COMPUTERS & SECURITY 45 (2014) 293–304

Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

Time and space interval record schedule consistency analysis for atomic items without interactions in open spaces with stationary locations☆

*Fred Cohen**, *Don Cohen*

Fred Cohen & Associates, PO Box 811, Pebble Beach, CA 93953, USA

ARTICLE INFO

Article history:
Received 4 January 2014
Received in revised form
25 January 2014
Accepted 2 March 2014
Available online 1 April 2014

Keywords:
Consistency analysis
Record schedule analysis
Digital forensics
Travel time analysis
Subversion detection

ABSTRACT

Attacks on systems often produce records that are distinguishable from normal records because, by the nature of the subversions they undertake, they produce records that the system could not produce under normal operation. This paper outlines a basis for understanding and determining one class of such discernible subversion inconsistencies associated with time and space interval record schedule consistency analysis for atomic items in open spaces without interactions as a method of questioned digital record examination. It starts with a brief introduction to the issues and description of the specific problem at hand, develops an approach to solving the problem, and identifies an algorithm for near-linear time detection of inconsistency or demonstration of a feasible schedule for special cases likely to occur in real-world record-keeping.

1. Objectives, methodology, background, and overview

1.1. Objectives

The objective of this paper is to describe an algorithm and method by which inconsistency analysis may be applied to detect attempts to subvert systems. The particular algorithm is suitable for time and space interval record schedule consistency analysis for atomic items without interactions in open spaces with stationary locations.

1.2. Methodology

The methodology applied was to (1) identify the nature of the problem, (2) partition the problem by identifying characteristics of relevance, (3) identify an approach to addressing the inconsistency analysis problem for the particular cases, (4) identify candidate algorithms based on knowledge, skills, training, education, and experience, (5) analyze these algorithms to determine their utility and complexity, (6) implement versions of these algorithms, (7) test these algorithms on sample data both generated and real, (8) write up the results, and (9) submit them to a peer reviewed journal for consideration.

1.3. Background

Attacks on systems often produce records that are distinguishable from normal records because, by the nature of the subversions they undertake, they produce records that the system could not produce under normal operation. One of a potentially unlimited number of examples of this is when a

☆ The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited.

* *Corresponding author*. Fred Cohen & Associates, PO Box 811, Pebble Beach, CA 93953, USA. Tel.: +1 925 454 0171.

http://dx.doi.org/10.1016/j.cose.2014.03.002
0167-4048/

user uses another user's identity through privilege escalation. Other examples include, without limit, deletion or alteration of logs, use of another user's account, altering ownership of files or directories, removal of a disk to duplicate it during system downtime, altering the names of files to avoid firewall rules, and use of another user's facility access badge. Some of these might be done by altering records, theft of devices, breaking and entering, exploiting backup and recover mechanisms, and any number of other mechanisms.

Rather than seeking to identify the mechanisms of privilege escalation, facility entry, disk removal, facility break-ins, backup interception and substitution, and every other mechanism that might produce these sorts of records and then identifying all of the conditions of those records indicative of each such mechanism, along with the extensive time and effort associated with doing such, we seek generic solutions that leverage computational advantage to the defender. One notion for such a defensive mechanism is consistency analysis.

Consistency analysis, as a broad concept, has an enormous range of possibilities, and if done rapidly enough, can be used as a detection mechanism with real-time response that allows it to form a preventive mechanism. But even if we cannot do detection and response in real-time, whether because of the lack of real-time access to records or the computational complexity of analysis, a consistency analysis approach that drives the computational complexity of undetected attack high enough to make it infeasible is an advantage to the defender, if it can be done with reasonable resources.

One class of such inconsistency detection methods deals with the movement of people, things, programs, data, or anything else for which there are records, and the record-keeping systems associated with such movements. This is the issue of time and space. Because all digital records must be of finite accuracy and precision in theory, and in practice all such records are so, rather than dealing with exact times and locations, we need to deal with less precise information, and thus we deal with intervals. The notion of things moving over time can be considered in light of the notions of scheduling, an area that has been studied for a long time as part of the field of operations research, and a field for which there are many known algorithms. Record schedule consistency analysis is then the set of analytical methods associated with detecting consistency (confirmation) or inconsistency (refutation) of the validity of a schedule of times and spaces reflected in records. A subfield of this area of study is cases where the things whose times and locations are associated with the schedules are not multi-part or separable, and thus cannot appear in two disjoint intervals of space simultaneously (i.e., they cannot be in two places at once). For example, people and other physically unique items can be considered atomic items for intervals large enough to envelope the items. Interactions between people and other atomic items can be considered in light of all interactions between them, but the present effort focuses on how to perform analysis without taking such interactions into account, and thus the current study is without interactions. The movement of items through space and time can be, and often is, restricted, for example by one way streets or impediments such as walls, which are stationary, and by things like ships, airplanes, and other moving things that may contain mechanisms capable of producing records. The present study examines only open spaces with stationary locations, a subset of the more complex overall problem. Thus, this paper is about identifying inconsistencies between records and realistic possibilities to detect subversions of systems resulting from attacks on those systems.

Like any mechanism producing traces, some knowledge of the nature of the mechanisms is required in order to understand the nature of what is being examined and meaningfully examine it. Many digital records self-indicate[1] the presence of an item at a location at a time. For example, a record may indicate that a particular credit card was present at a particular card reader at a particular time.[2] Multiple records associated with this credit card may be used to partially trace its movement over time. With a few assumptions, we can then trace the movement of the person using the credit card over time. But suppose the records are inconsistent in that they show the same credit card was used in Los Angeles, California, USA and London, England within a 15 min time span.[3] This would indicate a problem in terms of the use of these records for forensic purposes, and show the tracking of the card and/or the individual using it to be unreliable.[4]

Consistency analysis for digital records in limited contexts has been considered in the literature.[5] Papers on audit trail consistency,[6] semantic integrity checking,[7] and formalized event reconstruction[8] focussed largely on the theoretical basis for such analysis. As the field progressed, additional efforts were undertaken for more specific problems, such as rigorous checking for consistency in systems modeled with finite granularity,[9] hypothesis analysis for alternative hypotheses about time stamps,[10] and hypotheses based on investigative approaches.[11]

More recent papers associated with automated reconstruction tend to focus more on translating multiple traces into

[1] Self-indicating records, on their own, indicate the specified condition.

[2] When we indicate a "time" we mean it to include date, time, zone, and other relevant details.

[3] We will assume times indicated are reconciled to a common time base for the purposes of this paper.

