Insider Threat Identification by Process Analysis

Matt Bishop*, Heather M. Conboy†, Huong Phan†, Borislava I. Simidchieva†,
George S. Avrunin†, Lori A. Clarke†, Leon J. Osterweil†, Sean Peisert*‡

*Dept. of Computer Science, University of California at Davis, Davis, CA 95616-8562;
Email: bishop@ucdavis.edu
†Dept. of Computer Science, University of Massachusetts Amherst, Amherst, MA 01003-9264;
Email: {hconboy, hphan, bis, avrunin, clarke, ljo}@cs.umass.edu
‡Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720;
Email: sppeisert@lbl.gov

***Abstract*—The insider threat is one of the most pernicious in computer security. Traditional approaches typically instrument systems with decoys or intrusion detection mechanisms to detect individuals who abuse their privileges (the quintessential "insider"). Such an attack requires that these agents have access to resources or data in order to corrupt or disclose them. In this work, we examine the application of process modeling and subsequent analyses to the insider problem. With process modeling, we first describe how a process works in formal terms. We then look at the agents who are carrying out particular tasks, perform different analyses to determine how the process can be compromised, and suggest countermeasures that can be incorporated into the process model to improve its resistance to insider attack.**

I. Introduction

The insider problem is one of the most difficult problems in computer security, and indeed in all aspects of real-world security. The ancient Romans even had an aphorism about it: "Who will guard the guards themselves?" Recent events have focused attention on the threats that insiders can pose to the satisfactory performance of key societal systems and processes in domains such as finance, government, and even science. Many reports of insider attacks describe people trusted with access to sensitive information abusing that access to damage that information, compromise the privacy of that information, and collaborate with others (sometimes other insiders) to cause various kinds of failures, losses and serious harm.

Traditional approaches have focused on examining the actions of agents (people with access to the data or resources under consideration). In these approaches the actions themselves are analyzed to find suspicious or unexpected patterns, where agent behavior may be derived from technical logs or external indicators. This paper outlines an alternate approach. Rather than focusing on identifying insider attacks as they occur by monitoring event logs, we use a rigorous, process-based approach to identify places in organizational processes where insider attacks can be successfully launched. In knowing how such attacks might be launched, organizations will often be able to restructure their processes to reduce their exposure to such insider attacks. This knowledge should give organizations insights that will help them to know where to look first if an attack is suspected, and what steps in the process should be considered for more careful activity monitoring (e.g., using audit logging or video surveillance) or restructuring, e.g., to avoid single points of failure or add redundancy.

Thus, we advocate a systematic, well reasoned, and iterative approach to security analysis. We believe that this approach is very general and can be applied in a wide variety of environments and circumstances. It is most appropriate for important, long-lived processes. For such processes, an organization would need to be willing to invest in developing an accurate process model and then continue to update, and hopefully improve that model, as credible vulnerabilities are found, either through experience or process analysis, or a combination of both. Thus, for an initial validation, in this paper we apply our approach to a case study drawn from a critical process domain, namely elections.

For the purposes of this paper we consider a "process" to be the coordination of activities, often involving human agents, software applications, and hardware devices, needed to complete a task or reach a goal. For example, all the activities involved in conducting an election can be defined as a process involving precinct workers, election officials, voters, and voting equipment (computerized or otherwise). Building upon these notions, we informally define an "insider" to be "any agent whose participation in the performance of a process entails having access to data whose abuse or misuse constitutes an attack." A more rigorous definition will be presented later, but note that two key features of this informal definition are that it emphasizes that the notion of "insider" is relative to the process, the environment in which that process takes place, and the way in which the process can fail or be attacked, and the definition removes any notion of intent from the definition of insider. This is consonant with the definition of "threat," as "a potential violation of security" [1, p. 6] rather than an actual or attempted violation.

This paper discusses work that focuses on a range of insider attacks and develops a corresponding range of analyses that can help detect vulnerabilities to such attacks, and often even suggests process improvements to help thwart such attacks. More specifically, this work deals with two (not exclusive) categories of insider attacks:

1) Sabotage attacks, which we characterize informally as actions by insiders that cause the final results of a process to be incorrect. For these kinds of attacks we

DOI 10.1109/SPW.2014.40

consider an insider to be any agent that has write permission for artifacts that are outputs from the process, or are used directly or indirectly in the computation of any of these outputs. In this paper, we discuss how analysis approaches such as Fault Tree Analysis can be used to identify possible sabotage attacks and defend against them.

2) Data exfiltration attacks, which we characterize informally as actions by insiders that cause sensitive data to be made accessible to entities that are not entitled to such access. For these kinds of attacks we consider an insider to be any agent that has read permission for any data items that could be used, possibly in combination with other information, to obtain other data items that could then be made available to agents that are not authorized to have access to those data items. We explore how certain analysis approaches, such as Finite-State Verification, can be used to identify possible data exfiltration attacks and to suggest process improvements to defend against them.

Many current efforts to deal with insider attacks have been dynamic and reactive, and indeed often somewhat *ad hoc*. Some focus on detecting when an insider attack has taken place, or that an attack is currently taking place. Although detection may suggest how to attempt to compensate for the negative effects of an insider attack, this seems less desirable than preventing the attack in the first place. So other approaches try to predict who might launch an attack, based upon extensive psychological and sociological analysis (often embodied in background checks of employees or contractors). Such predictive approaches tend to focus rather narrowly on specific threat environments and seem to be difficult to generalize to alternate environments.

Access control is a common approach to inhibiting insider attacks. But some of the most common access control mechanisms, based upon the identity or the role of an agent, offer only weak protection from insider attacks. Such access control may allow insider agents to have access to critical data in order to perform tasks for which they are specifically authorized, but may not prevent these agents from abusing that authorization to damage or compromise that data. For example, biostatisticians have to be authorized to access the raw data from a drug trial as a necessary, integral part of the process for carrying out a drug company's trial of a new drug. But the biostatisticians are presumably *not* authorized to modify the data in order to make the drug appear to be more (or less) effective than it actually is (a sabotage attack), or to email the data to the drug company's competitor (a data exfiltration attack). Identity-based and role-based access controls are typically incapable of preventing such abuses. Accordingly more complex and powerful forms of access control are being explored. Our work is aimed at complementing that work by recognizing highly vulnerable situations and then considering process improvements that thwart such attacks, or at least make such attacks more difficult.

The work discussed in this paper is based upon rigorous analysis of precise models of the processes that create and manage critical data. Creating those process models is usually a joint effort involving computer scientists and domain experts. In the case study reported here, the domain experts were election officials. To create the process model, we interviewed those experts in considerable depth, extensively observed the elections ourselves, and read a considerable amount of written documentation (such as poll worker training materials) provided by the election officials. From this, we created a first cut at the process model. We then reviewed the model with the domain experts. The reviews often identified areas for expansion or led to other changes, and changing the model sometimes raised questions about the process, or our understanding of the process. We continued to iterate with the domain experts until we obtained the desired level of detail in the model. In this way, the model captures the relevant details about the real, existing process, and the results will be useful to the domain experts who have defined, and often carry out, the process. We are most interested in processes that are collaborations between humans, who perform those parts of the process that require creativity, initiative, and insight, and automated (hardware and software) devices that can perform large and complex data processing tasks rapidly. Of course, both humans and automated devices are capable of attacking the processes in which they participate. But, the effects of such attacks can be mitigated by processes that have been designed to incorporate checking and perhaps double-checking steps at appropriate critical points. The placement of such redundant checking steps is important, but no less important is the assurance that these steps are carried out only by the appropriate agents. Our work builds on existing software and safety engineering analysis techniques to determine whether or not a process adequately defends against these kinds of sabotage and data exfiltration attacks.

Thus, for example, static analysis of a process model might determine that a single insider is capable of damaging critical data, thereby corrupting the final output of the process. This is an example of a single-insider, single-artifact sabotage (SISAS) attack. The risk of this kind of attack can be mitigated by modifying the process to ensure that a different agent (human or automated) will, for all possible performances of the process, always review and verify every data item whose damage could corrupt the final output. Static analysis can also be used to ensure that all such artifacts must indeed always be reviewed in this way. This does not, however, rule out the possibility that the the verifying agent is not also a colluding participant in a multiple-insider, single-artifact sabotage (MISAS) attack. Analytic approaches for detecting and preventing a SISAS attack may be ineffective in dealing with a MISAS attack, and other analytic approaches and process improvements may be needed to detect and prevent such attacks. (For data exfiltration attacks, we similarly define SISADE, SIMADE, MISADE and MIMADE attacks.)

The goal of this work is to demonstrate that appropriate analyses can be used to identify vulnerabilities to relatively straightforward attacks, presumably causing attackers to have

to devise more extensive (and presumably more expensive) collusive attacks. This approach will help provide those entrusted with protecting critical processes from attack with tools that are useful in trying to ensure that any successful attack will cost attackers more than the attack gains them. Of course, this approach does not render a process completely invulnerable to an insider attack. *Any* process can be successfully attacked by a sufficiently large and comprehensive collusion among insider (and possibly also outsider) agents. But applying this approach can force an increase in the number of agents that must collude to launch a successful attack.

