Safety in Discretionary Access Control for Logic-based Publish-Subscribe Systems

Kazuhiro Minami, Nikita Borisov, and Carl A. Gunter
University of Illinois at Urbana-Champaign
{minami, nikita, cgunter}@illinois.edu

ABSTRACT

Publish-subscribe (pub-sub) systems are useful for many applications, including pervasive environments. In the latter context, however, great care must be taken to preserve the privacy of sensitive information, such as users' location and activities. Traditional access control schemes provide at best a partial solution, since they do not capture potential inference regarding sensitive data that a subscriber may make. We propose a logic-based pub-sub system, where inference rules are used to both derive high-level events for use in applications as well as specify potentially harmful inferences that could be made regarding data. We provide a formal definition of safety in such a system that captures the possibility of indirect information flows. We show that the safety problem is co-NP-complete; however, problems of realistic size can be reduced to a satisfiability problem that can be efficiently decided by a SAT solver.

Categories and Subject Descriptors: C.2.4 [Distributed Systems]: Distributed applications; K.6.5 [Management of Computing and Information Systems]: Security and Protection

General Terms: Security

Keywords: Access control, inference control, safety, logical language, publish-subscribe systems

1. INTRODUCTION

Applications in pervasive environments will often take advantage of context information regarding their users and the surrounding environment to change their behaviors dynamically and better meet users' needs. To support such context-aware applications, several researchers have proposed an information dissemination middleware [3, 4] based on the publish-subscribe (pub-sub) model. A pub-sub middleware [2] supporting many-to-many communications between entities can efficiently disseminate events from environmental sensors to context-aware applications.

Pervasive environments also introduce considerable privacy concerns [8]. Sensors can provide detailed information about users' activities, and even sensors that monitor the environment, rather than the user directly, can provide information that reveals much about the users. For example, sensor readings on power consumption [13] or air conditioning differential pressure [12] can be used to detect not only the presence of activities, but the types of the activities in great detail. Therefore, it can be difficult to set up appropriate confidentiality policies on sensor readings, especially if discretionary control is to be given to users, who may find it difficult to reason about possible inferences.

To address this problem, we propose adding a logical framework to the pub-sub system in order to model inferences. The pub-sub system can use derivation rules to generate high-level events based on low-level ones it receives from sensors. For example, a system that receives an event from a key-card reader on a door followed by a motion sensor detector may generate a high-level event stating that the key-card owner is inside the office. This allows applications to subscribe to high-level events, rather than raw sensor values. It also allows the system to model the inference of sensitive information that can be derived from lower-level events.

For example, if a person's location is sensitive, the above inference suggests that the key-card and motion-sensor events may also be sensitive. We suggest that experts define a set of event derivation rules modeling how sensitive information may be derived in the system. Ordinary users can then define discretionary access control policies that protect high-level events about them, without worrying about the low-level sensor events; e.g., each user can specify a discretionary policy on their own location.

To enforce these discretionary policies, we must be able to tell when they might be violated. We introduce a formal definition of safety of a set of subscriptions with respect to a discretionary policy and a set of derivation rules based on the notion of non-deducibility [18]. Intuitively, a system is safe if, for any possible view of a subscriber, both the presence of a sensitive event and its absence are logical possibilities and thus no inference can be made. Verifying the safety of a policy can be complex, as knowledge of high-level events can be used to infer low-level ones and vice versa; indeed, we show that the safety problem is co-NP-complete. However, we show that we can efficiently reduce a safety problem to a satisfiability problem that can be used as input to a SAT solver. Our experiments with the modern SAT solver show policies of small to moderate size can be decided in a short period of time.

The rest of the paper is organized as follows. In Section 2, we introduce a reference model of a logic-based pub-sub system and formally define the safety based on that model. In Section 3, we present a set of inference rules used by subscribers and prove that our verification algorithm based on those rules are sound and complete. We also show that the safety problem is co-NP-complete. In Section 4, we show that how an instance of the safety problem can be reduced to the corresponding SAT problem, demonstrating co-NP-completeness. Our experimental results show that we can solve

SACMAT'09, June 3–5, 2009, Stresa, Italy.

safety problems of a realistic size with a SAT solver efficiently. Section 5 discuss related work and Section 6 concludes.

2. SYSTEM MODEL

In this section, we first describe how we model a logic-based pub-sub system that derives high-level events from raw events using derivation rules. We next introduce access control policies for protecting confidential events in the system. Since a malicious subscriber could infer confidential events from other events he or she receives, we define two access control policies, one that specifies which events a user is allowed to see directly, and one that specifies which event a user is allowed to infer. We finally define the safety of a pub-sub system based on the notion of nondeducibility.

2.1 Logic-based pub-sub system

A pub-sub system receives events from publishers and maintains those events in its knowledge base. When a subscriber sends the system a subscription request, the system publishes events that satisfy the requirements of that subscription request to the subscriber. We assume that each publisher and subscriber is managed by some principal in the set $\mathcal{P}$ of all principals. Also, we assume that a pub-sub system is managed by a single principal p_{PS} in $\mathcal{P}$; that is, principal p_{PS} defines all the security policies of the system.

In our model, a pub-sub system represents each event as a fact in Datalog; that is, each event e is a predicate followed by a parenthesized list of constants. For example, a location event of Bob could be expressed as $loc(Bob, room214)$. A pub-sub system protects confidential events with access control policies. We assume that those policies are public knowledge among subscribers.[1] When a principal wishes to begin a subscription, he issues a subscription request to the pub-sub system. A subscription request is a set of principal-event pairs in $\mathcal{P} \times \mathcal{E}$ where $\mathcal{E}$ is a set of all events. Receiving a subscription request, the pub-sub system checks whether the subscriber satisfies the access control policies on the event in the subscription request. If the subscriber's request is granted, the pub-sub system starts publishing the subscriber the requested event e with a timestamp t periodically.

2.2 Event derivation rules

We consider a logic-based pub-sub system that derives high-level events from low-level events that are obtained from publishers (e.g., sensors). The system maintains a set I of event derivation rules, which are Datalog clauses of the form:

$$e \leftarrow e_1, \ldots, e_n.$$

Each atom in the rule consists of a predicate and an ordered list of variables and constants. Each variable in the head of a rule must also appear in the body of the same rule such that the set of all facts that can be derived from a set of Datalog clauses is finite. We represent event e as an atom (fact) containing only constants, and thus $\mathcal{E}$ is defined as a set of all ground facts. For example, a location event about Bob can be expressed as $location(bob, room25)$. Every time the system receives a new event, it derives all the high-level events by applying all the event derivation rules on that new event in a bottom-up way. For example, suppose that a pub-sub system maintains the rule below.

$$location(P, L) \leftarrow DoorBell(P, D), partOf(D, L)$$

The system derives a location event, $location(bob, room25)$, from a Doorbell event, $doorbell(bob, door25)$, and a containment event, $partOf(door25, room25)$, which represents the fact "$door25$ belongs to $room25$." (In this case, the containment event is a static fact maintained by the system permanently.)

A pub-sub system maintains a set of events, E_{PS}, which is a subset of event set $\mathcal{E}$. Set E_{PS} consists of two disjoint event sets E^L_{PS} and E^H_{PS}. Set E^L_{PS} contains raw events that a pub-sub system receives from publishers, and set E^H_{PS} contains events that the system derives from events in E^L_{PS} by applying derivation rules. Set E_{PS} is the least fixed point of the immediate consequence operator T_I below.

$$T_I(E') = \{e\theta \mid (e \leftarrow e_1, \ldots, e_n) \in I \text{ and } e_i\theta \in E' \text{ for each } i, \text{ or } e\theta \in E'\},$$

where θ is a substitution function that maps variables to constants.