[4] Per L. Duranti, "Diplomatics", Encyclopedia of Library and Information Sciences, Third Edition DOI: 10.1081/E-ELIS3-120043454, 2010, Taylor & Francis. Reliability: the record as a true statement of fact relates to the extent to which the record reflects the reality it purports.

[5] Summarized in F. Cohen, "Digital Forensic Evidence Examination – 4th Ed.", ASP Press, 2009–2012. ISBN # 1-878109-47-2.

[6] F. Cohen, "A Note on Detecting Tampering with Audit Trails", 1995, available at http://all.net/books/audit/audmod.html.

[7] T. Stallard and K. Levitt, "Automated Analysis for Digital Forensic Science: Semantic Integrity Checking", ACSAC-2003.

[8] P. Gladyshev, "Formalising event reconstruction in digital investigations." PhD Dissertation; University College Dublin; 2004-08.

[9] P. Gladyshev and A. Enbacka, "Rigorous Development of Automated Inconsistency Checks for Digital Evidence Using the B Method ", International Journal of Digital Evidence, Fall 2007, Volume 6, Issue 2.

[10] Svein Yngvar Willassen, "Hypothesis-based investigation of digital timestamps", chapter in Advances in Digital Forensics IV, Ray and Shenoi ed., Springer, ISBN# 978-0-387-84926-3, 2008.

[11] B. Carrier, "A Hypothesis Based Approach to Digital Forensic Investigation." PhD Dissertation; Purdue University; May, 2006.

commensurable records from diverse sources[12] and apply little of the theoretical analysis of consistency to the real problems of reconstruction that underlie the results they produce. While producing commensurate records is certainly a vital component of time analysis, essentially all such papers focus on identifying time-base offsets, reconciling times to those fixed offsets, and ordering records based only on those sets of assumptions. Strict orderings are essentially always assumed and provided as output, and ancillary information related to issues of time are largely ignored in most such analysis.

The problem with this approach and set of assumptions is that, when records are altered or orderings assumed, very different results may appear than the underlying reality actually supports. Again, the reliability of these methods when we loosen the false assumptions of perfect ordering or precision matching accuracy in time-related traces comes into potentially serious doubt, and in some cases direct refutation.

1.4. *Underlying notions of time and space*

We assume that causality exists and thus that mechanisms (m) turn causes (C) into effects (E). This is captured in the expression $C \rightarrow^{m} E$. Further, those mechanisms take time. Thus, in determining the (potential) orderings of events and associating times for such events, the notion that causes precede effects by times limited by the mechanisms of causality is fundamental.

Digital records, or more generally, traces associated with the execution of digital mechanisms, may be used to analyze causality relative to an assumed or measured underlying model of reality. For example, if traces hypothetically producible only by unique and finite (in time and space) devices indicate that an item traveled faster than the speed of light, then either our model of reality limiting travel to the speed of light is wrong, or the assumptions about unique and finite devices producing the trace is wrong. Which of these assumptions is considered wrong is up to the examiner and the court to decide. Modeling and detecting the inconsistency is our challenge.

Fundamental to this approach is the notion of travel time. That is, it takes time to get from place to place, even for digital information. Thus in $C \rightarrow^{m} E$, the delay from C to E associated with m is positive and bounded from below. Even when we properly assume that times are ranges and not exact values, as we identify more and more traces of things at times at locations, and assuming we know minimum travel times associated with $C \rightarrow^{m} E$ and pairs of locations, the ranges of times for things at places get more and more constrained (or at least never less constrained) with the addition of records. If and as records are added, they can either come to a final set of currently known and consistent time and location range constraints, or they can reach a point where some time-space condition indicated by the records becomes vacuous, in that there is no feasible m s.t.[13] $C \rightarrow^{m} E$, and thus the records are identified as inconsistent.

The general notion is particularized in this paper with regard to time and space interval record schedule consistency analysis for atomic items without interactions in open spaces with stationary locations. Breaking that down;

- Time as represented in digital records has finite precision and thus always represents a time interval. The size of the interval depends on a range of factors associated with the clocks that produce those records and the manner in which those clocks are used to produce those records.
- Space is typically identified in terms of recorded locations, often relative to other locations, such as an address of a building on Earth. It is also kept to finite granularity, and is typically associated with a physical object (e.g., a credit card reader), and thus represents a region of space. The size and location of the region may change over time (e.g., a credit card reader on an airplane in flight).
- A schedule is a sequence of places over time where an item may have appeared. A consistent schedule is a schedule which could actually have occurred.
- An atomic item is an item that cannot be separated or duplicated, and thus can only be in one region of space at any given region of time, and can only move in time-space from region to region at finite speed.
- Interactions encompass the issues surrounding meetings of two or more atomic objects. In this analysis, we ignore this issue except to note in passing that the interaction of atomic objects may further constrain consistent schedules.
- An open space is a space in which traveling from any distinct recorded location to any other distinct recorded location can be done without having to pass through a 3rd specific recorded location (e.g., the only way from A to B is through a door that records entries). In Euclidean space, transitivity of travel time applies (i.e., if you can go from A to B in time s and from B to C in time t, travel from A to C can be completed in s+t).
- Stationary locations don't move relative to the frame of reference within the time frame at issue. For example, a credit card reader at a cash register in a hotel in San Francisco is likely stationary over the period of a week to a spatial granularity of a few feet, while a credit card reader in a taxicab in San Francisco is likely not stationary over the same period relative to the same granularity.

1.5. *Implementation data model*

In this paper, our goal is to check the consistency of a set of records supposed to represent the locations of atomic items, such as people, at times. The times are not expected to be perfectly accurate, but it is assumed that we can bound their error. As a result, a timestamp from a record can be thought of as an interval. We also assume that we can map these intervals to a common time base, allowing them to be meaningfully compared with each other.[14]

[12] Christopher Hargreaves, Jonathan Patterson, "An automated timeline reconstruction approach for digital forensic investigations", Digital Investigation 9 (2012) S69–S79.

[13] "s.t." will stand for "such that" throughout this paper.

[14] Times are somewhat more complex than this since two times measured by the same clock normally have closely related offsets and error characteristics. Thus a second level of analysis relative to different clocks will improve interval reductions and detect more inconsistencies.

We model spatial locations as named places. The constraints below (static minimal travel times required to be positive between any two distinct locations and zero from any location to itself, along with the triangle inequality) assume that locations are single stationary points in space. However, in most cases, they can be reasonably thought of as relatively small regions of space, in that movement throughout the region of space they occupy by an atomic object takes time within time intervals small relative to the movement between such locations.