II. Technical Background

The essence of our approach is to use static analysis techniques, such as Fault Tree Analysis and Finite-State Verification to determine the possibility of successful insider attacks on a process. To help the reader understand our proposed approach, we first discuss the characteristics that a process model must have to represent complex processes and to support the analyses that we propose. We then define what we mean by an insider and the two types of insider attacks that we are focusing on, insider sabotage attacks and insider data exfiltration attacks.

After introducing key concepts related to Fault Tree Analysis and Finite-State Verification, we discuss how these analysis techniques can support identifying the possibility of insider sabotage and data exfiltration attacks and can be used to suggest defenses against these attacks.

A. Process Models

Our approach assumes the existence of a precise and detailed model of a process that needs to be protected from insider attacks. The approach rests upon analyzing the process model to identify vulnerabilities, thus, the model must be expressed in a process modeling language suitable for such analysis. Such a language must have several characteristics:

1) *Activity Specification*: Specification of the activities that comprise the process as well as control flow semantics that can be used to specify the order of execution of these activities. Our experience suggests that support for specifying iteration, choice, and concurrency is essential, as is support for specifying hierarchical decomposition. Finally, most complex real-world processes have exceptional situations that must be precisely specified and carefully managed.
2) *Artifact Specification*: Specification of the flow of artifacts through the activities. At the minimum the language must be able to specify which artifacts are used as inputs to, and outputs from, each activity.
3) *Agent Specification*: Specification of the types and characteristics of the entities required as the performers of each activity (e.g., the characteristics or qualifications that a human performer, such as an election official, must have, or the type or capabilities that a device, such as a voting machine, must have) and how these entities are assigned to activities. This implicitly assumes the existence at process execution of a repository enumerating the types, characteristics, and qualifications of all agents (human and non-human) that are available for assignment to perform process activities. It is critical to ensure that inappropriate agents cannot be assigned at runtime to the execution of critical process activities.
4) *Precision in Specification*: Specifications of the desired functionality of the process being modeled, at the desired level of detail, in a notation with precise semantics. The analytic results produced by this approach depend directly upon the precision of the process model upon which the analyses are based. Informal or intuitive results are the best that can be expected if the process model is itself imprecise. We advocate using a process specification notation having rigorously defined semantics. In a later section, we show how this enables rigorous and precise analytic results.

B. Insiders and Insider Attacks

The term *insider* has been defined in many different and sometimes inconsistent ways [2]–[11]. Recently, a basic consensus has emerged in which *insider threats* are defined in terms of someone who has some combination of increased access to and/or knowledge of an organization. We focus our process modeling on the *access* aspect of the insider problem because access is a particularly critical capability for the insider. The access can be direct, where the attacker reads or modifies the information directly, or indirect, where the attacker obtains information from an intermediary or has someone else modify the artifact as desired. In some cases, someone with access may be the kind of person classically defined as an insider, being actually inside of some kind of physical perimeter. An insider, however, can also be someone outside such a perimeter, accessing a system remotely (for example, through a virtual private network). Moreover, an insider may pose a threat to a process by accidentally causing harm to the system ("I wonder what this button does?") or being "socially engineered" into unwittingly doing harm to a system ("Hi! I'm from the help desk. I need you to send me your password to fix a system malfunction involving your account").

A process model that precisely specifies the coordination of activities in the process, the artifacts involved in each activity and how they flow through the process, and the agents assigned to perform the activities is well-suited for analysis of access. Assuming the existence of a process model that has the characteristics just enumerated, we can now define "insider" more precisely.

Definition. An *insider* is an activity execution agent who has access to any process artifact that can affect the outcome of the process and for which general access is restricted.

In a sabotage attack, an insider is an activity agent that can modify any artifact that can affect a final process output either directly or indirectly. Thus, a voter in an election is not an insider, because the voter only makes a request for another agent to record the voter's vote. If the request is addressed to

an electronic voting system, which then actually records the vote and adds it to a running tally, then the voting system *is* an insider. Similarly if the voter asks that a human election official properly handle the paper ballot on which the vote is recorded, then the election official is an insider. In both cases the voter is not an insider; but the electronic voting system and election officials involved in running the election and calculating and reporting the final tallies and results are all insiders. Similarly, suppose a biostatistician has emailed the data obtained from a drug trial to the drug company's competitor. The biostatistician was presumably *not* authorized to do this. Thus, by the definition of "insider" above, the biostatistician is an insider. Emailing the results comprises a data exfiltration attack, as it provides access to one of the outcomes of the process, namely the drug trial data, to entities not authorized to have that access.

We now define the types of insider attacks more precisely.
Definition. A *sabotage attack* is a sequence of agent actions that cause a value of a final artifact to be incorrect. A *sabotage attack insider* is an agent that is able to write (change the value of) an artifact used in the computation of any of the final outputs of the process.
Definition. A *data exfiltration attack* is a sequence of agent actions that provide access to process artifacts to entities not entitled to that access. Thus the specification of a data exfiltration attack relies upon a specification of which data entities must be guaranteed to be inaccessible to which entities. A *data exfiltration attack insider* is an agent that has access to any of the artifacts whose access should be restricted.

C. Analysis Technologies

Given the ways in which we have defined insider attacks, we can now describe ways in which two key static analysis technologies, Fault Tree Analysis and Finite-State Verification, can be used to identify vulnerabilities to insider attacks upon processes whose models are specified using a process language having the characteristics described above.

1) Fault Tree Analysis: Fault Tree Analysis (FTA) was first introduced in the 1960s and was used by NASA in evaluating the vulnerability to failure of space shots. Since that time the use of FTA has been adopted by various other industries, such as nuclear power, chemical processing, automotive [12], and health care [13]–[15]. Recently it has also been used experimentally in reasoning about the possibility of failures in election processes [16], [17].

FTA is a deductive, top-down analytical technique that begins with a specification of a *hazard*, namely some state in which a serious accident or undesired outcome is imminent, possibly dependent upon external conditions. Many hazards can be characterized as the arrival of one or more incorrect, unsuitable artifacts as inputs to a particularly critical step. An example of such a hazard is "the wrong type of blood is delivered to the bedside of a transfusion patient." (Note that the definition of a sabotage attack is precisely of this form, namely the arrival of incorrect artifacts at a step that produces as output a final result of the execution of a process.) Given the specification of such a hazard, FTA produces a fault tree, that is a graphical model of the various combinations of events that could possibly lead to the hazard.

Figure 2 in Section III is a depiction of an example fault tree. Note that the depiction is structured as a directed acyclic graph, rather than as a tree, in an attempt to reduce the size, and increase the comprehensibility, of the fault tree. The automated tool that we have used to create fault trees performs optimizations to address these improvements by combining identical subtrees.

A fault tree consists of 2 basic elements: *events* and *gates*. In this figure, events are represented by rounded rectangles. The event at the root (top) of the tree is the hazard. In the example, it is `Artifact ''finalTallies'' to step ''report final vote totals to Secretary of State'' is wrong`. In a fault tree, *intermediate events* are elaborated, while *primary events* are not further elaborated.

Events are connected by gates that permit or inhibit the passage of fault logic up the tree. A gate connects one or more *input events* (below) to a single *output event* (above). There are three types of gates:

- AND gate: the output event occurs only if all input events occur (inputs are assumed to be independent)
- OR gate: the output event occurs if any input event occurs
- NOT gate: the output event occurs only if the one single input event does not occur

Given the existence of a fault tree, simple Boolean algebra can be used to compute *cut sets*, which are sets of *event literals* (primary events or negations of primary events) that could cause the hazard, and *minimal cut sets (MCSs)*, which are cut sets that cannot be further reduced. Thus, the occurrence of all the events contained within an MCS in an execution of the process could cause the hazard at the root of the fault tree to arise. An MCS specifies one or more potential *system vulnerabilities*, which are flaws or weaknesses in a system's design, implementation, operation, or management that could be exploited to allow the hazard to occur.

An MCS of size 1 indicates a single point of failure, namely a single event whose occurrence might cause a hazard to occur. The performer of the activity causing this single event is, by our definition, an insider, and this insider can thus carry out a sabotage attack. Larger MCSs require the occurrence of multiple events and thus require either that the same attacking insider perform all such events or that there is some collusion among the set of all insiders performing the events in the MCS. Thus FTA can be used to suggest changes to a process aimed at ensuring that no single agent can be allowed to perform all the events of an MCS thereby forcing the need for collusion. This presumably makes the attack more difficult and more expensive, thus serving as an example of how our analysis can suggest possible defenses against the attack.

Many software tools, commercial as well as open-source, help in building fault trees manually and calculating MCSs. Most existing tools, however, do not go as far as building the fault trees automatically from rigorously-defined specifications of systems. Yet, when fault trees become large, as they

typically do when systems are large and complex, manual construction of fault trees becomes increasingly error-prone and time-consuming. In previous work we have developed a tool that builds fault trees and calculates their MCSs automatically from a carefully specified process model [15].

The approach just described has been shown to be effective in other work. But, Section VI discusses research challenges that must be met to apply the approach more effectively to wider classes of insider sabotage attacks.