2.3 Discretionary and operational access control policies

We introduce discretionary access control (DAC) policies that allow a pub-sub system to protect confidential events from unauthorized subscribers. DAC policies specify which events a user is allowed to learn their truth directly or through inference, and which events must be kept confidential. We consider that a user learns the truth of an event if he knows whether the system maintains that event in its knowledge base. We represent DAC policies by the function $dacl : \mathcal{E} \rightarrow 2^{\mathcal{P}}$ where $\mathcal{P}$ is a set of all principals; that is, a subscriber p_i is authorized to know the truth of an event e only if $p_i \in dacl(e)$.

We expect that, in practice, DAC policies would be specified explicitly on some subset of high-level events, with the rest being left in a permissive state, that is, if an administrator p_{PS} does not define an DAC policy on event e explicitly, the system assigns the policy $dacl(e) = \mathcal{P}$. This is because most events are sensitive only insofar as they allow inferences about other events to be made. For example, the event $motionSensor(BobsOffice)$ might be used to derive $location(Bob, BobsOffice)$ and thus would need to be protected. However, by properly specifying inference rules, we would need to only define the derived event of Bob's location as confidential by, for example, giving Bob discretionary control over $dacl(location(Bob, L))$. The low-level events that can be used to derive Bob's location would be *automatically* protected by the pub-sub system through requiring an appropriate operational access control (OAC) policy, without requiring users to explicitly reason about low-level events.

The OAC policy represents what events the pub-sub system will reveal to a user directly. We model it as $oacl : \mathcal{E} \rightarrow 2^{\mathcal{P}}$. The system will deliver an event e to subscriber p_i if and only if:

1. $e \in E_{PS}$,
2. $p_i \in oacl(e)$, and
3. p_i has subscribed to e.

In other words, users' subscriptions are restricted by the $oacl$ function. To protect confidential events from inferences, an OAC policy must be usually more restrictive than the DAC policy; that is, $oacl(e) \subseteq dacl(e)$ for every e. We next define what makes an OAC policy safe.

2.4 Safety conditions

The question we would like to answer is whether OAC policies, which define the operational semantics of a pub-sub system, protect facts listed as confidential in the DAC policies from inference.

[1] It is not actually necessary that the policies be public; our assumptions rather ensures that knowledge of access control policies does not help a user infer facts in violation of the policy. We also do not aim to protect policies themselves from inference.

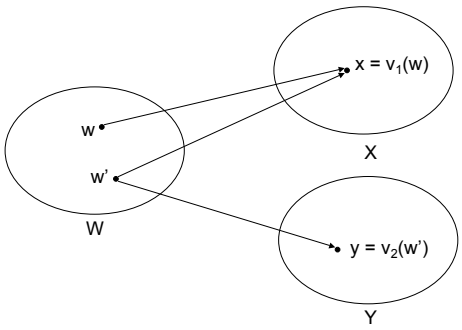

Figure 1: Concept of nondeducibility. For every world $w \in W$ and every value y in Y, there must exist world w' such that $v_1(w') = v_1(w)$ and $v_2(w') = y$.

Our safety conditions are based on the notion of *nondeducibility* introduced by Sutherland [18]. Sutherland modeled inferences by considering the set of all possible "worlds" $\mathcal{W}$; i.e., configurations of the system, and introduced the notion of an information function that represents a certain view of the system. That is, all the information about a given world $w \in \mathcal{W}$ is provided as the outputs of information functions that operate on w.

Sutherland describes information flows from function $v_1 : \mathcal{W} \rightarrow X$ to $v_2 : \mathcal{W} \rightarrow Y$ in a world $w \in \mathcal{W}$ as follows. Here $\mathcal{W}$ is a set of all the possible worlds, and X and Y are the ranges of the functions v_1 and v_2 respectively. Given $x = v_1(w)$, we can deduce that a world w belongs to a set of worlds $S \subseteq \mathcal{W}$ where $S = \{w' \mid w' \in \mathcal{W}, v(w') = x\}$. If there is no world $w' \in S$ such that $v_2(w') = y$ for some $y \in Y$, then we can eliminate the possibility that $v_2(w) = y$, and we thus learn the information on $v_2(w)$ from $v_1(w)$. Sutherland's nondeducibility in Definition 1 prohibits such information flow from function v_1 to v_2.

DEFINITION 1 (NONDEDUCIBILITY). *Given two information functions, $v_1 : \mathcal{W} \rightarrow X$ and $v_2 : \mathcal{W} \rightarrow Y$, we say that no information flows from v_1 to v_2 if for every world $w \in \mathcal{W}$, and for $y \in Y$, there exists $w' \in \mathcal{W}$ such that $v_1(w) = v_1(w')$ and $y = v_2(w')$.*

We apply the notion of nondeducibility to a pub-sub system to define the safety of the system. We consider a pub-sub system that is parameterized with a set of events $\mathcal{E}$, a set of derivation rules I, DAC policies $dacl$, and OAC policies $oacl$. We denote such a pub-sub system by $PS[\mathcal{E}, I, dacl, oacl]$. We assume that all the system parameters are public knowledge with subscribers. Particularly, every subscriber knows a set of derivation rules I, and thus our safety definition must ensure that no subscriber infers the truth of confidential events in the system from non-confidential events it receives by examining the derivation rules in I.

We define the safety of a pub-sub system by defining a set of possible worlds $\mathcal{W}$, and two information functions v_1 and v_2 in the following way. In a pub-sub system, a world $w \in \mathcal{W}$ corresponds to a set of events $E_{PS} \subseteq \mathcal{E}$ where E_{PS} is a set of events that are maintained by the system. The function v_1 in Definition 2 represents information that a subscriber p_i receives from the pub-sub system with respect to the system's OAC policies $oacl$.

DEFINITION 2. *We define function $v_1 : 2^{\mathcal{E}} \rightarrow 2^{\mathcal{E}}$ such that:*

$$v_1(E_{PS}) = \{e \mid e \in E_{PS} \wedge p_i \in oacl(e)\}$$

Note that function v_1 is defined with OAC policies $oacl$, which is given as a system parameter of the system. The function v_1 takes a set of events E_{PS} in $2^{\mathcal{E}}$ as an input and outputs the set of events that p_i receives from the system.

The function v_2 represents a subset of E_{PS}, which contains events that a subscriber p_i is not allowed to learn their truth with respect to the system's DAC policies. The range of function v_2 is the set $\overline{E_i}$ defined as follows.

DEFINITION 3.

$$\overline{E_i} = \{e \mid e \in \mathcal{E} \wedge p_i \notin dacl(e)\}$$

Set $\overline{E_i}$ contains events in event set $\mathcal{E}$ whose truth a subscriber p_i is prohibited to learn according to the $dacl$ policies. The function v_2 in Definition 4 takes as an input a set E_{PS} and outputs a set $\overline{E'} \subseteq \overline{E}$. Set $\overline{E'}$ represents confidential events that must be protected from p_i.

DEFINITION 4. *We define the function $v_2 : 2^{\mathcal{E}} \rightarrow \overline{E_i}$ such that:*

$$v_2(E_{PS}) = \{e \mid e \in E_{PS} \wedge p_i \notin dacl(e)\}$$

We now define the safety of a pub-sub system based on nondeducibility in Definition 1.

DEFINITION 5 (SAFETY). *We say that a pub-sub system $PS[\mathcal{E}, I, dacl, oacl]$ is safe with respect to a subscriber p_i if for every event set $E_{PS} \subseteq \mathcal{E}$ and every event set $\overline{E'_i} \subseteq \overline{E_i}$, there exists another event set $E'_{PS} \subseteq \mathcal{E}$ such that:*

1. *$v_1(E_{PS}) = v_1(E'_{PS})$, and*
2. *$\overline{E'_i} = v_2(E'_{PS})$.*

Intuitively, these safety conditions require a system to ensure that a principal p_i derives the same event set (i.e., $v_1(E_{PS})$) regardless of the system's maintaining any subset of events in $\overline{E_i}$ as part of E_{PS}. However, the safety conditions in Definition 5 turn out to be too strong for most applications in pervasive computing. For example, suppose that we consider two types of events $location(P, B)$ and $occupied(B)$. The event $location(P, L)$ represents the fact that principal P is in building B, and the event $occupied(B)$ represents the fact that building B is occupied with someone. We also have the inference rule $occupied(B) \leftarrow location(P, B)$ to derive an occupancy event from an location event. Suppose that location events about all the users are confidential from a subscriber p_i, but p_i receives an occupancy event derived from some confidential location event. Although the occupancy event does not reveal the identity of the person in the building, subscriber p_i can infer from the occupancy event that someone is in the building. According to the safety conditions in Definition 5, we are not allowed to leak this information to subscriber p_i because we need to construct an alternate event set E'_{PS} when $\overline{E'_i}$ is an empty set. However, it is not possible for the system to send the occupancy event if there is no location event in E'_{PS}.