We also assume that we can measure or estimate the minimum time required to get from one location to another. A set of records is consistent with the minimal travel times if there is a schedule of (exact) times and places matching the set of records, in a sense defined more precisely below, which does not involve traveling from one location to another in less than the minimal required time.

It turns out that the traveling salesman problem[15] is a special case of our problem, so we don't expect to be able to do this checking efficiently in all cases. We do expect that for our particular application area, this complexity will not be inherent in the vast majority of cases. In fact, for the most part, we expect the location records to be linearly ordered, that is, given two such records, it will usually be obvious which one must come first in time. For our purposes it will typically be adequate to recognize computationally complex situations and not actually solve them. We hope to process the records in something close to linear time, or more realistically in time proportional to n log(n) where n is the number of location records. This is the same complexity as the sorting problem. Since a subproblem of our problem, finding a complete ordering, sorts the records by time, a more efficient result would imply that we found a more efficient way of sorting. Finally we note that the field of operations research has long studied scheduling problems, but is generally oriented toward finding and optimizing feasible schedules, with a side effect of failure when no such schedule exists. We were unable to find any work in this area related to inconsistency analysis and optimizing finding inconsistencies.

1.6. *Overview of the rest of the paper*

The remainder of this paper is organized as follows:

- The problem is more formally defined.
- Some initial analysis is performed on the problem.
- Analysis is provided to determine whether a given order is consistent with travel time constraints.
- Preprocessing to derive tighter scheduling constraints is examined.
- Cost of processing pairs of scheduling constraints is examined for algorithmic complexity.
- Separation into independent subsets of scheduling constraints is identified to reduce high complexity situations and avoid intentional complexity-increasing attacks.
- Localizing inconsistencies is examined.
- Searching for an acceptable order is analyzed and examined.
- Evaluating search nodes is detailed.
- We summarize, conclude, and identify further work.

2. Problem definition

We are given a (finite) set of records, each indicating that:

- a given atomic object
- was at a given **location**
- at some time specified to be in a given **time interval**, i.e., at or between two given points in time.

We will refer to the earlier of the two times as the **start** time and the other one as the **end** time. We will refer to these records as **scheduling constraints**.

We are also given a minimal travel time for each ordered pair of locations. The minimal travel times are required to satisfy the following requirements (and we will assume that they do).

- the time is zero from any location to itself
- the time is positive from any location to any other location
- times satisfy the triangle inequality[16]

Minimal travel times are not required to be symmetric. For instance it might take longer to go up hill than down hill.

Atomic objects are not supposed to be in two disjoint locations at the same time (this is why we call them atomic objects), and are supposed to behave in accordance with the following travel time constraint (TTC):

If all of these conditions hold:

- atomic object A is at location L_1 at time T_1,
- A is at location L_2 at time T_2,
- $T_1 \leq T_2$,
- the minimal travel time from L_1 to L_2 is TT

then $TT \leq T_2\text{-}T_1$

Now that the input has been described, we turn to the output and its relation to the input. Our goal is to check whether the set of scheduling constraints is **consistent** with TTC. By this we mean that there is some **schedule** (defined below) which both **satisfies** the set of scheduling constraints (also defined below) and also **satisfies** TTC (defined below).

In order to formalize the notions above we define appearances. An **appearance** is like a scheduling constraint, except that the time interval is replaced with a single point in time. It indicates that a given atomic object was at a given location at a given point in time.

We say that an appearance, *a*, **satisfies** a scheduling constraint, *c*, if

[15] Garey, M. R.; Johnson, D. S. (1979), "A2.3: ND22-24", Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, pp. 211–212, ISBN 0-7167-104.

[16] David E. Joyce (1997). "Euclid's elements, Book 1, Proposition 20". Euclid's elements. Dept. Math and Computer Science, Clark University. Retrieved 2010-06-25.

- the atomic object of *a* is the same as the atomic object of *c*,
- the location of *a* is the same as the location of *c*,
- the time of *a* is in the time interval of *c*

A **schedule** is a set of appearances. We say that a schedule **satisfies** a set of scheduling constraints if for each scheduling constraint, *c*, in the set, the schedule contains some appearance, *a*, such that *a* satisfies *c*.

We will say that a schedule **violates** TTC if it contains a pair of appearances that require the same atomic object to be in two locations at two times where the difference between the two times is less than the minimal travel time from the location at the earlier time to the location at the later time. This condition is simply the negation of the TTC definition above, where an appearance of atomic object A at time T at location L is interpreted as "A is at location L at time T".

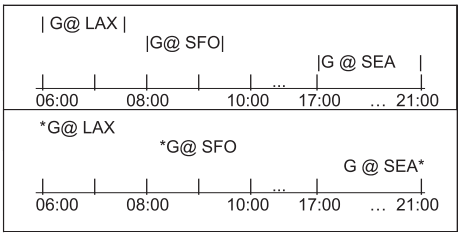
If a schedule does not contain such a pair of appearances we will say that it **satisfies** TTC.

As an example, a set of scheduling constraints is

- George is at LAX[17] some time between 06:00[18] and 07:30[19]
- George is at SFO[20] some time between 08:00 and 09:30
- George is at SEA[21] some time between 17:00 and 21:00

A schedule satisfying that set of scheduling constraints is

- George is at LAX at 06:00
- George is at SFO at 08:15
- George is at SEA at 21:00



This schedule would satisfy TTC iff all of these hold

- the minimal time from LAX to SFO is less or equal to 02:15[22]
- the minimal time from SFO to SEA is less or equal to 12:45
- the minimal time from LAX to SEA is less or equal to 15:00

[17] Los Angeles International Airport covers about 640 acres and is located about 16 miles West from downtown Los Angeles, California.

[18] Real times in this paper are in military notation (4 integers 00:00–23:59) in Pacific time on the same date with the actual date assumed irrelevant.

[19] A quick lookup indicates that there is a United Airlines flight from LAX at 06:02 arriving SFO 07:22.

[20] San Francisco International Airport covers about 150 acres and is located about 30 miles south of downtown San Francisco, California.

[21] Seattle/Tacoma International Airport is part of the Port of Seattle, which sits on about 1543 acres of waterfront and nearby properties between Seattle and Tacoma Washington.

[22] Differential times in this paper are in HH:MM notation.

