Towards an Access-Control Framework for Countering Insider Threats

Jason Crampton and Michael Huth

Abstract As insider threats pose very significant security risks to IT systems, we ask what policy-based approaches to access control can do for the detection, mitigation or countering of insider threats and insider attacks. Answering this question is difficult: little public data about insider-threat cases is available; there is not much consensus about what the insider problem actually is; and previous research in access control has by-and-large not dealt with this issue. We explore existing notions of insiderness in order to identify the relevant research issues. We then formulate a set of requirements for next-generation access-control systems, whose realization might form part of an overall strategy to address the insider problem.

1 Introduction

Most people have an intuitive, albeit informal, understanding of the terms "insiders" and "insider attacks". We all have seen spy movies in which agents and double agents exploit inside knowledge or privileged access to inflict damage on a hostile regime. The consequences of insider attacks can be extremely damaging. In January 2008, for example, Jerome Kerviel circumvented internal security mechanisms to place more than $70 billion in secret, unauthorized derivatives trades which, according to his employer Société Générale, resulted in a net loss of $7.2 billion to the bank [19].

It is less clear, though, how to formally define what insiders and insider attacks are, or what effective measures one could or should take to discover, prevent, miti-

Jason Crampton
Information Security Group, Royal Holloway, University of London, Egham, United Kingdom, e-mail: jason.crampton@rhul.ac.uk

Michael Huth
Department of Computing, Imperial College London, London, United Kingdom, e-mail: M.Huth@imperial.ac.uk

gate or counter threats from insiders. For a small family-run business, for example, insider threats may pose a negligible risk. For a national intelligence agency, on the other hand, insider threats may be by far the biggest source of risk and potential damage. For these reasons, which we will discuss in more detail in Section 2, we argue that one should not seek to provide a single universally applicable definition of "insider" or "insiderness".

In this chapter, we mean to explore the insider problem *from the perspective of access control within IT systems*. We realize that this confines the insider problem to a single aspect of an organization's security mechanisms. Moore *et al.* recently produced a "big picture" report into insider sabotage [18], and made a number of interesting observations about the factors that might contribute to an insider attack. We recognize that some of these factors cannot be addressed or phrased in terms of access control: Observation 1 in [18], for example, states that "most insiders had personal predispositions that contributed to their risk of committing IT sabotage." We cannot reasonably expect access-control systems to detect or to be aware of such predispositions. Such issues are considered to be outside the scope of this chapter.

But some factors that contribute to insider attacks [18, Observations 5–7] certainly can be associated with access-control systems as we know them or as we can conceive them to be in the future. We begin by stating and discussing Observation 5:

> *"Observation 5: In many cases organizations failed to detect technical precursors."* where technical precursors are defined as *"an individual action, event, or condition that involves computer or electronic media and that precedes and is associated with malicious insider activity."*

Observation 5 is relevant for access control since many actions, events or conditions within IT systems are mediated through access-control mechanisms. For example, an insider might have had a strange work pattern, may have forgotten to do backup procedures, and may have moved vast amounts of data across unusual folder or account boundaries. The creation of audit trails that pertain to such actions, events or conditions is the standard practice for documenting activity that may be part of an attack or that may increase the risk of a future attack. In technical language, the access-control decision may have the side effect of creating an audit-log entry. But this side effect does not normally interact with the access-control system itself. Audit trails are mostly used for "post mortem" analysis, once an attack has happened. Indubitably, audit trails are valuable for such analyses.

But audit trails represent a genuine opportunity for making access control context-dependent, where the context is stateful and (partially) determined by the data contained in the evolving audit trail. In other words, any access-control system that makes authorization decisions dependent on technical precursors or previously recorded suspicious activity (as well as the usual authorization policies) will help in addressing the insider problem. Nevertheless, using audit data in this way gives rise to new challenges. In particular, the sheer volume of data in audit trails and the

lack of a *lingua franca* for the (efficient) creation, processing and interpretation of audit records represent considerable technical hurdles – in addition to legal hurdles or concerns.