Therefore, we consider weaker safety that protects the truth of each confidential event rather than a set of truth of multiple confidential events. The following weak safety definition ensures that unauthorized subscribers cannot determine whether a pub-sub system maintains a given confidential event or not.

DEFINITION 6 (WEAK SAFETY). *We say that a pub-sub system $PS[\mathcal{E}, I, dacl, oacl]$ is safe with respect to a subscriber p_i if for every event set $E_{PS} \subseteq \mathcal{E}$ and every event $e \in \overline{E_i}$, there exist two alternate event sets E'_{PS} and E''_{PS} in $2^{\mathcal{E}}$ such that:*

1. *$v_1(E_{PS}) = v_1(E'_{PS}) = v_1(E''_{PS})$,*
2. *$e \in v_2(E'_{PS})$, and*
3. *$e \notin v_2(E''_{PS})$.*

This weaker safety definition still provides users in pervasive environments with plausible deniability [7] to avoid unwanted social obligations or socially embarrassing situations. In the rest of the paper, we use the term "safety" to refer to the weak safety in Definition 6. Note that we do not address inference using a series of historical events in this paper.

Example 1: Consider the system PS with the following system parameters:

$$\begin{aligned}
&\mathcal{E} = \{location(bob, bldg12), location(alice, bldg12),\\
&\qquad occupied(bldg12)\},\\
&E_{PS} = \{location(bob, bldg12), occupied(bldg12)\},\\
&I = \{occupied(B) \leftarrow location(P, B)\}, \text{ and}\\
&dacl = oacl = \{((location(bob, bldg12), \emptyset),\\
&\qquad\qquad (location(alice, bldg12), \emptyset),\\
&\qquad\qquad ((occupied, bldg12), \{dave\})\}.
\end{aligned}$$

We here assume that every subscriber knows that variables P and B in the derivation rules of set I can be instantiated only with constants *bob* and *alice*, and constant $bldg12$, respectively. Event $occupied(bldg12)$ is a high-level event, which could be derived either from low-level event $location(bob, bldg12)$ or $location(alice, bldg12)$ with the rule in set I. DAC policies specify that user *dave* can learn only the truth of the high-level occupancy event, while he cannot learn the truth of the location events. OAC policies, which are the same as the DAC policies, allow the system to publish only the occupancy event to *dave*. We thus obtain the function $v_1(E_{PS})$ in Definition 2 and the set of confidential events $\overline{E_i}$ in Definition 3 as follows:

$$v_1(E_{PS}) = \{occupied(bldg12)\}$$
$$\overline{E_i} = \{location(bob, bldg12), location(alice, bldg12)\}$$

For each event in $\overline{E_i}$, we need to construct two event sets E'_{PS} and E''_{PS} that satisfy the three conditions in Definition 6. We first consider event $location(bob, bldg12)$ in $\overline{E_i}$ and define E'_{PS} and E''_{PS} as follows:

$$\begin{aligned}
E'_{PS} = &\quad \{location(bob, bldg12), occupied(bldg12)\}\\
E''_{PS} = &\quad \{location(alice, bldg12), occupied(bldg12)\}
\end{aligned}$$

The first condition in definition 6 is satisfied because

$$v_1(E'_{PS}) = v_1(E''_{PS}) = \{occupied(bldg12)\};$$

that is, the occupancy event $occupied(bldg12)$ can be derived from either of the two location events $location(alice, bldg12)$ or $occupied(bldg12)$. The second and third conditions in definition 6 are also satisfied because

$$location(bob, bldg12) \in v_2(E'_{PS}) \text{ and}$$
$$location(alice, bldg12) \notin v_2(E''_{PS}).$$

We can clearly see that the other event $location(alice, bldg12)$ in $\overline{E_i}$ satisfies the three conditions with the same event sets E'_{PS} and E''_{PS}. Therefore, the system in this example is safe, and this conclusion matches our intuition that disclosing the occupancy event is safe if it can be derived from multiple location events about different users.

3. COMPLEXITY ANALYSIS

In this section, we first introduce inference rules that capture subscribers' inference. We next define the alternate safety definition based on those inference rules and show the equivalence of the alternate definition with the original definition in Definition 6. Finally, we show that the safety problem is co-NP-hard.

$$\frac{e \leftarrow e_1, \ldots, e_n \in I \quad \forall i : val(e_i) = T}{val(e) = T} \tag{S1}$$

$$\frac{\begin{array}{c} e \leftarrow e_{i,1}, \ldots, e_{i,n_i} \in I \text{ for } i = 1, \ldots, m \\ val(e_{i,j} = F) \end{array}}{val(e) = F} \tag{S2}$$

$$\frac{\begin{array}{c} e \leftarrow e_{i,1}, \ldots, e_{i,n_i} \in I \text{ for } i = 1, \ldots, m \\ val(e) = E \ \ (E = T \vee E = F) \end{array}}{\begin{array}{c} val((e_{1,1} \wedge \ldots \wedge e_{1,n_1}) \vee \ldots \\ \vee (e_{m,1} \wedge \ldots \wedge e_{m,n_m})) = val(E) \end{array}} \tag{S3}$$

$$\frac{val(x_1 \vee \ldots \vee x_m) = T \quad val(x_m) = F}{val(x_1 \vee \ldots \vee x_{m-1}) = T} \tag{S4}$$

$$\frac{val(x_1 \wedge \ldots \wedge x_m) = F \quad val(x_m) = T}{val(x_1 \wedge \ldots \wedge x_{m-1} = F)} \tag{S5}$$

$$\frac{val(x_1 \wedge \ldots \wedge x_m) = T}{\forall i : val(x_i) = T} \tag{S6}$$

$$\frac{val(x_1 \vee \ldots \vee x_m) = F}{\forall i : val(x_i) = F} \tag{S7}$$

Figure 2: S-inference rules I_s.

3.1 Inference by subscribers

We first consider what kinds of inferences are possible to a subscriber. We need to be concerned with both forward and backward inference as follows. We assume that a subscriber knows derivation rules I used by the system because the subscriber must understand the semantics of high-level events by knowing how those events are derived by the system. In the following discussion, we assume that all the derivation rules only contain constants; we can obtain such rules by instantiating derivation rules with constants. Consider the derivation rule below.

$$e \leftarrow e_1, \ldots, e_n$$

If a subscriber p_i receives events $e_1, \ldots, e_n$, then p_i can infer that e is true even if p_i is not authorized to receive e. Similarly, suppose that the above rule is the only rule to derive event e. If p_i does not receive e_j even if p_i is authorized to receive events e_j (i.e., $p_i \in oacl(e_j)$), then p_i can infer that event e is false. Similarly, if subscriber p_i receives e, p_i can infer that events $e_1, \ldots, e_n$ are also true. Finally, if p_i, who is authorized to receive event e (i.e., $p_i \in oacl(e)$), does not receive e, then p_i can infer that there exists some event e_j, which is false; that is, $e_1 \vee \ldots \vee e_n$ is false.

We next consider the case where there are multiple derivation rules for deriving the same event e as follows:

$$e \leftarrow e_1, e \leftarrow e_2, \ldots, \text{ and } e \leftarrow e_m.$$

In general, each e_j could be the conjunction of multiple events $e_{j,1}, \ldots, e_{j,n_j}$. When p_i receives event e, p_i infers that there is some event e_j, which is true, because at least one of the rules must be applied to derive event e. However, p_i cannot determine which e_j is true. On the other hand, when p_i who is authorized to receive e does not receive e, p_i knows that every e_j is false for $i = 1$ to m.

