Know Why Your Access Was Denied: Regulating Feedback for Usable Security*

Apu Kapadia†‡ akapadia@uiuc.edu
Geetanjali Sampemane† geta@cs.uiuc.edu
Roy H. Campbell rhc@uiuc.edu

Department of Computer Science
University of Illinois at Urbana-Champaign

February 2, 2004

Abstract

We examine the problem of providing useful feedback to users who are denied access to resources, while controlling the disclosure of the system security policies. High-quality feedback enhances the usability of a system, especially when permissions may depend on contextual information—time of day, temperature of a room and other factors that change unpredictably. However, providing too much information to the user may breach the confidentiality of the system policies.

To achieve a balance between system usability and privacy of security policies, we present *Know*, a framework that uses Ordered Binary Decision Diagrams (OBDDs) and cost functions to provide feedback to users about access control decisions. *Know* honors a system's privacy requirements, which are represented as a meta-policy, and generates permissible and relevant feedback to users on how to obtain access to a resource. To the best of our knowledge, our work is the first to address the need of access control feedback while honoring the privacy and confidentiality requirements of a system's security policy.

*This research is supported in part by the National Science Foundation NSF CCR 00-86094 and CISE EIA 99-72884 EQ

†These authors contributed equally and their names appear in alphabetical order

‡Apu Kapadia is funded by the U.S. Dept. of Energy's High-Performance Computer Science Fellowship through Los Alamos National Laboratory, Lawrence Livermore National Laboratory, and Sandia National Laboratory.

1 Introduction

Security policies for computing environments become increasingly complex with the number of devices in a typical computing environment. Ubiquitous computing [14] environments typically have users interacting with a plethora of computing, communication or I/O devices in their vicinity. Users interact with devices in such environments in several ways—voice, gestures and traditional keyboard-and-mouse input being some of them. Different sets of users are allowed access to different subsets of resources, and these permissions may change depending on contextual information such as the time of day, the current activity or the set of people involved. In such an environment, it may not be clear to a user who is denied access to certain resources why this happened, especially if the operation was permitted earlier. Thus, informative feedback about why access was denied becomes very important if the system is to avoid annoying users with apparently-arbitrary restrictions. However, unrestricted feedback about who is allowed to do what in the system could itself compromise system security and privacy; therefore, policies need to be protected against inadvertent disclosure.

In this paper, we propose a technique for providing useful feedback about security decisions to users, while still controlling the disclosure of the system security policies. We believe that our system is broadly applicable and can be a useful component of a wide range of security systems. In this paper, we apply our technique to ubiquitous computing environments, in which user permissions are affected by a variety of factors in the environment that change unpredictably. We augment the access

control system with a "feedback component" called *Know* which is invoked when access is denied. *Know* examines the "meta-policy" that protects the policies and determines the level of feedback that can be provided. For example, a student trying to access an audio device in a conference room may be told to return after the ongoing meeting has finished. However, a student trying to illegally modify administrative records need not be given any information that will help probe the security of the system.

The basic challenge in providing informative feedback is to identify the conditions required for the requested operation to be allowed, i.e. find a way to "satisfy" the access control rule guarding that operation. This may involve the user activating a different role (or presenting a different credential) or waiting for the context to change (e.g., time-of-day or current activity configuration of the space). Searching all the rules that guard a particular action will determine all the situations in which this operation is permitted. *Know* can use this information to suggest viable alternatives. A cost function is used to represent the relative difficulty of changing an attribute to satisfy an access condition—it may be easier for a Student to wait for the end of a meeting to be allowed access to a printer, rather than to become a room administrator to print the document immediately.

We use ordered binary decision diagrams (OBDDs) [10] as an efficient and compact representation of policies (or rules), and search for conditions that satisfy the access rules. The OBDDs are computed in advance by *Know*. Thus satisfiability of policies can be tested efficiently and feedback can be constructed by searching for paths leading to a satisfiable condition. We have implemented a prototype of this system and demonstrate how *Know* is able to provide useful feedback for an example access control policy.

This work is an initial attempt to address the problem of providing suitable feedback about security decisions in ubiquitous computing systems. System feedback is more useful if it is tailored to the intended recipient and based on his current permissions rather than a generic "Access Denied" message. We describe a framework to provide such feedback, and present results to show that it is possible to build a system that provides useful feedback about context-dependent security policies, and that using efficient representations such as OBDDs can achieve this with acceptable performance overhead.

The rest of the paper is organized as follows—Section 2 discusses some necessary background and related work. Section 3 describes our system's architecture including the use of OBDDs and cost functions to efficiently generate feedback. In Section 4 we describe our implementation of *Know* and show how it presents useful feedback for a realistic policy. We then discuss some of the issues with using OBDDs as data structures for representing boolean functions before concluding in Section 6.