Observation 6 notes that most insider attacks seem to happen when insiders are no longer "on the inside":

> *"Observation 6: Insiders created or used access paths unknown to management to set up their attack and conceal their identity or actions. The majority of insiders attacked after termination."* where an attack path is defined as *"a sequence of one or more access paths that lead to a critical system."*

Observation 6 is relevant to access control in IT systems since insider attacks based on IT systems necessarily need to go through access paths, be they legitimate or not. For example, two co-workers may decide to share their passwords for their user accounts, perhaps to circumvent some inconvenient system restrictions that "get in the way of getting work done." One of these co-workers may subsequently be fired and all his or her known access paths, such as personal user accounts, may be terminated with immediate effect. The disgruntled ex-employee may then launch a remote attack that crucially exploits his or her co-worker's user account in a hidden access path. A very similar scenario forms part of a hypothetical insider attack described by Moore *et al.* [18]. The access path was not known to management, which oversaw the termination of the attacker's known access paths, and the fact that a user shared a password with a co-worker to create this access path violated the organization's security policy. This raises at least two issues for access-control systems.

1. Such systems need to enforce an appropriate level of security whilst offering a sufficient degree of usability in the context of the particular organization's vital work practices. Otherwise people will invariably try to find ways that undermine the control of such systems in order to get work done. Establishing the correct balance between security and usability is increasingly recognized as an important aspect of effective security management [10].
2. The second issue is whether access-control systems could better detect or prevent hidden access paths. To illustrate, the sharing of passwords (an event that would occur completely outside of the scope of an access-control system) might be detectable indirectly.

To illustrate the second point, some office computer may be logged into a user account, doing routine office work whereas at the same time an office computer in another room is running a password-based authentication protocol for that same user account. These two contemporaneous events certainly indicate an abuse of this user account: the user identified with that account cannot be in two places at the same time. An access-control system, perhaps enhanced with an ability to process audit data, could detect this abuse and prevent the log-in attempt at the second computer.

One of the problems with such sophisticated access-control decisions is the possibility of (many) false positives. Even in the context of our example, the user may have to walk over to some other machine in order to launch a particular, secure application that can only be initiated from that machine, and he won't log out of the other machine as he will return to continue his routine office work. So one user uses two computers at the same time, but for legitimate reasons.

Observation 7 concerns the implementation of security within organizations:

> *"Observation 7: Lack of physical and electronic access controls facilitated IT sabotage"* where electronic access control is understood as *"the rules and mechanisms that control electronic access to information systems"* and physical access control is defined as *"the rules and mechanisms that control physical access to premises."*

Observation 7 is hardly surprising. In the context of insider attacks based on the misuse of IT systems it is evident that insider attacks are easier to plan, conduct, and conceal if no rules or mechanisms for restricting access to IT systems are implemented and enforced. More interestingly, perhaps, the above "lack of ... access control" might also be interpreted as "lack of *effective* ... access control". For example, a company may grant access to a sensitive database in its R & D department as follows: all employees have a standard, password-protected user account with user name as well as an identification number on their employee card; and the log-in for the database requires the user name and that id number, not the password for the user account. This gives insiders potentially wide access to all kinds of material in the R & D database: the user names of colleagues are pretty much public knowledge within an organization and it does not require great skills to look at an id card that is being swiped, say, at the organization's lunch cafeteria. Indeed, we had the opportunity to observe a very similar scenario in an actual organization.

Having introduced some of the issues of insider threats in the context of access control within IT systems, we now give an outline of the structure of this chapter.

Outline of chapter.

In Section 2 we motivate this work and discuss relevant related work. In Section 3 we introduce an alternative trust-based perspective on the insider problem. In Section 4 we identify requirements that arise from this new definition and sketch how we can realize these requirements by leveraging recent work in access-control policy languages. In Section 5 we discuss the sort of architecture that will be required to support access-control frameworks that are insider-aware. We conclude by discussing additional related work and our plans for future research in Section 6.

2 Motivation and related work

We have seen that there is evidence to suggest that the inadequacies of existing access-control mechanisms [18], whether automated or not, often play a crucial part in the successful launch of insider attacks. Apart from this empirical data, there are *technical reasons* to believe that a study of the insider problem from the perspective of access control in IT systems may be fruitful:

1. most of today's IT systems implement some form of access control – for example, to identify authorized users of the system or to limit the actions that each authorized user may perform – and understanding an organization's electronic access-control systems should help in discovering insiders, their degree of insiderness, and the level of risk they would represent;
2. an understanding of insiderness and insider threats in terms of access-control privileges and access history may allow IT systems to adapt access control in order to mitigate, counter or even prevent insider attacks;
3. the recent advances in *policy-based* access control may be leveraged to develop access-control frameworks in which policies can evolve dynamically or be retrofitted to deal with insider threats;
4. such policy-based access control systems may then be able to incorporate quantitative methods for the early detection of insider threats (such as host-based anomaly detection) in order to prevent, or limit the damage of, insider attacks.