3. Some initial analysis

A few facts that may seem obvious are still so useful as to be worth mentioning:

A violation of TTC involves only a single atomic object. (This is the point of "without interactions" in the title of the paper.) Therefore all atomic objects can be analyzed independently. For the remainder of this paper we will assume that the set of scheduling constraints has been sorted into sets by atomic object and we are dealing only with the set of scheduling constraints for one atomic object. Similarly, when we discuss a set of appearances, we will assume they all deal with the same atomic object, which is the one in all of the scheduling constraints under consideration.

If a schedule satisfies TTC then the result of deleting any subset of its appearances still satisfies TTC.

Conversely, if a schedule violates TTC then the result of adding appearances also violates TTC.

If a set of scheduling constraints is consistent with TTC then so is any subset of that set. Conversely, if a set of scheduling constraints is not consistent with TTC then every superset of that set is also not consistent with TTC.

It should be clear at this point how we can test whether a schedule satisfies TTC. However, a straight forward translation of the definition of a schedule satisfying TTC would require checking every pair of appearances, as shown in the example above. It is actually adequate to sort the appearances by time and test only the adjacent pairs. This is because of the triangle inequality. If there is any pair of appearances that violates TTC, then some adjacent pair must also violate TTC. This is important for our purposes because it reduces the cost of the check from quadratic in the number of appearances to n log(n) for sorting plus another linear pass for checking adjacent pairs.

Suppose an atomic object is scheduled to be in three locations,
L_1, L_2, L_3, at three times,
$T_1 < T_2 < T_3$.
We will refer to the minimal travel time from L_i to L_j as T_{ij}.
Suppose that TTC is satisfied for the adjacent pairs:
$T_{12} \leq T_2\text{-}T_1$, $T_{23} \leq T_3\text{-}T_2$
Then it follows (by adding the inequalities) that:
$T_{12}+T_{23} \leq (T_2\text{-}T_1) + (T_3\text{-}T_2)$
But the triangle inequality says that:
$T_{13} \leq T_{12}+T_{23} \leq (T_2\text{-}T_1) + (T_3\text{-}T_2)$
Replacing $(T_2\text{-}T_1) + (T_3\text{-}T_2)$ with $T_3\text{-}T_1$ then gives TTC for the non-adjacent pair of appearances.

A straight forward interpretation of our problem statement presents another more serious problem: there are infinitely many schedules that satisfy any set of scheduling constraints, for several reasons. Even if we can efficiently check one, we can't check them all.

The first reason there are infinitely many schedules is that a schedule could have more appearances than it needs just in

order to satisfy the given set of scheduling constraints. However, if there is such a schedule that satisfies TTC, then discarding the unnecessary appearances will result in another schedule that still satisfies TTC. Therefore, in order to determine whether there is a schedule that satisfies the set of scheduling constraints and also satisfies TTC, it is not necessary to consider schedules containing extra appearances. More specifically, for any set of scheduling constraints, C, if there is any schedule satisfying both C and TTC, then there is a schedule, S, also satisfying both C and TTC, which has the additional property that there is a bijection[23] between the scheduling constraints of C and the appearances of S which maps each scheduling constraint *c* in C to an appearance *a* in S where *a* satisfies *c*. In order to construct a schedule satisfying a set of scheduling constraints, C, we will construct for each *c* in C one appearance satisfying *c*.[24]

The other source of infinitely many schedules is the fact that, at least in the normal interpretation of time, there are infinitely many points in time in most intervals (the exceptions being intervals of size zero). This problem is solved as follows: given an **order**, c_1, c_2, … for the set of scheduling constraints, C, it is possible to determine whether that particular order is **consistent** with TTC, by which we mean that there is a schedule S that not only satisfies C and satisfies TTC, but also has the property that the times of the appearances of S are in the same order as the ordering of C. That is, for any two scheduling constraints c_i and c_j, if $i < j$, t_i is the time of the appearance satisfying c_i and t_j is the time of the appearance satisfying c_j, then $t_i \leq t_j$. Since we are dealing with finite sets of scheduling constraints, there are only finitely many orderings, so at worst we can check each ordering, and if none is consistent with TTC we will know that the set of scheduling constraints is not consistent with TTC.

[23] See http://en.wikipedia.org/wiki/Bijection or one of the many references it cites: "This topic is a basic concept in set theory and can be found in any text which includes an introduction to set theory. Almost all texts that deal with an introduction to writing proofs will include a section on set theory, so the topic may be found in any of these: … Wolf (1998). Proof, Logic and Conjecture: A Mathematician's Toolbox. Freeman. Sundstrom (2003). Mathematical Reasoning: Writing and Proof. Prentice-Hall. Smith; Eggen; St.Andre (2006). A Transition to Advanced Mathematics (6th Ed.). Thomson (Brooks/Cole). Schumacher (1996). Chapter Zero: Fundamental Notions of Abstract Mathematics. Addison–Wesley. O'Leary (2003). The Structure of Proof: With Logic and Set Theory. Prentice-Hall. Morash. Bridge to Abstract Mathematics. Random House. Maddox (2002). Mathematical Thinking and Writing. Harcourt/Academic Press. Lay (2001). Analysis with an introduction to proof. Prentice Hall. Gilbert; Vanstone (2005). An Introduction to Mathematical Thinking. Pearson Prentice-Hall. Fletcher; Patty. Foundations of Higher Mathematics. PWS-Kent. Iglewicz; Stoyle. An Introduction to Mathematical Reasoning. MacMillan. Devlin, Keith (2004). Sets, Functions, and Logic: An Introduction to Abstract Mathematics. Chapman & Hall/CRC Press. D'Angelo; West (2000). Mathematical Thinking: Problem Solving and Proofs. Prentice Hall. Cupillari. The Nuts and Bolts of Proofs. Wadsworth. Bond. Introduction to Abstract Mathematics. Brooks/Cole. Barnier; Feldman (2000). Introduction to Advanced Mathematics. Prentice Hall. Ash. A Primer of Abstract Mathematics. MAA."

[24] Note that a schedule may have two appearances that have the same atomic object, location and time. Similarly we could be given two identical scheduling constraints which we would view as separate constraints by virtue of being presented as different records. It is possible for one appearance to satisfy two scheduling constraints, but again we need not look for schedules containing fewer appearances than the number of scheduling constraints because if there is such a schedule, there will also be one of the expected size, the result of duplicating the appropriate appearances.