We capture such a subscriber's inferences with the *s-inference* rules in Figure 2. In each s-inference rule, the statements above the horizontal line are conditions to derive the conclusion below that line. We use the term, s-inference rules, to distinguish them from a pub-sub system's derivation rules in set I. In s-inference rules,

we use the function $val : \mathcal{E} \rightarrow \{U, T, F\}$ that maps an input event e into one of the three values. Each value on an event represents a subscriber's knowledge about the validity of event e. Statement $val(e) = U$ represents the facts that the subscriber does not know the truth of event e. Statements $val(e) = T$, and $val(e) = F$ represent the facts that the truth of event e is known to true or false to the subscriber respectively.

The first two rules S1 and S2 represent forward inferences; the first rule says that if all of the events on the left-hand side are true, the right-hand side can be assumed to be true as well, representing *modus ponens*. The second rule states that if every rule representing the derivation of an event e contains an event $e_{i,j}$ that is known to be false, then the derived event e must be false as well.

Rule S3 captures backward inference. The rule says that the value of an event e must be equal to that of the disjunction of the conjuctive clauses, each of which represents the body of a derivation rule for event e. The rest of the rules are used to perform simple logical elimination; rules S4 and S5 eliminate an element from a disjunctive or conjunctive clause if it is known to be false or true respectively, and rules S6 and S7 show that in a true conjunction (resp., false disjunction), all the constituent clauses must be true (resp., false).

Example 2: We give an example that shows how a subscriber could perform inferences using s-inference rules and deduce the truth of a confidential event. Consider the system PS with the following system parameters:

$$\begin{aligned} E_{PS} = \{&location(dave, seclab),\\ &ta(cs461, alice), ta(cs461, bob),\\ &ta\_room(cs461, seclab),\\ &occupied(seclab)\},\\ I = \{&occupied(L) \leftarrow location(P, L),\\ &ta\_available(C) \leftarrow ta\_room(C, L), ta(C, P),\\ &\qquad location(P, L)\},\\ dacl = oacl = \{&(location(P, seclab), \emptyset),\\ &(ta(cs461, P), \{tom\}),\\ &(ta\_room(cs461, seclab), \{tom\}),\\ &(occupied(seclab), \mathcal{P}),\\ &(ta\_available(cs461), \{tom\})\}. \end{aligned}$$

The system maintains the location event about Dave ($dave$), two events stating that Alice ($alice$) and Bob (bob) are a teaching assistant (TAs) for course CS461. The system also maintains the event about the occupancy of the security laboratory ($seclab$), which is derived from the fact that Dave is at the security laboratory (i.e., $location(dave, seclab)$). Subscriber Tom (tom), who has registered for course CS461, can receive all the events maintained by the system except for the location event about Dave. However, Tom can illegally figure out that Dave is in the security laboratory if he knows that:

$$\begin{aligned} val(ta\_available(cs461)) &= F\\ val(occupied(seclab)) &= T \end{aligned}$$

1 Apply s-inference rule S3 to the derivation rule,

$$occupied(L) \leftarrow location(P, L),$$

and infer that either Alice, Bob, or Dave must be in the security laboratory; i.e.:

$$\begin{aligned} val(location(alice, seclab) \vee location(bob, seclab)&\\ \vee location(dave, seclab)) &= T \end{aligned}$$

We here assume that the set of principals $\mathcal{P}$ contains only those three people.

2 Apply s-inference rule S3 to the derivation rule,

$$ta\_available(C) \leftarrow ta\_room(C, L), ta(C, P), location(P, L),$$

to obtain:

$$\begin{aligned} val((&ta\_room(cs461, seclab) \wedge ta(cs461, alice)\\ &\wedge location(alice, seclab))\\ \vee(&ta\_room(cs461, seclab) \wedge ta(cs461, bob)\\ &\wedge location(bob, seclab))) = \quad F \end{aligned}$$

Applying rule S7, Tom determines that:

$$\begin{aligned} val(&ta\_room(cs461, seclab) \wedge ta(cs461, alice)\\ &\wedge location(alice, seclab)) = \quad F\\ val(&ta\_room(cs461, seclab) \wedge ta(cs461, bob)\\ &\wedge location(bob, seclab)) = \quad F \end{aligned}$$

3 According to the $oacl$ rules, Tom knows that $ta\_room(cs461, seclab)$, $ta(cs461, alice)$, and $ta(cs461, bob)$ are all true. Therefore, he can use rule S5 to determine that:

$$\begin{aligned} val(location(alice, seclab)) &= F\\ val(location(bob, seclab)) &= F \end{aligned}$$

4 Finally, Tom uses uses rule S4, applied to his finding from the first step, to obtain that:

$$val(location(dave, seclab)) = T$$

Notice that, even in this simple example, knowledge of TA assignments and possible occupants of $seclab$ was necessary to determine that this policy violates safety. In general, we want an algorithm for verifying safety automatically.

3.2 Safety verification algorithm

We show the algorithm for verifying the safety in Definition 6. Figure 3 is a pseudocode of the function VERIFYSAFETY, which takes as inputs a set of events $\mathcal{E}$, DAC policies $dacl$, OAC policies $oacl$, a set of derivation rules I, and a subscriber p_i. The function VERIFYSAFETY returns TRUE if the system satisfies the safety condition with respect to subscriber p_i.

The algorithm iterates the following process for every T/F-value assignment function $A : \{e \mid p_i \in oacl(e)\} \rightarrow \{T, F\}$. In the beginning of each iteration, the algorithm assigns a boolean value $A(e)$ to event e if $p_i \in oacl(e)$, and a value U to the other events. The function next updates values of events by applying the s-inference rules in Figure 2. The function RULEAPPLICATION(r, r') applies an s-inference rule r in Figure 2 on a derivation rule $r' \in I$ and updates the values of events involved in the rule, and outputs a set of events e' whose values have been updated. Every time the value of an event e is updated, the algorithm checks whether p_i belongs to $dacl(e)$ or not. If $p_i \notin dacl(e)$, the algorithm returns FALSE meaning that the system is unsafe. Since the number of events to be updated is finite, the algorithm eventually terminates.

```
VERIFYSAFETY(E, dacl, oacl, I, p_i)
 1  for each T/F-value assignment A for events whose oacl includes p_i
 2      do ▷ Initialize the value of each event e to U.
 3         for each e ∈ E where p_i ∉ oacl(e)
 4             do val(e) ← U
 5         for each e ∈ E where p_i ∈ oacl(e)
 6             do val(e) ← A(e)
 7         Changed ← TRUE
 8         ▷ Update the values of events with s-inference rules in I_s
 9         while Changed
10         Changed ← FALSE
11             do for each s-inference rule r in I_s and each derivation rule r' in I
12                    do s ← RULEAPPLICATION(r, r')
13                       ▷ s contains a set of events whose values are updated
14                       if s ≠ ∅
15                          then Changed ← TRUE
16                       if there is some event e in s where p_i ∉ dacl(e)
17                          then return FALSE
18  return TRUE
```

Figure 3: Algorithm for safety verification

If there is no illegal assignment of value T or F, the function returns TRUE at the end. The algorithm is super-polynomial because the algorithm examines every possible value assignment function.

We show that a pub-sub system satisfies the safety condition in Definition 6, if and only if there is no confidential event derived by a subscriber applying the s-inference rules to a set of events received from the pub-sub system.

THEOREM 1 (SOUNDNESS). *If the function* VERIFYSAFETY $(\mathcal{E}, dacl, oacl, I, p_i)$ *returns* TRUE*, then a pub-sub system* $PS[\mathcal{E}, I, dacl, oacl]$ *is weakly safe with respect to a subscriber* p_i.

