used to extract data structures from raw data (Gladyshev and Almansoori 2010). Because of this, the analysis of file content, and the possibility of lower-level analysis – such as the cluster level, will be explored in future work. At a minimum, the inclusion of log content data will be explored to extract many more action instances. Likewise, the Windows Registry contains much information about past action instances. By combining signature-based methods with the use of snapshot data as described by Zhu (2011), more action instances over the life of the system may possibly be extracted.

11.3.4 Better Measurement of the Analysis Phase of an Investigation

During this research, measurement of inference extraction became an issue since the inference process is somewhat subjective and may be biased or incomplete. The proposed method to measure the accuracy of investigations will be further investigated, and applied to specifically measuring the accuracy of inferences in investigations. The idea is that the inference generated by more intelligent tools may be compared to a human investigator's inferences during an investigation. Without a way to measure how accurate the human is in their inference processes, there will be no way to measure how well the forensic tools perform in their inference processes.

Beyond the application of the proposed investigation accuracy measurement, the weight of derived objects – or inferences – should also be incorporated into the model. Since all inferences do not have the same weight on the final overall conclusion, a weighting system is required. This topic will also be further examined.

11.3.5 Generic Action Instance Analysis

An area that appears to have a high potential to produce more action instance information that may be interesting to investigators is the described generic action instance analysis. By examining the clustering of object updates on a system, it may be possible to extract a meaning of the update or relationship of the objects when no prior information is known. If action instances have already been extracted, then a generic analysis to extract more actions may provide more context about how the system has been used.

Generic action instance analysis using signatures has been proposed, but much research must be conducted into proper parameters to use when little information is known beforehand. For example, proper grouping thresholds for single action instances as well as grouping parameters to detect sessions.

Also, one method of keyword analysis of a group of objects was proposed as a way to demine what actions the objects are normally related to. This method of trace to action categorization will be further examined to determine if object keywords are a good indicator of the generic actions they are related to.

11.3.6 Other Topics

Other works that will be examined in the future include research into the communication and collaboration of criminal justice, academia and corporate entities around the world. Communication between these groups was identified as a major issue at national and international levels alike. Some organizations, such as the Korean National Police, have begun to put emphasis on international collaboration in dealing with digital crime (Jang 2009). Collaboration with these groups will be pursued.

Finally, the area of user behavior profiling was a recurring topic during this research. It may be possible to create both social and user-computer interaction profiles during a post-mortem analysis. This may give an investigator more information about what tasks the user is capable of, which many investigators are currently attempting to ascertain manually during their investigation (James 2011). Research into user behavior profiling based on a post-mortem computer analysis will be considered in the future in order to augment proposed automated analysis.

Thank you.

Bibliography

(1923). Frye v. United States. F., Court of Appeals, Dist. of Columbia. **293:** 1013.

(1970). In re Winship. US, Supreme Court. **397:** 358.

(1993). Daubert v. Merrell Dow Pharmaceuticals, US Supreme Court. **509:** 579.

(1997). General Electric Co. v. Joiner. US, Supreme Court. **522:** 136.

(1999). Kumho Tire Co. v. Carmichael. US, Supreme Court. **526:** 137.

(2000). Crawford v. Commonwealth of Virginia. Record No. 0683-99-1, Court of Appeals of Virginia. **534:** 332.

(2009, 08 July). "Digital forensics in a smarter and quicker way?" Info Security Retrieved 19 Apr., 2012, from http://www.infosecurity-magazine.com/view/2473/digital-forensics-in-a-smarter-and-quicker-way.

(2009). "Public-Key Cryptography Standards (PKCS)." 2.30. Retrieved 21 May, 2012, from http://www.rsa.com/rsalabs/node.asp?id=2176.

(2009). US v. Comprehensive Drug Testing, Inc. F. 3d, Court of Appeals, 9th Circuit. **579:** 989.

(2011, 27 Apr.). "Computer Forensics Tool Testing Program." Retrieved 23 Apr., 2012, from http://www.cftt.nist.gov/.

(2011, 31 Dec.). "Internet Usage Statistics: The Internet Big Picture." Retrieved 11 July, 2012, from http://internetworldstats.com/stats.htm.

(2012). Due Process. Merriam-Webster.com, Merriam-Webster, Inc.

(n.d.). Beyond a Reasonable Doubt. West's Encylopedia of American Law, The Gale Group, Inc.

(n.d.). "Law Handbook Online." Retrieved Nov. 13, 2011, from http://www.lawhandbook.sa.gov.au/go01.php.

ABA (2008). Adopted by the House of Delegates August 11-12, 2008. 301. A. B. Association.

AccessData (2009). AccessData Integrates e-fense's Live Response and Helix Technologies, AccessData.

AccessData. (2010). "Forensic Toolkit." Retrieved 4 Nov., 2010, from http://www.accessdata.com/forensictoolkit.html.

ACSR. (2006). "Association for Crime Scene Reconstruction." Retrieved 22 May, 2011, from http://www.acsr.org/.

Adelstein, F. (2006). "Live forensics: diagnosing your system without killing it first." Commun. ACM **49**(2): 63-66.

ADF. (2011). "ADF Solutions." Retrieved 31 Jan., 2011, from http://www.adfsolutions.com/.

Anderson, J. R. (1991). "The adaptive nature of human categorization." Psychological Review **98**(3): 409.

Anderson, T., D. A. Schum, et al. (2005). Analysis of evidence, Cambridge Univ Pr.

Anderson, T. and W. Twining (1998). Analysis of evidence: How to do things with facts based on Wigmore's Science of Judicial Proof, Northwestern University Press.

Anscombe, G. E. M. (1999). Causality and Determination. Introduction to Philosophy: Classical and Contemporary Readings. J. Perry and M. Bratman. New York, Oxford University Press.

Arasteh, A. R., M. Debbabi, et al. (2007). "Analyzing multiple logs for forensic evidence." Digital Investigation **4**: 82-91.

Ashworth, J. (2010). Is there a link between quality sandards & performance improvement?, Home Office.

Barnard, A. (2009) "Could Your Phone Testify Against You?" The New York Times Upfront **142**.

Beebe, N. L. (2009). "Digital forensic research: The good, the bad and the unaddressed." Advances in Digital Forensics V: 17-36.

Beebe, N. L. and J. G. Clark (2007). "Digital forensic text string searching: Improving information retrieval effectiveness by thematically clustering search results." Digital Investigation **4**: 49-54.

Bilby, D. (2006). Low Down and Dirty: Anti-forensic Rootkits.

Brenner, S. (2006). "Cybercrime jurisdiction." Crime, Law and Social Change **46**(4): 189-206.

Buchholz, F. (2004, 16 Aug. 2005). "Zeitline: a forensic timeline editor." Retrieved 20 Feb., 2012, from http://projects.cerias.purdue.edu/forensics/timeline.php.

Carney, M. and M. Rogers (2004). "The Trojan made me do it: A first step in statistical based computer forensics event reconstruction." International Journal of Digital Evidence **2**(4): 1-11.

Carrier, B. D. (2002). Defining digital forensic examination and analysis tools. DFRWS, Syracuse, NY, Citeseer.

Carrier, B. D. (2003). Open source digital forensics tools: The legal argument. @stake Research Report

Carrier, B. D. (2005). "The SleuthKit." Autopsy: Forensic Tools for Linux and other Unixes (www. sleuthkit. org).

Carrier, B. D. (2006a, 7 June). "Basic Digital Forensic Investigation Concepts." Retrieved 28 Jan., 2011, from http://www.digital-evidence.org/di_basics.html.

Carrier, B. D. (2006b). A hypothesis-based approach to digital forensic investigations. PhD, Purdue University.

Carrier, B. D. (2006c). "Risks of live digital forensic analysis." Commun. ACM **49**(2): 56-61.

Carrier, B. D. (2008, 21 Jan.). "A Brief Introduction To The Computer History Model." Retrieved 2 Feb., 2011, from http://www.digital-evidence.org/hist_model1.html.

Carrier, B. D. (2009). "Mactime output." Retrieved 31 Dec., 2012, from http://wiki.sleuthkit.org/index.php?title=Mactime_output&oldid=404.

Carrier, B. D. (2010). "The Sleuth Kit and The Autopsy Forensic Browser." Retrieved 17 Jan., 2011, from http://www.sleuthkit.org/.

Carrier, B. D. and E. H. Spafford (2003). "Getting physical with the digital investigation process." International Journal of Digital Evidence **2**(2): 1-20.

Carrier, B. D. and E. H. Spafford (2005). Automated digital evidence target definition using outlier analysis and existing evidence. 2005 Digital Forensic Research Workshop (DFRWS).

Casey, E. (2002). "Error, uncertainty, and loss in digital evidence." International Journal of Digital Evidence **1**(2): 1-45.

Casey, E. (2004). Digital evidence and computer crime : forensic science, computers and the Internet. Amsterdam, London, Elsevier Academic.

Casey, E. (2006). "Cutting corners: Trading justice for cost savings." Digital Investigation **3**(4): 185-186.

Casey, E. (2009). "Digital forensics: Coming of age." Digital Investigation **6**(1-2): 1-2.

Casey, E., M. Ferraro, et al. (2009). "Investigation Delayed Is Justice Denied: Proposals for Expediting Forensic Examinations of Digital Evidence*." Journal of forensic sciences **54**(6): 1353-1364.

CBS. (2010). "Survey: More Americans Unhappy at Work." Retrieved 24 Nov., 2010, from http://www.cbsnews.com/stories/2010/01/05/national/main6056611.shtml.

Chisum, W. and B. Turvey (2000). "Evidence dynamics: Locard's exchange principle & crime reconstruction." Journal of Behavioral Profiling **1**(1): 1-15.

CIPD. (2010). "Employee Outlook: Forecast from the front line remains unsettled." Quarterly survey Report Retrieved 24 Nov., 2010, from

http://www.cipd.co.uk/NR/rdonlyres/ECCCDC3C-05A3-4C41-9B63-FA250A61D3FD/0/5306_Employee_Outlook_SR.pdf.

Cisco. (2012, 14 Feb.). "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016." Retrieved 11 July, 2012, from http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.

Civie, V. and R. Civie (1998). Future technologies from trends in computer forensic science. Information Technology Conference, 1998. IEEE.

Clede, B. (1993). "Investigating Computer Crimes." Law and Order **41**(7): 99-102.

CNN. (2009). "CNN: Her name was Neda." from http://www.youtube.com/watch?v=b5KBrsz1oxs&feature=related.

Cohen, F. B. (2010). "Fundamentals of Digital Forensic Evidence." Handbook of Information and Communication Security: 789-808.

Collier, P. A. and B. J. Spaul (1992a). "A forensic methodology for countering computer crime." Artificial Intelligence Review **6**(2): 203-215.

Collier, P. A. and B. J. Spaul (1992b). "Forensic Science Against Computer Crime in the United Kingdom." Journal of the Forensic Science Society **32**(1): 27-34.

Congress, U. (2004). Proceedings and Debates of the 108th Congress. Congressional Record, Washington, United States Government Printing Office.

Connery, E. M. and S. B. Levy (1979). "Computer Evidence in Federal Courts." Commercial Law Journal **84**: 266-276.

Coopman, S. J. (2006, 26 May). "Fact, Observation & Inference." Retrieved 27 Feb., 2012, from http://www.roguecom.com/interview/facts.html.

de Vel, O. (2004). "File classification using byte sub-stream kernels." Digital Investigation **1**(2): 150-157.

DeHetre, J. D. (1975). "Data Processing Evidence-Is It Different?" Chicage-Kent Law Review **52**: 567-599.

Dempsey, J. and L. Forst (2009). An introduction to policing, Cengage Learning.

Dolinak, D., E. W. Matshes, et al. (2005). Forensic pathology: principles and practice, Academic Press.

Elman, J. L. (1990). "Finding Structure in Time." Cognitive Science **14**(2): 179-211.

EURIM-ippr. (2004). "EURIM - IPPR E-Crime Study: Partnership Policing for the Information Society." Third Discussion Paper. Retrieved 23 Nov., 2010, from http://www.eurim.org/consult/e-crime/may_04/ECS_DP3_Skills_040505_web.htm.

EURIM-ippr (2010). Separating Myth from Reality and Snake-Oil from Practicality. Partnership Policing for the Information Society, European Information Soceity Group (EURIM) and ippr.

Everett, C. (2005). "Cred or crud." Infosecurity Today **2**(6): 42-43.

Farmer, D. (2007). "A Forensic Analysis of the Windows Registry." Retrieved 28 Feb., 2012, from http://eptuners.com/forensics/contents/examination.htm.

Farmer, D. and W. Venema (2005). Forensic Discovery, Addison-Wesley Professional.

Fingerprinting. (2010). "Fingerprint Powder." Retrieved 6 July, 2010, from http://www.fingerprinting.com/fingerprint-powder.php.

Foster, J. C. and V. Liu (2005). Catch Me If You Can: Exploiting Encase, Microsoft, Computer Associates, and the rest of the bunch... Black Hat USA 2005. Las Vegas, USA.

Freiling, F. C. and B. Schwittay (2007). A common process model for incident response and computer forensics. Proceedings of Conference on IT Incident Management and IT Forensics.

Galton, F. (1892). Finger Prints. http://galton.org/books/finger-prints/galton-1892-fingerprints-1up.pdf.

Garfinkel, S. L. (2010). "Digital forensics research: The next 10 years." Digital Investigation **7**(Supplement 1): S64-S73.

Giannelli, P. (2006). Judicature - Scientific Evidence.

Giarrtano, J. C. and G. D. Riley (2005). Expert Systems Principles and Programming, Course Technology, Thomson Learning, Inc.

Gilovich, T. (1993). How we know what isn't so: The fallibility of human reason in everyday life. New York, NY, The Free Press.

Gladyshev, P. (2004). Formalising event reconstruction in digital investigations. PhD Thesis (Ph D) - University College Dublin, 2004, University College Dublin.

Gladyshev, P. (2005). "Finite state machine analysis of a blackmail investigation." International Journal of Digital Evidence **4**(1): 1-13.

Gladyshev, P. and A. Almansoori (2010). Reliable Acquisition of RAM dumps from Intel-based Apple Mac computers over FireWire. Second International Conference on Digital Forensics and Cyber Crime (ICDF2C). Abu Dhabi, UAE, ICST.

Gladyshev, P. and A. Enbacka (2007). "Rigorous development of automated inconsistency checks for digital evidence using the B method." International Journal of Digital Evidence (IJDE) **6**(2): 1-21.

Gladyshev, P. and A. Patel (2004). "Finite state machine approach to digital event reconstruction." Digital Investigation **1**(2): 130-149.

Gladyshev, P. and A. Patel (2005). "Formalising event time bounding in digital investigations." International Journal of Digital Evidence **4**(2).

Gogolin, G. (2010). "The Digital Crime Tsunami." Digital Investigation **7**(1-2): 3-8.

Goldstein, S. (1997). Managing attention and learning disorders in late adolescence and adulthood: A guide for practitioners, John Wiley & Sons Inc.

Goss, J. (2010). Forensic Triage: Managing the Risk. Master fo Science, University College Dublin.

Grinnell, R. M. (1993). Social Work Research and Evaluation, F. E. Peacock Publishers, Inc.

Gudjonsson, K. (2010). "Log2Timeline." Retrieved 20 Feb., 2012, from http://log2timeline.net/.

Guidance (2009). EnCase Legal Journal, September 2009 Edition. The practitioner's guide to legal issues related to digital invesigations and electrioic discovery.

Guidance. (2010). "EnCase Forensic." Retrieved 4 Nov., 2010, from http://www.guidancesoftware.com/forensic.htm.

Gutheil, T. G. and H. J. Bursztajn (2005). "Attorney Abuses of Daubert Hearings: Junk Science, Junk Law, or Just Plain Obstruction?" Journal of the American Academy of Psychiatry and the Law **33**(2): 150-152.

Halderman, J., S. Schoen, et al. (2009). "Lest we remember: cold-boot attacks on encryption keys." Communications of the ACM **52**(5): 91-98.

Hannan, M. (2004). To Revisit: What is Forensic Computing? 2nd Australian Computer, Network & Information Forensics Conference. Perth, Australia.

Heimdahl, M. P. E. and N. G. Leveson (1996). "Completeness and consistency in hierarchical state-based requirements." Software Engineering, IEEE Transactions on **22**(6): 363-377.

Herrera, G. (2006). "Cyberspace and Sovereignty." Retrieved 28 Mar., 2011, from http://www.allacademic.com/meta/p98069_index.html.