4. Determining whether a given order is consistent with TTC

Given an ordered set of scheduling constraints, along with minimal travel time data, if there is a schedule of appearances which satisfies TTC and also has the property that the i'th appearance satisfies the i'th scheduling constraint, then one such schedule is produced by the algorithm below. This algorithm constructs the schedule in which every appearance is as early as possible:

- For the first scheduling constraint in the order, c_1, create an appearance where the time is the start time of c_1.
- As long as there are more scheduling constraints to schedule, do the following to schedule the appearance for the next scheduling constraint, c_{i+1}, using the time just assigned to the appearance for the last one scheduled, c_i:

  Find the minimal travel time from the location of c_i to the location of c_{i+1}.
  Add that to the time just scheduled for c_i.
  If this result is later than the end time of c_{i+1}
  then there is no schedule (with this order) that satisfies TTC
  Otherwise,
  assign as the time for the appearance for c_{i+1}, the later of the computed result above and the start time of c_{i+1}.

A similar procedure could be used starting from the last appearance to construct a schedule in which every appearance is as late as possible.

At this point we have an algorithm that, as far as anyone currently knows, cannot be significantly improved in the worst case: iterate over all orders of the scheduling constraints checking for an order that is consistent with TTC. However, this can be improved a great deal in the cases we expect to encounter. In particular, we describe the following optimizations below:

- separation of a sequence of scheduling constraints into independent subsets
- ways to "improve" scheduling constraints
- ways to prune the search

5. Preprocessing to derive tighter scheduling constraints

A single scheduling constraint cannot be inconsistent with travel time data. A pair of scheduling constraints can be. If

there were a good chance of finding a pair of inconsistent scheduling constraints (a property of the data source) then it might be worth while to test consistency of pairs as a way to avoid other work. We have other reasons for considering pairs of scheduling constraints. In addition to being either consistent or inconsistent with TTC, a pair of scheduling constraints, in conjunction with TTC, can imply additional constraints that we will find useful below. As usual in this problem, how useful it actually turns out to be is a property of the data source.

Consider the following two scheduling constraints:

- *George is at LAX some time between 06:30 and 07:00*
- *George is at SFO some time between 07:30 and 08:30.*

Suppose that it takes at least 01:20[25] to get from LAX to SFO.[26] If George left LAX as early as possible, 06:30, he wouldn't get to SFO until 07:50. Therefore we are justified in claiming

- George is at SFO some time between 07:50 and 08:30

This is just a stronger version of the original scheduling constraint involving SFO (being at SFO some time between 07:50 and 08:30 implies being there some time between 07:30 and 08:30), so we can simply replace the original version with the new one. That is, the set of schedules that satisfy the new set of scheduling constraints will be the same as the set of schedules that satisfy the old set. Tighter scheduling constraints better constrain the search for an acceptable schedule in section "searching for an acceptable order" and thereby lead to more efficient search.

Similarly, in order to get to SFO as late as possible, 08:30, George would have to leave LAX by 07:10. This justifies the claim

- George is at LAX some time between 06:30 and 07:10

This is a stronger version of the LAX scheduling constraint, and again replacing the old one with the new one does not affect the set of schedules but may result in more efficient search later on.

More generally, when we consider two scheduling constraints, c_1 and c_2, in conjunction with TTC, we consider two different questions:

- Is it possible for c_1 to precede c_2? That is, would starting at the start time of c_1 from the location of c_1 and then traveling as quickly as possible to the location of c_2 get there before the end time for c_2?
- Is it possible for c_2 to precede c_1? (The translation is analogous to above.)

If neither is possible, then we know that the two scheduling constraints are inconsistent with TTC, and therefore so is the larger set of scheduling constraints. If only one order is possible, then it may be possible to derive some tighter versions of one or both scheduling constraints, as illustrated above. In particular, in the case where the time intervals of the two scheduling constraints overlap, if it turns out that TTC requires one to precede the other, the time intervals will no longer overlap in the tighter versions. One advantage of using the tighter versions is that in certain subsequent processing it will be more immediately evident that one has to precede the other.

Finally, it is possible that, even including TTC, the two scheduling constraints could be satisfied by appearances in either order. This is the normal case in the traveling salesman problem, e.g.,

- *George is at LAX some time between 0:900 and 18:00*
- *George is at SFO some time between 09:00 and 18:00*

and it takes at least 1:20 to get either from LAX to SFO or back.

In this case it is still possible to derive a stronger result, namely

- Either
  George is at LAX some time between 09:00 and 16:40 and at SFO some time between 10:20 and 18:00
  Or
  George is at SFO some time between 09:00 and 16:40 and at LAX some time between 10:20 and 18:00

However, this does not have the same form as our scheduling constraints, and reasoning with disjunctions tends to lead to exponential complexity. Therefore, at present we do not attempt to use such results.

It is also possible to derive additional constraints from triples, quadruples, or more generally n-tuples of scheduling constraints. In particular, for any size, n, it is possible for a subset of size n to be inconsistent with TTC, even though every subset of size less than n is consistent with TTC. Of course, the cost of checking n-tuples increases more than linearly in n.[27] In any case, our current implementation does not go beyond pairs.

6. Cost of processing pairs of scheduling constraints

Processing every pair of scheduling constraints would take time at least quadratic in the number of scheduling constraints. However, in cases of interest, most of this cost can normally be avoided. In particular, the processing described above does no good for two scheduling constraints, c_1 and c_2, if the end time of c_1 is earlier than the start time of c_2 by more than the minimal time required for getting from the location

[25] Current air traffic schedules indicate 1:20 minimum nonstop departure to arrival.

[26] Distance runway to runway is 338 miles per http://www.webflyer.com/travel/mileage_calculator/.

[27] The diagonal stripe argument used below for pairs could be applied here to argue that the number of triples, quadruples, or more generally n-tuples that we need to process is actually linear in the number of scheduling constraints. Furthermore, the cost of processing an n-tuple, while more than the cost of processing an $(n-1)$-tuple, is still constant.

of c_1 to the location of c_2. It may be complicated to arrange to spend time proportional to the number of pairs that are within their travel times, but it is relatively easy to take advantage of a simpler version of this observation: Nothing is gained by processing a pair in which the time intervals are separated by more than the maximum of all minimal travel times.

Suppose, for example, that our data source is a security system that records when people enter and leave various areas within a facility. Suppose the maximum of all minimal travel times is 1 h, i.e., you can get from any place to any other place within an hour. Now suppose we are given a set of 1000 records (scheduling constraints) for one person over a month, an average of 50 records per day over 20 work days, mostly concentrated during 8 work hours per day. Most records in this case are within an hour of only about a dozen other records, far fewer than the 1000 total records. If we imagine a square matrix in which the rows and columns are each labeled with the scheduling constraints, instead of processing the entire matrix we are now processing a diagonal stripe. The width of that stripe is the number of scheduling constraints (for a given atomic item) that occur within the maximum minimal travel time of a given scheduling event. This we regard as approximately linear in the number of scheduling constraints, at least for the types of problems we expect.