PROOF. We prove the theorem by induction as follows.
Base case: Suppose that the function VERIFYSAFETY returns TRUE with no iteration in the for-block in lines 11–17 for every assignment A. That implies that there is no derivation rule $e \leftarrow e_1, \ldots, e_n$ in I where $p_i \in oacl(e)$ or $p_i \in oacl(e_j)$ for $j = 1$ to n. Therefore, we can construct two event sets E_{PS} and E'_{PS} as follows:

$$E_{PS} = \{e \in \mathcal{E} \mid p_i \in oacl(e)\} \cup \{e \in \mathcal{E} \mid p_i \notin dacl(e)\} \text{ and}$$
$$E'_{PS} = \{e \in \mathcal{E} \mid p_i \in oacl(e)\}.$$

Two sets E_{PS} and E'_{PS} satisfy the three conditions in Definition 6 for every confidential event e_c where $p_i \notin dacl(e_c)$.
Induction step: Suppose that the system is safe if the function VERIFYSAFETY returns TRUE with no more than k iterations in the for-block in lines 11–17 for every assignment A. Then, if the value of event e is T or F, subscriber p_i belongs to $dacl(e)$. Suppose that we get a confidential event e_c violating the safety of the system at step $k+1$. Subscriber p_i must infer the truth of e_c from the events whose values become T or F at the kth iteration of the for-block. Otherwise, p_i should be able to infer the truth of e_c at an earlier iteration.

We claim that there must exist derivation rule r involving the confidential event e_c whose value is U because it is impossible to infer the truth of an event that does not have any logical dependency with other events. If subscriber p_i learns the truth of event e_c, it must use a different form of inference that is not captured by s-inference rules in Figure 2. However, this is not possible; we show that if subscriber p_i performs such inference, we can construct two event sets E_{PS} and E'_{PS} that satisfy the conditions in Definition 6. We first consider the case that the confidential event e_c appears in the head of rule r and next consider the case that e_c appears in the body of rule r. When event e_c is the head of rule r, the body of r contains events with value T and other events with value U. Otherwise, the function VERIFYSAFETY assigns value T or F by applying s-inference rule S1 or S2 on rule r respectively. We can construct two event sets E_{PS} and E'_{PS} for event e_c satisfying the safety conditions in Definition 6 as follows:

$$E_{PS} = \{e \mid val(e) = T\} \cup \{e_c\} \cup \{e_i \mid e_c \leftarrow e_1, \ldots, e_i, \ldots, e_n \in I \wedge val(e_i) = U\},$$
$$E'_{PS} = \{e \mid val(e) = T\}.$$

Set E_{PS} contains all the events with a value T after we run the function VERIFYSAFETY, the confidential event e, and the events with a value U in the body of rule r for deriving e. Set E'_{PS} only contains events with a value T.

Next, we consider the case that confidential event e_c appears in the body of derivation rule r for deriving event e_0. If the value of event e_0 is F, there must exist another event e_i whose value is either F or U in the body of rule r. Then, we can construct two event sets E_{PS} and E'_{PS} for event e_c satisfying the safety conditions as follows:

$$E_{PS} = \{e \mid val(e) = T\} \cup \{e_c\},$$
$$E'_{PS} = \{e \mid val(e) = T\} \cup \{e_i \mid e_0 \leftarrow e_1, \ldots, e_i, e_c, \ldots, e_n \in I \wedge val(e_i) = U\}.$$

Set E'_{PS} contains the events with a value U in the body of rule r as well as all the events with a value T. If the value of event e is T, there must exist another rule $e_0 \leftarrow e'_1, \ldots, e'_n$ where $val(e'_i) = T$ or $val(e'_i) = U$ for $i = 1$ to n. Then, we can construct two event sets E_{PS} and E'_{PS} satisfying the safety conditions as follows:

$$E_{PS} = \{e \mid val(e) = T\} \cup \{e_c\},$$
$$E'_{PS} = \{e \mid val(e) = T\} \cup \{e'_i \mid e_0 \leftarrow e'_1, \ldots, e'_n \in I \wedge val(e'_i) = U\}.$$

We also need to consider the case that subscriber p_i infers the truth of event e_c by knowing a conjunction or disjunction of events involving e_c. However, we can easily construct two event sets satisfying the safety when subscriber p_i does not use s-inference rules S4–S7. Since we show that there is no confidential event e_c with a value T or F at step $k+1$, we conclude that the function VERIFYSAFETY is sound. □

THEOREM 2 (COMPLETENESS). *If a pub-sub system* $PS[\mathcal{E}, I, dacl, oacl]$ *is weakly safe with respect to a subscriber* p_i*, then the function* VERIFYSAFETY$(\mathcal{E}, dacl, oacl, I, p_i)$ *returns* TRUE.

PROOF. We show that if the function VERIFYSAFETY returns FALSE, the system is not safe. If the function updates the value of an event e to either T or F where $e \notin dacl(e)$, it is clearly impossible to construct an alternate world where e has the opposite value. For example, consider the rule S1. If a subscriber p_i knowing the inference rule $e \leftarrow e_1, \ldots, e_n$ receives events $e_1, \ldots, e_n$, p_i learns that event e is also true. Even if $p_i \notin dacl(e)$, we cannot construct an alternate event set E'_{PS}, which does not include e while system PS publishes $e_1, \ldots, e_n$ to p_i. The same argument holds for the the other s-inference rules, and thus the safety verification with the function VERIFYSAFETY is complete. □

3.3 Complexity of the safety problem

We define the UNSAFE problem, which is the complement of the safety problem, as follows:

$$\text{UNSAFE} = \{(PS[\mathcal{E}, I, dacl, oacl], p_i) \mid \text{VERIFYSAFETY}(\mathcal{E}, dacl, oacl, I, p_i) = \text{FALSE}\}$$

THEOREM 3. *UNSAFE is NP-complete.*

PROOF. We first show that the UNSAFE problem is in NP. We consider a particular T/F-value assignment A as a certificate that shows that a given system PS is unsafe. We remove the outer most for-loop in the function VERIFYSAFETY such that it only computes the values of events from the initial value assignment given by particular A. This algorithm, which is similar to the computation of a fixpoint in Datalog, runs in polynomial time.

Next, we prove that UNSAFE is NP-hard by showing 3-CNF-SAT is polynomial-time reducible to the UNSAFE problem (i.e., 3-CNF-SAT $\leq_P$ UNSAFE). Let $\phi = C_1 \wedge \ldots \wedge C_n$ be a boolean formula in 3-CNF with k clauses. Our goal is to construct a set of derivation rules I, DAC policies $dacl$, and OAC policies $oacl$ for the UNSAFE problem such that the formula ϕ is satisfiable if and only if a pub-sub system $PS[\mathcal{E}, I, dacl, oacl]$ is unsafe. To achieve this goal, we introduce a confidential event s such that a subscriber learns the truth of s if and only if the formula ϕ is satisfiable.

We first show the construction of such a system $PS[\mathcal{E}, I, dacl, oacl]$. We use the following formula ϕ as a running example.