Higginbotham, S. (2010, 14 Apr.). "Ericsson CEO Predicts 50 Billion Internet Connected Devices by 2020." Retrieved 27 Jan., 2011, from http://gigaom.com/2010/04/14/ericsson-sees-the-internet-of-things-by-2020/.

Hilley, S. (2007). "Anti-forensics with a small army of exploits." Digital Investigation **4**(1): 13ñ15.

Hillier, S. (1996). "The System Registry." Retrieved 28 Feb., 2012, from http://msdn.microsoft.com/en-us/library/ms970651.aspx.

HomeOffice (2011). Codes of Practice and Conduct for forensic science providers and practitioners in the Criminal Justice System. F. S. Regulator, Crown Copyright**:** 48.

HTCIA (2010). 2010 Report on Cyber Crime Investigation. Roseville, CA, High Technology Crime Investigation Association (HTCIA).

Huber, E. (2010). "Certification, Licensing, and Accreditation in Digital Forensics." A Fistful of Dongles http://ericjhuber.blogspot.com/2010/11/certification-licensing-and.html 2010.

Hume, D. and L. Selby-Bigge (1886). A treatise of human nature, Longmans.

Hunton, P. (2012). "Managing the technical resource capability of cybercrime investigation: a UK law enforcement perspective." Public Money & Management **32**(3): 225-232.

Ianuzzi, T. (2007). "Automating Computer Forensics." Retrieved 4 Nov., 2010, from http://www.forensicautomation.org/introduction.htm.

IBTimes (2010) "Number of Internet users in emerging markets to double by 2015: report." International Business Times.

IC3 (2011). 2011 Internet Crime Report, Internet Crime Complaint Center.

INTERPOL. (2008). "INTERPOL asserts neutrality in seized FARC computer evidence probe in Colombia." Retrieved 25 Nov., 2010, from http://www.interpol.int/public/ICPO/PressReleases/PR2008/PR200812.asp.

Irons, A. D., P. Stephens, et al. (2009). "Digital Investigation as a distinct discipline: A pedagogic perspective." Digital Investigation **6**(1-2): 82-90.

James, J. (2010). "Survey of Evidence and Forensic Tool Usage in Digital Investigations " Retrieved 22 Dec., 2010, from http://cci.ucd.ie/content/survey-evidence-and-forensic-tool-usage-digital-investigations.

James, J., P. Gladyshev, et al. (2010). "Analysis of Evidence Using Formal Event Reconstruction." Digital Forensics and Cyber Crime **31**: 85-98.

James, J., P. Gladyshev, et al. (2010). Signature Based Detection of User Events for Post- Mortem Forensic Analysis. 2nd International ICST Conference on Digital Forensics & Cyber Crime (ICDF2C). Abu Dhabi, UAE.

James, J. I. (2011). An Garda Síochána Computer Crime Investigation Capability and Needs Analysis. Dublin, Centre for Cybersecurity and Cybercrime Investigation, UCD.

James, J. I. (2012). An Garda Síochána Preliminary Analysis Unit Process Model Definition and Verification DRAFT 2012-05-17. Dublin, University College Dublin Centre for Cybersecurity and Cybercrime Investigation.

James, J. I. and P. Gladyshev. (2010). "2010 Report of digital forensic standards, processes and accuracy measurement." Retrieved 22 Dec., 2013, from http://www.forensicfocus.com/2010-digital-forensics-standards-processes-accuracy.

James, J. I., M. Koopmans, et al. (2011). Rapid Evidence Acquisition Project for Event Reconstruction. The Sleuth Kit & Open Source Digital Forensics Conference, McLean, VA, Basis Technology.

James, J. I., A. F. Shosha, et al. (2012). Digital Forensic Investigation and Cloud Computing. Cybercrime and Cloud Forensics: Applications for Investigation Processes. K. Ruan, IGI Global.

Jang, Y. (2009). The Current Situation and Countermeasures to Cybercrime and Cyber-Terror in the Republic of Korea. 143rd UNAFEI Internatioanl Training Course. M. Sasaki. Tokyo, Japan, United Nataions Asia and Far East Institute for the Prevention of Crime and the Treatment of Offenders (UNAFEI).

Jenkins, M. M. (1975). "Computer-Generated Evidence Specially Prepared for Use at Trial." Chicage-Kent Law Review **52**: 600-609.

Jiang, X., F. Buchholz, et al. (2007). "Tracing worm break-in and contaminations via process coloring: A provenance-preserving approach." IEEE Transactions on Parallel and Distributed Systems: 890-902.

Jones, A. and C. Valli (2008). Building a Digital Forensic Laboratory: Establishing and Managing a Successful Facility, Butterworth-Heinemann.

Jones, N. (2004). "Training and accreditation - who are the experts?" Digital Investigation **1**(3): 189-194.

Jones, R. (2012). Towards a Global Criminology? Working papers series. Edinburgh, Edinburgh School of Law Research**:** 28.

Kahvedzic, D. and T. Kechadi (2008). Extraction of User Activity through Comparison of Windows Restore Points, Citeseer.

Kanable, R. (2007, July 15). "The New Frontier In Digital Evidence." Retrieved Nov. 21, 2011, from http://www.officer.com/article/10249649/the-new-frontier-in-digital-evidence.

Keane, A. (2008). The modern law of evidence, Oxford University Press, USA.

Kelman, A. and R. Sizer (1982). Computer in Court - A Guide to Computer Evidence for Lawyers and Computing Professionals.

Kent, K., S. Chaevalier, et al. (2006). Guide to Integrating Forensic Techniques into Incident Response, National Institute of Standards and Technology**:** 121.

Keppens, J. (2007). Towards qualitative approaches to Bayesian evidential reasoning, ACM.

Keppens, J., Q. Shen, et al. (2005). Probabilistic abductive computation of evidence collection strategies in crime investigation, ACM.

Kessler, R., W. Chiu, et al. (2005). "Prevalence, severity, and comorbidity of twelve-month DSM-IV disorders in the National Comorbidity Survey Replication (NCS-R)." Archives of General Psychiatry **62**(6): 617-627.

Khan, M., C. Chatwin, et al. (2007). "Extracting Evidence from Filesystem Activity using Bayesian Networks." International journal of Forensic computer science **1**: 50-63.

Khan, M., C. Chatwin, et al. (2007). "A framework for post-event timeline reconstruction using neural networks." Digital Investigation **4**(3-4): 146-157.

Khan, M. and I. Wakeman (2006). Machine Learning for Post-Event Timeline Reconstruction.

Khan, M. N. A. (2008). Digital Forensics using Machine Learning Methods. Ph.D., University of Sussex.

Kim, J. S., D. G. Kim, et al. (2004). A fuzzy logic based expert system as a network forensics, IEEE.

King, S. T. and P. M. Chen (2005). "Backtracking intrusions." ACM Transactions on Computer Systems (TOCS) **23**(1): 51-76.

Koen, R. and M. S. Olivier (2008). "The Use of File Timestamps in Digital Forensics."

Kohtz, D. (2011). Dealing with the Digital Evidence Backlog. Digital Forensics Magazine.

Koopmans, M. (2010). The Art of Triage with (g)PXE. Master of Science, University College Dublin.

Kovar, D. (2009a). "The value of push button forensics." Intergriography: A Journal of Broken Locks, Ethics, and Computer Forensics http://integriography.wordpress.com/2009/11/17/the-value-of-push-button-forensics/ 2010.

Kovar, D. (2009b). "Push button forensics - managing the downsides." Intergriography: A Journal of Broken Locks, Ethics, and Computer Forensics http://integriography.wordpress.com/2009/11/19/push-button-forensics-managing-the-downsides/ 2010.

Kwan, M., K. P. Chow, et al. (2008). "Reasoning about evidence using Bayesian networks." Advances in Digital Forensics IV: 275-289.

Lee, R. (2008). Draft Forensic Common Body of Knowledge, SANS Institute.

Li, B., Q. Wang, et al. (2006). Forensic Analysis of Document fragment based on SVM. International Conference on Intelligent Information Hiding and

Multimedia Signal Processing (IIH-MSP'06). Pasadena, California, USA**:** 236-239.

Lim, K.-S., A. Savoldi, et al. (2012). "On-the-spot digital investigation by means of LDFS: Live Data Forensic System." Mathematical and Computer Modelling **55**(1,Äì2): 223-240.

Lyle, J. (2010). "If error rate is such a simple concept, why don't I have one for my forensic tool yet?" Digital Investigation **7**: S135-S139.

MacForensicsLab. (2010). "Field triage tool benefits." Retrieved 2 Dec., 2010, from http://www.macforensicslab.com/ProductsAndServices/index.php?main_page=document_general_info&cPath=5_24&products_id=200.

Manes, G. and E. Downing (2009). "Overview of licensing and legal issues for digital forensic investigators." Security & Privacy, IEEE **7**(2): 45-48.

Manning, C., P. Raghavan, et al. (2008). Introduction to information retrieval, Cambridge University Press.

Mansfield-Devine, S. (2010). "Fighting forensics." Computer Fraud & Security(1): 17-20.

Marrington, A., G. Mohay, et al. (2007). Event-based computer profiling for the forensic reconstruction of computer activity.

Marrington, A., G. Mohay, et al. (2010). A Model for Computer Profiling, IEEE.

Marrington, A. D. (2009). Computer profiling for forensic purposes. PhD, Queensland University of Technology.

Martin, A. (2007). "Firewire memory dump of a Windows XP computer: a forensic approach."

McAfee (2010). A Good Decade for Cybercrime. Santa Clara, California, McAfee, Inc.

McGguire, C., R. E. Hurley, et al. (1964). "Auscultatory Skill: Gain and Retention after Intensive Instruction." Journal of Medical Education **39**(2): 120-131.

McKemmish, R. (1999). "What is forensic computing." Trends and issues in crime and criminal justice **118**.

Meeker, M. (2012, 12 June). "Internet Trends." D10 Conference Retrieved 11 July, 2012, from http://allthingsd.com/20120612/mary-meeker-explains-internet-2012-in-17-minutes-the-full-d10-interview-video/?refcat=d10.

Meissner, C. A. and S. M. Kassin (2002). "He's guilty!: Investigator Bias in Judgments of Truth and Deception." Law and Human Behavior **26**(5): 469-480.

Microsoft. (2010). "Computer Online Forensic Evidence Extractor (COFEE)." Retrieved 4 Feb., 2011, from http://www.microsoft.com/industry/government/solutions/cofee/default.aspx.

Mislan, R. P., E. Casey, et al. (2010). "The growing need for on-scene triage of mobile devices." Digital Investigation **6**(3-4): 112-124.

NFPA (2005). User's manual for the National Fire Protection Association 921: guide for fire and explosion investigations, Jones & Bartlett Pub.

Nguyen, L. (2012, 20 Apr.). "Tori Stafford trial: Cellphone record shows gap during abduction, murder." Retrieved 13 Jul., 2012, from http://www.canada.com/life/Tori+Stafford+trial+Cellphone+record+shows+during+abduction+murder/6486178/story.html.

NIJ. (2008, 14 Apr.). "Electronic Crime Scene Investigation: A Guide for First Responders, Second Edition." Second Edition. Retrieved 2 Feb., 2011, from http://www.ojp.usdoj.gov/nij/publications/ecrime-guide-219941/welcome.htm.

O'Connor, T. (2010, 21, Aug.). "Admissibility of Scientific Evidence Under Daubert." Retrieved 26, Jan., 2011, from http://www.drtomoconnor.com/3210/3210lect01a.htm.

Obasogie, O. (2009, 1 Apr.). "Phantom of Heilbronn Revealed!" Retrieved 16 Dec., 2010, from http://www.biopoliticaltimes.org/article.php?id=4616.

Ogawa, A., Y. Yamazaki, et al. (2010). "Neural correlates of species-typical illogical cognitive bias in human inference." Journal of cognitive neuroscience **22**(9): 2120-2130.

Overill, R. E., J. A. M. Silomon, et al. (2010). Sensitivity Analysis of a Bayesian Network for Reasoning about Digital Forensic Evidence, IEEE.

Palmer, G. (2001). DFRWS Technical Report: A Road Map for Digital Forensic Research. Digital Forensic Research Workshop. G. Palmer. Utica, New York.

Palmer, G. (2002). "Forensic analysis in the digital world." International Journal of Digital Evidence **1**(1): 1-6.

Parsonage, H. (2009). The Meaning of (L)inkfiles (I)n (F)orensic (E)xaminations. http://computerforensics.parsonage.co.uk/downloads/TheMeaningofLIFE.pdf.

Paul, W. (2012). "Cyber War, Formal Verification and Certified Infrastructure." Verified Software: Theories, Tools, Experiments: 1-1.

Pollitt, M. (1995). Computer Forensics: An Approach to Evidence in Cyberspace. National Information Systems Security Conferences. Baltimore.

Pollitt, M. (1995). Principles, practices, and procedures: an approach to standards in computer forensics.

Pollitt, M. (2007). An ad hoc review of digital forensic models, IEEE.

Polsson, K. (2011, 1 Jan.). "Chronology of Personal Computers." Retrieved 25 Jan., 2011, from http://www.islandnet.com/~kpolsson/comphist/.

Pouzol, J. P. and M. Ducassé (2002). Formal specification of intrusion signatures and detection rules.

Purdy, C. (2010, 11 Aug.). "Industry's First Forensic-base Critical Infrastructure Security Solution." Retrieved 29 Jan., 2011, from https://www.guidancesoftware.com/Media/NewsRoom/NewsRoomBlog.aspx?B=BlogContentDetails&Blog_S=NewsRoomMenu&newsroommenu_id=3503&image_id=1000000303&md_id=1000000296&id=1000000267&blogid=2523.

Raasch, J. (2010, 12 July). "Child porn prosecutions delayed by backlog of cases." Retrieved 25 Sept., 2010, from http://www.easterniowanewsnow.com/2010/07/12/child-porn-prosecutions-delayed-by-backlog-of-cases/.

Reith, M., C. Carr, et al. (2002). "An examination of digital forensic models." International Journal of Digital Evidence **1**(3): 1-12.

Roberts, J. J. (1974). "A Practitioner's Primer on Computer-Generated Evidence." The University of Chicago Law Review **41**(2): 254-280.

Rogers, M., J. Goldman, et al. (2006). "Computer forensics field triage process model." Journal of Digital Forensics, Security and Law **1**(2): 27-40.

Roiter, N. (2007, May 16). "When signature based antivirus isn't enough." Retrieved Nov. 15, 2011, from http://www.computerweekly.com/news/1280096452/When-signature-based-antivirus-isnt-enough.

Ruan, K., J. Carthy, et al. (2011). "Cloud forensics: An overview." Advances in Digital Forensics VII.

Ruan, K., J. I. James, et al. (2012). Cloud Forensics: Key Terms for Service Level Agreements. Eigth Annual IFIP WG 11.9 International Conference on Digital Forensics. Pretoria, South Africa.

Russell, S. and P. Norvig (2009). Artificial intelligence: a modern approach Prentice hall.

Russinovich, M. (n.d.). "Inside the Registry." Retrieved Feb. 3, 2010, from http://technet.microsoft.com/en-us/library/cc750583.aspx.

Sambandaraksa, D. (2010). Seeking unified global approach to cyber crime. Bangkok Post.

Scarfone, K. and P. Mell (2007). Guide to Intrustion Detection and Prevention Systems (IDPS). Gaithersburg, NIST: National Institute of Science and Technology.

Schlicher, B. (2008). Emergence of cyber anti-forensics impacting cyber security, ACM.

Schneier, B. (2010). "The Threat of Cyberwar Has Been Grossly Exaggerated." Schneier on Security http://www.schneier.com/blog/archives/2010/07/the_threat_of_c.html 2011.

Senge, P. M. (2006). The fifth discipline: The art and practice of the learning organization, Vintage.

Shafer, G. (1976). A Mathematical Theory of Evidence, Princeton University Press.

Shen, Q., J. Keppens, et al. (2006). "A scenario-driven decision support system for serious crime investigation." Law, Probability and Risk **5**(2): 87.

Shinder, D. and M. Cross (2008). Scene of the Cybercrime, Syngress Media.

Shipley, T. G. and B. Door. (2012, 27 Jan.). "Forensic Imaging of Hard Disk Drives-What we thought we knew." Retrieved 15 July, 2012, from http://articles.forensicfocus.com/2012/01/27/forensic-imaging-of-hard-disk-drives-what-we-thought-we-knew-2/.