2 Background and Related Work

While our method of providing feedback is widely applicable to security systems, we focus on its use in ubiquitous computing systems, which is an area of active research [11, 7]. Such environments are currently used in experimental ways while users and researchers devise useful applications for them. Security mechanisms must be a part of the infrastructure at this stage so that application developers have a feel for security in such environments and applications are designed with reasonable security models. However, users tend to disable or work around security systems that are too obstructive. Usability concerns are therefore important for these security systems, so that users are aware of what actions are disallowed and why. Since such environments are particularly dynamic, with lots of mobile users and devices, feedback is particularly important, as user permissions may change with the context of the system in non-obvious ways. Hence we believe that ubiquitous systems are an ideal testbed for *Know*.

Testbed The Gaia [11] system at the University of Illinois takes an operating system approach to ubiquitous computing environments called Active Spaces. A "space OS" interacts with all the devices in the space and provides a uniform programming interface to application developers. The Gaia OS provides infrastructure services such as naming and context that applications can use, as well as security services for authentication and access control. We use Gaia as a test-bed for *Know*.

The Gaia access control system [13] uses an extension of the role-based access control system to assign roles to users within an Active Space. The "space role" assigned to a user decides the permissions available to the user at that time. The user's space role may change depending on contextual in-

formation such as the time of day, the current activity being undertaken in the space or even the set of people currently in the space. Thus permissions available to a user at any point in time depend on a variety of factors, and good feedback about access control decisions is very important.

Policy Protection UniPro [16] provides a scheme to model protection of resources, including policies, in trust negotiation. It allows policies to be treated as resources in the system, and allows the specification of policies to protect them. *Know* can use the policies that protect policies (meta-policies) to decide what information can be provided to a user.

Representation Ordered binary decision diagrams (OBDDs) [10] are a canonical-form representation for boolean formulae where two restrictions are placed on binary decision diagrams: the variables should appear in the same order on every path from the root to a terminal, and there should be no isomorphic subtrees or redundant vertices in the diagram. A binary decision diagram is a rooted, directed acyclic graph with two types of vertices: terminal and nonterminal. Each nonterminal vertex v is labeled by a variable $var(v)$ and has two successors, $low(v)$ and $high(v)$. We call the edge connecting v to $low(v)$ the 0-edge of v (since it is the edge taken if $v = 0$) and the edge connecting v to $high(v)$ the 1-edge of v. Figure 1 gives an example of OBDDs that represents the simple boolean formula $a \vee (b \wedge c)$. To test for satisfiability, we start at the root node and test whether the variable at the root is *true* or *false*. If it is false, we follow the 0-edge, and if it is true, we follow the 1-edge, and repeat this process. Eventually we reach either the T-node or the F-nodes (also called the 1-node and 0-node respectively). If we reach the T-node, then the given assignment satisfies the formula, and if we reach the F-node, it does not satisfy the formula. For example, applying the assignment $\langle a = false, b = true, c = false \rangle$ to either of the OBDDs in Figure 1 tells us that the formula is not satisfied. The figure also illustrates that multiple OBDDs can represent the same formula.

We express access control rules using OBDDs, and *Know* can efficiently search them for paths that would satisfy them. When access is currently denied, the OBDD can provide information about alternate paths that would allow access. *Know* provides information to the user about such paths. With this background, we proceed to describe the *Know* architecture.

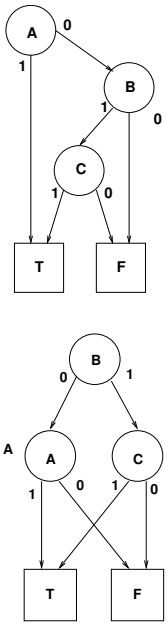

Figure 1: Example OBDDs for $a \vee (b \wedge c)$

3 Architecture

We augment Gaia's access control mechanism with our feedback component (*Know*). Requests for services are intercepted by Gaia's access control component. If the requested access is denied by the system security policy, the request is forwarded to *Know*, which prepares a feedback message for the user. This message will include a list of alternatives that would allow a user to access the given service. Since Gaia is heavily context-driven, feedback may contain a temporal component. For example, if a printer is inaccessible due to the current context (a meeting in the room may disallow the use of noisy printers), feedback may be of the form, "Printer X will be accessible after 5pm today".

When providing a user with feedback, a system must not compromise sensitive components of the system policy. For example, feedback of the form, "Sorry, this room is accessible by Motorola and IBM employees only" might indicate to the user that IBM and Motorola are collaborating on a project, which may be sensitive information to both the companies. To protect such information *Know* augments the policies with meta-policies. A meta-policy governs access to a policy, thereby treating policies as objects themselves. When determining feedback for a user (Alice), *Know* first checks the meta-policy to see what parts of the policy Alice is allowed

to read, and constructs feedback using only those parts. UniPro [16] provides a generalized framework to protect parts of the policy with a policy, which in turn can be protected by another policy, and so on. Here, in the interest of simplicity, we focus on policies and their meta-policies (and not several levels of indirection). Henceforth our notation will resemble that of UniPro, since we use a subset of its functionality. We now provide some concrete examples of access policies, and their associated meta-policies.