2) Finite-State Verification: Finite-State Verification (FSV) is a technology used to infer characteristics about the executions of some or all paths through a specified system. Simple inspection of a process model may detect the possibility that an agent might be assigned the execution of an activity that provides access to one or more artifacts used to compute a final output of the process. This information could then be used to determine if a single agent has the ability to corrupt a final output of the process, thereby carrying out a SISAS attack. Simple inspection is usually insufficient to identify when and how MISAS or MIMAS attacks might occur, but FSV analysis can be effective in identifying the possibility of such attacks. For example, FSV can identify if certain suspect sequences of events, such as write accesses to particular artifacts by specific kinds of performers, can occur in any execution through the process. More sophisticated FSV could track data dependencies as well as agent relationships.

For data exfiltration attacks, May Happen in Parallel analysis [18] can be used to document all of the activities in a process that might possibly be executed concurrently. These combinations could then be examined to determine the agents that could have simultaneous access to the artifacts involved. If there is only one such agent, than that agent could then potentially reveal inappropriate information causing a SISADE or SIMADE attack. If there is a collection of agents, then that collection could cause a MISADE or MIMADE attack if the agents in that collection are in collusion with each other. If the data does not have to be accessed simultaneously, then FSV can be used to find sequences of access events that look troubling.

III. Examples

This section illustrates some of the details of our approach by presenting an example process and showing how some vulnerabilities might be identified. Our example is based upon the election process used in Yolo County, California [19]. The model has been extensively validated through numerous previous discussions with Yolo County election officials. However, the approach is not specific to this process, and could be applied equally well to other election processes, and indeed to other types of processes in general.

A. Modeling an Election Process

We used the Little-JIL [20] visual process definition language to model the election process used in Yolo County. Little-JIL is a process definition language that incorporates all of the characteristics enumerated in Section II-A. In particular, Little-JIL's semantics support precise definition of process activity coordination among multiple human agents and software and hardware components; the creation, use, and modification of artifacts; the specification of complex exceptional situations and their mitigation; and the types of agents to be assigned to execute process activities, referred to as *steps* in Little-JIL. The diagrams presented here elide many of these details to avoid visual clutter.

We begin with an informal description of a part of the election process. After the polls close in Yolo County, every precinct brings its ballots along with a summary cover sheet (indicating how many ballots were issued to the precinct, and how many were used, spoiled or blank after the polls closed) to Election Central for tabulation. Election officials first count the votes from each of the precincts, then perform a random audit, and then, finally, if no discrepancies are noted, report final vote totals to the California Secretary of State. The votes from each precinct are tallied separately before being added to a total tally. Before a precinct's ballots are actually scanned, they are counted, and these counts are compared to the ballot counts on the precinct's summary sheet. After reconciling the actual and reported numbers the ballots are scanned to obtain the actual vote counts. Random auditing (or a mandatory manual recount of 1% of ballots to ensure consistency) is a state requirement in California as well as in many other states [21], [22]. Yolo County primarily uses paper ballots, which are scanned and counted by automated optical scanners. To comply with the Help America Vote Act (HAVA), each precinct also provides a direct-recording electronic (DRE) voting machine that voters can ask to use. All DRE machines have an attached printer to generate a voter-verified paper audit trail (VVPAT). This paper trail is dispositive, and in California, the electronic record cannot be used in the mandatory 1% audit. Indeed, Yolo County election officials use the VVPAT to count the votes cast on DREs because so few people use the DREs.

The Little-JIL process coordination model shown in Figure 1 is a model of the process that we have just described, simplified for space considerations. The model is a hierarchical structure of *steps*, where a step is shown as a black rounded rectangle with the step name above it. Each step specifies the characteristics that must be satisfied by the agent that will be assigned to be responsible for the step's execution. The agent may be a human, such as an election official or a voter, or a hardware or software component such as a ballot-scanning device. A Little-JIL step is akin to a procedure or method definition that, once specified, can be invoked by reference from anywhere in the process model. The way in which a leaf step (a step without substeps) is executed is left entirely to the step's agent. The behavior of a non-leaf step is defined by its substeps, connected by edges emanating from the left side of the parent step bar. The step bar has a sequence badge at the left that specifies the order in which the substeps are to be executed, as described below.

Thus, the root step `count votes` in Figure 1 is a sequential step (indicated by a right arrow), which means its substeps will be executed in order from left to right, so after `initialize`

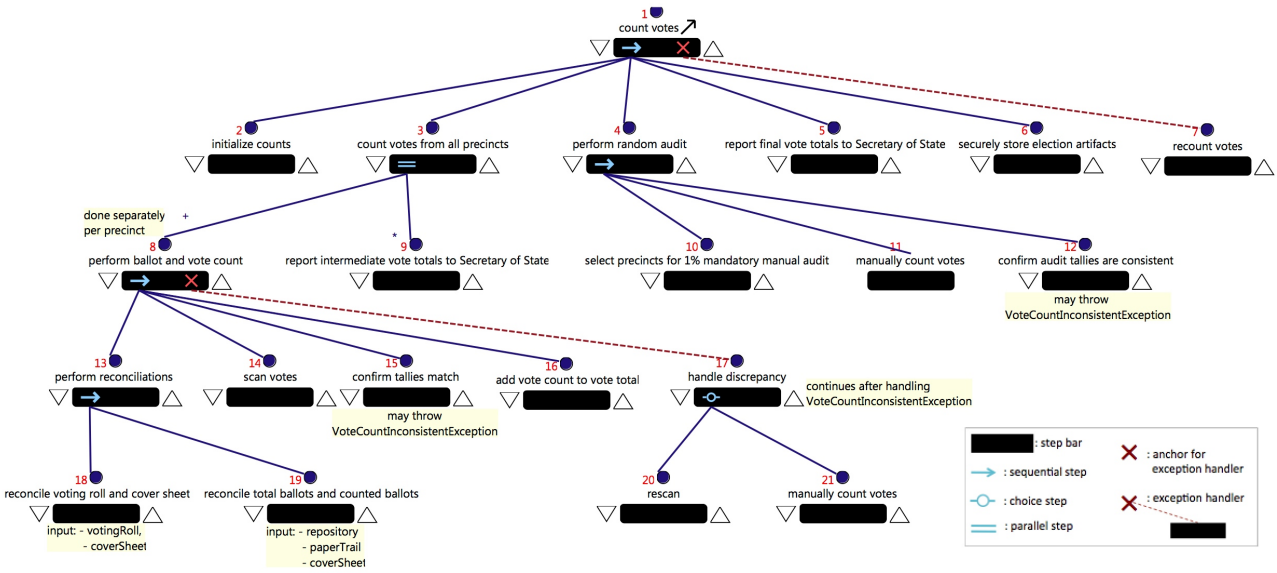

Fig. 1: Little-JIL process model: elaboration of "count votes" sub-process

(Reference #) Step	Artifacts		Agent
	Inputs	Outputs	
(1) count votes	coverSheet, paperTrail, repository, votingRoll	totalTallies	ElectionOfficial
(2) initialize counts		totalTallies	ElectionOfficial
(3) count votes from all precincts	coverSheet, paperTrail, repository, votingRoll, tallies, totalTallies	tallies, totalTallies	ElectionOfficial
(4) perform random audit	paperTrail, repository, auditTallies, tallies	autditTallies, tallies	ElectionOfficial
(5) report final vote totals to Secretary of State	finalTallies		ElectionOfficial
(6) securely store election artifacts	paperTrail, repository, votingRoll, tallies, totalTallies		ElectionOfficial
(7) recount votes	paperTrail, repository, originalTallies	recountedVoteTotals	ElectionOfficial, Observer
(8) perform ballot and vote count	coverSheet, paperTrail, repository, votingRoll	tallies, auditTallies	ElectionOfficial
(9) report intermediate vote totals to Secretary of State	immediateTallies		ElectionOfficial
(10) select precinct for 1% mandatory manual audit	tallies	tallies	Observer
(11) manually count votes	paperTrail, repository, tallies	auditTallies	ElectionOfficial, Observer
(12) confirm audit tallies are consistent	tallies, auditTallies		ElectionOfficial, Observer
(13) perform conciliations	coverSheet, paperTrail, repository, votingRoll		ElectionOfficial
(14) scan votes	repository, paperTrail	tallies	ElectionOfficial
(15) confirm tallies match	coverSheet, tallies		ElectionOfficial
(16) add vote count to vote total	tallies, totalTallies	totalTallies	ElectionOfficial
(17) handle discrepancy	paperTrail, repository, tallies	tallies	ElectionOfficial
(18) reconcile voting roll and cover sheet	votingRoll, coverSheet		ElectionOfficial
(19) reconcile total ballots and counted ballots	coverSheet, repository, paperTrail		ElectionOfficial
(20) rescan	paperTrail, repository, tallies	tallies	ElectionOfficial
(21) manually count votes	paperTrail, repository, tallies	tallies	ElectionOfficial

TABLE I: Artifacts and agents for the steps in the "count votes" sub-process

`counts` (step #2) is completed, `count votes from all precincts` (step #3) will be followed by `perform random audit` (step #4), followed by `report final vote totals to Secretary of State` (step #5), and, finally, `securely store election artifacts` (step #6). Step `count votes from all precincts` is itself decomposed into the step `perform ballot and vote count` (step #8), executed one or more times (denoted by the Kleene "+" on the edge connecting it to its parent), once for each precinct and the vote counting for the different precincts with `report intermediate vote totals to Secretary of State` (step #9), a step that is executed zero or more times (note the Kleene star on the edge). These steps can be done in parallel (denoted by the equal sign in the step bar of their parent step). This indicates that, although optional, the intermediate vote totals could be provided to the Secretary of State at any time during the counting upon request. Continuing the hierarchical decomposition further, `perform ballot and vote count` (is the sequential execution of `perform reconciliations` (step #13), `scan votes` (step #14), `confirm tallies match` (step #15), and `add vote count to vote total` (step #16). As the essence of these steps is checking for errors and discrepancies, we now explain Little-JIL exception handling semantics.