Shosha, A. F., J. I. James, et al. (2011). A Novel Methodology for Malware Intrustion Attack Path Reconstruction. 3rd International ICST Conference on Digital Forensics & Cyber Crime (ICDF2C). Dublin, Ireland.

Shosha, A. F., J. I. James, et al. (2012). Automated Forensic Event Reconstruction of Malicious Code. 15th International Symposium on Research in Attacks, Intrusions and Defenses (RAID'12).

Shosha, A. F., J. I. James, et al. (2012). Towards Automated Forensic Event Reconstruction of Malicious Code. 4th International Conference on Digital Forensics and Cyber Crime (ICDF2C'12).

SkillsforJustice. (2010). "What are NOS and how do they work?" Retrieved 25 Nov., 2010, from http://www.skillsforjustice.com/template01.asp?pageid=37.

Smith, S. E. (2011). "What is Cyberwar?" Retrieved 28 Jan., 2011, from http://www.wisegeek.com/what-is-cyberwar.htm.

Spafford, E. H. and S. A. Weeber (1993). "Software forensics: Can we track code to its authors?" Computers & Security **12**(6): 585-595.

Spinoza, B. (1677). Ethics.

Stallard, T. and K. Levitt (2003). "Automated analysis for digital forensic science: Semantic integrity checking."

SWGDE (2009). SWGDE/SWGIT Digital & Multimedia Evidence Glossary Version: 2.3, Scientific Working Group on Digital Evidence.

Sy, B. (2005). "Signature-based approach for intrusion detection." Machine Learning and Data Mining in Pattern Recognition: 633-633.

Tapper, C. (1974). "Evidence From Computers." Rutgers Journal of Computers and the Law **4**: 324-406.

Taylor, C., B. Endicott-Popovsky, et al. (2007). "Specifying digital forensics: A forensics policy approach." Digital Investigation **4**: 101-104.

Teubner, A. L. (1978). "The Computer as Expert Witness: Toward a Unified Theory of Computer Evidence." Jurimetrics Journal **19**: 274-297.

Thompson, W. C. (2002). DNA Testing. Encyclopedia of crime and punishment. D. Levinson. Thousand Oaks, California, Sage Publications, Inc. **2:** 537-544.

Thornton, J. I. and J. L. Peterson (1997). "The general assumptions and rationale of forensic identification." David L. Faigman, David H. Kaye, Michael J. Saks, & Joseph Sanders, Modern Scientific Evidence: The Law And Science Of Expert Testimony **2**.

Turvey, B. E. (2008). Criminal profiling: An introduction to behavioral evidence analysis, Academic Press.

USDoJ (2002). Prosecuting Computer Crimes. S. Eltringham, United States Department of Justice, Computer Crime & Intellectual Property Section.

Wagner, D. and P. Soto (2002). Mimicry attacks on host-based intrusion detection systems, ACM.

Wender, P., L. Wolf, et al. (2001). "Adults with ADHD. An overview." Annals of the New York Academy of Sciences **931**: 1.

Wik, L., H. Myklebust, et al. (2002). "Retention of basic life support skills 6 months after training with an automated voice advisory manikin system without instructor involvement." Resuscitation **52**(3): 273-279.

Willassen, S. Y. (2008a). "Hypothesis-based investigation of digital timestamps." Advances in Digital Forensics IV: 75-86.

Willassen, S. Y. (2008b). Timestamp evidence correlation by model based clock hypothesis testing, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Williams, C. (2011) "Cybercrime gang 'responsible for a third of all data thefts'." The Telegraph.

Wilsdon, T. and J. Slay (2005). Digital forensics: exploring validation, verification & certification. First International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'05), IEEE.

Xia, Y., K. Fairbanks, et al. (2008). "A program behavior matching architecture for probabilistic file system forensics." ACM SIGOPS Operating Systems Review **42**(3): 4-13.

Yeoman, F. (2009, 27 Mar.). "The Phantom of Heilbronn, the tainted DNA and an eight-year goose chase." Retrieved 16 Dec., 2010, from http://www.timesonline.co.uk/tol/news/world/europe/article5983279.ece.

Zhu, Y. (2011). A Novel Event Extraction Model for Snapshot Data in Digital Forensics. PhD, University College Dublin.

Zhu, Y., P. Gladyshev, et al. (2009). Identifying Newly Updated Data Values of MRU Keys Between Registry Snapshots Fifth Annual IFIP WG 11.9 International Conference on Digital Forensics. Orlando, Florida, USA, Springer Berlin Heidelberg. **306**.

Zhu, Y., P. Gladyshev, et al. (2009). "Temporal Analysis of Windows MRU Registry Keys." Advances in Digital Forensics V **306/2009**: 83-93.

Zhu, Y., P. Gladyshev, et al. (2009). "Using shellbag information to reconstruct user activities." Digital Investigation **6**: S69-S77.

Zhu, Y., J. James, et al. (2009). "A comparative methodology for the reconstruction of digital events using Windows Restore Points." Digital Investigation **6**(1-2): 8-15.

Zhu, Y., J. James, et al. (2010). "A Consistency Study of the Windows Registry." Advances in Digital Forensics VI: 77-90.

Appendices

Appendix A: Results of precision of investigation vs. the gold standard in case 1.

Analysis 1:

Table Appxs.1 Examination 1 artifacts identified compared to the gold standard

	Inculpatory	Exculpatory	False Positive	Total
Gold Standard	12	0	N/A	12
Triage Examination	4	0	2	6

$$P = \frac{4}{6} = 0.67 \qquad R = \frac{4}{12} = 0.33 \qquad F = 2 \cdot \frac{0.67 \cdot 0.33}{0.67 + 0.33} = 0.44$$

Analysis 2:

Table Appxs.2 Examination 2 artifacts identified compared to the gold standard

	Inculpatory	Exculpatory	False Positive	Total
Gold Standard	0	1	N/A	1
Triage Examination	0	0	5	5

$$P = \frac{0}{5} = 0 \qquad R = \frac{0}{1} = 0 \qquad F = 2 \cdot \frac{0 \cdot 0}{0 + 0} = 0$$

Analysis 3:

Table Appxs.3 Examination 3 artifacts identified compared to the gold standard

	Inculpatory	Exculpatory	False Positive	Total
Gold Standard	200	0	N/A	200
Triage Examination	200	0	0	200

$$P = \frac{200}{200} = 1 \qquad R = \frac{200}{200} = 1 \qquad F = 2 \cdot \frac{1 \cdot 1}{1+1} = 1$$

Analysis 4:

Table Appxs.4 Examination 4 artifacts identified compared to the gold standard

	Inculpatory	**Exculpatory**	**False Positive**	**Total**
Gold Standard	30	0	N/A	30
Triage Examination	16	0	200	216

$$P = \frac{16}{216} = 0.07 \qquad R = \frac{16}{30} = 0.53 \qquad F = 2 \cdot \frac{0.07 \cdot 0.53}{0.07 + 0.53} = 0.12$$

Analysis 5:

Table Appxs.5 Examination 5 artifacts identified compared to the gold standard

	Inculpatory	**Exculpatory**	**False Positive**	**Total**
Gold Standard	34	0	N/A	34
Triage Examination	4	0	26	26

$$P = \frac{4}{26} = 0.15 \qquad R = \frac{4}{34} = 0.12 \qquad F = 2 \cdot \frac{0.15 \cdot 0.12}{0.15 + 0.12} = 0.13$$

Appendix B: Results of precision of investigation vs. the gold standard in case 2.

- **Media Number 1**

Table Appxs.6 Results of a full examination on media number 1

Fully-Examined Case	
Illegal Objects	**Notes**
0	No illegal content was detected in a full analysis

Table Appxs.7 Results of preliminary analysis on media number 1 from five examiners

Preliminary Analysis Results			
Examiner	**Further Analysis**	**Suspect Objects**	**Notes**
Examiner 1	Yes	4	Decision made based on found images suspicious deleted files and searching activity
Examiner 2	Yes	6	Decision made based on found images, cleaner programs, Internet activity and evidence of P2P activity
Examiner 3	Yes	4	Decision made based on found images, movies and Internet search and browser history
Examiner 4	Yes	30	Decision made based on large amount of highly suspicious images and some movie files
Examiner 5	Yes	903	Decision made based on a large amount suspicious images

Table Appxs.8 Preliminary analysis object identification error rates for media number 1

Object Identification Error Rate				
Examiner	**Num. of False Positives**	**False Positive Error**	**Num. of False Negatives**	**False Negative Error**
Examiner 1	4	1	0	0
Examiner 2	6	1	0	0
Examiner 3	4	1	0	0
Examiner 4	30	1	0	0
Examiner 5	903	1	0	0

Table Appxs.9 Preliminary analysis accuracy rates for media number 1

Accuracy Rate			
Examiner	**Precision**	**Recall**	**F-measure**
Examiner 1	n/a	n/a	n/a
Examiner 2	n/a	n/a	n/a
Examiner 3	n/a	n/a	n/a
Examiner 4	n/a	n/a	n/a
Examiner 5	n/a	n/a	n/a

- **Media Number 2**

Table Appxs.10 Results of a full examination on media number 2

Fully-Examined Case	
Suspect Objects	**Notes**
19	All illegal objects were images

Table Appxs.11 Results of preliminary analysis on media number 2 from five examiners

Examiner	Further Analysis	Suspect Objects	Notes
Examiner 2	Yes	44	Decision made based on suspicious images, cookies and installed cleaner
Examiner 5	Yes	9	Decision made based on suspicious images. Note: more suspicious images available not listed in report.
Examiner 1	Yes	6	Decision made based on suspicious movie, porn chat (cookies), possible disk wiping, undeleted, and nothing in the live set
Examiner 3	Yes	75	Decision made based on suspicious images, undeleted and cookies
Examiner 4	Yes	40	Decision made based on many suspicious undeleted images and trace cleaning software

Table Appxs.12 Preliminary analysis object identification error rates for media number 2

Object Identification Error Rate				
Examiner	**Num. of False Positives**	**False Positive Error**	**Num. of False Negatives**	**False Negative Error**
Examiner 2	25	.56	0	0
Examiner 5	0	0	10	.47
Examiner 1	1	.05	13	.68
Examiner 3	56	.74	0	0
Examiner 4	21	.53	0	0

Table Appxs.13 Preliminary analysis accuracy rates for media number 2

Accuracy Rate			
Examiner	**Precision**	**Recall**	**F-measure**
Examiner 2	.43	1	.60
Examiner 5	1	.47	.64
Examiner 1	.83	.26	.40
Examiner 3	.25	1	.41
Examiner 4	.48	1	.64

- **Media Number 3**

Table Appxs.14 Results of a full examination on media number 3

Fully-Examined Case	
Suspect Objects	**Notes**
0	No evidence or trace evidence relevant to the investigation

Table Appxs.15 Results of preliminary analysis on media number 3 from five examiners

Examiner	**Further Analysis**	**Suspect Objects**	**Notes**
Examiner 3	Yes	0	Decision made based on presence of virtual machines
Examiner 5	No	0	n/a
Examiner 4	Yes	0	Decision made based on evidence that user is highly computer literate
Examiner 2	Yes	0	Decision made based on deleted files that could not be processed – user also highly computer literate
Examiner 1	No	0	n/a

Table Appxs.16 Preliminary analysis object identification error rates for media number 3

Object Identification Error Rate				
Examiner	**Num. of False Positives**	**False Positive Error**	**Num. of False Negatives**	**False Negative Error**
Examiner 3	0	0	0	0
Examiner 5	0	0	0	0
Examiner 4	0	0	0	0
Examiner 2	0	0	0	0
Examiner 1	0	0	0	0

Table Appxs.17 Preliminary analysis accuracy rates for media number 3

Accuracy Rate			
Examiner	**Precision**	**Recall**	**F-measure**
Examiner 3	n/a	n/a	n/a
Examiner 5	n/a	n/a	n/a
Examiner 4	n/a	n/a	n/a
Examiner 2	n/a	n/a	n/a
Examiner 1	n/a	n/a	n/a

- **Media Number 4**

Table Appxs.18 Results of a full examination on media number 4

Fully-Examined Case	
Suspect Objects	**Notes**
0	No evidence or trace evidence relevant to the investigation

Table Appxs.19 Results of preliminary analysis on media number 4 from five examiners

Examiner	Further Analysis	Suspect Objects	Notes
Examiner 5	Yes	45	Decision made based on found images
Examiner 1	No	0	n/a
Examiner 3	No	0	n/a
Examiner 4	No	0	Images found, but appear to be non-exploitation stock photos
Examiner 2	No	0	n/a

Table Appxs.20 Preliminary analysis object identification error rates for media number 4

Object Identification Error Rate				
Examiner	**Num. of False Positives**	**False Positive Error**	**Num. of False Negatives**	**False Negative Error**
Examiner 5	45	1	0	0
Examiner 1	0	0	0	0
Examiner 3	0	0	0	0
Examiner 4	0	0	0	0
Examiner 2	0	0	0	0

Table Appxs.21 Preliminary analysis accuracy rates for media number 4

Accuracy Rate			
Examiner	**Precision**	**Recall**	**F-measure**
Examiner 5	n/a	n/a	n/a
Examiner 1	n/a	n/a	n/a
Examiner 3	n/a	n/a	n/a
Examiner 4	n/a	n/a	n/a
Examiner 2	n/a	n/a	n/a

- **Media Number 5**

Table Appxs.22 Results of a full examination on media number 5

Fully-Examined Case	
Suspect Objects	**Notes**
182	More images appear to be one the machine but have yet to be categorized.

Table Appxs.23 Results of preliminary analysis on media number 5 from five examiners

Examiner	**Further Analysis**	**Suspect Objects**	**Notes**
Examiner 4	Yes	66	Decision made based on found images, keywords and encryption
Examiner 3	Yes	165	Decision made based on found images, movies, keywords, Real Player history, evidence of disk wiping tools, evidence of encryption tools
Examiner 2	Yes	96	Decision made based on found images, movies, encryption software, P2P, cleaner software
Examiner 5	Yes	31	Decision made based on found images and movies
Examiner 1	Yes	85	Decision made based on found images, movies

Table Appxs.24 Preliminary analysis object identification error rates for media number 5

Object Identification Error Rate				
Examiner	**Num. of False Positives**	**False Positive Error**	**Num. of False Negatives**	**False Negative Error**
Examiner 4	0	0	116	.64
Examiner 3	0	0	16	.09
Examiner 2	0	0	86	.47
Examiner 5	0	0	151	.83
Examiner 1	0	0	97	.53

Table Appxs.25 Preliminary analysis accuracy rates for media number 5

Accuracy Rate			
Examiner	**Precision**	**Recall**	**F-measure**
Examiner 4	1	.36	.53
Examiner 3	1	.91	.95
Examiner 2	1	.53	.69
Examiner 5	1	.17	.29
Examiner 1	1	.47	.64

Appendix C: Start and end time tests for the action instances "Starting Internet Explorer" and "Starting Firefox".