Example 1 An electronic door lock's access policy might allow access only to professors in Computer Science or members of the CIA. When a person is denied access to the room, feedback for the form, "Sorry, you must be a professor or a member of the CIA to enter this room" is potentially dangerous. Collaboration between the Computer Science department and the CIA could be sensitive information. Outsiders may also gather intelligence information about where CIA members meet. Clearly, we may not want to reveal parts of the access rule. Feedback of the form, "Sorry, if you are a professor in Computer Science, you may access the room" may be acceptable. A meta-policy would control this flow of information to denied users. Formally, we can represent the policy and meta-policy as shown below:

Policy:

$$\begin{aligned} R &: P \\ P &\leftrightarrow P1 \vee P2 \\ P1 &\leftrightarrow User.role = Professor \wedge \\ &\quad\ User.department = CS \\ P2 &\leftrightarrow User.role = CIA \end{aligned}$$

Meta-Policy:

$$\begin{aligned} P &: true \\ P1 &: User.department = CS \\ P2 &: false \end{aligned}$$

A policy definition includes two types of expressions. An expression of the form $O : P$ means that an object O is protected by policy P, where policies themselves can be objects (since policies may be protected by meta-policies). Expressions of the form $P \leftrightarrow E$, tells us that the policy P is as defined by expression E. The access policy for the room is $R : P$, which means that access to the room R is protected by policy P. P is defined as the disjunction of policies $P1$ and $P2$. $P1$ is the policy, "User must be a professor in Computer Science." $P2$ is the policy, "User must be a member of the CIA." Hence the policy P to access the room is "User must be a professor in Computer Science or the user must be a member of the CIA." The meta-policy $P1 : User.department = CS$ indicates that the policy $P1$ may be revealed only to subjects in the Computer Science department, while the meta-policy $P2 : false$ does not reveal $P2$ under any circumstances. For example, a denied student in Computer Science would receive the feedback "Sorry, if you are a professor in Computer Science, you may access this room," while a student in Civil Engineering will be informed, "Sorry, access is denied." In either case, no policy information involving the CIA is revealed. Figure 2 shows the associated OBDD for this policy. We discuss how to apply meta-policies to OBDDs through cost functions in Section 3.1.

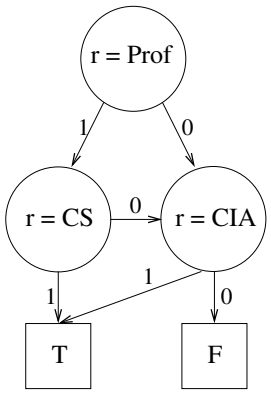

Figure 2: An OBDD for example 1, where $r \equiv User.role$

Example 2 Since we are interested in providing useful feedback in the face of complex policies, we provide a more complex example. We first present the policy P for a printer A, and then augment it with a meta-policy. The policy $A : P$ may be as follows:

Policy:

$$\begin{aligned} A &: P \\ P &\leftrightarrow P1 \vee P2 \\ P1 &\leftrightarrow Context.activity \neq meeting \wedge \\ &\quad\ (P3 \vee P4 \vee P5) \\ P3 &\leftrightarrow User.role = TeachingAssistant \\ P4 &\leftrightarrow Context.labAssistantPresent = true \\ P5 &\leftrightarrow Context.time \geq 9am \wedge \\ &\quad\ Context.time \leq 5pm \\ P2 &\leftrightarrow Context.activity = meeting \wedge P6 \\ P6 &\leftrightarrow User.currentRole = MeetingChair \end{aligned}$$

To understand this policy, first note that printer A

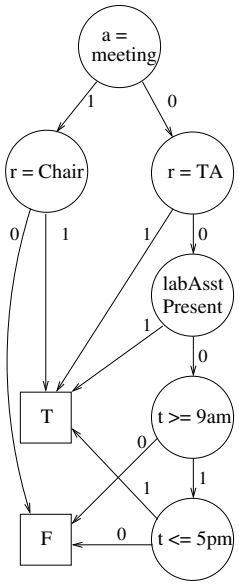

Figure 3: An OBDD for example 2, where $r \equiv User.role$, $a \equiv Contex.Activity$, $t \equiv Context.time$

is noisy, and so, disruptive to meetings being held in the room. During meetings, we would like to restrict printer access to the person in charge of the meeting. $P2$ and $P6$ ensure that nobody will disturb the meeting by using the printer, but the meeting chair may use the printer if needed. When there is no meeting in effect, we would like to grant access to the printer to anybody during normal business hours($P1, P5$). Outside of these hours, in the absence of an ongoing meeting, users are only allowed to access this printer in the presence of a Lab Assistant ($P1, P4$). Teaching Assistants have 24-hour access ($P1, P3$) to the printer to perform their duties (again, as long as there is no ongoing meeting in the room). Figure 3 shows the associated OBDD for this policy.

Consider the case when a Student is denied access to the printer. An "Access Denied" message may be confusing to the Student who was able to access the printer the previous day. In the spirit of offering the user a consistent view of the system's policies, we would like to inform the user why the access was denied. Was it because a meeting was in effect? Should the user come back during regular business hours when there is no meeting? Should the user be informed that there is no lab-assistant in the room and that it is past business hours? Clearly there are several useful options available to the user. Now consider some other options that may not be of much help to the user. Let us say that access was denied at 7pm, and no meeting was taking place in the room. Feedback of the form "Sorry, access is denied, but if you become a Teaching or Lab Assistant you will have access to the printer" is clearly less useful to the user, since becoming a Teaching or Lab Assistant is a non-trivial task.