It is also our hope that the understanding of the insider problem gained from studying the well-defined and structured context of policy-based access control may be transferrable to policies that are merely descriptive and not enforced within IT systems. Jerome Kerviel, for example, managed to launch his attack because, over time, he was able to take on two roles that should not have been held by any single individual (even at different points in time). But this role-based separation of duty [26] was not implemented in any part of the IT system. Support for role-based access control (RBAC) within that IT system, in combination with proper identity management, may have prevented or at least detected this insider attack.

We explore the potential of these technical ideas in later sections. Now we introduce three examples that further illustrate the diversity of the insider problem. We then discuss some existing work on defining insiders and the insider problem, concluding the section with a discussion of previous work on access control and the insider problem.

2.1 Illustrative scenarios

Inevitably, any organization of even moderate size and complexity must trust (some of) its employees with sensitive information and resources. For this reason, most organizations have security policies that specify who is authorized to access what resources. To ensure that these policies are enforced, all modern operating systems

and many applications provide authentication and authorization services. In using these services, an organization allows a number of individuals access to privileged knowledge of mission-critical information, and special access rights to mission-critical resources or both. To us, the *insider problem* refers to the inherent threats that organizations face when granting individuals such special privileges. Here are some examples of what many people would consider to be insider problems, the first one corresponding loosely to a case reported in [18]:

1. a system administrator may be given the right to manipulate folders that contain a substantial portion of her organization's intellectual property;
2. urgent building work at a company's headquarters may require contractors, employed by a third party, to be given permission to enter security-sensitive parts of that building; or
3. the personal assistant of a chief financial officer (CFO) would have access to the CFO's diary and contents or patterns of communication.

Part of the problem is that these individuals enjoy some particular trust relationship with the organization:

1. the system administrator *could* encrypt all files with a secret key in order to extort money from her employer, but *she is trusted not to do so*;
2. the building workers *could* be IT security specialists from competitors who want to infiltrate the company's premises and intranet, but *are assumed not to be*;
3. the personal assistant of a senior officer *could* divulge the existence and status of confidential merger talks, but *is expected not to*;

Organizations have to trust individuals, otherwise we cannot distinguish between authorized and unauthorized users. However, it may not be possible to articulate optimal trust relationships from a perspective of security for a number of reasons: lack of software support (e.g. for the enforcement of role-based separation of duty in the attack by Jerome Kerviel), excessive cost, staff constraints, etc. Practical and economic considerations are often reason for compromising on security. Probst & Hunker [25], for example, note that many organizations take basic steps toward preventing insider attacks but rarely engage in addressing serious insider attacks; moreover, they argue that this appears to be rational economic behavior. In the context of the above scenarios:

1. A small start-up company may simply not be able to afford several system administrators, thus leaving the entire management of stored intellectual property (including backup procedures) to that sole administrator.
2. It may be impractical and expensive to vet every contractor that enters the building. It is much more cost-efficient to rely on (trust) the third party contractor to employ reliable staff.
3. A CFO naturally insists on a convenient single point of contact for arranging meetings, taking minutes, etc. But such convenience increases the risk and threat level of an insider attack by the personal assistant.

The above examples already indicate the difficulty of obtaining a working definition of "insider" that will fit all instances of the "insider problem". This has already been emphasized by Bishop in [3]. To illustrate this difficulty, it is too simplistic to say that every and only employees of an organization are insiders for that organization, considering that the above builders might be employees of an "outside" contractor but do have physical access to the "inside", the company's headquarters.

Even with a working definition of "insider" in hand, it is not clear how to put that definition into use so that it may aid with the detection, mitigation or countering of insider threats. Moreover, given the limited budget that is available for security solutions, there is economic pressure for research in this area to deliver solutions that can be deployed and managed in a manner that is perceived to be economically viable.

2.2 Definitions of insiders

Matt Bishop organized a panel at NSPW'05 on the insider problem and noted the diversity of existing definitions of insiders [6], which we cite here: Brackney and Anderson define an insider to be *"someone with access, privilege, or knowledge of information systems and services"* and a definition of the insider problem as *"malevolent (or possibly inadvertent) actions by an already trusted person with access to sensitive information and information systems"* [4]; in contrast, Patzakis defines an insider (implicitly) as *"anyone operating inside the security perimeter"* [20].