Image	Action	Time start	Time end	Time span	Seconds
XP 1713	IE	16:03:04	16:04:17	00:01:13	73
XP 1713	FF	13:54:52	13:55:39	00:00:47	47
Win7 1631	IE	09:41:49	09:42:43	00:00:54	54
Win7 1631	FF	14:11:59	14:12:15	00:00:16	16
Win7 1723	IE	12:13:39	12:14:20	00:00:41	41
Win7 1723	FF	15:10:21	15:11:04	00:00:43	43
XP 1525	IE	12:21:37	12:22:13	00:00:36	36
XP 1525	FF	16:36:19	16:36:48	00:00:29	29
XP 1528	IE	12:21:37	12:22:13	00:00:36	36
XP 1528	FF	16:06:09	16:06:12	00:00:03	3
XP 1529	IE	12:21:37	12:22:13	00:00:36	36
XP 1529	FF	16:04:00	16:04:03	00:00:03	3
XP 1536	IE	16:15:19	16:15:30	00:00:11	11
XP 1536	FF	14:33:10	14:33:48	00:00:38	38
XP 1540	IE	16:25:27	16:25:37	00:00:10	10
XP 1540	FF	16:30:33	16:30:41	00:00:08	8
XP 1542	IE	15:42:15	15:42:32	00:00:17	17
Xp 1542	FF	15:41:22	15:41:44	00:00:22	22
XP 1545	IE	12:38:03	12:38:16	00:00:13	13
XP 1545	FF	12:15:21	12:15:44	00:00:23	23
XP 1546	IE	12:21:37	12:22:13	00:00:36	36
XP 1546	FF	16:16:17	16:16:32	00:00:15	15
XP 1556	IE	15:37:28	15:38:04	00:00:36	36
XP 1556	FF	15:35:20	15:35:42	00:00:22	22
XP 1600	IE	14:38:41	14:38:58	00:00:17	17
XP 1600	FF	16:44:28	16:45:09	00:00:41	41
XP 1603	IE	12:21:37	12:22:13	00:00:36	36
XP 1603	FF	12:11:43	12:12:15	00:00:32	32
XP 1611	IE	16:47:11	16:47:19	00:00:08	8
XP 1611	FF	16:45:17	16:45:26	00:00:09	9
XP 1615	IE	12:02:17	12:02:28	00:00:11	11
XP 1615	FF	16:52:02	16:52:16	00:00:14	14
XP 1621	IE	12:21:37	12:22:13	00:00:36	36
XP 1621	FF	17:49:19	17:49:55	00:00:36	36
XP 1630	IE	16:27:19	16:27:40	00:00:21	21
XP 1630	FF	16:28:06	16:28:42	00:00:36	36
XP 1632	IE	17:29:15	17:29:25	00:00:10	10
XP 1632	FF	Not Ran			
XP 1644	IE	16:41:55	16:42:05	00:00:10	10
XP 1644	FF	16:43:16	16:43:45	00:00:29	29
XP 1648	IE	16:58:51	16:59:10	00:00:19	19
XP 1648	FF	14:52:28	14:52:43	00:00:15	15

XP 1650	IE	12:21:37	12:22:13	00:00:36	36
XP 1650	FF	11:36:25	11:36:49	00:00:24	24
XP 1653	IE	16:53:02	16:53:23	00:00:21	21
XP 1653	FF	16:51:10	16:51:41	00:00:31	31
XP 1658	IE	15:06:06	15:06:56	00:00:50	50
XP 1658	FF	11:19:38	11:20:17	00:00:39	39
XP 1703	IE	17:14:50	17:15:01	00:00:11	11
XP 1703	FF	17:54:32	17:54:45	00:00:13	13

Appendix D: Excerpt of objects updated by the action 'Open Internet Explorer' as reported by Process Monitor

```
HKLM\System\CurrentControlSet\Control\Nls\CustomLocale
HKLM\System\CurrentControlSet\Control\Nls\CustomLocale\en
-US
HKLM\System\CurrentControlSet\Control\Nls\CustomLocale
HKLM\System\CurrentControlSet\Control\Nls\ExtendedLocale
HKLM\System\CurrentControlSet\Control\Nls\ExtendedLocale
HKLM\System\CurrentControlSet\Control\Nls\ExtendedLocale\
en-US
HKLM\System\CurrentControlSet\Control\Nls\ExtendedLocale
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
ser Shell Folders
HKLM\Software\Policies\Microsoft\Windows\Explorer
HKCU\Software\Policies\Microsoft\Windows\Explorer
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-7B8E7F157091}
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\Category
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\Name
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\ParentFolder
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\Description
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\RelativePath
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\ParsingName
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\InfoTip
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\LocalizedName
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\Icon
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
```

7B8E7F157091}\Security
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\StreamResource
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\StreamResourceType
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\LocalRedirectOnly
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\Roamable
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\PreCreate
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\Stream
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\PublishExpandedPath
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\Attributes
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\FolderTypeID
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\InitFolderHandler
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-
7B8E7F157091}\PropertyBag
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{F1B32785-6FBA-4FCF-9D55-7B8E7F157091}
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\S
essionInfo\2
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\S
essionInfo\2\KnownFolders
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\S
essionInfo\2
HKCU
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
ser Shell Folders
HKCU
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
ser Shell Folders\Local AppData
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
ser Shell Folders
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-EA3317B67173}
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Category
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Name
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\ParentFolder
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Description
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\RelativePath
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\ParsingName
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\InfoTip
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\LocalizedName
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Icon
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Security
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\StreamResource
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\StreamResourceType
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\LocalRedirectOnly
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Roamable
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\PreCreate
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Stream

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\PublishExpandedPath
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\Attributes
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\FolderTypeID
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\InitFolderHandler
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-
EA3317B67173}\PropertyBag
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{5E6C858F-0E22-4760-9AFE-EA3317B67173}
C:\Program Files\Internet Explorer\profapi.dll
C:\Program Files\Internet Explorer\profapi.dll
C:\Windows\System32\profapi.dll
C:\Windows\System32\profapi.dll
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{3EB685DB-65F9-4CF6-A03A-
E3EF65729F3D}\InitFolderHandler
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{3EB685DB-65F9-4CF6-A03A-
E3EF65729F3D}\PropertyBag
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\F
olderDescriptions\{3EB685DB-65F9-4CF6-A03A-E3EF65729F3D}
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\S
essionInfo\2
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\S
essionInfo\2\KnownFolders
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\S
essionInfo\2
HKCU
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
ser Shell Folders
HKCU
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
ser Shell Folders\AppData
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
ser Shell Folders
C:\Users\Administrator\AppData\Roaming
C:\Users\Administrator\AppData\Roaming
C:\Users\Administrator\AppData\Roaming
C:\Users\Administrator\AppData\Roaming
C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\
Cookies
```

Appendix E: Excerpt of objects updated by the action 'Open Firefox' as reported by Process Monitor

```
HKLM\SOFTWARE\Microsoft\CTF\KnownClasses
HKLM\SOFTWARE\Microsoft\CTF\KnownClasses
HKLM\SOFTWARE\Microsoft\CTF\KnownClasses
HKLM\SOFTWARE\Microsoft\CTF\KnownClasses
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
serAssist\{CEBFF5CD-ACE2-4F4F-9178-
9926F41749EA}\Count\P:\Hfref\Nqzvavfgengbe\Qrfxgbc\Cebpzb
a.rkr
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
serAssist\{CEBFF5CD-ACE2-4F4F-9178-
9926F41749EA}\Count\P:\Hfref\Nqzvavfgengbe\Qrfxgbc\Cebpzb
a.rkr
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\U
serAssist\{CEBFF5CD-ACE2-4F4F-9178-
9926F41749EA}\Count\HRZR_PGYFRFFVBA
HKCU\Software\Classes
HKCU\Software\Classes\.lnk
HKCR\.lnk
HKCR\.lnk
HKCU\Software\Classes\.lnk
HKCR\.lnk\(Default)
HKCR\.lnk
HKCU\Software\Classes
HKCU\Software\Classes\.lnk\OpenWithProgids
HKCR\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Classes
HKCU\Software\Classes\.lnk
HKCR\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\UserChoice
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
```

```
ileExts\.lnk
HKCU\Software\Classes
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile\CurVer
HKCR\lnkfile\CurVer
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile\IsShortcut
HKCR\lnkfile
HKCU\Software\Classes\lnkfile\ShellEx\DataHandler
HKCR\lnkfile\ShellEx\DataHandler
HKCR\.lnk
HKCU\Software\Classes\.lnk\ShellEx\DataHandler
HKCR\.lnk\ShellEx\DataHandler
HKCU\Software\Classes
HKCU\Software\Classes\*
HKCR\*
HKCR\*
HKCU\Software\Classes\*\ShellEx\DataHandler
HKCR\*\ShellEx\DataHandler
HKCU\Software\Classes
HKCU\Software\Classes\AllFilesystemObjects
HKCR\AllFilesystemObjects
HKCR\AllFilesystemObjects
HKCU\Software\Classes\AllFilesystemObjects\ShellEx\DataHa
ndler
HKCR\AllFilesystemObjects\ShellEx\DataHandler
HKCR\AllFilesystemObjects
HKCR\*
HKCR\.lnk
HKCR\lnkfile
HKCU\Software\Classes
HKCU\Software\Classes\.lnk
HKCR\.lnk
HKCR\.lnk
HKCU\Software\Classes\.lnk
```

```
HKCR\.lnk\(Default)
HKCR\.lnk
HKCU\Software\Classes
HKCU\Software\Classes\.lnk\OpenWithProgids
HKCR\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\OpenWithProgids
HKCU\Software\Classes
HKCU\Software\Classes\.lnk
HKCR\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk\UserChoice
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\F
ileExts\.lnk
HKCU\Software\Classes
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile\CurVer
HKCR\lnkfile\CurVer
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile\IsShortcut
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCU\Software\Classes\lnkfile\shellex\ContextMenuHandlers
\{00021401-0000-0000-C000-000000000046}
```

HKCR\lnkfile\shellex\ContextMenuHandlers\{00021401-0000-0000-C000-000000000046}
HKCR\lnkfile\shellex\ContextMenuHandlers\{00021401-0000-0000-C000-000000000046}
HKCU\Software\Classes\lnkfile\shellex\ContextMenuHandlers\{00021401-0000-0000-C000-000000000046}
HKCR\lnkfile\shellex\ContextMenuHandlers\{00021401-0000-0000-C000-000000000046}\SuppressionPolicy
HKCR\lnkfile\shellex\ContextMenuHandlers\{00021401-0000-0000-C000-000000000046}
HKCR\lnkfile\shellex\ContextMenuHandlers
HKCR\lnkfile\shellex\ContextMenuHandlers
HKCR\AllFilesystemObjects
HKCU\Software\Classes\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCU\Software\Classes\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCU\Software\Classes\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu
HKCU\Software\Classes\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu\SuppressionPolicy
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu
HKCU\Software\Classes\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu\(Default)
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\CopyAsPathMenu
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers
HKCU\Software\Classes\AllFilesystemObjects\shellex\ContextMenuHandlers\SendTo
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\SendTo
HKCR\AllFilesystemObjects\shellex\ContextMenuHandlers\SendTo
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.lnk
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.lnk

```
HKCU\Software\Classes
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile\CurVer
HKCR\lnkfile\CurVer
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCR\lnkfile
HKCU\Software\Classes\lnkfile
HKCR\lnkfile\IsShortcut
HKCR\lnkfile
HKCU\Software\Classes\lnkfile\ShellEx\{0000000C-0000-
0000-C000-000000000046}
HKCR\lnkfile\ShellEx\{0000000C-0000-0000-C000-
000000000046}
HKCR\.lnk
HKCU\Software\Classes\.lnk\ShellEx\{0000000C-0000-0000-
C000-000000000046}
HKCR\.lnk\ShellEx\{0000000C-0000-0000-C000-000000000046}
HKCU\Software\Classes
HKCU\Software\Classes\*
HKCR\*
HKCR\*
HKCU\Software\Classes\*\ShellEx\{0000000C-0000-0000-C000-
000000000046}
HKCR\*\ShellEx\{0000000C-0000-0000-C000-000000000046}
HKCU\Software\Classes
HKCR\AllFilesystemObjects
HKCU\Software\Classes\AllFilesystemObjects\ShellEx\{00000
00C-0000-0000-C000-000000000046}
HKCR\AllFilesystemObjects\ShellEx\{0000000C-0000-0000-
C000-000000000046}
HKCR\AllFilesystemObjects
HKCR\*
HKCR\.lnk
HKCR\lnkfile
C:\Users\Public\Desktop\Mozilla Firefox.lnk
```

Appendix F: SigTest.pl Perl script to output object time stamp values given a list of file and Registry objects

```
#! c:\Perl\Bin\Perl.exe

use File::stat;
use Time::localtime;
use Switch;

      # Signature List - What files to get timestamps from
      # Generated from ProcMon - run more than once and combine to
remove noise
      $signatureList=glob("*.sig");
      # Saved Registry file after each run (manual)
      $rawReg="raw.txt";
      $convReg="reg.txt";

      # Read in the sig file and close
      open (FH, $signatureList) or die "Can't find signature file
'test.sig'! : $!\n";
      @sig_list=<FH>;
      close (FH);

      if ( -e $rawReg ) {
            if ( -e "utf16to8.pl" ) {
                  system("perl utf16to8.pl raw.txt > reg.txt");
                  if ( "$?" == "0" ) {
                        unlink( "raw.txt" );
                  } else {
                        print "Could not convert the registry
file!\n";
                  }
            }
      }

      # Slurp the reg file and close (make sure it has been converted
to UTF8)
      my $holdTerm = $/;
      undef $/;
      open $inf, "<" . $convReg;
      my $Registry = <$inf>;
      close $inf;
      $/ = $holdTerm;

      # Two files will be created with the file time stamps (fts) and
the reg timestamps (rts)
      open (fts, ">fts.xls") or die "Can not open fts output file. :
$!\n";
      open (rts, ">rts.xls") or die "Can not open rts output file! :
$!\n";

      # Counters
      $countregistry = 0;
      $countfiles = 0;
      $counttotal = 0;

      # Create the xls headders
      print fts "name\taccessed time\tmodified time\tcreated time\n";
      print rts "name\tmodified date\tmodified time\n";

      foreach $line (@sig_list) {
```

```
            $line =~ s/^"//;
            $line =~ s/"$//;
            $line =~ s/[\r\n]//g;
            if (-f $line && ($line =~ m/.:*/)) {
                  $st = stat($line);
                  if (!$@) {
                        $astring = ctime($st->atime);
                        $mstring = ctime($st->mtime);
                        $cstring = ctime($st->ctime);
                        #print
"$line\t$astring\t$mstring\t$cstring\n";
                        print fts
"$line\t$astring\t$mstring\t$cstring\n";
                        $countfiles++
                  }

            } elsif     (($line =~ m/HK[CLU][RUM]/) || ($line =~
m/^\\/)) {
                  switch ($line) {
                        case (m/HKCR/) { $line =~
s/HKCR/HKEY_CLASSES_ROOT/ }
                        case (m/HKCU/) { $line =~
s/HKCU/HKEY_CURRENT_USER/ }
                        case (m/HKLM/) { $line =~
s/HKLM/HKEY_LOCAL_MACHINE/ }
                        case (m/HKU/) { $line =~ s/HKU/HKEY_USERS/ }
                  }

                  if ( $Registry =~ m/(\Q$line\E)\nClass
Name:.*\nLast Write Time:\s{3}(\d+\/\d+\/\d+)\s-\s(\d+:\d+\s[AP]M)/ )
{
                        #print "$1 Date: $2 Time: $3\n";
                        print rts "$line\t$2\t$3\n";
                        $countregistry++;
                  } else { #print "$line\n"

                  };
            }
            $counttotal++
      }
      close (fts);
      close (rts);