Consider a final scenario when a person is denied access to the printer because of an ongoing meeting. If the user is not a member of the meeting, feedback of the form "Sorry, access is denied, but the meeting chair is allowed access" may influence users to request the chair to print documents. This is clearly disruptive, so we would like to disclose this fact only to people who are participating in the meeting. Consider the following meta-policy for the policy provided above:

Meta-Policy:

$$P3 \;:\; false$$
$$P6 \;:\; User \in Context.activityMembers$$

Using this meta-policy, we restrict feedback provided to users. Users who are denied access are not informed that Teaching Assistants have 24-hour access ($P3 : false$), since this may result in several students accosting their Teaching Assistants for their personal printing needs. Further, users who are denied access to the printer during an ongoing meeting, are only informed about the meeting chair's printing capability if they are a member of the current meeting ($P6 : User \in Context.activityMembers$).

We have presented examples of feedback that a user may, or may not, find useful. Furthermore, there were some examples of feedback that were restricted or eliminated by the meta-policy due to privacy concerns. Providing the user with useful feedback in the face of restrictions by a meta-policy, and the relative usefulness of options available to the user, suggests the use of a *cost function*. This cost function can evaluate each feedback option, and generate an ordering that says one option is better than another. For example, the system may provide the user with the three most useful options as determined by the cost function. We now describe cost functions in more detail and formalize the notion of "feedback."

3.1 Cost Functions

When the access policy for a resource is not satisfied, we can try to compute all the paths from the root of the OBDD of the policy to the *true* node. Essentially, a set of such paths will be presented as feedback to the user since they represent assignments that satisfy the policy. Some of the edges followed in the path will correspond to conditions that do not currently hold. Consider Example 2. Suppose a meeting is in progress, and a TA tries to access the printer. One possible path from the root to *true* is $Context.activity \neq meeting, r = TA$. This requires one change since there is a meeting in progress. Other options may require more changes. The number of changes that must be made for each option, and the relative difficulty in making certain changes over others, suggests the use of cost functions to rank the options. We first introduce some notation and a formal definition of *feedback*.

We use the notation $S \models P$ to indicate that a policy P is satisfied under the atomic propositions specified by S. For example we could have $S = \{Context.meeting = \text{false}, Context.time = \text{9:05am}\}$. In the notation $S[a] \models P$, $S[a]$ is the set of atomic propositions in S along with any update provided by a. In our example above, $S[Context.meeting = \text{true}] = \{Context.meeting = \text{true}, Context.time = \text{9:05am}\}$. This notation can be naturally extended for a set of updates, e.g., $S[A]$, where A is a set of atomic propositions. Let C be the set of atomic propositions relating to the context of the system and U be the set of atomic propositions specific to the user (identity, role, etc.). Given a policy P and a user U, the user is granted access when $C \cup U \models P$, and denied access when $C \cup U \not\models P$. In essence, if $C \cup U \not\models P$, then a set of updates X such that $C \cup U[X] \models P$ constitutes feedback to the user.

We now formalize the notion of feedback. Let $\Pi = \{\pi_1, \ldots, \pi_n\}$ be the set of paths from the root node to the *true* node in the OBDD of P. Let π'_i be the set of atomic propositions that appear in $\pi_i \in \Pi$ and not in $C \cup U$, i.e., the set of propositions that must be changed (or a set of updates to the state) for the policy to be satisfied. Let $\mathcal{F} = \{\pi'_1, \ldots, \pi'_n\}$. Note that $(C \cup U)[\pi'_i] \models P$ for all $\pi'_i \in \mathcal{F}$. We define any subset F of $\mathcal{F}$ to be the $feedback$ offered to the user. In other words, each *feedback option* f_i in the *feedback* F corresponds to a set of atomic propositions the user must change to be granted access. $\mathcal{F}$ is the set of all possible feedback options available to the user. Since $\mathcal{F}$ can be very large, our primary goal is to find a way to offer the user only a few relevant feedback options in $\mathcal{F}$. We do this through the use of cost functions. The cost function assigns a cost to all $f \in \mathcal{F}$, and returns the k lowest-cost feedback options, where k is a tunable parameter.

A naïve cost function could assign the same cost to all changes, in which case the user would be given feedback with the least number of changes that need to be made to access a resource. For example we could sort the elements f_i of $\mathcal{F}$ in ascending order based on $|f_i|$ (number of atomic propositions in f_i) and return the first k choices. However, changing roles might be more difficult than changing context. For example, a Student may be able to come back at a later time, but it would be extremely difficult to acquire a Professor role. This suggests the use of more sophisticated cost functions.