Any Little-JIL step may throw one or more *exceptions*, and may define one or more *exception handlers*. As in a programming language, exceptions are typed objects, and exception handling is scoped. Thus an exception is handled

by the appropriately typed handler located at the nearest ancestor step. Exception handlers are (potentially hierarchically decomposed) steps themselves, and are attached by a dotted edge to the right side of a step bar, below a red capital "X". Although the details are not shown, in this example, `perform ballot and vote count` has a handler named `handle discrepancy`. This handler is specified to catch an exception of the type `VoteCountInconsistentException`, which can be thrown by the `confirm tallies match` step (step #15). The handler, `handle discrepancy` (step #17), is specified using the choice step kind, denoted by a slashed circle, which means the step can be carried out by performing either the `rescan` step (#20) or the `manually count votes` step (#21). After the `handle discrepancy` step has completed, execution continues after the step that threw the exception, in this case with `add vote count to vote total` (step #16). Other continuation semantics (not used in this example) may specify, for example, that the step throwing the exception is to be repeated after the exception has been handled. The explanation of the rest of this process model is similar and is omitted here.

Analysis of the possibility of insider attacks on a process requires specification of the artifact flow and agent assignments to steps as well as specification of the structure of process activities. In Little-JIL a step specification includes an enumeration of the artifacts used as step inputs and outputs, and (as noted above) of the characteristics of the agent required to perform the step. Table I provides some examples of these specifications.

The row numbers in the table correspond to the numbers on the steps in Figure 1 (the step numbers are included here for convenience but are not part of the actual Little-JIL activity diagram). Thus, for example, line 3 of Table I specifies that `coverSheet`, `paperTrail`, `repository`, `votingRoll`, `tallies`, and `totalTallies` are all inputs to the `count votes from all precincts` step, and `tallies` and `totalTallies` are both outputs from that step. Little-JIL artifact passing is usually best thought of as procedure parameter passing. Artifacts are typically passed between parent steps and their substeps, but Little-JIL also supports message passing to any step in the process, and passing arguments to exception handlers.

In Little-JIL, a specific agent, who is the performer of the step, is bound dynamically at the time of the step's execution. When this binding is done, an available agent having the specified characteristics and capabilities is selected and assigned by a resource manager, which is a separate component of the Little-JIL run time system. An agent may be either human or automated. Thus, both `Election Official` (a human), and `Scanner` (an optical scanning machine) are examples of types of agents for this process.

B. Identifying an Insider Sabotage Attack

We now consider possible insider sabotage attacks on the process just described. Our approach, as suggested previously, consists of:

- identifying a hazard (namely a successful insider attack) as the delivery of an incorrect artifact to a step in the process that delivers the artifact as a final process output;
- using that hazard definition and a sufficiently precisely-defined model (e.g. a model defined using Little-JIL) of a process to generate a fault tree that shows how the hazard can occur;
- calculating the fault tree's MCSs and identifying the steps corresponding to the events in those MCSs; and finally
- identifying the combinations of insider agents who can execute all of the steps associated with each of the MCSs.

If a single agent can execute all the steps associated with an MCS, the process is shown to be vulnerable to a single insider attack. When multiple agents are needed to execute all the steps in an MCS, a multiple insider attack is possible, assuming that all of these agents are colluding. The larger the number of agents that must collude, the greater the difficulty (and presumably the cost) of the insider attack, hopefully making the attack impractical. Similarly, the kinds and number of artifacts involved would impact the cost of an attack.

We now consider the hazard that the wrong `finalTallies` artifact is delivered to the `report final vote totals to Secretary of State` step (i.e. the reported election results are incorrect). We use a tool we developed to generate the fault tree shown in Figure 2. (Note, again, that the graph in Figure 2 reflects some optimizations that combine different instances of the same event, and thus is not actually a tree but an equivalent directed acyclic graph). Generating this fault tree requires both the activity structure in Figure 1, and the artifact flow specifications shown in Table I, both of which are part of a complete Little-JIL process definition. We then use a fault tree analysis tool to calculate the MCSs. We are particularly interested in sets of activities where all the agents are insiders and are able to modify either the final process output or any artifact used to create the final output. For this example `finalTallies` is the final process output, and the fault tree identifies steps involved in the artifact's creation. Thus our analysis concludes by identifying the agents that perform these steps. Our tool finds a total of 12 MCSs for this hazard and the process model shown in Figure 1. Three of these MCSs are shown in Figure 3, one (MCS 1) having two events, one (MCS 2) having three events, and one (MCS 3) consisting of only one event, a single point of failure.

MCS 3 indicates that the hazard occurs if the `recount votes` step produces an incorrect result. Elaboration of this step, not done here to save space, reinforces the point that more process detail is sometimes needed to shed light on indicated vulnerabilities. Further elaboration of `recount votes` is needed to provide details of how the recount process can be attacked. In Section VI we suggest an incremental analysis approach that could be useful in helping to guide the acquisition and analysis of such needed details, thereby providing better understandings of such indicated vulnerabilities.

MCS 2 requires the step `rescan` to produce an incorrect `tallies` artifact and the step `perform random audit`, which is intended to detect the discrepancy in tallies, to fail

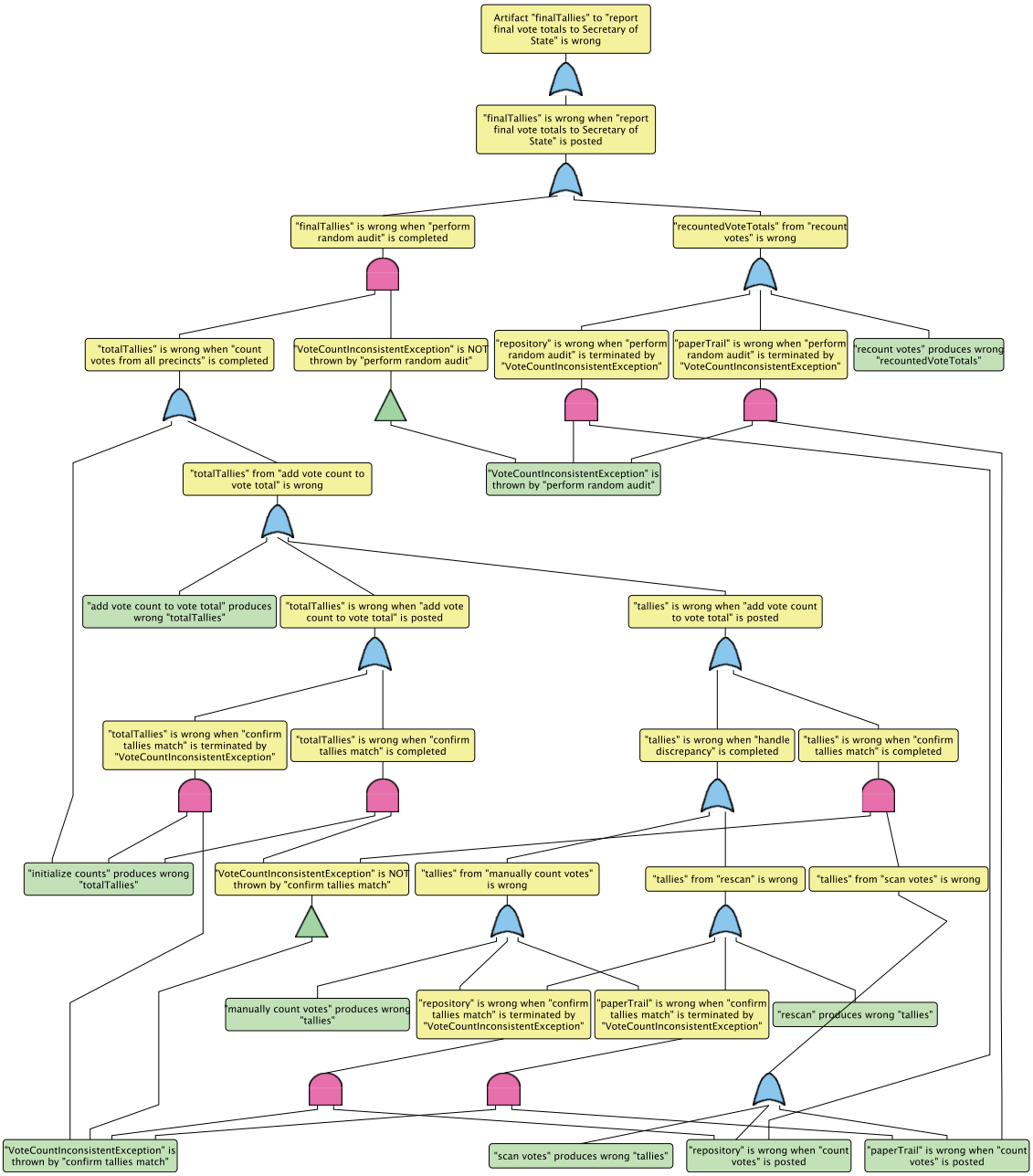

Fig. 2: Fault tree for hazard "wrong `finalTallies` artifact is delivered to step `report final vote totals to Secretary of State`"

to throw the `VoteCountInconsistentException`, which in fact is propagated from the substep `confirm audit tallies are consistent`[1]. It is important to note that there are a number of possible explanations of how these two events could occur. One explanation is that the incorrect results produced by the rescan are never presented as part of the random audit and that the audit is done correctly. But it is also possible that the incorrect results of the rescan are indeed presented to the audit but the inconsistency is not reported when the tallies are found to be inconsistent. In the first case, the agent for the rescanning step can carry out a SISAS attack, but risks detection by the random audit. To be sure of avoiding detection, the agent who performs `confirm audit tallies`

[1]The fact that the MCS event indicates the step `perform random audit` instead of `confirm audit tallies are consistent` is the result of an optimization for scalability of our FTA tool [15]. We might reconsider this optimization since it seems to decrease the understandability of the resulting fault trees and therefore MCSs.