In terms of an algorithm to take advantage of this observation, suppose we wish to iterate over all pairs of scheduling constraints, c_1, c_2 where the time intervals of c_1 and c_2 are separated by less than some time, T:

```
1. create an array A containing the scheduling constraints
2. sort A by start time
3. for i from 1 to A.length do
     for j from i+1 to A.length while A[i].end+T <= A[j].start
     do
          processpair(A[i],A[j])
```

The for while syntax indicates that the loop should end when either the index reaches the limit or the test fails.

The cost of sorting A is presumably on the order of n log(n) where n is the number of scheduling constraints. This algorithm only processes pairs c_1 and c_2 where c_1 is earlier in the sort order than c_2. In addition to the actual processing of pairs, it has to iterate over and test at most one more j value than will end up actually being processed for each i value.

Now comes some less pleasant news: Whenever we derive a tighter interval for a scheduling constraint, it would be useful to re-process relevant pairs involving that scheduling constraint so as to find further improvements of intervals. The order in which these pairs are processed will generally affect the amount of work involved. If processing a pair updates the time interval of a scheduling constraint that is in use by the algorithm above, then subsequent use of that scheduling constraint already uses the improved time interval. In general, the algorithm above tends to propagate forward in time the updates that increase start times, e.g., the travel time between the first two locations will result in increasing the start time at the second location, and that new start time, in conjunction with the travel time from the second location to the third will affect the start time for the third location. Intuitively, this is due to the fact that earlier pairs are processed before later pairs.

However the propagation of the other type of update, decreasing the end times, is very poorly handled by that algorithm. For that purpose it would be better to use a mirror image version of the algorithm that processes later pairs before earlier ones. Our current implementation alternates between the forward and backward algorithms until an entire pass through the data fails to improve any bounds. If the scheduling constraints were linearly ordered, it would seem that one pass in each direction should be enough, but of course this is not true in general.[28]

The improvement of time intervals in one pass can reduce the number of pairs processed in later passes, since it may turn out that a pair that was previously separated by less than T is now separated by more. Furthermore, since the improved bounds also improve the ordering of the scheduling constraints, the propagation of constraints in one direction or the other tends to be improved.

We do not currently have good bounds on the number of passes that can be required, but we expect that more than two will be unusual (forward, backward and then another forward to verify that no new results are discovered).

7. Separation into independent subsets of scheduling constraints

For any problem with high complexity, it is clearly advantageous if the problem can be separated into smaller pieces that can be solved independently, at least if it's not too difficult to recombine the solutions for the pieces into a solution for the original problem. One obvious example of separation is that the scheduling constraints involving different atomic objects can be processed independently. Another way to separate a set of scheduling constraints uses the previously described maximum of all minimal travel times. In our earlier example, we expect that a person will be observed at various places during the work day, but not be observed at all from the end of the work day until the beginning of the next work day, a time greater than the hour required to get from any known location to any other. This allows us to independently test the consistency of his set of scheduling constraints for each day, thereby replacing one problem of size 1000 with 20 problems of size 50, or perhaps

[28] Note that the algorithm above relies on start times remaining sorted (and the mirror image algorithm relies on end times being sorted) to determine when to terminate the inner loop, so we don't want these changed by processpair. The current implementation makes copies of the original start and end times of the scheduling constraints and uses the copies to control the loop. The "real" start and end times are altered by processpair and these values are propagated. At the beginning of each forward or backward pass the scheduling constraints must be resorted, since the updates to their intervals affects the order. At this time the updated interval data is copied again so that it can be used to terminate the inner loop in the next pass.

even further if he stays in one place for more than an hour at a time.

An algorithm for separating scheduling constraints into subsets separated by at least time T is straight forward:

1. For each scheduling constraint, create a start time record and an end time record (the start time record contains the start time of the scheduling constraint plus the fact that it is a start time, and similarly for the end time record)
2. Create a single array containing all of these records and sort it by time
3. initialize the variable C to 0 (C counts the scheduling constraints in progress)
4. iterate through the array from smallest time to largest, at each step doing:
   - if the next entry is a start record, add 1 to C
   - if the next entry is a stop record, subtract 1 from C (C should never be negative)
   - Whenever C is set (decremented) to 0, compute the difference between the (end) time of the record that caused the decrement to the (start) time of the next record. If this difference is more than T, then the set can be separated by the time interval between the (stop) time of this record and the (start) time of the next record.

8. Localizing inconsistencies

Up to here we have presented the goal as simply determining whether a set of scheduling constraints is consistent with TTC. In reality, if the set is inconsistent, then further information is generally of interest. To the extent possible, we wish to present to humans some explanation of the inconsistency. To begin with, which atomic object has an inconsistent set of scheduling constraints? Furthermore, if more than one such atomic object exists, it will be of interest to identify all of them. That is, we should not stop after the first inconsistency is found. Even for a given atomic object, the separation into subsets of scheduling constraints that are separated by the maximum minimal travel time is useful in that it will be of interest to separately report different subsets that are inconsistent.

In the case where a search shows that there is no consistent schedule, the explanation is easy to understand in some abstract sense, but difficult to understand in the sense of a succinct (and yet still compelling) argument. Human consumers of such information would always prefer a succinct argument. Although such an argument is not available in general, the case where two scheduling constraints prove to be inconsistent with TTC is an exception where we can provide such an explanation, and we wish to do so. This is another reason for the preprocessing phase.

Even after finding an inconsistent pair in a subset of scheduling constraints that can no longer be separated, it would be useful to continue to search for "unrelated" inconsistencies. Our current implementation skips any further processing of scheduling constraints sorted between the two found to be inconsistent. In particular, if processpair(A[i],A[j]) discovers an inconsistency, the response is to exit the inner loop and resume the outer loop with i set to j.