      $addtotal = $countfiles + $countregistry;
      print "$countfiles files and $countregistry registry ($addtotal
total) processed out of $counttotal total.\n
```

Appendix G: Excerpt of file timestamp update data for the action Open Internet Explorer where gray is the first run and blue is the second run.

name	accessed	modified	created
C:\Documents and Settings\Administrator\ntuser.dat.LOG	Fri Feb 12 10:16:57 2010	Fri Feb 12 10:16:57 2010	
C:\WINDOWS\system32\config\software.LOG	Fri Feb 12 10:16:57 2010	Fri Feb 12 10:16:57 2010	
C:\WINDOWS\system32\shell32.dll	Fri Feb 12 10:16:03 2010		
C:\WINDOWS\Prefetch\IEXPLORE.EXE-27122324.pf	Fri Feb 12 10:14:56 2010	Fri Feb 12 10:14:56 2010	
C:\WINDOWS\system32\ieapfltr.dat	Fri Feb 12 10:14:52 2010		Mon Feb 12 16:10:12 2007
C:\Documents and Settings\Administrator\Cookies\administrator@live[1].txt	Fri Feb 12 10:14:51 2010	Fri Feb 12 10:14:51 2010	Fri Feb 12 10:14:51 2010
C:\Documents and Settings\Administrator\Cookies\administrator@msn[1].txt	Fri Feb 12 10:14:51 2010	Fri Feb 12 10:14:51 2010	Fri Feb 12 10:14:51 2010
C:\Documents and Settings\Administrator\ntuser.dat.LOG	Fri Feb 12 16:44:05 2010	Fri Feb 12 16:44:05 2010	
C:\WINDOWS\system32\config\software.LOG	Fri Feb 12 16:44:05 2010	Fri Feb 12 16:44:05 2010	
C:\WINDOWS\Prefetch\IEXPLORE.EXE-27122324.pf	Fri Feb 12 16:42:15 2010	Fri Feb 12 16:42:15 2010	
C:\Documents and Settings\Administrator\Cookies\administrator@live[1].txt	Fri Feb 12 16:42:09 2010	Fri Feb 12 16:42:09 2010	Fri Feb 12 10:14:51 2010
C:\Documents and Settings\Administrator\Cookies\administrator@msn[1].txt	Fri Feb 12 16:42:09 2010	Fri Feb 12 16:42:09 2010	Fri Feb 12 10:14:51 2010

Appendix H: List of category 3 traces that are irregularly updated during the execution of Internet Explorer 8

C:\Documents and Settings\Administrator\Cookies*,

C:\WINDOWS\system32\ieapfltr.dat,

C:\Documents and Settings\Administrator\Application Data\Microsoft\IdentityCRL\production\ppcrlconfig.dll,

C:\Documents and Settings\All Users\Application Data\Microsoft\IdentityCRL\production\ppcrlconfig.dll,

C:\Documents and Settings\Administrator\Application Data\Microsoft\CryptnetUrlCache\Content\7B2238AACCEDC3F1FFE8E7EB5F575EC9,

C:\Documents and Settings\Administrator\Application Data\Microsoft\CryptnetUrlCache\MetaData\7B2238AACCEDC3F1FFE8E7EB5F575EC9,

C:\WINDOWS\system32\xmllite.dll,

C:\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Internet Explorer\frameiconcache.dat,

C:\Documents and Settings\Administrator\Favorites\Links\desktop.ini,

C:\Documents and Settings\Administrator\Favorites\Desktop.ini,

C:\WINDOWS\system32\winhttp.dll,

C:\Program Files\Common Files\Microsoft Shared\Windows Live\WindowsLiveLogin.dll,

C:\Program Files\Common Files\Microsoft Shared\Windows Live\msidcrl40.dll

C:\WINDOWS\system32\ieui.dll

C:\WINDOWS\system32\msls31.dll

C:\WINDOWS\system32\ieapfltr.dll

C:\Program Files\Internet Explorer\xpshims.dll

C:\WINDOWS\system32\mshtml.dll

C:\WINDOWS\system32\msfeeds.dll

C:\WINDOWS\system32\activeds.dll

C:\WINDOWS\system32\adsldpc.dll

C:\WINDOWS\system32\credui.dll

C:\WINDOWS\system32\cryptnet.dll

C:\WINDOWS\system32\cscdll.dll

C:\WINDOWS\system32\cscui.dll

C:\WINDOWS\system32\dhcpcsvc.dll

C:\WINDOWS\system32\dot3api.dll

C:\WINDOWS\system32\dot3dlg.dll

C:\WINDOWS\system32\eapolqec.dll

C:\WINDOWS\system32\eappcfg.dll

C:\WINDOWS\system32\eappprxy.dll

C:\WINDOWS\system32\esent.dll

C:\WINDOWS\system32\mprapi.dll

C:\WINDOWS\system32\msxml3r.dll

C:\WINDOWS\system32\netman.dll

C:\WINDOWS\system32\netshell.dll

C:\WINDOWS\system32\onex.dll

C:\WINDOWS\system32\psapi.dll

C:\WINDOWS\system32\qutil.dll

C:\WINDOWS\system32\rasadhlp.dll

C:\WINDOWS\system32\rsaenh.dll

C:\WINDOWS\system32\winlogon.exe

C:\WINDOWS\system32\winrnr.dll

C:\WINDOWS\system32\wintrust.dll

C:\WINDOWS\system32\wmi.dll

C:\WINDOWS\system32\wtsapi32.dll

C:\WINDOWS\system32\wzcsapi.dll

C:\WINDOWS\system32\wzcsvc.dll

C:\Program Files\Messenger\msmsgs.exe

C:\WINDOWS\system32\mswsock.dll

C:\WINDOWS\system32\msxml3.dll

C:\WINDOWS\system32\atl.dll

C:\Program Files\Internet Explorer\sqmapi.dll

C:\WINDOWS\system32\schannel.dll

C:\WINDOWS\AppPatch\aclayers.dll

C:\WINDOWS\system32\urlmon.dll

C:\Program Files\Internet Explorer\ieproxy.dll

C:\WINDOWS\system32\iertutil.dll

C:\WINDOWS\system32\ieframe.dll

C:\WINDOWS\system32\actxprxy.dll

C:\WINDOWS\system32\apphelp.dll

C:\WINDOWS\system32\crypt32.dll

C:\WINDOWS\system32\cryptdll.dll

C:\WINDOWS\system32\digest.dll

C:\WINDOWS\system32\iphlpapi.dll

C:\WINDOWS\system32\ir32_32.dll

C:\WINDOWS\system32\ir41_32.ax

C:\WINDOWS\system32\ir41_qc.dll

C:\WINDOWS\system32\ir41_qcx.dll

C:\WINDOWS\system32\ir50_32.dll

C:\WINDOWS\system32\ir50_qc.dll

C:\WINDOWS\system32\ir50_qcx.dll

C:\WINDOWS\system32\mlang.dll

C:\WINDOWS\system32\msapsspc.dll

C:\WINDOWS\system32\msisip.dll

C:\WINDOWS\system32\msnsspc.dll

C:\WINDOWS\system32\msvcrt40.dll

C:\WINDOWS\system32\rasapi32.dll

C:\WINDOWS\system32\rasman.dll

C:\WINDOWS\system32\rtutils.dll

C:\WINDOWS\system32\sensapi.dll

C:\WINDOWS\system32\setupapi.dll

C:\WINDOWS\system32\sxs.dll

C:\WINDOWS\system32\tapi32.dll

C:\WINDOWS\system32\winspool.drv

C:\WINDOWS\system32\ws2_32.dll

C:\WINDOWS\system32\ws2help.dll

C:\WINDOWS\system32\xpsp2res.dll

C:\WINDOWS\system32\msv1_0.dll

C:\WINDOWS\system32\msasn1.dll

C:\WINDOWS\system32\wshext.dll

C:\WINDOWS\system32\dnsapi.dll

C:\Documents and Settings\Administrator\Cookies\administrator@live[1].txt

C:\Documents and Settings\Administrator\Cookies\administrator@msn[1].txt

Appendix I: Returned objects discovered through the use of generic signature trace update clustering.

```
2010 10 18 Mon 17:13:21,..c.,/Program Files/Internet
Explorer/iexplore.exe
2010 10 18 Mon 17:13:21,..c.,/WINDOWS/system32/url.dll
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/user/Application
Data/Microsoft/SystemCertificates
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/user/Application
Data/Microsoft/SystemCertificates/My
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/user/Application
Data/Microsoft/SystemCertificates/My/CTLs
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/user/Application
Data/Microsoft/SystemCertificates/My/CRLs
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/user/Application
Data/Microsoft/SystemCertificates/My/Certificates
2010 10 18 Mon 17:13:22,mac.,/WINDOWS/Prefetch
2010 10 18 Mon 17:13:22,.a..,/WINDOWS/system32/ras
2010 10 18 Mon 17:13:22,macb,/WINDOWS/Prefetch/RUNDLL32.EXE-
211063BE.pf
2010 10 18 Mon 17:13:22,macb,/Documents and
Settings/user/Cookies/user@msn[3].txt (deleted)
2010 10 18 Mon 17:13:22,macb,/Documents and
Settings/user/Cookies/user@zune[1].txt (deleted)
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/All
Users/Application Data/Microsoft/Network/Connections
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/All
Users/Application Data/Microsoft/Network/Connections/Pbk
2010 10 18 Mon 17:13:22,macb,/Documents and
Settings/user/Cookies/user@windowsmarketplace[1].txt (deleted)
2010 10 18 Mon 17:13:22,macb,/WINDOWS/system32/wbem/fastprox.dll
(deleted)
2010 10 18 Mon 17:13:22,macb,/Documents and
Settings/user/Cookies/user@bing[1].txt (deleted)
2010 10 18 Mon 17:13:22,macb,/WINDOWS/system32/wbem/framedyn.dll
(deleted)
2010 10 18 Mon 17:13:22,macb,/Documents and
Settings/user/Cookies/user@atdmt[2].txt (deleted)
2010 10 18 Mon 17:13:22,macb,/WINDOWS/system32/wbem/krnlprov.dll
(deleted)
2010 10 18 Mon 17:13:22,macb,/Documents and
Settings/user/Cookies/user@live[3].txt (deleted)
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Office
2010 10 18 Mon 17:13:22,.a..,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Office/Groove
2010 10 18 Mon 17:13:25,...b,/$OrphanFiles/Repository/..
2010 10 18 Mon 17:13:25,...b,/Documents and Settings/user/Local
Settings/Temporary Internet
Files/Content.IE5/WL2XKB0R/google_ie[1].txt (deleted)
2010 10 18 Mon 17:13:25,macb,/Documents and
Settings/user/Cookies/user@google[2].txt (deleted)
2010 10 18 Mon 17:13:25,...b,/Documents and
Settings/user/Cookies/user@google[1].txt (deleted)
2010 10 18 Mon 17:13:26,mac.,/$OrphanFiles/Repository/..
2010 10 18 Mon 17:13:26,mac.,/Documents and Settings/user/Local
Settings/Temporary Internet
Files/Content.IE5/WL2XKB0R/google_ie[1].txt (deleted)
2010 10 18 Mon 17:13:26,macb,/Documents and Settings/user/Local
Settings/Temporary Internet
Files/Content.IE5/S9OHK3S1/26b57add41fbd610[1].js (deleted)
```

```
2010 10 18 Mon 17:13:26,macb,/Documents and Settings/user/Local
Settings/Temporary Internet
Files/Content.IE5/S9OHK3S1/nav_logo14[1].png (deleted)
2010 10 18 Mon 17:13:26,macb,/WINDOWS/system32/wbem/wbemcons.dll
(deleted)
2010 10 18 Mon 17:13:26,macb,/Documents and Settings/user/Local
Settings/Temporary Internet Files/Content.IE5/S9OHK3S1/logo1w[1].png
(deleted)
2010 10 18 Mon 17:13:26,macb,/WINDOWS/system32/wbem/wmipsess.dll
(deleted)
2010 10 18 Mon 17:13:26,...b,/Documents and Settings/user/Local
Settings/History/History.IE5/MSHist012010101820101019 (deleted)
2010 10 18 Mon 17:13:26,...b,/Documents and Settings/user/Local
Settings/History/History.IE5/MSHist012010101820101019/index.dat
(deleted)
2010 10 18 Mon 17:13:26,macb,/Documents and Settings/user/Local
Settings/Temporary Internet
Files/Content.IE5/WL2XKB0R/kYdwQLTp1rU[1].js (deleted)
2010 10 18 Mon 17:13:26,macb,/Documents and Settings/user/Local
Settings/Temporary Internet Files/Content.IE5/WL2XKB0R/favicon[2].ico
(deleted)
2010 10 18 Mon 17:13:26,mac.,/Documents and
Settings/user/Cookies/user@google[1].txt (deleted)
2010 10 18 Mon 17:13:31,mac.,/WINDOWS/Prefetch/IEXPLORE.EXE-
27122324.pf
2010 10 18 Mon 17:13:31,macb,/Documents and Settings/user/Local
Settings/Temp/~DF10EA.tmp (deleted)
2010 10 18 Mon 17:13:31,macb,/WINDOWS/system32/xolehlp.dll (deleted)
2010 10 18 Mon 17:13:31,.a.b,/Documents and Settings/user/Local
Settings/Temp/~DF11A9.tmp (deleted)
2010 10 18 Mon 17:13:31,macb,/Documents and Settings/user/Local
Settings/Temp/~DF11CC.tmp (deleted)
2010 10 18 Mon 17:13:31,.a.b,/Documents and Settings/user/Local
Settings/Temp/~DF1245.tmp (deleted)
2010 10 18 Mon 17:13:31,.a.b,/WINDOWS/system32/mtxoci.dll (deleted)
2010 10 18 Mon 17:13:31,macb,/Documents and Settings/user/Local
Settings/Temp/~DF1268.tmp (deleted)
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/Fonts/desktop.ini
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/Tasks/desktop.ini
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/desktop.ini
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/Downloaded Program Files
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/Downloaded Program
Files/desktop.ini
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/Offline Web Pages
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/Offline Web Pages/desktop.ini
2010 10 18 Mon 17:13:32,.a..,/WINDOWS/assembly/Desktop.ini
2010 10 18 Mon 17:13:35,mac.,/WINDOWS/Debug/UserMode/userenv.log
2010 10 18 Mon 17:13:35,.a..,/WINDOWS/$NtUninstallKB951376-v2$
2010 10 18 Mon 17:13:36,.a..,/WINDOWS/system32/mscoree.dll
2010 10 18 Mon 17:13:36,mac.,/WINDOWS/Prefetch/VERCLSID.EXE-
3667BD89.pf
2010 10 18 Mon 17:13:36,.a..,/WINDOWS/system32/en-us/occache.dll.mui
2010 10 18 Mon
17:13:36,.a..,/WINDOWS/Microsoft.NET/Framework/v2.0.50727/shfusion.dl
l
2010 10 18 Mon
17:13:36,.a..,/WINDOWS/Microsoft.NET/Framework/v2.0.50727/Culture.dll
2010 10 18 Mon 17:13:36,.a..,/WINDOWS/system32/occache.dll
2010 10 18 Mon
17:13:36,.a..,/WINDOWS/Microsoft.NET/Framework/v2.0.50727/ShFusRes.dl
l
```

```
2010 10 18 Mon
17:13:36,.a..,/WINDOWS/Microsoft.NET/Framework/v2.0.50727/fusion.dll
2010 10 18 Mon 17:13:37,.a..,/WINDOWS/system32/webcheck.dll
2010 10 18 Mon 17:13:37,.a..,/WINDOWS/system32/shell32.dll
2010 10 18 Mon 17:13:44,..c.,/WINDOWS/system32/shell32.dll
2010 10 18 Mon 17:13:45,mac.,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Windows/UsrClass.dat.LOG
2010 10 18 Mon 17:13:57,macb,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Internet
Explorer/frameiconcache.dat (deleted)
2010 10 18 Mon 17:13:57,m.c.,/Documents and Settings/user/Local
Settings/Temp/~DF11A9.tmp (deleted)
2010 10 18 Mon 17:13:57,m.c.,/Documents and Settings/user/Local
Settings/Temp/~DF1245.tmp (deleted)
2010 10 18 Mon 17:13:57,m.c.,/WINDOWS/system32/mtxoci.dll (deleted)
2010 10 18 Mon 17:13:57,macb,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Internet Explorer/Recovery/Last
Active/RecoveryStore.{B58C96A2-DAD2-11DF-8CD1-0019B95246E6}.dat
(deleted)
2010 10 18 Mon 17:13:57,macb,/Documents and Settings/user/Local
Settings/Temp/~DF3CF6.tmp (deleted)
2010 10 18 Mon 17:13:57,macb,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Internet Explorer/Recovery/Last
Active/{B58C96A3-DAD2-11DF-8CD1-0019B95246E6}.dat (deleted)
2010 10 18 Mon 17:13:57,macb,/Documents and Settings/user/Local
Settings/Temp/~DF3D82.tmp (deleted)
2010 10 18 Mon 17:13:58,m...,/Documents and Settings/user/Local
Settings/Temporary Internet Files/Content.IE5/index.dat
2010 10 18 Mon 17:13:58,m...,/Documents and Settings/user/Local
Settings/History/History.IE5/index.dat
2010 10 18 Mon 17:13:58,m...,/Documents and
Settings/user/IECompatCache/index.dat
2010 10 18 Mon 17:13:58,m...,/Documents and
Settings/user/Cookies/index.dat
2010 10 18 Mon 17:13:58,m...,/Documents and
Settings/user/UserData/index.dat
2010 10 18 Mon 17:13:58,m.c.,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Internet Explorer/Recovery/Active
2010 10 18 Mon 17:13:58,.a..,/Documents and Settings/user/PrivacIE
2010 10 18 Mon 17:13:58,m...,/Documents and
Settings/user/PrivacIE/index.dat
2010 10 18 Mon 17:13:58,m...,/Documents and
Settings/user/IETldCache/index.dat
2010 10 18 Mon 17:13:58,m...,/Documents and Settings/user/Local
Settings/Application Data/Microsoft/Feeds Cache/index.dat
2010 10 18 Mon 17:13:58,m...,/Documents and Settings/user/Local
Settings/History/History.IE5/MSHist012010101820101019/index.dat
(deleted)
```

Appendix J: Results of searching for all objects updated on a suspect system at time "2008 03 07 Fri 11:35".