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3).$$

Let A be a set of literals that appear in ϕ. We introduce a set of events corresponding to literals in A, another set of events corresponding to clause C_i in ϕ, and a confidential event that is used to violate the safety conditions. We introduce an event x_i for each variable x_i in A and introduce nx_j for each atom $\neg x_j$ in A. We need event nx_j to handle the negation of a variable in ϕ. In the example, $A = \{x_1, \neg x_1, v_2, \neg x_2, x_3, \neg x_3\}$, and thus we include $\{x_1, nx_1, x_2, nx_2, x_3, nx_3\}$ in event set $\mathcal{E}$. System PS publishes both x_i and nx_i to subscriber p_i; that is,

$$p_i \in oacl(x_i) \text{ and } p_i \in oacl(nx_i).$$

We also introduce a triplet of events (u_i, z_i, z_i') for each event x_i and introduce a triplet of events (nu_i, nz_i, nz_i') for each event nz_i. In the above example, we introduce pairs of events (u_1, z_1, z_1'), (nu_1, nz_1, nz_1'), (u_2, z_2, z_2'), (nu_2, nz_2, nz_2'), (u_3, z_3, z_3'), and (nu_3, nz_3, nz_3'). The purpose of those triplets of events will become clear when we define derivation rules in I below. System PS does not publish events of the types u_i, z_i, z_i', nu_i, nz_i, nz_i' to subscriber p_i. However, DAC policies do not prohibit p_i from knowing the truth of those types of events; that is, for every i:

$$\begin{array}{ll} p_i \notin oacl(u_i), & p_i \in dacl(u_i), \\ p_i \notin oacl(z_i), & p_i \in dacl(z_i), \\ p_i \notin oacl(z_i'), & p_i \in dacl(z_i'), \\ p_i \notin oacl(nu_i), & p_i \in dacl(nu_i), \\ p_i \notin oacl(nz_i), & p_i \in dacl(nz_i), \\ p_i \notin oacl(nz_i'), & p_i \in dacl(nz_i'). \end{array}$$

We also introduce two events y_i and w_i corresponding to clause C_i in ϕ. System PS does not publish y_i and w_i to subscriber p_i, but DAC policies do not prohibit p_i from knowing the truth of y_i and w_i respectively; that is,

$$\begin{array}{ll} p_i \notin oacl(y_i), & p_i \in dacl(y_i), \\ p_i \notin oacl(w_i), & p_i \in dacl(w_i). \end{array}$$

We finally introduce confidential event s that subscriber p_i is not allowed to know its truth; that is, $p_i \notin dacl(s)$.

Next, we describe how we define derivation rules in I. For each clause C_i, we define three rules for deriving event y_i corresponding to clause C_i. For example, if $C_1 = x_1 \vee \neg x_2 \vee \neg x_3$ as in the example above, we have three rules as follows:

$$y_1 \leftarrow x_1,\ y_1 \leftarrow nx_2, \text{ and } y_1 \leftarrow nx_3.$$

We need to make sure that each pair of events (x_i, nx_i) are consistent; that is, $x_i = \neg nx_i$ for every x_i in $\mathcal{E}$. For example, variable x_1 appears in clause C_1 and $\neg x_1$ appears in C_2. Therefore, we have to define rules such that $x_1 = \neg nx_1$. We define a pair of rules for each event in the body of each rule above as follows:

$$\begin{array}{ll} x_1 \leftarrow nx_1, z_1, & x_1 \leftarrow u_1, z_1', \\ nx_2 \leftarrow x_2, nz_2, & nx_2 \leftarrow nu_2, nz_2', \\ nx_3 \leftarrow x_3, nz_3, & nx_3 \leftarrow nu_3, nz_3'. \end{array}$$

The first two rules ensure that subscriber p_i infers that event u_1 is true only when event x_1 is true and nx_1 is false. If p_i knows that x_1 is true, he infers that either $nx_1 \wedge z_1$ or $u_1 \wedge z_1'$ is true. If p_i also knows that nx_1 is false, then he concludes that $u_1 \wedge z_1'$ must be true, and thus u_1 is true. If x_1 is false, p_i infers that both conjunctions $nx_1 \wedge z_1$ and $u_1 \wedge z_1'$ are false, but p_i cannot determine whether u_1 is true because p_i does not know the truth of the other event z_1'. If event nx_1 as well as x_1 is true, p_i cannot infer the truth of event u_i because which one of the first two rules is used to derive x_1.

We add rules for deriving w_i for clause C_i. For clause C_1, we define the rules below:

$$w_1 \leftarrow u_1, w_1 \leftarrow nu_2, \text{ and } w_1 \leftarrow nu_3.$$

The above three rules ensure that subscriber p_i knows that event w_1 is true if and only if event y_1, which corresponds to the truth of clause C_1, is true and there is no inconsistent assignment with a pair of (x_i, nx_i) in event set $\mathcal{E}$. We can define derivation rules for clause C_2 in the same way.

$$\begin{array}{lll} y_2 \leftarrow nx_1, & y_2 \leftarrow x_2, & y_2 \leftarrow x_3, \\ nx_1 \leftarrow x_1, nz_1, & nx_1 \leftarrow nu_1, nz_1', & \\ x_2 \leftarrow nx_2, z_2, & x_2 \leftarrow u_2, z_2', & \\ x_3 \leftarrow nx_3, z_3, & x_3 \leftarrow u_3, z_3', & \\ w_2 \leftarrow nu_1, & w_2 \leftarrow u_2, & w_2 \leftarrow u_3. \end{array}$$

We finally add the following derivation rule for a confidential event s, which requires w_i is true for every clause C_i in ϕ.

$$s \leftarrow w_1, w_2.$$

We introduce five events for each variable x_i in ϕ, two events for each clause C_i in ϕ, and a single event for ϕ. We also define a DAC and a OAC policy of a constant size for each event. We introduce nine rules of a constant size for each clause C_i and a single rule whose size is proportional to the number of clauses in ϕ. Therefore, this reduction can be performed in polynomial time.

We now show that if a formula ϕ is satisfiable, then subscriber p_i can infer the truth of confidential event s, which violates the safety conditions of the pub-sub system PS. If ϕ is satisfiable, there exists truth/false assignments to the variables in ϕ such that every clause C_i is true. If we assign the truth of events in $\mathcal{E}$ such that $val(x_i) = T$ and $val(nx_i) = F$ if and only if x_i is assigned to be true in ϕ, then, as we describe above, subscriber p_i can infer that $val(s) = T$. Now we prove the converse. If system PS is unsafe, event s is known to be true because s is the only event in $\mathcal{E}$ where subscriber p_i does not satisfy its DAC policies. Also, if s is known to be true, every event w_i must be known be true. If w_i is known to be true, then one of the three events for clause C_i must be known to be true while satisfying constraints regarding the negation of an atom in formula ϕ. $\square$

4. EXPERIMENTS

In this section, we first show the reduction from the UNSAFE problem to the SAT problem to leverage the techniques of SAT solvers. We next show our experimental results using a SAT solver.

4.1 Reduction to the SAT problem

We convert a pub-sub system $PS(\mathcal{E}, I, dacl, oacl)$ into a SAT formula ϕ_j such that there is safety violation with respect to principal p_j if and only if the formula ϕ_j is satisfiable. Our basic approach is to have ϕ_j encode the application of a sequence of s-inference rules leading to a safety violation.

To model this, we introduce for each event $e_i \in \mathcal{E}$ two variables, Te_i and Fe_i, modeling the final state of the variable after application of the inference rules. We add a clause that ensures that each event e_i is known to be either true or false, but not both:

$$e_i \in \mathcal{E} \Longrightarrow (\neg Te_i \vee \neg Fe_i) \tag{1}$$

We represent a conversion rule from PS to formula ϕ_j using a long right arrow '$\Longrightarrow$'; that is, for each element in PS on the left-hand side of an arrow, we add a clause on the right into formula ϕ_j. For an event e_i that is released (i.e., $p_j \in oacl(e_i)$), we also include the constraint:

$$e_i \in \mathcal{E} \wedge p_j \in oacl(e_i) \Longrightarrow (Te_i \vee Fe_i) \tag{2}$$

to represent that the subscriber knows that the event is either true or false.

We then add constraints representing the s-inference rules. To simplify the problem, we first rewrite the inference rules into a normalized form such that, for every event $e_i \in \mathcal{E}$, there is either a single rule with e_i on the left-hand side ($e_i \leftarrow e_1, \ldots, e_n$) or all of the rules with e_i on the left-hand side contain a single event on the right-hand side ($e_i \leftarrow e_1$; …; $e_i \leftarrow e_n$), by means of introducing auxiliary variables. This ensures that each conjunction that appears in s-inference rule S3 is represented by a single event, and hence has a corresponding Te_i and Fe_i variable.