9. Searching for an acceptable order

When a set of scheduling constraints has been separated as far as possible into independent subsets, we still have the problem of testing a subset for consistency with TTC. This is done by searching for an ordering for that set of scheduling constraints that is consistent with TTC. In the worst case, of course, this might end up testing all n! orders of n scheduling constraints. As noted above, it is reasonable to limit the effort in such a process and report a result of failure due to high complexity when that limit is exceeded. In the best case, only one order is consistent with the time intervals of the scheduling constraints, independent of TTC. One would hope that in such cases the effort involved would be approximately proportional to the number of scheduling constraints. By making the limit (approximately) proportional to the number of scheduling constraints we can keep the cost of the processing (approximately) proportional to the number of scheduling constraints, while still finding acceptable schedules in the normal cases. The algorithms below create new search nodes in constant time, and the number of search nodes created for a linearly ordered set of n scheduling constraints is n (or possibly less if they are not consistent with TTC). Of course, all of this ultimately depends on the actual data that arrives. If the real world does not conform to our expectations then we will either have to accept longer processing times or more failures due to high complexity than we hope for. Note, however, that the frequency of high complexity failures is also highly dependent on the effectiveness of the algorithms that control the search, which are described below.

Our current implementation does a depth first search for an ordering on scheduling constraints, where each node in the search tree represents an initial segment of the order and each child of a search node represents the result of adding one scheduling constraint to the end of the segment represented by its parent. It should be clear that if the end time of scheduling constraint c_1 is earlier than the start time of scheduling constraint c_2, then c_1 must be scheduled earlier than c_2. This can be incorporated into the depth first search as follows: given a search node, limit the set of child nodes to those that assign as the next scheduling constraint, c_1, those scheduling constraints for which there is no other scheduling constraint remaining to be added, c_2, where the end time of c_2 is earlier than the start time of c_1. In the case where the scheduling constraints are linearly ordered (where no intervals overlap), this leads to a linear search, at least in terms of the number of search nodes created.

However, this does not mean that the entire process takes linear time. If it takes time proportional to n to find the single allowable next of n scheduling constraints, then creating n search nodes will have taken time proportional to n^2. The current implementation arranges to find each of the allowable next scheduling constraints in constant time by maintaining a representation of the set of scheduling constraints

still to be added in a data structure that contains two sorted lists of the scheduling constraints, one sorted by start time and the other by end time. In order to iterate over the set of allowable next scheduling constraints, one first finds the earliest end time, i.e., the end time of the first in the list of scheduling constraints ordered by end time. Then one iterates through the list of scheduling constraints ordered by start time, stopping when the start time exceeds the earliest end time. In order to maintain these lists incrementally, they can be represented by doubly linked lists, where deleting an element can be done in constant time. This involves following the forward and backward links from that element and then making those previous and next elements point to each other.

Moving deeper into the search tree involves removing a scheduling constraint from the set remaining to be scheduled, which is done in constant time, but backtracking to an earlier state requires adding a scheduling constraint back to that data structure. This would seem to involve finding its start and end times in the sorted lists, which would take at least log(n) time and even that would require a more complicated data structure than the linked list. That would be the case if we were adding an arbitrary scheduling constraint to the set, but recall that we are actually only returning to a previous state. Given that the new state was reached by splicing this element out of the list we can simply record the data needed to undo that operation. Saving and using that data takes a (small) constant amount of time and space.

The current implementation manages to avoid even the small task of allocating space and filling it with undo data on the way into the search tree. The representation of the ordered lists created at the start of the search involves a few arrays. Suppose we have n scheduling constraints identified by numbers 1 through n, and that given an index, i, we can find scheduling constraint i in constant time. When we sort these scheduling constraints by start time we get some permutation, e.g., 4 3 5 2 1.

We will add to this two more parallel arrays for next and previous pointers. The next pointer for the last element and the previous pointer for the first element will be zero. We will also add some variables to hold the index of the first and last elements:

```
element  4 3 5 2 1
   next  2 3 4 5 0   first = 1
   prev  0 1 2 3 4   last = 5
```

In other words, the first element is the one at index 1 (the value of first), and at index 1 (the left most column of numbers) we see that this is scheduling constraint 4, the next element (in this case the one with next larger start time) is the one at index 2, and the previous element is the one at index 0 (meaning that there isn't any previous). Removing the element at index i is just splicing it out of the list:

```
nextind = next[i]
prevind = prev[i]
if prevind==0 then first=nextind else next[prevind]=nextind
if nextind==0 then last=prevind else prev[nextind]=prevind
```

So, for example, the result of removing the second element of this array (which happens to be the scheduling constraint identified as element 3) would be

```
element  4 3 5 2 1
   next  3 3 4 5 0   first = 1
   prev  0 1 1 3 4   last = 5
```

Notice that we have not changed the pointers associated with element 2 itself. These now serve as our undo information! In order to reverse the change we simply look at these pointers to see which elements were the ones before and after this element. We want the one that was before to point to this one as its next element, and the one that was after to point to this one as its previous. So to undo the removal of element i:

```
nextind = next[i]
prevind = prev[i]
if prevind==0 then first=i else next[prevind]=i
if nextind==0 then last=i else prev[nextind]=i
```

10. Evaluating search nodes

When we add a scheduling constraint to a sequence of earlier scheduling constraints we have to check that the result is consistent with TTC. If it is not, then clearly this search node can be pruned. We saw earlier how the entire sequence could be checked in linear time. Now we want to check it incrementally in constant time for a single addition. Fortunately, the entire sequence was tested by considering the elements in order, and we can arrange to do that incrementally as we traverse the search tree. As we add scheduling constraints to the sequence we keep track of the earliest start time of each one due to the earlier ones. As an example, consider the following path through a search tree:

- George is at LAX some time between 06:00 and 07:00
- George is at SFO some time between 07:00 and 09:00
- George is at SEA some time between 09:00 and 10:30

If the minimal travel time from LAX to SFO is 01:20 we should record when we add SFO that it cannot be reached until at least 07:20. At this point we can also test whether SFO had to be reached before that time, and if so, prune this path through the search tree. Otherwise, this information only has to be available whenever we try to extend the sequence by adding an additional scheduling constraint after SFO. If the minimal travel time from SFO to SEA[29] is 02:00[30] we would record when we add SEA that it cannot be reached until at least 09:20, and so on.[31]

If we find that we cannot reach SEA in time from SFO, one might expect that a depth first search would then resume by

[29] Distance runway to runway is 678 miles per http://www.webflyer.com/travel/mileage_calculator/.

[30] Current air schedules indicate a minimum travel time of 1:50 nonstop departure to arrival.

[31] Again, looking at commercial schedules, the next available flight is SFO 8:35AM→SEA 10:35AM.

selecting an alternative scheduling constraint to try adding after SFO. However this cannot succeed. By failing to reach SEA in time from SFO we have actually shown that the schedule up to SFO is not going to work, and we should therefore seek an alternative to SFO as the location to visit after LAX. The reasoning that justifies this conclusion is that if we don't visit SEA immediately after SFO, we still have to visit it later, and as we have shown earlier, visiting some other location first cannot help us get to SEA in time.