```
cat mactime.out | grep "2008 03 07 Fri 11:35" | awk -F, '{print $NF}'
C:/WINDOWS/msagent/agentctl.dll
C:/WINDOWS/system32/devenum.dll
C:/WINDOWS/system32/ksuser.dll
C:/WINDOWS/system32/ipconf.tsp
C:/WINDOWS/system32/kmddsp.tsp
C:/WINDOWS/system32/ndptsp.tsp
C:/WINDOWS/system32/rastapi.dll
C:/WINDOWS/system32/unimdm.tsp
C:/WINDOWS/system32/uniplat.dll
C:/WINDOWS/system32/hid.dll
C:/WINDOWS/system32/hidphone.tsp
C:/WINDOWS/system32/ntlsapi.dll
C:/WINDOWS/system32/rasppp.dll
C:/Documents and Settings/Yuandong/Local Settings/Application
Data/Microsoft/Windows Media/9.0
C:/Documents and Settings/Yuandong/Local Settings/Application
Data/Microsoft/Windows Media/9.0/WMSDKNS.XML
C:/Documents and Settings/Yuandong/Local Settings/Application
Data/Microsoft/Windows Media/9.0/WMSDKNS.DTD
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EP23UTY5/bg_b[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/GVK5INUX/hptg[2].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/GVK5INUX/ovr13[2].css
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/ETKB6XYL/blu[2].css
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/GVK5INUX/ushp[2].css
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EP23UTY5/ie[2].css
C:/Documents and Settings/Yuandong/Cookies/yuandong@c.msn[1].txt
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EP23UTY5/312070BCE01CB4C36B8984D6858B1[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EP23UTY5/92947F101AA77A8129E31215C8033[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/ETKB6XYL/glow_b[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EP23UTY5/pipe[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/ETKB6XYL/primedns[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EDG5KXAD/73EA3A497EB807310219A1C4D1E9E[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/ETKB6XYL/528E19AA57C59BD28F9241C1469F1[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EP23UTY5/WL_b[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EDG5KXAD/37276FB1F39BE69AD3AA1528087[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/EDG5KXAD/search[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/ETKB6XYL/CF3D42982D1982D557E124F8BA983D[1].jpg
C:/Documents and
Settings/Yuandong/Cookies/yuandong@msnportal.112.2o7[1].txt
```

C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/11[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/4C88F0DFE17E3BCF32326190C1CE0[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/hpb[2].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/FDCAAC85D66BE7CB4D71155977E9CC[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/57627C8031578A8F7E5D628C671AE5[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/bullet[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/msn_b2[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/9[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/84B2F5CD2F23E45C9AF89D1265209B[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/msft[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/B025C5D96DAF8A8961143BDE5511CC[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/494C38D81F69D4863B7240AAC3D439[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/8153BBDB41659C3502E18DEA4C9B7[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/F2D98818F41BDC36847174FFD7785[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/F81ABC39FE7CB357D2E8993DC5975[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/ieminwidth[2].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/4E381C752EC4BDE97298CD86591073[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/0000000001_000000000000000506109[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/buttons2[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/A177725C178F4C6B9E88CC927C9C0[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/pp[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/mail[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/31891541001[1].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/free_sim_300x250_2[1].swf
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/windowsupdate.microsoft[1].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/redirect[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/redirect[1].js
C:/Documents and Settings/Yuandong/UserData
C:/Documents and Settings/Yuandong/UserData/index.dat
C:/Documents and Settings/Yuandong/UserData/JSC3D5O5
C:/Documents and Settings/Yuandong/UserData/QTRSXO3U
C:/Documents and Settings/Yuandong/UserData/BNX3FDGW
C:/Documents and Settings/Yuandong/UserData/0FFR6GTH

C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/default[2].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/tgar[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/redirect[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/commontop[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/webcomtop[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/webcomtop[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/spupdateids[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/resultslist[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/resultslist[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/toc[1].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/mstoolbar[1].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/footer[1].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/tgar[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/tgar[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/toc[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/windows_masthead_ltr[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/content[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/hcp[1].css
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/content[1].css
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/arrow[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/update_webtrends[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/blank[1].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/errorinformation[1].htm
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/tgar[2].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/banner-right[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/content[1].js
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/GVK5INUX/information[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EP23UTY5/warning[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/EDG5KXAD/banner-bg[1].jpg
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet Files/Content.IE5/ETKB6XYL/CAX07EF1.htm

```
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/GVK5INUX/trans_pixel[1].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/GVK5INUX/trans_pixel[2].gif
C:/Documents and Settings/Yuandong/Local Settings/Temporary Internet
Files/Content.IE5/GVK5INUX/trans_pixel[1].gif
```

Appendix K: Results of searching for groups of objects by path known to correlate to a specific action “Internet-related activity”, rounded by minute.

```
cat mactime.out | grep -i "Temporary Internet Files" | awk -F,
'{print $1}' | cut -b -20 | sort | uniq
2008 03 05 Wed 11:42                2008 09 22 Mon 09:32
2008 03 05 Wed 12:11                2008 09 22 Mon 09:33
2008 03 05 Wed 12:12                2008 09 22 Mon 09:34
2008 03 05 Wed 12:13                2008 09 22 Mon 09:38
2008 03 05 Wed 12:15                2008 09 22 Mon 09:50
2008 03 05 Wed 12:16                2008 09 22 Mon 09:58
2008 03 05 Wed 22:50                2008 09 22 Mon 09:59
2008 03 07 Fri 11:35                2008 09 22 Mon 10:11
2008 03 07 Fri 11:37                2008 09 22 Mon 10:12
2008 03 07 Fri 11:38                2008 09 22 Mon 10:14
2008 03 07 Fri 11:39                2008 09 22 Mon 10:15
2008 03 07 Fri 11:40                2008 09 22 Mon 10:16
2008 03 07 Fri 12:20                2008 09 22 Mon 10:58
2008 06 23 Mon 10:53                2008 09 22 Mon 10:59
2008 06 23 Mon 15:54                2008 09 22 Mon 11:36
2008 06 23 Mon 17:11                2008 09 22 Mon 12:40
2008 06 23 Mon 17:12                2008 09 22 Mon 12:58
2008 06 25 Wed 05:24                2008 09 22 Mon 12:59
2008 06 26 Thu 09:00                2008 09 23 Tue 10:33
2008 06 26 Thu 09:15                2008 09 23 Tue 10:34
2008 07 03 Thu 10:14                2008 09 23 Tue 10:35
2008 09 17 Wed 12:41                2008 09 23 Tue 10:36
2008 09 17 Wed 12:42                2008 09 23 Tue 10:37
2008 09 17 Wed 12:43                2008 09 23 Tue 10:38
2008 09 17 Wed 12:44                2008 09 23 Tue 10:39
2008 09 17 Wed 12:45                2008 09 23 Tue 10:40
2008 09 17 Wed 12:46                2008 09 23 Tue 10:41
2008 09 17 Wed 12:47                2008 09 23 Tue 10:42
2008 09 17 Wed 16:48                2008 09 23 Tue 10:43
2008 09 17 Wed 16:49                2008 09 23 Tue 10:47
2008 09 18 Thu 10:32                2008 09 23 Tue 10:48
2008 09 18 Thu 10:58                2008 09 23 Tue 10:50
2008 09 18 Thu 11:28                2008 09 23 Tue 10:52
2008 09 19 Fri 10:07                2008 09 23 Tue 10:55
2008 09 19 Fri 10:08                2008 09 23 Tue 10:56
2008 09 19 Fri 10:09                2008 09 23 Tue 10:57
2008 09 19 Fri 10:10                2008 09 23 Tue 10:58
2008 09 19 Fri 10:16                2008 09 23 Tue 11:05
2008 09 19 Fri 10:19                2008 09 23 Tue 15:33
2008 09 19 Fri 10:37                2008 09 23 Tue 15:34
2008 09 19 Fri 11:00                2008 09 23 Tue 15:39
2008 09 19 Fri 16:35                2008 09 23 Tue 15:40
2008 09 19 Fri 16:40                2008 09 23 Tue 15:41
2008 09 19 Fri 16:45                2008 09 23 Tue 15:42
2008 09 21 Sun 11:09                2008 09 23 Tue 15:43
2008 09 21 Sun 11:10                2008 09 23 Tue 15:44
2008 09 21 Sun 11:12                2008 09 23 Tue 15:45
2008 09 21 Sun 11:13                2008 09 23 Tue 15:46
2008 09 21 Sun 11:14                2008 09 23 Tue 15:47
2008 09 21 Sun 11:15                2008 09 23 Tue 15:48
2008 09 21 Sun 11:16                2008 09 23 Tue 15:49
2008 09 21 Sun 11:20                2008 09 23 Tue 15:51
2008 09 21 Sun 11:21                2008 09 23 Tue 15:52
2008 09 21 Sun 12:23                2008 09 23 Tue 19:22
2008 09 21 Sun 12:24                2008 09 23 Tue 19:23
2008 09 21 Sun 12:25                2008 09 24 Wed 10:11
```

```
2008 09 24 Wed 10:12
2008 09 24 Wed 10:13
2008 09 24 Wed 10:14
2008 09 24 Wed 10:33
2008 09 24 Wed 11:02
2008 09 24 Wed 11:03
2008 09 24 Wed 13:02
2008 09 24 Wed 13:03
2008 09 24 Wed 19:09
2008 09 24 Wed 19:35
2008 09 24 Wed 19:36
2008 09 24 Wed 19:37
2008 09 24 Wed 19:39
2008 09 25 Thu 16:49
2008 09 25 Thu 16:50
2008 09 25 Thu 17:13
2008 09 25 Thu 17:14
2008 09 25 Thu 17:15
2008 09 25 Thu 17:16
2008 09 25 Thu 17:17
2008 09 25 Thu 17:18
2008 09 28 Sun 21:22
2008 09 29 Mon 11:28
2008 09 29 Mon 12:19
2008 10 04 Sat 19:27
2008 10 07 Tue 19:01
2008 10 07 Tue 19:07
2008 10 07 Tue 19:45
2008 10 13 Mon 10:12
2008 10 13 Mon 10:13
2008 10 13 Mon 10:23
2008 10 13 Mon 10:28
2008 10 26 Sun 10:16
2008 10 26 Sun 10:17
2008 10 26 Sun 10:21
2008 10 26 Sun 10:34
2008 10 26 Sun 10:43
2008 10 26 Sun 10:44
2008 10 26 Sun 10:48
2008 10 26 Sun 11:03
2008 11 08 Sat 20:49
2008 11 28 Fri 10:44
2008 11 28 Fri 10:46
2008 12 03 Wed 11:15
2008 12 03 Wed 11:22
2008 12 03 Wed 11:23
2008 12 03 Wed 11:25
2008 12 03 Wed 11:26
2008 12 03 Wed 11:32
2008 12 03 Wed 12:24
```

Appendix L: Commonly occurring words generically detected at times when a program was known to execute.

Computer 1

Time	User Action	Commonly Occurring Words	Process
07/19/2011 22:04:43	Close	1 3 C 2 2 VOLUME 3 2 SYSTEM 4 2 RP191 5 2 RESTORE 6 2 MY 7 2 INI 8 2 INFORMATION 9 2 F871D828 10 2 DOCUMENTS	firefox.exe
07/19/2011 23:22:51	Open	None	firefox.exe
07/19/2011 23:41:58	Open	1 2 SETTINGS 2 1 QUICKTIME 3 1 LOCAL 4 1 DOWNLOADS 5 1 DOCUMENTS 6 1 *USERNAME* 7 1 DATA 8 1 COMPUTER 9 1 C 10 1 APPLICATION	firefox.exe
07/19/2011 23:41:59	Close	" "	firefox.exe
07/20/2011 01:21:42	Close	1 7 VOLUME 2 7 SYSTEM 3 7 RP192 4 7 RESTORE 5 7 INFORMATION 6 7 F871D828 7 7 C 8 7 A148 9 7 969FDFB59BDE 10 7 4810	firefox.exe
07/20/2011 11:55:09	Open	1 173 C 2 154 VOLUME 3 154 SYSTEM 4 154 RP192 5 154 RESTORE 6 154 INFORMATION 7 154 F871D828 8 154 A148 9 154 969FDFB59BDE 10 154 4810	firefox.exe
07/20/2011 13:49:57	Close	1 13 LNK 2 13 C 3 12 VOLUME	firefox.exe

		4 12 SYSTEM 5 12 RESTORE 6 12 INFORMATION 7 12 F871D828 8 12 A148 9 12 969FDFB59BDE 10 12 4810	
07/20/2011 15:10:33	Open	1 3 C 2 2 VOLUME 3 2 SYSTEM 4 2 RP192 5 2 RESTORE 6 2 LNK 7 2 INFORMATION 8 2 F871D828 9 2 A148 10 2 969FDFB59BDE	firefox.exe
07/20/2011 16:19:40	Open	1 4 MY 2 4 DOCUMENTS 3 3 NA 4 2 SETTINGS 5 2 MUSIC 6 2 ITUNES 7 2 *USERNAME* 8 2 C 9 2 AND 10 1 RIGHT	firefox.exe
07/20/2011 16:19:41	Close	“ “	firefox.exe
07/20/2011 20:23:52	Close	1 4 C 2 3 VOLUME 3 3 SYSTEM 4 3 RESTORE 5 3 LNK 6 3 INFORMATION 7 3 F871D828 8 3 A148 9 3 969FDFB59BDE 10 3 4810	firefox.exe
07/20/2011 22:07:06	Open	1 35 SETTINGS 2 34 DOCUMENTS 3 34 *USERNAME* 4 34 C 5 34 AND 6 33 OFFICE07ENTUCD 7 33 DESKTOP 8 23 EN 9 22 US 10 10 ADMIN	firefox.exe
07/21/2011 01:24:09	Close	1 7 C 2 5 VOLUME 3 5 SYSTEM 4 5 RESTORE 5 5 INFORMATION 6 5 F871D828	firefox.exe

		7 5 A148 8 5 969FDFB59BDE 9 5 4810 10 5 4649	
07/21/2011 11:29:01	Open	None	firefox.exe
07/21/2011 11:31:39	Close	1 45 C 2 27 FILES 3 26 PROGRAM 4 26 MOZILLA 5 26 FIREFOX 6 11 DLL 7 9 VOLUME 8 9 SYSTEM 9 9 RP193 10 9 RESTORE	firefox.exe
07/21/2011 19:29:30	Open	1 26 C 2 17 SETTINGS 3 15 WINDOWS 4 15 MSIL 5 15 GAC 6 15 ASSEMBLY 7 14 GRAPHICS 8 14 CLI 9 14 ASPECT 10 9 SHARED	firefox.exe
07/22/2011 02:57:05	Close	None	firefox.exe
07/22/2011 11:35:29	Open	None	firefox.exe
07/23/2011 03:07:32	Close	1 1 SQMNOOPT08 2 1 SQM 3 1 SETTINGS 4 1 MSN 5 1 MICROSOFT 6 1 MESSENGER 7 1 DOCUMENTS 8 1 *USERNAME* 9 1 DATA 10 1 C	firefox.exe
07/23/2011 11:36:13	Open	1 10 SETTINGS 2 6 C 3 5 PROFILES 4 5 MOZILLA 5 5 LOCAL 6 5 FIREFOX 7 5 DOCUMENTS 8 5 DEFAULT 9 5 *USERNAME* 10 5 DATA	firefox.exe
07/24/2011 02:26:40	Close	1 15 SETTINGS 2 13 C 3 11 DATA 4 11 APPLICATION 5 10 PROFILES	firefox.exe

		6 10 MOZILLA 7 10 FIREFOX 8 10 DOCUMENTS 9 10 DEFAULT 10 10 *USERNAME*	
07/24/2011 13:23:58	Open	1 27 C 2 19 VOLUME 3 19 SYSTEM 4 19 RESTORE 5 19 INFORMATION 6 19 F871D828 7 19 A148 8 19 969FDFB59BDE 9 19 4810 10 19 4649	firefox.exe
07/24/2011 15:02:30	Open	1 59 C 2 51 MOZILLA 3 51 FIREFOX 4 42 FILES 5 39 PROGRAM 6 28 SETTINGS 7 20 APPLICATION 8 19 DOCUMENTS 9 19 *USERNAME* 10 19 DATA	firefox.exe
07/24/2011 15:02:31	Close	“ ”	firefox.exe

Computer 1

Time	User Action	Commonly Occurring Words	Process
07/20/2011 19:15:48	Open	None	iexplore.exe
07/20/2011 19:15:50	Open	“ “	iexplore.exe
07/20/2011 19:24:03	Close	1 2 MY 2 2 ITUNES 3 2 DOCUMENTS 4 1 SETTINGS 5 1 MUSIC 6 1 MEDIA 7 1 DOWNLOADS 8 1 *USERNAME* 9 1 C 10 1 AND	iexplore.exe
07/20/2011 19:24:03	Close	“ “	iexplore.exe
07/21/2011 16:08:09	Open	1 21 SETTINGS 2 12 C 3 11 DOCUMENTS 4 11 AND	iexplore.exe

		5 10 TEMPORARY 6 10 LOCAL 7 10 INTERNET 8 10 IE5 9 10 FILES 10 10 *USERNAME*	
07/21/2011 16:08:10	Open	" "	iexplore.exe
07/21/2011 18:34:38	Close	1 17 C 2 14 SETTINGS 3 12 DOCUMENTS 4 10 *USERNAME* 5 10 AND 6 6 TEMP 7 5 LNK 8 4 VOLUME 9 4 SYSTEM 10 4 RP193	iexplore.exe
07/21/2011 18:34:38	Close	" "	iexplore.exe
07/23/2011 14:56:45	Open	1 78 C 2 51 SETTINGS 3 45 FILES 4 33 *USERNAME* 5 32 DOCUMENTS 6 32 AND 7 31 PROGRAM 8 21 INTERNET 9 19 DLL 10 18 LOCAL	iexplore.exe
07/23/2011 14:56:46	Open	" "	iexplore.exe
07/23/2011 19:20:09	Close	1 16 SETTINGS 2 11 *USERNAME* 3 9 DOCUMENTS 4 9 C 5 9 AND 6 7 MICROSOFT 7 7 LOCAL 8 7 DATA 9 7 APPLICATION 10 4 RECOVERY	iexplore.exe
07/23/2011 19:20:09	Close	" "	iexplore.exe

Appendix M: Forensic Investigation of Timelines using Signatures (FITS) bash scripts that allow for action instance detection when the object-trace list, action update threshold, and a consistency function are given compared to the meta-data of a system in mactime format.