- MCS 1
  1) Step `rescan` produces wrong artifact `tallies`
  2) Step `perform random audit` does not throw exception `VoteCountInconsistentException`
- MCS 2
  1) Step `scan votes` produces wrong artifact `tallies`
  2) Step `confirm tallies match` does not throw exception `VoteCountInconsistentException`
  3) Step `perform random audit` does not throw exception `VoteCountInconsistentException`
- MCS 3
  1) Step `recount votes` produces wrong artifact `recountedVoteTotals`

Fig. 3: Some MCSs for the hazard "wrong `finalTallies` artifact is delivered to step `report final vote totals to Secretary of State`"

`are consistent` would have to either be the same as the agent who performed the rescanning step (a successful SISAS attack), or in collusion with the agent who performed the rescanning step (a successful MISAS attack).

Table I specifies that the agents for both `perform random audit` and `rescan` may be of type ElectionOfficial, and so the process specification does not explicitly prevent the same agent from performing both steps, thus revealing a possible SISAS attack. If the process incorporated a check to ensure that the same agent could not perform both steps the SISAS attack is thwarted, but it could still succeed as a MISAS attack if the two performers are in collusion.

We observed that the MCS does not distinguish between the case where the random audit detects the problem and where it does not. The scenario and analysis leave some other questions unanswered as well. Note that the `rescan` step is only executed as part of handling an exception thrown by the `confirm tallies match` substep of `perform ballot and vote count`, and the MCS sheds no light on why the `confirm tallies match` step threw the exception in the first place. This suggests the need for sharpened analysis that might present more detailed scenarios, such as where the scanning device has a defect, where the human operating the scanner used it incorrectly, or where scanner output was recorded incorrectly. Approaches to addressing these problems are discussed in Section VI.

C. Identifying Insider Data Exfiltration Attacks

One of the requirements for voting is the secret ballot; no one should be able to associate a specific voter with a specific ballot. One danger is that, during the election process, some information might enable an observer to match a voter's name to a specific ballot. Indeed, in Ohio, researchers were able to do exactly that [23] by obtaining and correlating two artifacts, a list of voters and a paper trail of the actual votes. The time-stamped paper trail was the standard output of the DRE voting machines being used, and, according to state law, anyone could request a copy of it, as well as a copy of the list of voters (which included the order in which they voted). Given that both lists were chronologically ordered, it was easy to match the voter names to the voter preferences, which clearly violated voter confidentiality, and is a good example of a data exfiltration attack. This attack was actually carried out by voters requesting these two publicly available lists and has since been thwarted by making one of the lists confidential. We now consider this attack within the context of insiders, considering the possibility that election officials who have access to this information could carry out this attack.

We consider a slight variant of the election process, where when a voter enters the polling station, he or she is authenticated in some fashion, and a notation is made in the voting roll (sometimes called a "poll book") of the time the voter arrived. The voter then goes to a DRE, and casts his or her vote. As part of the process, the DRE time-stamps the paper record of the ballot. At the end of the day, the voting rolls and paper trails (and other ballots) are taken to Election Central to be counted.[2]

Consider the process modeled in Figure 4. Step #33 in this diagram, `count votes`, is the root of the process for counting votes shown in Figure 1 that we have been discussing thus far. As previously explained, Little-JIL steps can be thought of as being much like procedures, so consider Figure 4 to be the context within which `count votes` takes place. Intuitively, this process indeed describes the high-level process for carrying out elections, thus `conduct election` consists of sequentially performing `pre-polling activities`, `prepare for and conduct election at precinct` (done once for each precinct as indicated by the Kleene "+"), and, finally, `count votes`. Focus now on the activities that happen on Election Day, namely the elaboration of `prepare for and conduct election at precinct`, which includes as one of its sub steps `authenticate and vote`, a sequential subprocess that is performed once for each voter (again indicated by the Kleene "+" on the edge leading to the step from its parent). This subprocess dictates that election officials first `perform pre-vote authentication`,

[2]We have taken some liberties with the actual procedure in Yolo County. Almost all voters use paper ballots, and they are *never* time-stamped. However, the process modifications were inspired by the incident in Ohio and are representative of real-world procedures.

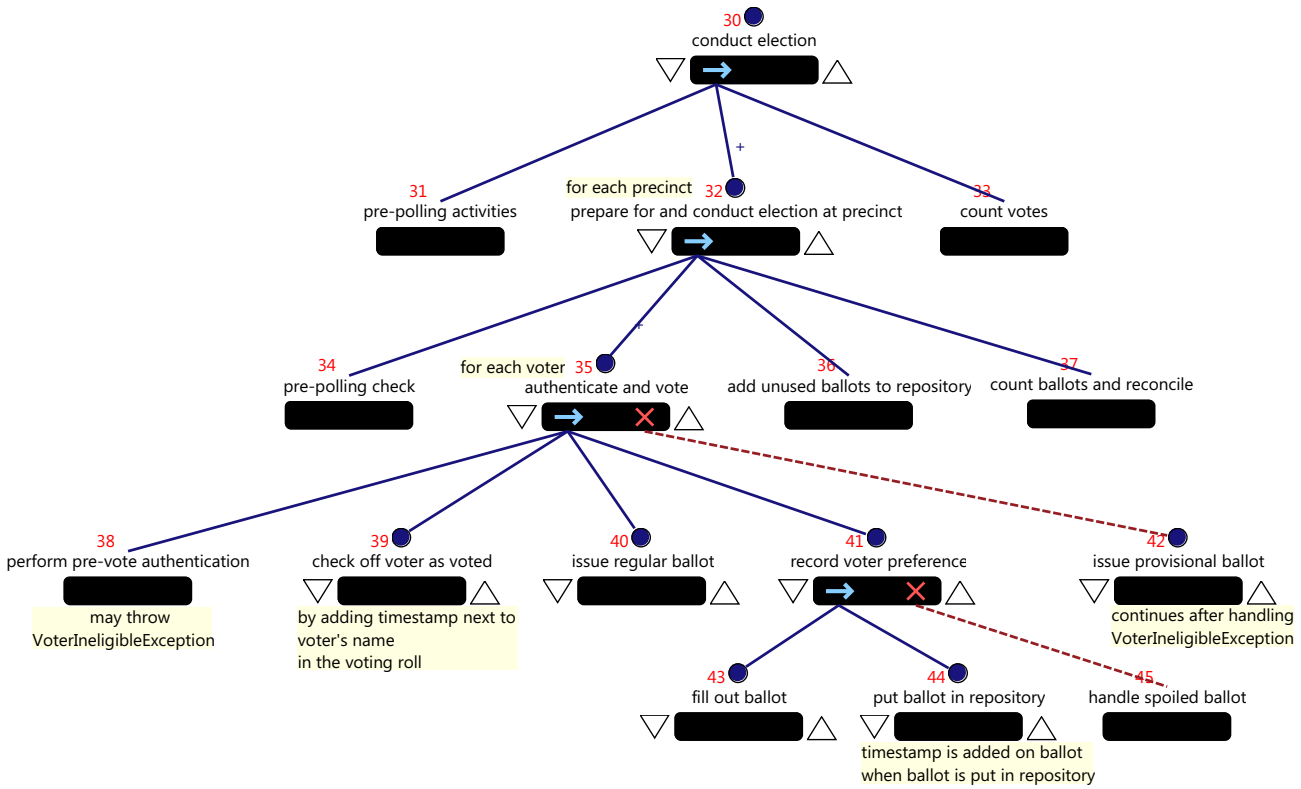

Fig. 4: Little-JIL process model: "conduct election" process

then `check off voter as voted`, and `issue regular ballot` if the authentication is completed correctly and the voter has not voted yet. Only then can the process go on to `record voter preference`, where a voter will first `fill out ballot`, then `put ballot in repository`.