Rules S1 and S2 are modeled in ϕ_j as follows:

$$\begin{aligned} e_i \leftarrow & e_1, \ldots, e_n \in I \Longrightarrow \\ & ((Te_1 \wedge \ldots \wedge Te_n) \Rightarrow Te_i) \wedge && (3) \\ & ((Fe_1 \vee \ldots \vee Fe_n) \Rightarrow Fe_i) && (4) \end{aligned}$$

For a conjunctive rule, we insert clauses to ϕ_j modeling rule S3 combined with rules S6 and S5:

$$\begin{aligned} e_i \leftarrow & e_1, \ldots, e_n \in I \Longrightarrow \\ & (Te_i \Rightarrow (Te_1 \wedge \ldots \wedge Te_n)) && (5) \\ & \wedge\ ((Fe_i \wedge Te_2 \wedge \ldots \wedge Te_n) \Rightarrow Fe_1) && (6) \\ & \ldots \\ & \wedge\ ((Fe_i \wedge Te_1 \wedge \ldots \wedge Te_{n-1}) \Rightarrow Fe_n) && (7) \end{aligned}$$

Disjunctive clauses are modeled analogously:

$$\begin{aligned} (e_i \leftarrow & e_1; \ldots; e_i \leftarrow e_n) \in I \Longrightarrow \\ & (Fe_i \Rightarrow (Fe_1 \wedge \ldots \wedge Fe_n)) && (8) \\ & \wedge\ ((Te_i \wedge Fe_2 \wedge \ldots \wedge Fe_n) \Rightarrow Te_1) && (9) \\ & \ldots \\ & \wedge\ ((Te_i \wedge Fe_1 \wedge \ldots \wedge Fe_{n-1}) \Rightarrow Te_n) && (10) \end{aligned}$$

We also add a clause to ϕ_j modeling a safety violation:

$$e_i \in \mathcal{E} \wedge p_j \notin dacl(e_i) \Longrightarrow (Te_i \vee Fe_i) \tag{11}$$

A combination of these clauses ensures that, whenever there is a sequence of inference rules that produces a safety violation, ϕ_j is satisfiable. However, we need to add more clauses to make sure that ϕ_j is satisfiable *only when* there is a safety violation. To do so, we introduce an extra requirement that a variable Te_i or Fe_i be only true if it was derived by some inference rule, or it was released through the $oacl$ policies.

For each variable Te_i (likewise, Fe_i) such that $p_j \notin oacl(e_i)$, we collect all the clauses introduced above that have the form $x \Rightarrow Te_i$. Then we introduce an additional clause in formula ϕ_j:

$$Te_i \Rightarrow (x_1 \vee \ldots \vee x_n) \tag{12}$$

If there are no such clauses, then we instead add the clause $(\neg Te_i)$, since Te_i can never be derived by an inference rule.

This nearly solves the problem, but it introduces the possibilities of inference loops. For example, if there are two variables, e_1 and e_2 with a simple rule $e_1 \leftarrow e_2$, then we would introduce the clauses in formula ϕ_j:

$$Te_1 \Rightarrow Te_2$$

$$Te_2 \Rightarrow Te_1$$

Setting both Te_1 and Te_2 would be a satisfying assignment, even if there are no inference rules to derive the values of either e_1 or e_2 in the rest of the system. To address this problem, we add auxiliary variable $B_{j,i}$ for each event pair $(e_i, e_j) \in \mathcal{E}$ that tracks the order of inference rules used to derive the value of each variable. We update clause (12) to include the requirements that the variables on the left-hand side be known before event e_i:

$$Te_i \Rightarrow ((x_1 \wedge \bigwedge_{e_j \in V(x_1)} B_{j,i}) \vee \ldots \vee (x_n \wedge \bigwedge_{e_j \in V(x_n)} B_{j,i})), \tag{13}$$

where $V(x_k)$ denotes a set of variables that appear in clause x_k. This ensures that any variable in x_k is known before e_i. Thus, the above clauses regarding rule $e_1 \leftarrow e_2$ are modified as follows:

$$Te_1 \Rightarrow (Te_2 \wedge B_{2,1})$$

$$Te_2 \Rightarrow (Te_1 \wedge B_{1,2})$$

We also introduce the requirement that the 'before' relation $B_{i,j}$ is antisymmetric and transitive:

$$\begin{aligned} & e_i, e_j \in \mathcal{E} \Longrightarrow (\neg B_{i,j} \vee \neg B_{j,i}) && (14) \\ & e_i, e_j, e_k \in \mathcal{E} \Longrightarrow ((B_{i,j} \vee B_{j,k}) \Rightarrow B_{i,k}) && (15) \end{aligned}$$

Together, these rules ensure that, if any variable Te_i (or, likewise, Fe_i) is found to be true in a satisfying assignment, there exists a sequence of inference rules deriving Te_i that starts from variables release through the $oacl$ policies.

Example 3: Consider a pub-sub system PS with the following system parameters: $\mathcal{E} = \{e_1, e_2, e_3\}$, $I = \{e_1 \leftarrow e_2, e_3\}$, $p_l \in dacl(e_1)$, and $p_l \in oacl(e_3)$. We convert the safety problem in

PS into the following formulas ϕ_j.

$$\phi_j =$$
$$[(1)](\neg Te_1 \vee \neg Fe_1) \wedge (\neg Te_2 \vee \neg Fe_2) \wedge (\neg Te_3 \vee \neg Fe_3) \quad (16)$$
$$[(2)] \wedge (Te_3 \vee Fe_3) \quad (17)$$
$$[(4)] \wedge (Fe_2 \vee Fe_3) \Rightarrow Fe_1 \quad (18)$$
$$[(3)] \wedge (Te_2 \wedge Te_3) \Rightarrow Te_1 \quad (19)$$
$$[(5)] \wedge Te_1 \Rightarrow (Te_2 \wedge Te_3) \quad (20)$$
$$[(6)] \wedge (Fe_1 \wedge Te_2) \Rightarrow Fe_3 \quad (21)$$
$$[(7)] \wedge (Fe_1 \wedge Te_3) \Rightarrow Fe_2 \quad (22)$$
$$[(13)] \wedge Fe_1 \Rightarrow ((Fe_2 \wedge B_{2,1}) \vee (Fe_3 \wedge B_{3,1})) \quad (23)$$
$$[(13)] \wedge Te_1 \Rightarrow ((Te_2 \wedge Te_3 \wedge B_{2,1} \wedge B_{3,1})) \quad (24)$$
$$[(13)] \wedge Te_2 \Rightarrow (Te_1 \wedge B_{1,2}) \quad (25)$$
$$[(13)] \wedge Fe_2 \Rightarrow (Fe_1 \wedge Te_3 \wedge B_{1,2} \wedge B_{3,2}) \quad (26)$$
$$[(14)] \wedge (\neg B_{1,2} \vee \neg B_{2,1}) \wedge (\neg B_{1,3} \vee \neg B_{3,1}) \wedge (\neg B_{2,3} \vee \neg B_{3,2}) \quad (27)$$
$$[(15)] \wedge (\forall i, j, k : (B_{i,j} \wedge B_{j,k}) \Rightarrow B_{i,k}) \quad (28)$$
$$[(11)] \wedge (Te_2 \vee Fe_2). \quad (29)$$

This formula is unsatisfiable, and hence the system is safe. To see this, suppose Te_2 were true. Then Te_1 and $B_{1,2}$ must be true (25). This implies that $(Te_2 \wedge Te_3 \wedge B_{2,1} \wedge B_{3,1})$ must be true (24). But $B_{1,2}$ and $B_{2,1}$ cannot be true at the same time (27).

On the other hand, if Fe_2 is true, then we must also set Fe_1, Te_3, $B_{1,2}$, and $B_{3,2}$ all to be true (26). On the other hand, Fe_1 requires that $(Fe_2 \wedge B_{2,1})$ be true or $(Fe_3 \wedge B_{3,1})$ is true (23). The former clause violates antisymmetry (27) and the latter requires that e_3 be both true and false at the same time (16).