We need to define an appropriate cost function that is applied to edges in the OBDD as edge weights. Using these weights we can use shortest path algorithms from the root to *true* to provide feedback with lowest total cost. Running Dijkstra's algorithm gives us a path with lowest total cost in polynomial time. There are several proposed algorithms for k shortest paths for graphs. Eppstein [3] presents an algorithm that computes k shortest paths in time $O(m + n\log n + k)$, where n is the number of vertices, and m is the number of edges in the graph. This is the best known bound for k shortest paths in directed acyclic graphs.

Let A be the set of atomic propositions (this corresponds to the condition being tested within a node of the OBDD) for the policy P. We define a cost function $c : A \times \{0, 1\} \rightarrow \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of non-negative real numbers. This function tells us the cost to follow a 0-edge or a 1-edge for a node in the OBDD. When a request for access is denied, let $T \subseteq A$ be the set of propositions that evaluate to true, and $F \subseteq A$ be those that evaluate to false (note that T, F form a partition of A). We define $c(t, 1) = 0, \forall t \in T$ and $c(f, 0) = 0, \forall f \in F$ since there is no cost to follow an edge that is satisfied under the current conditions ($C \cup U$), and we would like to assign non-zero cost when a user must *change* some atomic proposition. Cost functions will differ according to their assignments to $c(t, 0), \forall t \in T$ and $c(f, 1), \forall f \in F$. Now, for all $a \in A$, assign the weight $c(a, 0)$ to the 0-edge of a, and $c(a, 1)$ to the 1-edge of a. What results is a directed acyclic graph

with weights assigned to each edge. We can now apply k-shortest path algorithms to this graph to get the k lowest-cost paths. For small k the running time for such algorithms is dominated by the structure of the OBDD and not k. Specifically, since we expect to have $k < n$ (for example $k = 3$ might be sufficient), then the running time is $O(m + n\log n)$. Our naïve cost function that considers all changes to be equally expensive would set $c(t, 0) = 1, \forall t \in T$ and $c(f, 1) = 1, \forall f \in F$. Hence the total cost of any path is equal to the number of propositions that need to be changed under the given conditions.

3.2 Meta-policies

Now that we have described the basic algorithm for computing feedback using OBDDs and shortest path algorithms, we must modify the algorithm to honor the meta-policies. Each meta-policy determines whether a user can read certain nodes in the policy's OBDD. Let $D \subset A$ be the set of nodes forbidden by the meta-policy. For each $d \in D$, we assign infinite cost to the edge that effects a change in the current value of d. This does two things: firstly, it prevents shortest path algorithms from exploring a change in d and hence does not return any feedback options that require a change in d. Secondly since this proposition d cannot be changed, it will not appear within a feedback option, which includes only those propositions that must be changed. Since no atomic proposition that is precluded by the meta-policy appears in any feedback option, the feedback given to the user honors the meta-policy.

3.3 A Useful Cost Function

We have already discussed the shortcomings of a naïve cost function that assigns equal cost to all changes made to atomic propositions. In the future we would like to experiment with learning-based solutions that determine the various cost parameters for a user based on feedback provided by the user. Over time, the system can learn a user's preferences and provide more useful feedback. For now, we concentrate on a simpler cost function that does not rely on any learning algorithms. In essence, we augment the naïve cost function by forbidding the exploration of certain, obviously undesirable, options. Hence paths to the true nodes will still be graded according to the number of changes, where certain changes are forbidden.

Activities Gaia policies for a smart space depend on the current activity in that space. Example 2 showed the policy for the activities *meeting* and *no meeting*. Consider a space with the possible activities of *meeting, conference, reception*, *presentation* and *no activity*. If a user is denied access to a resource during a *meeting*, feedback of the form "During a *conference* you need to be a Chair" is not very useful. Hence we only provide the user feedback for the current activity, and the absence of any real activities (*no activity*). Hence we apply an infinite cost to all 1-edges for the activities that are not current (the 0-edges will have 0 cost) and apply a cost of 1 to the 0-edge for the current activity. Hence *Know* will not explore feedback for other activities, but will explore feedback for both, the current activity and no activity.

Roles We argued earlier that it may be difficult for a user to obtain a new role. Let N be the set of nodes in the OBDD that tests for a role that the user has not activated. For each node in N, assign infinite cost to the 1-edges (the 0-edges will have 0 cost). Hence the system will not provide any feedback that requires a user to obtain a new role. Due to privacy concerns, a user may choose to activate only certain roles in the system. If the user is not satisfied with the options, the user may activate another role and get better feedback.

Meta-policy As described in Section 3.2, we assign infinite costs to all edges that require a change to variable assignments forbidden by the meta-policy.

Given a feedback option (path from root to the *true*-node in the OBDD), the cost is the number of changes that need to be made. Since certain edges were forbidden by applying infinite cost, feedback provided to the user will not require any changes in the current role, and will only be for the current activity, or no activity. We believe that this cost function is useful in the context of current policies and activities in Gaia. In our analysis, we provide a detailed example policy and show how this cost function provides more useful feedback to the user than the uniform cost function.

4 Implementation

We have built a prototype of the *Know* system. In this section we describe the implementation and re-

sults from a preliminary evaluation. We present results of *Know* running with an example access control policy for videoconferencing equipment located in a kiosk in a multi-purpose business center.