Now focus on the fact that a voter's vote should be confidential. If this process is executed as specified, the voter should be the only person that knows how she or he voted (clearly, the DRE voting machine that executes the step `put ballot in repository` knows the voter preference, but does not know the voter's identity, and therefore cannot perform the data exfiltration attack). However, when an election official checks off the voter as having voted (in step #39), a timestamp is added next to the voter's name in the voting roll to indicate when the ballot was issued. Also note that when a voter casts a ballot on a DRE machine (step #44), the ballot is automatically timestamped.

Therefore, one artifact in the process contains a list of voters' names and timestamps indicating when they voted (the voting roll in which the election officials `check off voter as voted`, step #39), and another artifact in the process contains a list of timestamped votes (the ballot that the DRE has `put ballot in repository`, step #44). Clearly, if these two artifacts are combined, then voter confidentiality can be compromised, comprising a data exfiltration attack. A careful consideration of Figure 1 where the vote counting occurs, and Table I which describes what artifacts are accessible at what steps, reveals how this combination may happen. In Figure 1, step #18, `reconcile voting roll and cover sheet` requires the voting roll as input, and step #19, `reconcile total ballots and counted ballots`, requires the repository (the collection of all timestamped votes in this case) as input. Both of these artifacts are passed to (and through) step #13, `perform reconciliations`, which is executed by an agent of type "ElectionOfficial", as shown in Table I. Thus, the agent performing step #13 is able to carry out a data exfiltration attack.

Identifying this possible data exfiltration attack is easy in this example, but can be much more difficult in the case of a very large and complex process. Indeed, note that the agent executing step #19 in this example could also carry out the data exfiltration attack if that agent received the voting roll as input, suggesting the need for careful specification and control of artifact flow. Or, indeed if steps # 18 and #19 were executed in parallel, rather than sequentially, vulnerability to a single insider attack (SIMADE) or collusion (MIMADE) attack would exist. As noted above, may happen in parallel and Finite-State Verification could be used to determine all possible combinations of steps that might possibly be executed at the same time. Such algorithms can then be used to systematize searches for these kinds of vulnerabilities.

IV. Evaluation

We believe it is important that this approach be evaluated both subjectively and objectively. We will continue our use of the election domain and the running example of the election process definition to show how this is being done; again, however, we emphasize that the approach outlined below is general and will work for other election processes and for processes in other domains. The previous section outlined two different analysis approaches used to verify that the model of the election process is robust with respect to certain attacks. These analysis results can then be used to inform and improve the real-world process, given that the model was an accurate representation. Therefore, before it is analyzed, the process model is first *validated*.

In the election domain, this validation is carried out by working with election officials, who can identify the types of insider sabotage that they are most concerned about, and those having the greatest impact upon an election. They also provide detailed information about the election process being examined and the agents (voters, election officials, and others) that perform the steps in these processes. Then, as we developed and analyzed early versions of these models, the domain experts provided initial evaluations of how effectively our methods seem to be able to identify agents who might launch insider attacks either deliberately or through errors.

Once the model has been validated through interactions with domain experts, objective evaluations focus on the effectiveness and efficiency of our process definition and analysis approaches. The clarity and efficiency of the Little-JIL process definitions in providing sufficient detail and supporting effective analyses are key measures of the usefulness of the results. In previous research, these kinds of practical applications of Little-JIL have led to important language improvements. As noted above, some of our analyses seem to produce diagnoses of hazards that may not be explained sufficiently well to satisfy domain experts, thereby necessitating iterative analytic explorations. There are several approaches to supporting such analytic iteration, although formulating and performing more detailed and in-depth analysis proactively can help to obviate the need for analytic iteration in the first place. Undertaking these more complex and detailed analysis problems necessitated larger, more complex process definitions, and analyses that generate larger (sometimes far larger) problem spaces. Thus a key part of our objective evaluation has been to study the scalability of our approaches, investigating both theoretical complexity bounds and problem sizes actually encountered.

Finally, the evaluation approaches are being integrated by having the domain experts provide their views of the relative practical value and importance of the objective measures that were developed. The feedback from the subjective evaluations improved the development of the objective measures above.

V. Related Work

Much work has examined insider threats and attacks. Hu et al. [24] use role-based access control (RBAC) model to look for unusual behavior, or behavior that violates job requirements to construct rules for intrusion detection. This approach, however, leads to false positives. Park and Giordano [25] reverse the Hu et al. approach, analyzing a user's behavior and checking that it is as expected. Several researchers [26]–[28] incorporate risk assessment into an extended access control framework, adapting user privileges based on role, attributes, and level of trust, which drops when a user's behavior becomes suspicions. Other work extends the RBAC model by focusing on generalized attributes of people and data, and placing the insider threat in the context of modeling policies using layers of abstraction [29], [30].

Schultz [31] proposes a framework based on attack-related behaviors and symptoms, and correlates behaviors that typically precede an attack, with patterns of use, verbal behaviors, and personal characteristics. Kandias et al. [6] extend this work with a model identifying users who need additional monitoring. Greitzer and Frincke [32] develop a predictive system that identifies insiders through psychosocial indicators and anomalies in system use, rather than focusing on the process and artifacts as we do.

Other work focuses on user psychology. Chinchani et al. [33] present a theory of insider assessment that models the user's view of the organization and of key information and capabilities. Vance et al. [34] argue that user interface designs that makes users feel more accountable for their actions discourage insider attacks. Posey et al. [35] study how the perception of the organization's trust affects an insider's mindset. This type of work focuses on agents as organization members, in contrast to our focus on them as process performers.

Probst et al. [36] use process algebras to model and reason about the actions of entities in the context of the insider problem. Magklaras and Furnell [37] predict insider threats by projecting what agent actions can affect based on an analysis of agent sophistication. This contrasts with our focus on critical artifacts and the agents who can access them.

Some researchers [3], [38]–[40] propose using decoys to detect insiders. Others study how to create effective decoys [41] and honeypots [42]. Stolfo et al. [43] extend this work to the cloud.

Kammüller and Probst [44] build vectors for insider attacks by using organizational structure to identify sequences of actions that lead to security policy violations. This work focuses on the organization, whereas our work focuses on the processes that the organization uses. Similarly, Ard et al. [45] focus on workflows of documents as opposed to processes. Intrusion detection approaches such as system call analysis [46], [47], graph-based approaches [48], [49], and signature analysis [50] have been explored, as have threat specification languages [8]. Crampton and Huth [51] look to next-generation access control mechanisms to address the insider problem. Neumann [52] takes a holistic approach, arguing that systems—their architecture, development, and operation—must be made more trustworthy if insider misuse is to be addressed effectively.

Attack trees, first introduced as such by Schneier [53], are

widely used, and several variants (such as the requires/provides model [54]) enable the analyst to examine conditions necessary for the attack to succeed. Unlike attack trees, fault trees focus on *threats* rather than attacks. Fault tree analysis helps identify vulnerabilities that are potentially exploited by attacks. Also, in our work, fault trees are automatically derived from the model of the process which takes into account the temporal order of activities and the artifact flows between them, while such specifications are not explicitly considered in attack trees.

VI. Future Directions and Work

As described earlier, the FTA techniques we have developed already can identify vulnerabilities to some kinds of insider attacks. In this section, we briefly describe the significant research challenges that must be addressed to improve the utility and generality of these techniques.

As noted above, the MCSs identified by our techniques for automated analysis of processes identify vulnerabilities. But our techniques are not always precise enough to fully describe the vulnerabilities and explain how they arise. For instance, the two-event MCS discussed earlier fails to distinguish between the case where two steps must be performed incorrectly and the case where only one step must be compromised. Moreover, our analysis does not indicate that this attack scenario only occurs in an exceptional situation.

These problems reflect the fact that our analysis does not take into account the full control and data dependencies of all the steps in the process. Identifying such dependencies is itself nontrivial, especially when interprocedural (or interstep) data flow must be taken into account. Our current tool uses a relatively simple template-based approach which would need to be extended or replaced to make use of more precise dependence information. We intend to investigate techniques for efficiently selecting what additional control and data flow dependence information can be included in order to produce more "useful" cut sets and more easily identify vulnerabilities and ways to modify the process to defend against them. We do not yet know the appropriate tradeoffs between using enough information to facilitate this and using too much information, leading to too many cut sets, therefore making the approach unhelpful to process designers. It might be possible to incrementally refine the FTA, using additional control and data flow information when requested by the analyst.

Another important issue of the precision of our analyses has to do with the modeling and analysis of the agent assignment mechanism. The coarse approach described earlier depends on identifying the type of agent required for each step. But the particular assignment mechanism used by a given process might give much more precise information about the possible agents who could be assigned to execute a given step, depending on the history of agent assignment on that particular execution. The coarse analysis might detect that agents of the same type are assigned to the steps in a cut set and indicate the possibility that a single agent could sabotage the process. But a more precise analysis might show that the agent assignment mechanism would never assign the same agent instance to those steps. Such an analysis would require consideration of all possible process executions leading to those steps.

Given the concurrency and complicated branching inherent in many processes, techniques like the flow analysis used by our FLAVERS FSV tool [55] might be more appropriate than FTA-based approaches for this kind of analysis. The FLAVERS tool, which we have previously used for verification of temporal properties of processes (e.g., [56], [57]), builds a graph representing the control flow of the process (including concurrency, using a May Happen in Parallel analysis [18] to determine which steps can overlap in time) and uses data flow techniques to propagate the states of automata through this graph. The automata can represent a temporal property to be checked as well as constraints that restrict the execution according to data values. The use of constraints representing the assignment of agents, including the assignment mechanism and the characteristics of the agents that might be used in doing that assignment, would enable flow analysis to determine which agents could be assigned to what steps at what times.