4.2 Experimental results

We measured latency for solving the SAT problems, which are reduced from UNSAFE problems. We used SAT4J [1], which is a Java-based SAT solver, and conducted our experiments with a machine running Mac OS X version 1.4.2 and Sun Microsystem's Java runtime (v.1.5.0) with 2GHz Intel Core Duo processor and 1.5GB RAM.

We generated system parameters, event set $\mathcal{E}$, derivation rules I, DAC policies $dacl$ and OAC policies $oacl$ randomly by specifying the number of events in $\mathcal{E}$ (ENUM), the number of derivation rules in I (INUM), the number of events (OAC_NUM) that subscriber p_l receives, the number of events (NO_DAC_NUM) that p_l is not authorized to learn with respect to DAC policies, and the average size of events in the bodies of the derivation rules (SIZE). We used the parameters in Table 1. Although those parameters do not consider particular applications, we believe that we covered the case where number of events and derivation rules are large enough to meet the needs of practical pub-sub systems in pervasive computing.

Figure 4 shows latencies for evaluating the reduced SAT problems. We measured the latencies of ten different policies for each number of events in set $\mathcal{E}$ and took the average of them. Although the latency grows as the number of events grows, the latency for the SAT problem, which corresponds to 70 events and 56 derivation rules in a pub-sub system, is still less than ten milliseconds.

We wrote a program for converting a safety problem into the corresponding SAT problem in Python. The conversion process took an order of ten minutes when we chose a system configuration involving 70 events. However, we believe that we can make this process an order of magnitude or two faster by implementing the conversion program with a low-level language such as C.

ENUM	INUM	OAC_NUM	NO_DAC_NUM	SIZE
10	8	3	1	2
20	16	6	2	2
30	24	9	3	2
40	32	12	4	2
50	40	15	5	2
60	48	18	6	2
70	56	21	7	2

Table 1: Parameters for the experiments.

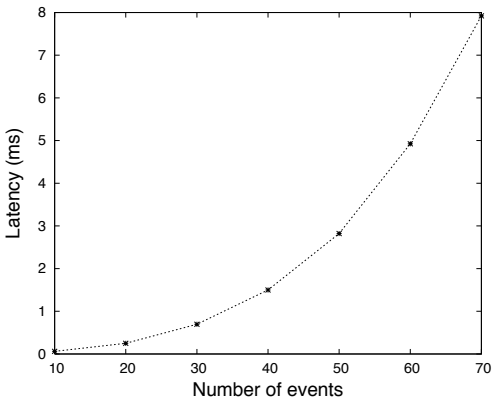

Figure 4: Latency for satisfiability check.

5. RELATED WORK

Several researchers [11, 14, 20] have studied policy languages and enforcement mechanisms for limiting access to events in pub-sub systems. However, none of the researchers consider safety in a pub-sub system that derives high-level events using logical derivation rules.

Inference control has been studied extensively in the field of statistical database systems [5]. Statistical database systems prevent unauthorized inferences by limiting certain types of queries (e.g., query size and query overlap control) and/or by modifying query results (e.g., cell suppression and response perturbation). On the other hand, our approach based on OAC policies does not require the modification of events to be published.

Inference control in multilevel secure database systems [16, 17] prevents a low-clearance user from inferring high-classification data using functional dependencies in the database system. However, the inference control in those systems does not address the issue of backward inferences concerning functional dependencies.

Considerable research has been done for safety analysis of DAC systems (e.g., [6, 9, 15]) that decides whether a system reaches an unsafe state where an unauthorized subject has a particular access right. More recently, Li et al. [10] studied the computational complexity of the safety problem for trust management systems where role memberships (i.e., access rights) of users are updated based on logical rules. The safety problem in the previous studies is orthogonal to that of ours whose focus is to ensure that discretionary access control policies are preserved under the presence of malicious users performing inferences.

Winsborough and Li [19] formally define safety in the framework of automatic trust negotiation. Their safety definition is based on the notion of *indistinguishability* about the possession of a set of credentials by a negotiating party. They studied inferences in ATN through the behaviors of the negotiating party, whereas our focus

is a subscriber's inferences that consider the derivation rules of a pub-sub system.

6. SUMMARY

We consider pub-sub systems that derives high-level events from raw events using derivation rules, and provide the formal definition of safety that considers a subscriber's inferences about confidential events. We develop a safety verification algorithm, which is sound and complete, by introducing a set of inference rules that capture what a subscriber can infer. We prove that the safety problem is co-NP-complete. However, we show that the safety problem is polynomially reducible to the SAT problem, which can be solved efficiently with an existing SAT solver. Our experimental results show that it is feasible to verify the safety of a pub-sub system with moderate number of events and derivation rules.

Acknowledgments

We would like to thank Mahesh Tripunitara for his help with the final version of the paper. This work was supported in part by NSF CNS 07-16626, NSF CNS 07-16421, NSF CNS 05-24695, ONR N00014-08-1-0248, NSF CNS 05-24516, NSF CNS 05-24695, DHS 2006-CS-001-000001, and grants from the MacAruthur Foundation and Boeing Corporation. The views expressed are those of the authors only.

7. REFERENCES

[1] Sat4j: Bringing the power of sat technology to the java platform, http://www.sat4j.org/.

[2] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[3] Guanling Chen, Ming Li, and David Kotz. Design and implementation of a large-scale context fusion network. In *Proceedings of the International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 246–255, August 2004.

[4] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction Journal*, 16(2-4):97–166, 2001.

[5] Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases, From Theory to Practice*. Springer-Verlag, London, UK, 2002.

[6] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[7] Jason I. Hong and James A.Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the international conference on Mobile systems, applications, and services*, pages 177–189, New York, NY, USA, 2004. ACM.

[8] Marc Langheinrich. Privacy by design— principles of privacy-aware ubiquitous systems. In *Proceedings of the International Conference on Ubiquitous Computing (Ubicomp)*, volume 2201 of *Lecture Notes in Computer Science*, pages 273–291. Springer-Verlag, 2001.

[9] Ninghui Li and Mahesh V. Tripunitara. On safety in discretionary access control. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 96–109, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Ninghui Li, William H. Winsborough, and John C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 123, Washington, DC, USA, 2003. IEEE Computer Society.

[11] Zoltan Miklos. Towards an access control mechanism for wide-area publish/subscribe systems. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 516–524, Washington, DC, USA, 2002. IEEE Computer Society.

[12] Shwetak N. Patel, Matthew S. Reynolds, and Gregory D. Abowd. Detecting human movement by differential air pressure sensing in HVAC system ductwork: An exploration in infrastructure mediated sensing. In *Proceedings of the International Conference on Pervasive Computing*, Berlin, Ireland, May 2008.

[13] Shwetak N. Patel, Thomas Robertson, Julie A. Kientz1, Matthew S. Reynolds1, and Gregory D. Abowd. At the flick of a switch: Detecting and classifying unique electrical events on the residential power line. In *Proceedings of the international conference on Ubiquitous computing*, pages 271–288, New York, NY, USA, 2007. ACM.

[14] Lauri I. W. Pesonen, David M. Eyers, and Jean Bacon. A capability-based access control architecture for multi-domain publish/subscribe systems. In *Proceedings of the International Symposium on Applications on Internet*, pages 222–228, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, page 122, Washington, DC, USA, 1992. IEEE Computer Society.

[16] Mark E. Stickel. Elimination of inference channels by optimal upgrading. In *Proceedings of the IEEE Symposium on Security and Privacy*, page 168, Washington, DC, USA, 1994. IEEE Computer Society.

[17] Tzong-An Su and Gultekin Ozsoyoglu. Controlling FD and MVD inferences in multilevel relational database systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):474–485, 1991.

[18] David Sutherland. A model of information. In *Proceedings of the National Computer Security Conference*, pages 175–183, September 1986.

[19] William H. Winsborough and Ninghui Li. Safety in automated trust negotiation. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 147–160. IEEE Computer Society, May 2004.

[20] Yuanyuan Zhao and Daniel C. Sturman. Dynamic access control in a content-based publish/subscribe system with delivery guarantees. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, page 60, Washington, DC, USA, 2006. IEEE Computer Society.