As currently implemented, the system access control policy is converted into an OBDD, which is then transformed into a weighted graph using an appropriate cost function to assign weights to the edges. The system meta-policy also affects these weights. Finding the k shortest paths to the 1-node of the OBDD gives us k sets of assignments to the variables that will satisfy the access control rules, and thus, describe k situations under which the particular operation is allowed. We describe the process in more detail below.

The first step is to generate an OBDD from the system access control policy. Each proposition in an access control rule forms a node of the OBDD. The 1-edge represents the situation when this proposition is satisfied, and the 0-edge represents the proposition being false. We use the BuDDy [8] library which can use some heuristics for optimizing the generated OBDDs. The end result of this is an OBDD that represents all allowable ways to perform a particular action (or access a particular resource). Given a user credential and the current context, a path from the root to either the 1-leaf or the 0-leaf of the OBDD is determined by the current values of all propositions. If the path reaches the 1-leaf, the action is permitted and *Know* is not needed. If the path reaches the 0-leaf, it means that the current credential and context do not permit this action. *Know* now attempts to find alternative paths in the OBDD that would permit the operation.

Alternative paths are found by using the Eppstein [3, 5] algorithm to find the k shortest paths from the root to the 1-node in this OBDD. Weights are assigned to the edges of the OBDD graph based on the cost function and the current values of the user roles and context variables. We provide results from the two cost functions described earlier—the naïve cost function (which just counts the number of changes required) and the sophisticated cost function (which treats role changes as more difficult to achieve than context changes). Selecting a suitable cost function is site-specific—the weights assigned to the different changes will depend on the nature of tasks that are normally performed by users of the system.

Know then outputs the necessary changes that need to be made to allow the alternative paths. It is up to the user to decide which of these suggestions are most useful, and to retry the request after following the suggestion.

4.1 Evaluation

We illustrate this entire process with its application to a sample policy that governs the access to devices in the business center of a hotel. In addition to computers, the business center also contains devices such as printers, fax machines, cameras for videoconferencing and so on. The business center is located in the conference hall, and hotel guests and other members who have signed up are normally allowed to use the devices as per the security policy. The conference hall is also rented out for activities such as meetings, conferences or receptions, during which time use is restricted to participants of this activity, as per the policy configuration by the organizers. Users present their credentials to enter the business center, in the form of a smartcard (a conference badge or a hotel room key) and the system uses this information to restrict access and provide useful feedback. We present here the rules that affect access control to the camera for the videoconferencing system.

The basic policy is as follows:

- When no activity is scheduled for the room, hotel guests or other registered users can use the videoconferencing equipment during the business day. Visitors are also allowed to use the facilities if an *operator* is present. Hotel guests may also use the system during non-business hours, but others may not.
- When an activity (such as a videoconference) is scheduled, only registered activity participants are allowed to use the system.
- Use of the videocamera is disallowed for regular participants if the activity being undertaken in the conference center is labeled as confidential. However, the meeting supervisor may still turn on the videocamera if all participants have the required security clearance.
- Each activity being conducted in the center may define its own policies for its users for its duration.

- Maintenance activities are performed by designated personnel.

- Finally ambient temperature above 30C indicates some problem with the air-conditioning/cooling system, and camera use is prohibited until temperature reaches the allowed range. Similarly, overcrowding the room will violate the fire safety codes and cause access to the camera to be denied.

The meta-policy that governs feedback contains the following rules:

- Certain "blacklisted users" (known trouble-makers) are denied access and should be provided with no further information.

- Information about confidential activities is only provided to the meeting supervisor. Thus an unauthorized user trying to access the video-camera during a confidential activity will not be informed that a confidential activity is going on, but just that access is denied at that time.

- Information about maintenance activities is not provided to other users.

The access control rules for the policy above are presented below. In our implementation, access to the camera C is protected by policy P. Policies $P1, \ldots, P9$ describe the various rules presented above, where $P7$ and $P8$ are rules pertaining to the Video Conferencing activity. In the interest of brevity, we only present the rules relevant to the Video Conferencing activity.

Policy:

$$
\begin{aligned}
C &: P \\
P &\leftrightarrow P1 \vee \ldots \vee P9 \\
&\ldots \\
VC &\leftrightarrow activity = VideoConference \\
CA &\leftrightarrow Context.isConfidential = true \\
RS &\leftrightarrow User.role = Supervisor \\
NU &\leftrightarrow Context.UnclearedUsersPresent \\
&\quad = false \\
NH &\leftrightarrow Context.cameraOverheated \\
&\quad = false \\
NF &\leftrightarrow Context.roomFull = false \\
NB &\leftrightarrow User.blackListed = false \\
P7 &\leftrightarrow VC \wedge CA \wedge RS \wedge NU \wedge NH \\
&\quad \wedge NF \wedge NB \\
RP &\leftrightarrow User.role = Participant \\
P8 &\leftrightarrow VC \wedge \neg CA \wedge (RP \vee RS) \wedge NH \\
&\quad \wedge NF \wedge NB
\end{aligned}
$$