It might also be possible to integrate some of this analysis with FTA-based approaches to improve the analysis of agent assignments to the steps associated with events in an MCS.

Given a significant vulnerability identified by one of these analyses, process designers will be interested in modifying the process to reduce or eliminate the vulnerability. While repeating the analysis on a modified process can show whether or not a proposed modification actually works, in some cases the information produced in the original analysis might be used to suggest (in an automated or semi-automated way) appropriate modifications. How to use the analyses to identify suitable process improvements is an open question.

Thus far, we have discussed only attacks in which insider agents may perform a process step incorrectly by modifying an output artifact or by causing an exception to be incorrectly thrown or not thrown. But other kinds of deviation from correct execution can be equally interesting and provide insights for the domain experts. For instance, agents refusing to carry out certain steps, or delaying them sufficiently, could produce denials of service or related problems. In some cases, analysis for deadlock or starvation might detect the potential for such attacks in appropriate models. In other cases, the flow analysis might need to include certain classes of executions not allowed by the original process model in order to represent more substantial deviations from the specified process.

VII. Conclusion

Process model-based analysis is a powerful tool, and has been used to analyze processes ranging from medical procedures [56], [58] to scientific workflows [59] to processes by which elections are conducted [16], [17]. Many of these efforts produce large and complex process models and fault trees. As the analysis is automated, this has not been a problem. Given the variety of domains in which the process model-based analysis has been applied, we expect this approach — using process model-based analysis to identify insider threats — to also be general and applicable in various domains.

Our technique approaches the insider problem in a way that is different from the approaches of prior research. Past research has focused on analyses of how insiders might attack, and has sought to identify attackers and attacks by tactics such as decoys and anomaly detection, and to inhibit attacks by adding technological constraints in accordance with the principle of separation of privilege or separation of duty [60]. Those approaches essentially take the attacker's point of view.

In contrast, our approach takes the point of view of the process that is the target of attacks. We leverage a detailed knowledge of the structure of the activities of the process, and the ways in which different kinds of agents can be assigned to carry out those activities. We focus on the assurance of the process that the insider is trying to disrupt, including violations of properties such as confidentiality of the data used by the process, integrity of the process itself, and availability of both results and the process, rather than how the insider is trying to disrupt it. We look for places where the process could be disrupted by modeling the process rigorously and formally, and then using techniques such as fault tree analysis to examine how to ameliorate or prevent the disruption. Thus, rather than anticipating how an insider will attack, our approach identifies points where the insider (or insiders) *could* disrupt the process, and determines how to prevent that disruption if possible, and if not possible, how to increase the cost to the attacker.

How effective this approach will be remains to be seen. It is a new and promising direction for insider threat analysis.

ACKNOWLEDGMENTS

Matt Bishop and Sean Peisert acknowledge the support of the National Science Foundation (NSF) under grant CNS-1258577 and the National Institute of Standards and Technology (NIST) under grant 60NANB13D165. Sean Peisert also acknowledges the support of the NSF under grant CCF-1018871.

George Avrunin, Lori Clarke, Heather Conboy, Lee Osterweil, Huong Phan, and Borislava Simidchieva acknowledge the support of NIST under grant 60NANB13D165 and of the NSF under grants IIS-1239334 and CNS-1258588.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or NIST.

We also thank Freddie Oakley, Yolo County Clerk/Recorder; Tom Stanionis, her chief deputy; and Elaine Ginnold, the Registrar of Voters in Marin County, for their help.

REFERENCES

[1] M. Bishop, *Computer Security: Art and Science*. Boston, MA, USA: Addison Wesley Professional, Dec. 2002. [Online]. Available: http://www.amazon.com/gp/product/0201440997

[2] R. Brackney and R. Anderson, "Understanding the Insider Threat: Proceedings of a March 2004 Workshop," RAND Corporation, Santa Monica, CA, Tech. Rep., Mar. 2004.

[3] B. M. Bowen, M. Ben Salem, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Designing host and network sensors to mitigate the insider threat," *IEEE Security & Privacy*, vol. 7, no. 6, pp. 22–29, Nov. 2009.

[4] D. Cappelli, A. Moore, R. Trzeciak, and T. J. Shimeall, "Common sense guide to prevention and detection of insider threats, 3rd edition — version 3.1," CERT, Tech. Rep., Jan. 2009. [Online]. Available: http://www.cert.org/archive/pdf/CSG-V3.pdf

[5] J. Hunker and C. W. Probst, "Insiders and insider threats—an overview of definitions and mitigation techniques," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 2, no. 1, pp. 4–27, 2011.

[6] M. Kandias, A. Mylonas, N. Virvilis, M. Theoharidow, and D. Gritzalis, "An insider threat prediction model," in *Proceedings of the 7th International Conference on Trust, Privacy and Security in Digital Business*, S. Katsikas, J. Lopez, and M. Soriano, Eds., vol. 6264. Berlin, Germany: Springer-Verlag, 2010, pp. 26–37.

[7] M. Maloof and G. Stephens, "Elicit: A system for detecting insiders who violate need-to-know," in *Recent Advances in Intrusion Detection*. Springer, 2007, pp. 146–166.

[8] G. B. Magklaras, S. M. Furnell, and P. J. Brooke, "Towards an insider threat prediction specification language," *Information Management and Computer Security*, vol. 14, no. 4, pp. 361–381, 2006.

[9] J. Patzakis, "New incident response best practices: Patch and proceed is no longer acceptable incident response," Guidance Software, Pasadena, CA, Tech. Rep., Sep. 2003.

[10] C. W. Probst and R. R. Hansen, "Analysing access control specifications," in *Proceedings of the Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)*, Oakland, CA, 2009, pp. 22–33.

[11] ——, "An extensible analysable system model," *Information Security Technical Report*, vol. 13, no. 4, pp. 235–246, 2008.

[12] C. A. Ericson II, "Fault tree analysis—a history," in *17th International System Safety Conference*, 1999.

[13] J. Ward, M. Lyons, S. Barclay, J. Anderson, P. Buckle, and P. Clarkson, "Using fault tree analysis (FTA) in healthcare: A case study of repeat prescribing in primary care," in *Patient Safety Research: Shaping the European Agenda*, 2007.

[14] E. Hyman, William A.; Johnson, "Fault tree analysis of clinical alarms," *Journal of Clinical Engineering*, pp. 85–94, 2008.

[15] B. Chen, "Improving processes using static analysis techniques," Ph.D. dissertation, University of Massachusetts Amherst, 2010.

[16] B. I. Simidchieva, S. J. Engle, M. Clifford, A. C. Jones, S. Peisert, M. Bishop, L. A. Clarke, and L. J. Osterweil, "Modeling and analyzing faults to improve election process robustness," in *Proceedings of the 2010 Electronic Voting Technology/Workshop on Trustworthy Elections*. Berkeley, CA, USA: USENIX Association, Aug. 2010. [Online]. Available: http://www.usenix.org/events/evtwote10/tech/full_papers/Simidchieva.pdf

[17] H. Phan, G. Avrunin, M. Bishop, L. A. Clarke, and L. J. Osterweil, "A systematic process-model-based approach for synthesizing attacks and evaluating them," in *Proceedings of the 2012 USENIX/ACCURATE Electronic Voting Technology Workshop*. Berkeley, CA, USA: USENIX Association, Aug. 2012. [Online]. Available: https://www.usenix.org/system/files/conference/evtwote12/evtwote12-final26.pdf

[18] G. Naumovich and G. S. Avrunin, "A conservative data flow algorithm for detecting all pairs of statements that may happen in parallel," in *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 1998, pp. 24–34.

[19] Yolo county elections office. [Online]. Available: http://www.yoloelections.org

[20] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, J. S. M. Sutton, and A. Wise, "Little-JIL/Juliette: A process definition language and interpreter," in *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000, pp. 754–757.

[21] "One percent manual tally," California Elections Code Section 15360. [Online]. Available: http://www.leginfo.ca.gov/cgi-bin/displaycode?section=elec&group=15001-16000&file=15360

[22] "Postcanvass risk-limiting audit pilot program," California Elections Code Section 15560. [Online]. Available: http://www.leginfo.ca.gov/cgi-bin/displaycode?section=elec&group=15001-16000&file=15560

[23] D. McCullagh, "E-voting predicament: Not-so-secret ballots," http://news.cnet.com/2100-1014_3-6203323.html, Aug. 2007.

[24] N. Hu, P. G. Bradford, and J. Liu, "Applying role based access control and genetic algorithms to insider threat detection," in *Proceedings of the 44th Annual ACM Southeast Regional Conference*, 2006, pp. 790–791.

[25] J. S. Park and J. Giordano, "Role-based profile analysis for scalable and accurate insider-anomaly detection," in *Proceedings of the 25th IEEE International Performance, Computing, and Communications Conference*. Piscataway, NJ, USA: IEEE, Apr. 2006, pp. 463–469.