This suggests that after finding that the sequence LAX–SFO–SEA is inconsistent with TTC, we could remember that and recognize any other sequence in which this appears as a subsequence. For instance, if we decide to change SFO to DEN,[32] we should not then try to go from DEN to SFO[33] since we would still have to visit SEA and the resulting schedule would then include as a subsequence the impossible LAX–SFO–SEA. It is not clear how much it would cost to try to record and use such forbidden subsequence results, nor is it clear how much good it would do in the cases of interest. The current implementation does not try to do this.

11. Summary, conclusions, and future work

In seeking more generic and higher computational leverage approaches to detecting subversions of systems, the area of consistency analysis has emerged as one of the potential areas for detecting a broader spectrum of attack activities with less computation than alternative approaches, and doing so in such a manner as to make it very difficult for an attacker to subvert a system without being detectable.

This paper describes a near-linear time O(n log(n)) solution to a subset of consistency problems. In particular, we address time and space interval record schedule consistency analysis for atomic items without interactions in open spaces with stationary locations. The same algorithms may be used to solve related problems in more time, but we expect the general case to take exponential time. We also expect that similar methods will generalize to closed spaces with interactions and non-stationary locations, and these will be the focus of future work.

A number of simplifications have been made in the model described here and it might be useful to extend results to a broader set of situations:

- We currently model locations as single points in space, but it might be more accurate to use regions of space just as we use intervals of time. This was not done here because it leads to problems in the meaning of minimal travel time.[34] Thus the necessary assumption that regions of space associated with locations are small relative to travel times at issue is used above.
- We currently model locations as stationary points in space. But many records are associated with non-stationary locations, such as ships, cars, trains, and planes.
- In the current model, all timestamps are independently translated into intervals in a common time base. But in cases where timestamps come from the same or otherwise related clocks, we may know their difference to much higher accuracy than we do from unrelated clocks. Thus we may be able to determine that a set of records otherwise undetectable as inconsistent to be inconsistent by using the associated additional constraints on intervals.[35]

In addition to relaxing these simplifications, substantial future work is required in order to deal with the more general related problem set. In particular, time and space Interval record schedule consistency analysis for atomic and non-atomic items with Interactions in open and closed spaces with stationary and non-stationary locations.

The current implementation and approach are already useful in questioned document examination of digital records for two purposes.

- It identifies sets of records that are unreliable and isolates them from other records that are not demonstrably unreliable by these methods, thus "sealing the breech" of unreliable records it identifies while demonstrating that something is wrong with those records, and
- It produces feasible event sequences for records not shown to be unreliable by these methods, thus producing a basis for further investigation. In addition, through investigation, additional records or facts may be identified and added to the analysis so as to further reduce the number of feasible schedules and better particularize the event sequence and time intervals at issue.

Another beneficial side effect of the particularization process is that it reduces the investigative search and asserted claim spaces by reducing the intervals of records and locations of items with added data. Thus as more and more details are identified and added, the time frames for each item and event become tighter and tighter, leading to potentially smaller and smaller sets of suspected activities, interactions, alternative claims of event sequences, and potential leads. Inaccurate statements and claims are harder and harder to claim without violating a constraint, and testing such claims against the current body of constraints becomes more efficient. For the investigator going into an interview, the

[32] Distance runway to runway is 860 miles (LAX→DEN) and 1020 miles (SEA→DEN) per http://www.webflyer.com/travel/mileage_calculator/.

[33] Distance runway to runway is 965 miles per http://www.webflyer.com/travel/mileage_calculator/.

[34] For example, if records indicate that John is in Pasadena at 12:01 and Los Angeles at 12:02, this seems fine because they have a common border. Likewise, if records indicate that John is in Los Angeles at 12:02 and Santa Monica at 12:03, that seems fine because they also have a common border. But there are no places in Santa Monica that can be reached in 2 min from any places in Pasadena by people. This violates the triangle inequality, which is the basis for reducing the complexity in our first example from quadratic to linear.

[35] For example, suppose Joe is recorded at $location_1$ at 12:05 and again at 12:15, and at $location_2$ at 12:10. If all of the timestamps are known to have errors of less than 5 min, then minimal TT between the two locations of 10 min would be consistent, since the first appearance at $location_1$ could be at 12:00 and the second at 12:20. But if we know that the two timestamps at $location_1$ are measured by related clocks with differentials less than 1 s per day (e.g., if they use NTP synchronization), then this is not possible.

additional knowledge presents improved opportunities for detecting falsehood, jogging of memories, and identifying lines of inquiry.

It has been suggested that an attack against such a system would be to intentionally create conditions driving the complexity of algorithms high so as to cause them to be unable to perform timely detection. In our experience, such high complexity cases are readily identified by long algorithm run times, and if the run times are sufficiently long to be highly variant from normal system operation, this too is an indicator of the system operating inconsistently with its normal operation. Thus the attack against the mechanism self-indicates that the mechanism and thus the system are being attacked. Avoiding detection in this way is detectable.

For the more distant future, there is the more general problem of moving from schedule analysis to a more general model of location over time for sets of interacting objects. This is similar to the sorts of challenges faced in optimal robot path planning, where paths of multiple manipulators starting at a given set of places, times, and poses and ending at another set of places and poses within a 4-dimensional time-space avoid collisions and facilitate hand-offs of workpieces from manipulator to manipulator. The notional forensic equivalent would be the presentation of a multi-dimensional depiction of all objects and all possible interactions with the ability to add constraints and see the implications. For example, imagine an analysis fusing together all relevant records from all available sources to reconstruct all of the actors and actions related to a criminal act from the initial idea formation and planning through committing the crime and subsequent attempts to get away. While we currently believe that this problem will be intractable, we also speculate that large and useful subsets of this problem may be tractable under some reasonable assumptions.

Fred Cohen is best known as the person who defined the term "computer virus" and the inventor of most of the widely used computer virus defense techniques, the principal investigator whose team defined the information assurance problem as it relates to critical infrastructure protection, as a seminal researcher in the use of deception for information protection, as a leader in advancing the science of digital forensic evidence examination, and as a top flight information protection consultant and industry analyst.

Don Cohen has done work in artificial intelligence, automatic programming and computer security, including attribution, packet flooding defense and attack detection.