```
#!/bin/bash
# FITS: Forensic Investigation of Timelines using Signatures
# Script to check signatures against input file
# Input: Fullpath to a file to test signatures against - image
[vol,part], fls, mactime
# Output: list of detected signatures, and time > "output/detected"
# Version: 3/8/2010
# TODO: Detect Operating System and don't check sigs for other OS's
# Support for split image

# Create timeplot/timeline output?
output_timeline=true

# Global vars
outputdir="output"
datafile="${outputdir}/output.mac"
includesdir="inc"

# Includes
. ${includesdir}/checkInputFile.sh 255
. ${includesdir}/processInputFile.sh 255
. ${includesdir}/signatureTester.sh 255
. ${includesdir}/output.sh 255

#
==================================Functions=======================
================
# sigTest - get all signatures file "signature_list" in same dir, and
compare to datafile
sigTest() {
      if [ -f "signature_list" ]; then
            echo "[I] Signature File List Detected"
            echo "[P] Detecting Events..."
            while read signature; do
                  # Split signature into array 0-type, 1-os, 2-
action, 3-sig, 4-timerange
                  sigArr=(`echo $signature | tr ',' ' '`)
                  if [ -f "${sigArr[3]}" ]; then
                        if [ "${sigArr[0]}" == "core" ]; then
                              # $1 input file - arg3 sig - arg4 time
range
                              sigtest_core "$1" "${sigArr[3]}"
"${sigArr[4]}"
                        elif [ "${sigArr[0]}" == "support" ]; then
                              # $1 input file - arg3 sig - arg4 time
range
                              sigtest_support "$1" "${sigArr[3]}"
"${sigArr[4]}"
                        elif [ "${sigArr[0]}" == "shared" ]; then
                              echo "[I] Shared Signature Found"
                        # Test generic signature based on RE over
directory
                        elif [ "${sigArr[0]}" == "general" ]; then
                              # $1 input file - arg3 sig
                              sigtest_general "$1" "${sigArr[3]}"
                        fi
                  else
```

```
                        echo "[I] Signature file ${sigArr[3]} not
found!"
                fi
            done < "signature_list"
      fi
return 0
}
# ================================End
Functions==================================

# check_file sets var $filetype [vol,part,fls,mac] - sourced
check_file "$1"
if [ $? -eq 0 ]; then
      output_setup_env
      echo "[P] Raw file set to: $1"
      rawfile=$1
      # process_file creates output/output.mac from given file -
sourced
      process_file "$rawfile" "$filetype"
      if [ -f $datafile ]; then
            echo "[P] Data file set to: ${outputdir}/output.mac"
            sigTest "$datafile"
      fi
      # If events have been detected, make timeplot data
      if [ -f "${outputdir}/detected" ]; then
            echo "[P] Exporting the data to XML"
            create_timeplot "output"
      fi
      output_close
else
      echo "$0 [raw_file]"
      echo "      [raw_file] may be:"
      echo "            volume image file"
      echo "            partition image file"
      echo "            fls output file (fls -r -l -m \"/\")"
      echo "            mactime output file (mactime -d -m -y)"
      exit 1
fi

exit 0
```

FITS Included Scripts:

checkInputFile.sh

```
#!/bin/bash
# Script to test the if input file is image, FLS, or MACTIME
compliant
# Input: Fullpath to a file - accepts Disk/Parition Image, FLS
output, MACTIME output
# Output: var filetype set to [vol,part,fls,mac] or null
# Unusable exit 0 - Usable return "1"
# If sourcing file send "255" as first argument
# Version: 3/8/2010
# TODO: Check for split files (*)

# Includes
# Script to process disk image and fls to get mactime format?

#
==================================Functions=======================
================
# check_file - determines the type of file submitted
# must be image, fls, or mactime formats
check_file() {
if [ -n "$1" ] && [ "$1" != "255" ]; then
      if [ -n "$(file $1 | grep boot)" ]; then
            # Should be an image
            # Check for sleuthkit - if no, exit with error
            if [ "$(fls -V)" ]; then
                  if [ -n "$(file $1 | grep partition)" ]; then
                        # Is volume image
                        if [ "$(mmls $1)" != "" ]; then
                              echo "[I] Volume image detected"
                              filetype="vol"
                              return 0
                        fi
                  else
                        # Is partition image
                        if [ "$(fls $1)" != "" ]; then
                                    echo "[I] Partition image
detected"
                                    filetype="part"
                                    return 0
                        fi
                  fi
            else
                  echo "[I] Submitted file looks like disk image, but
Sleuthkit is not installed"
                  echo "[I] Please install Sleuthkit, or submit the
evidence as FLS or MACTIME formats"
                  exit 1
            fi
      else
            firstline=$(tail -n 1 $1)
            flsRE="[0-9]+\|.*\|.*\|.*\|[0-9]+\|[0-9]+\|[0-9]+\|[0-
9]+\|[0-9]+\|[0-9]+$"
            mactimeRE="[0-9]{4}.[0-9]{2}.[0-9]{2}.{5}[0-9]{2}.[0-
9]{2}.[0-9]{2}\,[0-9]*\,.*\,[0-9]+\,[0-9]+\,.*\,.*$"
            # Check for correct FLS output
            if [[ $firstline =~ $flsRE ]]; then
                  echo "[I] Correct FLS format detected"
                  filetype="fls"
```

```
                return 0
          else
                if [[ $firstline =~ $mactimeRE ]]; then
                      echo "[I] Correct MACTIME format detected"
                      filetype="mac"
                      return 0
                else
                      echo "File type could not be detected. Check
the file format and try again."
                      echo "      Correct FLS output: fls -r -l -m
\"/\" [image]"
                      echo "      Correct mactime output: mactime -
d -m -y -b [bodyfile]"
                      exit 1
                fi
          fi
      fi
fi

return 1
}

# ================================End
Functions==================================

# Check arguments
if [ "$1" != "255" ]; then
      if [ -n $1 ] && [ -f "$1" ]; then
            check_file "$1"
            result=$?
      else
            echo "$0 [evidence_file]"
            echo "      Where evidence_file may be:"
            echo "            * Disk image [raw, aff, afd, afm,
afflib, ewf, split]"
            echo "                  (check current sleuthkit image
support)"
            echo "            * Disk partition [check current
sleuthkit fs support]"
            echo "            * FLS output [fls -r -l -m \"/\"]"
            echo "            * MACTIME output [mactime -d -m -y]"
            exit 1
      fi

exit $result
fi
```

Compare.sh

```
#!/bin/bash
# Script to compare two time strings sent in "YYYY MM DD DDD
HH:MM:SS" format (mactime -d -m -y)
# Can set the time range allowance in minutes with a third argument.
# If sourcing file send "255" as first argument
# Version: 29/7/2010
# TODO: Allow for date and hour change within the set range i.e. 2010
01 02 23:59=>2010 01 03 00:00
```

```
#
====================================Functions=======================
===============
# Compare two given times are equal to the minute - input: two times,
output: 0 false 1 true
# Need to take into account hour change and day change while action
takes place
compare_times() {
      if [ "$3" ]; then
            range=$3
            #echo "Range set to $3"
      else
            range="1"
      fi
      if [ "$1" ] && [ "$2" ]; then
            time1=$(echo $1 | awk -F, '{ print $1 }' | cut -c 16-20)
            time2=$(echo $2 | awk -F, '{ print $1 }' | cut -c 16-20)
            htime1=$(echo $time1 | awk -F: '{print $1}')
            htime2=$(echo $time2 | awk -F: '{print $1}')
            mtime1=$(echo $time1 | awk -F: '{print $2}')
            mtime2=$(echo $time2 | awk -F: '{print $2}')
            let rangepls=$mtime2+$range
            let rangemns=$mtime2-$range
            if [ "$htime1" == "$htime2" ]; then
           if [ $mtime1 -eq $mtime2 ] || [ $mtime1 -le $rangepls ] &&
[ $mtime1 -ge $rangemns ]; then
                        return 0
                fi
        fi
      fi
return 1
}

# Compare two dates from a mactime string
compare_dates() {
      if [ "$1" ] && [ "$2" ]; then
            # Get time string in HH:MM
            date1=$(echo $1 | awk '{print $1" "$2" "$3" "$4}')
            date2=$(echo $2 | awk '{print $1" "$2" "$3" "$4}')
            if [ "$date1" == "$date2" ]; then
                  return 0
            fi
      fi
return 1
}

# Compare both dates and times
compare_datetime() {
        compare_dates "$1" "$2"
        if [ $? -eq 0 ]; then
                if [ "$3" ]; then
                        compare_times "$1" "$2" "$3"
                        return $?
                else
                        compare_times "$1" "$2"
                        return $?
                fi
        fi
return 1
}
```

```
# ================================End
Functions================================

# Check arguments
if [ $1 -ne 255 ]; then
      if [ $# -ne 2 ] && [ $# -ne 3 ]; then
            echo "$0 [time1] [time2] [range]"
            echo "      Date 1 and Date 1 in time format 'YYYY MM DD
DDD HH:MM:SS'"
            echo "      (mactime -m -y)"
            echo "      Range is the range allowance in minutes -
default 1"
            exit 1
      else
            compare_datetime "$1" "$2"
            result=$?
      fi
exit $result
fi
```

processInputFile.sh

```
#!/bin/bash
# Script to process the input file - image [vol,part], fls, mactime
# Input: Fullpath to a file and file type
# Output: process each type to get proper mactime format - output in
folder "output"
# Output: to name output.fls & ${datafile}=output.mac
# If sourcing file send "255" as first argument
# Version: 19/8/2010
# TODO: Error checking - what about unknown file systems?
checkInputFile should help

# Includes

#
==================================Functions=======================
================
# check_file - determines the type of file submitted
# must be image, fls, or mactime formats
process_file() {
if [ -n "$1" ] && [ -n "$2" ] && [ "$1" != "255" ]; then
      # Check what type of file
      case $2 in
            "vol")
                  process_volume "$1"
                  process_fls "${outputdir}/output.fls";;
            "part")
                  process_partition "$1"
                  process_fls "${outputdir}/output.fls";;
            "fls")
                  is_fls "$1"
                  process_fls "${outputdir}/output.fls";;
            "mac")
                  process_mactime "$1";;
            *)
```

```
                  echo "Check the file type, and try again
(checkInputFile.sh)"
                  exit 1;;
      esac
fi

return 0
}

# Process a volume image to get fls output
process_volume () {
      echo "[P] Processing Volume Image"
      offsets=$(mmls $1 | tail -n+6 | awk '{print $3}')
      for offset in ${offsets[*]}
      do
            $(fls -o ${offset} -r -l -m "/" $1 >>
${outputdir}/output.fls)
      done
}

# Process a partition image to get fls output
process_partition () {
      echo "[P] Processing Partition Image"
      $(fls -r -l -m "/" $1 > ${outputdir}/output.fls)
}

# If file is fls file, copy to output directory
is_fls() {
      # Copy fls file to output folder for consistency
      $(cat $1 > ${outputdir}/output.fls)
}

# Process an fls file to get mactime output
process_fls () {
      echo "[P] Processing FLS File"
      $(mactime -b $1 -d -m -y > ${datafile})
}

# If correct mactime, copy data file to output folder (for
consistency)
process_mactime () {
      echo "[P] Processing MACTIME File"
      $(cat $1 > ${datafile})
}

# ================================End
Functions=================================

# Check arguments
if [ "$1" != "255" ]; then
      outputdir="output"
      if [ -n "$1" ] && [ -n "$2" ] && [ -f "$1" ]; then
            datafile="${outputdir}/output.mac"
            process_file "$1" "$2"
            result=$?
      else
            echo "$0 [evidence_file] [file_type]"
            echo "      Where evidence_file may be:"
            echo "            * Disk image [raw, aff, afd, afm,
afflib, ewf, split]"
```

```
            echo "                  (check current sleuthkit image
support)"
            echo "            * Disk partition [check current
sleuthkit fs support]"
            echo "            * FLS output [fls -r -l -m \"/\"]"
            echo "            * MACTIME output [mactime -d -m -y]"
            echo "  And file type is [vol,part,fls,mac]"
            exit 1
      fi

exit $result
fi
```

signatureTester.sh

```
#!/bin/bash
# Script to test a signature against a given data file
# Input: Datafile in mactime [-d -m -y] format - signature file -
optional time range (default 1 min)
# If sourcing file send "255" as first argument
# Version: 29/7/2010
# TODO: Time returned by sigtest_core needs to be earliest time found
# Move General Signature timeplot formatting to createTimeplot.sh
# Get core event time to test supporting consistency

#
====================================Functions=======================
================
# sigtest_core tests core signatures against datafile
# and returns the time of the event or is inconsistent (throw error
stop)
sigtest_core() {
if [ -n $1 ] && [ -n $2 ]; then
      if [ "$3" ]; then
            range=$3
      else
            range="1"
      fi
      if [ "$1" ] && [ "$2" ]; then
            local sig_name=$(basename $2 | awk 'sub("....$","")')
            while read trace; do
                  local cmptime1=$cmptime2
                  # First element is file and path (as specific as
possible)
                  local trace_file=$(echo $trace | awk -F, '{print
$1}')
                  # Second element is timestamp to look at in mactime
format (macb)
                  local trace_ts=$(echo $trace | awk -F, '{print
$2}')
                  # Search the data file for the file + time - return
time
                  local trace_file_clean=$(echo $trace_file | sed
's/\//\\\//g')
                  # Possible to get multiple times from search -
treat like array (after split on '-' )
```