Meta-Policy:

$$
\begin{aligned}
P &: true \\
NB &: false \\
CA &: User.role = supervisor
\end{aligned}
$$

This policy states that during a confidential Video Conference, only a Supervisor can access the camera as long as there are no uncleared users present. During a non-confidential Video Conference, any Participant or Supervisor can access the camera. At all times during a Video Conference the room must not be over-heated, the room should not be full, and the user should not be blacklisted. The second line in the meta-policy states that black-listed users should not be informed that they are blacklisted. In fact no user is ever made aware of the existence of a black-list. The third line in the meta-policy states that only Supervisors will be made aware of confidential activities. Hence if an ordinary user is denied access to a camera, the user will not be told that there is a confidential conference in progress (this information itself is deemed sensitive). Only supervisors can receive feedback about confidential activities.

The OBDD generated by the above policy has 19 variables and 38 nodes (in contrast, a binary decision tree would have at least 2^{19} nodes).

To evaluate *Know*, we try to access the videocamera

under a variety of situations, and present the suggestions provided by *Know* using each of the two cost functions described earlier, which we designate as the "naïve" and the "sophisticated" cost function. Since the sophisticated cost function just restricts information about role and activity change, feedback from the sophisticated cost function will just be a (more useful) subset of the feedback from the naïve cost function. We describe some of the experiments below for $k = 4$. The run-time overhead for *Know* to find these suggestions was negligible—in the order of milliseconds. Since OBDDs are just a representation of the access control policy, they can be constructed ahead of time and only need to be recomputed if the policy changes. Assigning weights to the edges of the OBDD has to be performed each time a request arrives, since the weights depend on the current values of the context variables and user credentials. Since *Know* only runs when access is denied, it has no performance overhead on successful requests.

- A *visitor* tries to use the camera during business hours, but no *operator* is present. There is no activity in session. With the naïve cost function, *Know* suggests that the user come back a) as a *hotel guest* b) as a *registered room user* c) when an *operator* is present, or d) on a workday as a *hotel guest*. The sophisticated cost function suppresses the suggestions involving a role change, and only advises the user to come back when an *operator* is present. This simple example illustrates the basic functionality of *Know*.

- If a *hotel guest* tries to use the equipment when the room is too hot and there is no activity in session, *Know* correctly suggests that the user try again when the temperature is within the allowed range. Specifically, with the naïve cost function, the three options suggested by *Know* are: come back a) when room is not *overHeated* b) when room is not *overHeated* and it is out of business hours and c) when room is not *overHeated* and as a registered room user, or d) when room is not *overHeated*, as a *visitor*, and when an operator is present. The sophisticated cost function only offers the first two suggestions because it does not recommend role changes. Clearly, the only change *required* is for the temperature to be reduced, but *Know* does not presently try to recognize if some of its suggestions are subsets of others. This may be a useful test in some situations.

  Maintenance operations are allowed even in overheated conditions, and a straightforward search through the policy might have offered the suggestion to try coming back as a *maintenance worker*. However, the system meta-policy forbids disclosure of information about maintenance permissions, so this option is correctly ignored by *Know*.

- If a *participant* tries to access the videocamera during a *confidential* activity when users without the required security clearance are present, the naïve cost function also suggests the user come back a) as a supervisor when no uncleared users are present, b) when there is no activity and as a *hotel guest*, c) when there is no activity and as a *registered user*, or d) when there is no activity and as a *visitor*. The clever cost function does not offer any feedback, because there is no useful option for the *participant*.

  One possible suggestion is to inform the user that this operation is not permitted during a *confidential* activity and to suggest re-trying when no confidential activity is being undertaken, but the system meta-policy precludes any information about confidential activities from being revealed, so this suggestion is not offered.

- If a *supervisor* tries to use the camera when the room is reserved for a confidential *videoconference* and uncleared users are present, the uniform cost function suggests that the user come back a) after changing the activity type to be non-confidential b) when no uncleared users are present, c) when there is no activity scheduled, or d) as a *cleaner* during the cleaning activity. The sophisticated cost function suggests the first three options. Note how the *supervisor* is given feedback regarding confidential activities, as opposed to a *participant* in the previous scenario.

While the above examples are fairly simple, they validate our hypothesis that *Know* can provide useful information about alternatives when access is denied, that it can do so without compromising privacy or confidentiality requirements of the security policies, and that this can be achieved with negligible performance overheads. We are in the process of integrating *Know* fully with the Gaia system, after which we can perform larger-scale studies.

An interesting avenue for future work is the study of

suitable cost functions. Local system administrators may select a cost function based on their estimates of site usage patterns. Another option could be to allow users to specify their own cost function to tailor the *Know* feedback. Finally, learning algorithms could be used to improve *Know* feedback over time by allowing users to rate the feedback received.