[26] M. Bishop, S. Engle, D. A. Frincke, C. Gates, F. L. Greitzer, S. Peisert, and S. Whalen, "A risk management approach to the "insider threat"," in

Insider Threats in Cyber Security, ser. Advances in Information Security, C. W. Probst, J. Hunker, D. Gollmann, and M. Bishop, Eds. New York, NY, USA: Springer Science+Business Media, LLC, Jan. 2010, vol. 49, pp. 115–137.

[27] N. Baracaldo and J. Joshi, "A trust-and-risk aware RBAC framework: Tackling insider threat," in *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '12. New York, NY, USA: ACM, June 2012, pp. 167–176.

[28] S. Peisert and M. Bishop, "Dynamic, flexible, and optimistic access control," Dept. of Computer Science, University of California at Davis, Davis, CA, USA, Technical Report CSE-2013-76, Mar. 2013.

[29] M. Bishop, S. Engle, S. Peisert, S. Whalen, and C. Gates, "We have met the enemy and he is us," in *Proceedings of the 2008 Workshop on New Security Paradigms*, ser. NSPW '08. New York, NY, USA: ACM, Sep. 2008, pp. 1–12.

[30] ——, "Case studies of an insider framework," in *Proceedings of the 42nd Hawaii International Conference on System Science*, Jan. 2009.

[31] E. E. Schultz, "A framework for understanding and predicting insider attacks," *Computers & Security*, vol. 21, no. 6, pp. 526–531, Oct. 2002.

[32] F. L. Greitzer and D. A. Frincke, "Combining traditional cyber security audit data with psychosocial data: Towards predictive modeling for insider threat mitigation," in *Insider Threats in Cyber Security*, ser. Advances in Information Security, C. W. Probst, J. Hunker, D. Gollmann, and M. Bishop, Eds. New York, NY, USA: Springer Science+Business Media, LLC, Jan. 2010, vol. 49, pp. 85–113.

[33] R. Chinchani, A. Iyer, H. Q. Ngo, and S. Upadhyaya, "Towards a theory of insider threat assessment," in *Proceedings of the 2005 Internation Conference on Dependable Systems and Networks*. Los Alamitos, CA, USA: IEEE Computer Society Press, June 2005, pp. 108–117.

[34] A. Vance, B. Molyneux, and P. B. Lowry, "Reducing unauthorized access by insiders through user interface design: Making end users accountable," in *Proceedings of the 45th Hawaii International Conference on System Sciences*. Los Alamitos, CA, USA: IEEE Computer Society Press, Jan. 2012, pp. 4623–4632.

[35] C. Posey, R. J. Bennett, and T. L. Roberts, "Understanding the mindset of the abusive insider: An examination of insiders' causal reasoning following internal security changes," *Computers & Security*, vol. 30, no. 6–7, pp. 486–497, Sep. 2011.

[36] C. Probst, R. Hansen, and F. Nielson, "Where can an insider attack?" in *Proceedings of the Fourth International Workshop on Formal Aspects in Security and Trust*, Aug. 2006.

[37] G. Magklaras and S. Furnell, "Insider threat specification as a threat mitigation technique," in *Insider Threats in Cyber Security*, ser. Advances in Information Security, C. W. Probst, J. Hunker, D. Gollmann, and M. Bishop, Eds. New York, NY, USA: Springer Science+Business Media, LLC, Jan. 2010, vol. 49, pp. 219–244.

[38] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *Proceedings of the 5th International ICST Conference on Security and Privacy in Communication Networks*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Y. Chen, T. D. Dimitriou, and J. Zhou, Eds., vol. 19. Berlin, Germany: Springer, Sep. 2009, pp. 51–70.

[39] B. M. Bowen, M. Ben Salem, A. D. Keromytis, and S. J. Stolfo, "Monitoring technologies for mitigating insider threats," in *Insider Threats in Cyber Security*, ser. Advances in Information Security, C. W. Probst, J. Hunker, D. Gollmann, and M. Bishop, Eds. New York, NY, USA: Springer Science+Business Media, LLC, Jan. 2010, vol. 49, pp. 197–217.

[40] M. Ben Salem and S. J. Stolfo, "Decoy document deployment for effective masquerade attack detection," in *Proceedings of the 8th International Conference on the Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, T. Holz and H. Bos, Eds., vol. 6739. Berlin, Germany: Springer-Verlag, Aug. 2011.

[41] J. Voris, N. Boggs, and S. J. Stolfo, "Lost in translation: Improving decoy documents via automated translation," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy Workshops*, May 2012, pp. 129–133.

[42] L. Spitzner, "Honeypots: Catching the insider threat," in *Proceedings of the 19th Annual Computer Security Applications Conference*. Los Alamitos, CA, USA: IEEE Computer Society Press, Dec. 2003, pp. 170–179.

[43] S. J. Stolfo, M. Ben Salem, and A. D. Keromytis, "Fog computing: Mitigating insider data theft attacks in the cloud," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy Workshops*, May 2012, pp. 125–128.

[44] F. Kammüller and C. W. Probst, "Invalidating policies using structural information," in *Workshop on Research for Insider Threat*, May 2013.

[45] J. Ard, M. Bishop, C. Gates, and M. Sun, "Information behaving badly," in *Proceedings of the 2013 Workshop on New Security Paradigms*, Sep. 2013, pp. 107–118.

[46] A. Liu, C. Martin, T. Hetherington, and S. Matzner, "A comparison of system call feature representations for insider threat detection," in *Proceedings of the Sixth Annual IEEE Systems, Man and Cybernetics Information Assurance Workshop*. Piscataway, NJ, USA: IEEE, June 2005, pp. 340–347.

[47] N. Nguyen, P. Reiher, and G. H. Kuenning, "Detecting insider threats by monitoring system call activity," in *Proceedings of the 2003 IEEE Workshop on Information Assurance*. Los Alamitos, CA, USA: IEEE Computer Society, June 2003, pp. 45–52.

[48] W. Eberle and L. Holder, "Insider threat detection using graph-based approaches," in *Proceedings of the 2009 Conference for Homeland Security, Cybersecurity Applications & Technology*, Mar. 2009, pp. 237–241.

[49] ——, "Applying graph-based anomaly detection approaches to the discovery of insider threats," in *Proceedings of the 2009 IEEE International Conference on Intelligence and Security Informatics*, June 2009, pp. 206–208.

[50] Y. Liu, C. Corbett, K. Chiang, R. Archibald, B. Mukherjee, and D. Ghosal, "Detecting sensitive data exfiltration by an insider attack," in *Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '08. New York, NY, USA: ACM, May 2008, pp. 16:1–16:3.

[51] J. Crampton and M. Huth, "Towards an access-control framework for countering insider threats," in *Insider Threats in Cyber Security*, ser. Advances in Information Security, C. W. Probst, J. Hunker, D. Gollmann, and M. Bishop, Eds. New York, NY, USA: Springer Science+Business Media, LLC, Jan. 2010, vol. 49, pp. 173–195.

[52] P. G. Neumann, "Combatting insider threats," in *Insider Threats in Cyber Security*, ser. Advances in Information Security, C. W. Probst, J. Hunker, D. Gollmann, and M. Bishop, Eds. New York, NY, USA: Springer Science+Business Media, LLC, Jan. 2010, vol. 49, pp. 17–44.

[53] B. Schneier, "Attack trees," in *Dr. Dobb's Journal*, vol. 24, no. 12, Dec. 1999, pp. 21–29.

[54] S. J. Templeton and K. Levitt, "A requires/provides model for computer attacks," in *Proceedings of the 2000 New Security Paradigms Workshop*, ser. NSPW '00. New York, NY, USA: ACM, 2000, pp. 31–38.

[55] M. B. Dwyer, L. A. Clarke, J. M. Cobleigh, and G. Naumovich, "Flow analysis for verifying properties of concurrent software systems," *ACM Transactions on Software Engineering and Methodology*, vol. 13, no. 4, pp. 359–430, Oct. 2004.

[56] B. Chen, G. S. Avrunin, E. A. Henneman, L. A. Clarke, L. J. Osterweil, and P. L. Henneman, "Analyzing medical processes," in *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*. ACM, May 2008, pp. 623–632.

[57] S. Christov, G. S. Avrunin, L. A. Clarke, L. J. Osterweil, and E. A. Henneman, "A benchmark for evaluating software engineering techniques for improving medical processes," in *SEHC '10: Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*, L. A. Clarke and J. Weber-Jahnke, Eds., Cape Town, South Africa, May 2010, pp. 50–56.

[58] G. Avrunin, L. Clarke, L. Osterweil, S. Christov, B. Chen, E. Henneman, P. Henneman, C. L., and W. Mertens, "Experience modeling and analyzing medical processes: Umass/baystate medical safety project overview," in *Proceedings of the First ACM International Health Informatics Symposium*, Nov. 2010, pp. 316–325.

[59] L. J. Osterweil, L. A. Clarke, A. M. Ellison, R. Podorozhny, A. Wise, E. Boose, and J. Hadley, "Experience in using a process language to define scientific workflow and generate dataset provenance," in *Proceedings of the ACM SIGSOFT 16th International Symposium on Foundations of Software Engineering*, Nov. 2008, pp. 319–329.

[60] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, Sep. 1975.