```
                  local detected_traces=$(cat "$1" | egrep
$trace_file_clean$ | egrep $trace_ts | awk -F, '{print $1}' | awk
'{print $1"-"$2"-"$3"-"$4"-"$5}')
                  # loop through all returned times
                  local let elements=0 # Need counter because
$trace_times returns 1 element even if more
                  for times in $detected_traces; do
                        let elements=$elements+1
                  done
                  for found_time in $detected_traces; do
                        local trace_time=$(echo $found_time | awk -F-
'{print $1" "$2" "$3" "$4" "$5}')
                        if [ "$cmptime1" == "" ] && [ $elements -eq 1
]; then
                              local cmptime2=$trace_time
                              break
                        elif [ "$cmptime1" == "" ] && [ $elements -gt
1 ]; then
                              echo "[I] Multiple times found, but no
time to compare them to: $trace_times"
                              break
                        elif [ "$cmptime1" != "" ] && [ $elements -gt
1 ]; then
                              if [ $(compare_datetime "$cmptime1"
"$trace_time" "$range") ]; then
                                    cmptime2=$trace_time
                              else
                                    echo "[I] Questionable (multi-
return) time found in $sig_name: $trace_time"
                              fi
                        fi
                  done
                  #echo $cmptime1
                  #echo $cmptime2
                  if [ "$cmptime1" != "" ] && [ "$cmptime2" != "" ];
then
                        compare_datetime "$cmptime1" "$cmptime2"
"$range"
                        local cmp_result=$?
                        if [ $cmp_result -eq 1 ]; then
                              echo "[I] Inconsistency found in
signature $sig_name"
                              # Clear vars
                              cmptime1=""
                              cmptime2=""
                              cmp_result=1
                              break
                        fi
                  fi
            done < "$2"
      fi
      if [ "$cmptime1" == "" ] && [ "$cmptime2" != "" ]; then
            # Only one trace in signature, and trace is detected
            cmp_result=0
      fi
      if [ "$cmp_result" != "" ] && [ $cmp_result -eq 0 ]; then
            echo "[I] Core event $sig_name detected at $trace_time"
            $(echo "core,$sig_name,$trace_time" >>
${outputdir}/detected)
            # Clear vars
            cmptime1=""
```

```
            cmptime2=""
            cmp_result=1
            return 0
      fi
fi
return 1
}

# sigtest_support tests supporting signatures against datafile
# and returns the time(s) of the event or is inconsistent (throw
error stop)
# Sigtest_support may be an array!
# ----------------- Need to check for the core to support - get core
time -----------
sigtest_support() {
if [ -n $1 ] && [ -n $2 ]; then
      if [ "$3" ]; then
            range=$3
      else
            range="1"
      fi
      if [ "$1" ] && [ "$2" ]; then
            local sig_name=$(basename $2 | awk 'sub("....$","")')
            # Each trace could return one or more elements - treat as
array
            while read trace; do
                  # First element is file and path (as specific as
possible)
                  local trace_file=$(echo $trace | awk -F, '{print
$1}')
                  # Second element is timestamp to look at in mactime
format (macb)
                  local trace_ts=$(echo $trace | awk -F, '{print
$2}')
                  # Search the data file for the file + time - return
time
                  local trace_file_clean=$(echo $trace_file | sed
's/\//\\\//g')
                  local detected_traces=$(cat "$1" | egrep
$trace_file_clean$ | egrep $trace_ts | awk -F, '{print $1}' | awk
'{print $1"-"$2"-"$3"-"$4"-"$5}')
                  # loop through all returned times
                  local let elements=0 # Need counter because
$trace_times returns 1 element even if more
                  for times in $detected_traces; do
                        let elements=$elements+1
                  done
                  #echo "Number of elements: $elements"
                  for found_time in $detected_traces; do
                        local trace_time=$(echo $found_time | awk -F-
'{print $1" "$2" "$3" "$4" "$5}')
                        echo "[I] Supporting event $sig_name detected
at $trace_time"
                        $(echo "support,$sig_name,$trace_time" >>
${outputdir}/detected)
                  done
            done < "$2"
            return 0
      fi
fi
return 1
```

```
}

# sigtest_general - iterates through a list to test a RE pattern
# and returns the time(s) of the event or is inconsistent (throw
error stop)
# Groups files by a common known pattern for a specific action like
"browse" or "install"
sigtest_general() {
if [ -n "$1" ] && [ -n "$2" ]; then
      datafile="$1"
      while read line; do
            local sig_name=$(echo $line | awk -F, '{print $1}')
            local sig_RE=$(echo $line | awk -F, '{print $2}')
            local gen_traces=$(cat $datafile | egrep "$sig_RE" | awk
-F, '{print $1}' | sort | uniq | awk -F, '{print $1","}')
            if [ -n "${gen_traces[@]}" ]; then
                  echo "[I] Occurrences of general event - $sig_name
- found"
                  # Assign array elements to arg variable to pass to
function
                  returned_array=$( process_gen_array
"${gen_traces[@]}" )
                  # Might want to output to file and have
createTimeplot take care of formatting
                  OIFS=$IFS
                  IFS=";"
                  evt_range=( $returned_array )
                  IFS=$OIFS
                  for range in "${evt_range[@]}"; do
                        start_time=$(echo $range | awk -F, '{print
$1}')
                        # Remove space in front of ending time or get
xml error
                        end_time=$(echo $range | awk -F, '{print $2}'
|  sed -e 's/^[ \t]*//')
                        # output to gen signature file
                        #$(echo "<event start=\"$start_time GMT-
0000\" end=\"$end_time GMT-0000\" title=\"$sig_name\"></event>" >>
${outputdir}/genevents.xml)
                        $(echo
"general,$sig_name,$start_time,$end_time" >> ${outputdir}/detected)
                  done
            fi
      done < $2
fi
return 0
}
# ================================End
Functions==================================

# Check arguments
# Currently treats every sent signature as a core
if [ "$1" != "255" ]; then
      . compare.sh 255
      . processGeneral.sh 255
      if [ $# -ne 2 ] && [ $# -ne 3 ]; then
            echo "$0 [data_file] [signature_file] [range]"
            echo "      Datafile should be in mactime (-d -m -y)
format"
            echo "      Signature file should be in
[fullpath,timestamp] format"
```

```
            echo "      Range is the range allowance in minutes -
default 1"
            exit 1
      else
            sigtest_core "$1" "$2"
            result=$?
      fi

exit $result
else
      . ${includesdir}/processGeneral.sh 255
      . ${includesdir}/compare.sh 255
fi
```

processGeneral.sh

```
#!/bin/bash
# Script to process results of a general signature match
# Input: Array of times to group
# Output: Events with a time range
# If sourcing file send "255" as first argument
# Version: 5/8/2010
# TODO:

#
==================================Functions=======================
================
# process_gen_array accepts an array of mactime-format timestamps
YYYY MM DD DDD HH:MM:SS
# and returns array of csv start and end times [evt start,evt end]
# FOR TESTING: ONLY GET TIME RANGE PER DAY
process_gen_array() {
      local passed_array
      local timespan_array
      # Array passed as csv string
      OIFS=$IFS
      IFS=","
      passed_array=( $1 )
      IFS=$OIFS
      # List should already be chronological so get first and compare
date until non-match
      # (testing) - fix this later
      for ts in "${passed_array[@]}"; do
            if [ "$time1" == "" ]; then
                  time1="$ts"
                  continue
            fi
            if [[ $time1 =~ [0-9]{4}.[0-9]{2}.[0-9]{2}.{5}[0-
9]{2}.[0-9]{2}.[0-9]{2} ]]; then
                  #echo "Comparing $time1 and $ts"
                  compare_dates "$time1" "$ts"
                  if [ $? -eq 0 ]; then
                        time2="$ts"
                        continue
                  else
                        compare_dates "$time1" "$time2"
                        if [ $? -eq 0 ]; then

      timespan_array[$[${#timespan_array[@]}+1]]="$time1,$time2;"
```

```
                        else

      timespan_array[$[${#timespan_array[@]}+1]]="$time1,$time1;"
                        fi
                        time1="$ts"
                  fi
            else
                  time1="$ts"
            fi
      done
echo ${timespan_array[@]}
}

# ================================End
Functions================================

# Check arguments
# Currently treats every sent signature as a core
if [ "$1" != "255" ]; then
      outputdir="output"
      . compare.sh 255
      if [ "$1" == "" ]; then
            echo "$0 [timestamp array]"
            echo "      Where timestamp array is an array of
timestamps returned by a general signature."
            exit -1
      else
            process_gen_array "$1"
            result=$?
      fi

exit $result
fi
```

output.sh

```
#!/bin/bash
# Script to manage the output of the FITS script
# OLDInput: output dir - will detect event detection and fls/mactime
files from output dir
# OLDOutput: if not dir is given, will default to 'output' dir
# OLDOutput: 'fsactivity.txt' for plotting, 'events.xml' for event
plotting - 'preinstall.txt' if detected
# If sourcing file send "255" as first argument
# Version: 19/8/2010
# TODO: Split mactime file based on [os]_install if no fls file
exists
# Include ability to send GMT offset

# Vars
time_offset="0000"
# Output event files per signature
evt_gen="events_general.xml"
evt_core="events_core.xml"
evt_support="events_support.xml"
evt_shared="events_shared.xml"
evt_del="events_deleted.xml"
```

```
declare -a evt_array=($evt_gen $evt_core $evt_support $evt_shared
$evt_del)


# Includes

#
====================================Functions=======================
================
# Sets up output files - one for each signature and assigns the
variables
output_setup_env () {
      # Cleanup output directory
      $(rm -r -f ${outputdir} && mkdir -p ${outputdir})
      $(echo "<data>" > ${outputdir}/events.xml)
      # It output timeline = true create new files and vars for each
sig type
      if [ $output_timeline ]; then
            for evt_file in "${evt_array[@]}"; do
                  $(echo "<data>" > "${outputdir}/${evt_file}")
            done
      fi
}

# Adds closing brackets to any files that were opened with
output_setup_env
output_close () {
      $(echo "</data>" >> ${outputdir}/events.xml)
      # It output timeline = true create new files and vars for each
sig type
      if [ $output_timeline ]; then
            for evt_file in "${evt_array[@]}"; do
                  $(echo "</data>" >> "${outputdir}/${evt_file}")
            done
      fi
}

# Input - evt_type,sig_name,trace_time
# Writes event information in the correct file
output_event() {
      echo "Ouput Event! Not used yet"
}

# Create_timeplot creates timeline/timeplot data from 'detected'
events
create_timeplot() {
if [ -d $1 ] && [ "$1" != "255" ]; then
      while read line; do
            sig_type=$(echo "$line" | awk -F, '{print $1}')
            evt_name=$(echo "$line" | awk -F, '{print $2}')
            evt_start=$(echo "$line" | awk -F, '{print $3}')
            evt_end=$(echo "$line" | awk -F, '{print $4}')

            if [ $output_timeline ]; then
                  case $sig_type in
                        "core")
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/events.xml);
```

```
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/${evt_core});;
                        "support")
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/events.xml);
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/${evt_support});;
                        "shared")
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/events.xml);
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/${evt_shared});;
                        "general")
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" end=\"$evt_end\" title=\"$evt_name\">$sig_type event
$evt_name detected from $evt_start to $evt_end</event>" >>
${outputdir}/events.xml);
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" end=\"$evt_end\" title=\"$evt_name\">$sig_type event
$evt_name detected from $evt_start to $evt_end</event>" >>
${outputdir}/${evt_gen});;
                        "deleted")
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/events.xml);
                              $(echo "<event start=\"$evt_start GMT-
${time_offset}\" title=\"$evt_name\">$sig_type event $evt_name
detected at $evt_start</event>" >> ${outputdir}/${evt_del});;
                  esac
            fi
            # Check for OS install time
            case $evt_name in
                  "win_install")
                        local install_time=$( echo $evt_start | awk
'{print $1"-"$2"-"$3}');
                        output_OS_install_filter "$install_time";;
                  "mac_install")
                        echo "MAC Install Detected at $evt_start";;
                  "unix_install")
                        echo "Unix Install Detected at $evt_start";;
            esac

      done < "${outputdir}/detected"
else
      echo "[I] No output directory found"
      exit -1
fi

return 0
}

# Split timeplot/timeline graphs based on the OS_Install time, if
detected
# Write filter to split mactime file directly
output_OS_install_filter() {
      local install_time="$1"
```

```
      if [ -f "${outputdir}/output.fls" ]; then
            # Get beginning time from old mactime file
            if [ "$install_time" != "" ] && [ -f "${datafile}" ];
then
                  echo "[I] OS Install time detected, filtering
plots"
                  local fs_start=$(head -2 ${datafile} | tail -1 |
awk -F, '{print $1}' | awk '{print $1"-"$2"-"$3}')
                  # Write preinstall.txt
                  $(mactime -b ${outputdir}/output.fls -d -m -y
$fs_start..$install_time | awk -F, '{print $1}' | cut -c -10,15- |
cut -c -16 | uniq -c | awk '{print $2"-"$3"-"$4" "$5","$1}' >
${outputdir}/preinstall.txt)
                  # Write fsactivity.txt
                  $(mactime -b ${outputdir}/output.fls -d -m -y
$install_time | awk -F, '{print $1}' | cut -c -10,15- | cut -c -16 |
uniq -c | awk '{print $2"-"$3"-"$4" "$5","$1}' >
${outputdir}/fsactivity.txt)
                  return 0
            fi
      else
            echo "[I] Could not find FLS file to split plot data"
            # So we dont get error from timeplot
            $(echo "" > ${outputdir}/preinstall.txt)
            $(mactime -b ${outputdir}/output.fls -d -m | awk -F,
'{print $1}' | cut -c -10,15- | cut -c -16 | uniq -c | awk '{print
$2"-"$3"-"$4" "$5","$1}' > ${outputdir}/fsactivity.txt)
            return 1
      fi
}

# ================================End
Functions==================================

# Check arguments
if [ "$1" != "255" ]; then
      local outputdir="output"
      local datfile="${outputdir}/output.mac"
      local output_timeline=false
      if [ -n $1 ] && [ -d "$1" ]; then
            create_timeplot "$1"
            result=$?
      else
            # Default to 'output' directory
            create_timeplot "output"
            result=$?
      fi

exit $result
fi
```

createTimeplot.sh

```
#!/bin/bash
# Script to create timeplot files for viewing
# Input: output dir - will detect event detection and fls/mactime
files from output dir
# Input: if not dir is given, will default to 'output' dir
```

```
# Output: 'fsactivity.txt' for plotting, 'events.xml' for event
plotting - 'preinstall.txt' if detected
# If sourcing file send "255" as first argument
# Version: 3/8/2010
# TODO: Split mactime file based on [os]_install if no fls file
exists
# Include ability to send GMT offset

# Includes

#
====================================Functions=======================
================
# check_file - determines the type of file submitted
# must be image, fls, or mactime formats
create_timeplot() {
if [ -d $1 ] && [ "$1" != "255" ]; then
      outputdir="$1"
      echo "[P] Creating Event xml file"
      $(echo "<data>" > ${outputdir}/events.xml)
      while read line; do
            evt_name=$(echo "$line" | awk -F, '{print $1}')
            evt_time=$(echo "$line" | awk -F, '{print $2}')
            $(echo "<event start=\"$evt_time GMT-0000\"
title=\"$evt_name\"></event>" >> ${outputdir}/events.xml)
            # Check for OS install time
            case $evt_name in
                  "win_install")
                        install_time=$( echo $evt_time | awk '{print
$1"-"$2"-"$3}');;
                  "mac_install")
                        echo "MAC Install Detected at $evt_time";;
                  "unix_install")
                        echo "Unix Install Detected at $evt_time";;
            esac

      done < "${outputdir}/detected"

      if [ -f ${outputdir}/genevents.xml ]; then
            echo "[P] Adding Detected General Events"
            $(cat ${outputdir}/genevents.xml >>
${outputdir}/events.xml)
      fi

      $(echo "</data>" >> ${outputdir}/events.xml)

      # Get beginning time from old mactime file
      if [ "$install_time" != "" ] && [ -f "${outputdir}/output.mac"
]; then
            echo "[I] OS Install time detected, filtering plots"
            fs_start=$(head -2 ${outputdir}/output.mac | tail -1 |
awk -F, '{print $1}' | awk '{print $1"-"$2"-"$3}')
            # Write preinstall.txt
            $(mactime -b ${outputdir}/output.fls -d -m -y
$fs_start..$install_time | awk -F, '{print $1}' | cut -c -10,15- |
cut -c -16 | uniq -c | awk '{print $2"-"$3"-"$4" "$5","$1}' >
${outputdir}/preinstall.txt)
            # Write fsactivity.txt
            $(mactime -b ${outputdir}/output.fls -d -m -y
$install_time | awk -F, '{print $1}' | cut -c -10,15- | cut -c -16 |
```

```
uniq -c | awk '{print $2"-"$3"-"$4" "$5","$1}' >
${outputdir}/fsactivity.txt)
      else
            # So we dont get error from timeplot
            $(echo "" > ${outputdir}/preinstall.txt)
      fi

else
      echo "[I] No output directory found"
      exit -1
fi

return 0
}

# ================================End
Functions==================================

# Check arguments
if [ "$1" != "255" ]; then
      if [ -n $1 ] && [ -d "$1" ]; then
            create_timeplot "$1"
            result=$?
      else
            # Default to 'output' directory
            create_timeplot "output"
            result=$?
      fi

exit $result
fi
```