5 Discussion

Usability has been recognized as an important concern for security systems since the early days [12] of research in computer protection systems; however, in practice, usability issues have not been a primary consideration for security designers. Usability concerns are especially important in ubiquitous computing environments, since the objective of ubiquitous computing is to blend into the background and allow the user to perform his tasks without having to pay attention to the computing environment. Work on the human-computer interface aspects of security [18, 15] have identified consistent feedback as an important aspect of usability. We posit that providing useful feedback about access control decisions is a step in the right direction. Users can then obtain a better picture of the security policies and can access resources accordingly. Suggesting alternative methods of access may potentially be computationally expensive; however, we can bound the time spent by *Know* in such calculations. In the worst case, *Know* provides no additional information, but it does not make things worse. Also, since *Know* only kicks in when access is denied, it does not slow down successful requests.

Since our policies translate into boolean functions, OBDDs are a natural choice for testing satisfiability of the function. In general computing assignments for satisfiability (SAT) of a boolean formula is NP-complete, and hence we must resort to state space exploration through binary decision trees. For n variables, we have a tree with at least 2^n nodes. OBDDs are an efficient and compact representation of this state space, and tests for satisfiability are efficient. It is well known that the size of OBDDs depends on the variable ordering (the order in which variables are tested in an OBDD), and in certain cases is not able to reduce the exponential state space of decision trees. In fact, determining a suitable variable ordering (yielding a minimum sized OBDD) has been shown to be NP-complete [1]. Given these challenges it is comforting to know that commonly encountered functions have reasonably sized OBDDs and there are several heuristics (e.g., group-sifting [9] is one of the popular methods, also see [2, 4]) to determine adequate variable orderings for small (non-exponentially sized) OBDDs. Degenerate cases usually involve functions that behave differently for all possible assignments (e.g., output of an integer multiplier [10], integer division [6], etc.), and we do not expect such state space explosion in practice. In fact OBDDs are widely used in verification of digital systems and computer-aided design [6].

6 Conclusions

We have presented *Know*, a system for providing feedback to users about access control policy decisions. When the system denies a user access to a resource, *Know* suggests alternatives for the user to try. While a list of all possible conditions under which a particular resource may be accessed is likely to be large and not very useful, we try to restrict the options presented to the user to a smaller set of useful options by the use of appropriate cost functions. An important consideration is that this process should not leak any information that would compromise system security or confidentiality. This is achieved by using a meta-policy to represent the required protection for the policies themselves. We present qualitative performance results from a prototype implementation.

We are currently extending our prototype for integration with Gaia, our operating environment for ubiquitous computing environments. This will allow us to evaluate *Know* with more extensive, and real, policies with actual users. Future work in the area of selecting appropriate cost functions is also indicated. Apart from the system-selected cost function that our prototype implementation uses, user-selected cost functions may be useful since different users may assign different levels of difficulty to the same task. Artificial intelligence techniques such as learning or planning algorithms [17] might also be helpful to learn a user's preferences. Another question of interest is the feedback mechanism, since ubiquitous computing environments use a variety of mechanisms for users to interact with them, and text messages on a display monitor may not always be available or appropriate for system

feedback. Audible feedback may reveal one user's feedback to other nearby users, which may not be appropriate.

Lastly, we believe that research in the area of providing useful feedback to denied users has been vastly neglected, and to the best of our knowledge this is the first attempt at integrating policy feedback with policy protection.

7 Acknowledgments

We thank Mahesh Viswanathan for his helpful comments.

References

[1] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Computers*, 45(9):993–1001, Sept. 1996.

[2] K.M. Butler, D.E. Ross, R. Kapur, and M.R. Mercer. Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. In *Proc. 28th ACM/IEEE Design Automation Conf.*, 1991.

[3] David Eppstein. Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. IEEE, November 1994.

[4] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proc. European Design Automation Conf.*, 1991.

[5] Jon Graehl. Implementation of eppstein's k shortest paths algorithm.

[6] Takashi Horiyama and Shuzo Yajima. Exponential lower bounds on the size of OBDDs representing integer divistion. In *Proceedings ISAAC*, pages 163–172, 1997.

[7] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces project: Experiences with ubiquitous computing environments. *IEEE Pervasive Computing magazine*, 1(2), Apr–Jun 2002.

[8] J. Lind-Nielsen. BuDDy – a binary decision diagram package. Technical Report IT-TR: 1999-028, Technical University of Denmark, 1999.

[9] Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In *Proc. International Conference on Computer-Aided Design (ICCAD '95)*, Nov. 1995.

[10] R.E.Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[11] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. GaiaOS: A middleware infrastructure to enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct–Dec 2002.

[12] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, September 1975.

[13] Geetanjali Sampemane, Prasad Naldurg, and Roy H. Campbell. Access control for Active Spaces. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, Las Vegas, NV, December 2002.

[14] Mark Weiser. The computer for the 21st century. *Scientific American*, September 1991.

[15] Ka-Ping Yee. User interaction design for secure systems. In *Proceedings of the 4th International Conference on Information and Communications Security*, pages 278–290. Springer-Verlag, 2002.

[16] Ting Yu and Marianne Winslett. A unified scheme for resource protection in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and privacy*, May 2003.

[17] Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning. *AI Magazine*, 24(2):73–96, 2003.

[18] Mary Ellen Zurko and Richard T. Simon. User-centered security. In *Proceedings of New Security Paradigms Workshop*, 1996.