The summary of the recent Dagstuhl Seminar on countering insider threats [2] proposes several definitions of an insider, e.g.,

1. as someone "defined with respect to a resource, leading to *degrees of insiderness*[1]"
2. as "somebody with legitimate", past or present, "access to resources"
3. as a "wholly or partially trusted subject"
4. as "a system user who can misuse privileges"
5. as someone with "authorized access who might attempt unauthorized removal or sabotage of critical assets or who could aid outsiders in doing so".

It should be clear that the protagonists in the scenarios described earlier fit some of these definitions better than others. Any attempt at a comprehensive definition of insiders will have shortcomings. For example, if an "outsider" manages to break into an IT system by masquerading as a legitimate inside user, does this constitute an insider attack? In Section 3 we offer our own definition, which is based on and recalls definitions of trust and trustworthiness from the 1970s.

[1] Our italics.

2.3 Access control

For the sake of completeness, we now provide a brief description of what we understand by access control. In order to implement enterprise security policies, every attempted interaction between an authorized user and a protected resource (which could be as specific as a particular entry in a database table or as general as a computer system) is "trapped" by the access-control system. This interaction is modeled as an *access request*. The access-control system determines whether the access request is *authorized* (according to some policy and relevant security-configuration data) and then either allows the interaction to proceed (if the request is authorized) or prevents the interaction otherwise. Hence, an access-control system typically comprises:

1. a *policy-enforcement point* (PEP), which traps attempted interactions, generates access requests and enforces authorization decisions;
2. a *policy-decision point* (PDP), which accepts access requests and returns authorization decisions; and
3. a *policy repository* (PR), which stores authorization policies.

The typical architecture of an access control system is illustrated in Figure 1. The user and resource information is typically provided by trusted entities, perhaps the simplest example being (trusted) operating-system components such as an authentication service and a resource manager.

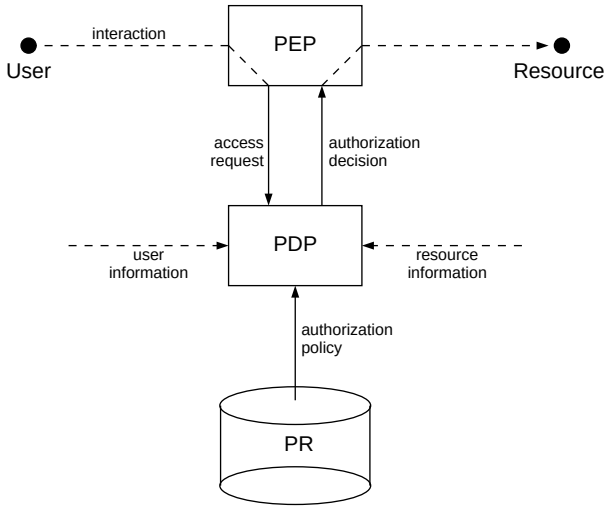

Fig. 1 A generic high-level access control architecture

In order to specify an access-control model, therefore, we need to define a language for articulating authorization policies and an algorithm that is used to evaluate access requests with respect to such policies.

2.4 The insider problem and access control

It can be seen that many problems that arise when discussing the insider problem are related to abuse of privileges granted by authentication or authorization services, both of which may be regarded as providing some form of access control. So how can an understanding of access control enable us to better understand and address the insider problem? There is very little existing work in this area, although Park and Giordano suggest the following issues need to be addressed in order to counter the insider threat in large-scale, dynamic IT systems [23]:

1. The management of privileges should not be based on identities, as this won't scale to distributed, highly dynamic and heterogeneous systems. The authors propose RBAC as a potential solution.
2. Access control needs to also support finer levels of granularity. For example, most systems grant access to an entire structured file and not, say, to specific fields in that file.
3. Most approaches to access control decide an access request by a static and predefined set of rules. Rather there is a need for *context-aware* access control, which makes decisions dependent on context.

We believe that these requirements are rather generic and do little to address the insider problem. Indeed, it is hard to believe that RBAC, which is by design unconcerned with individual users, is an appropriate underlying model for an access-control system that provides some protection against insider threats. Moreover, there is little evidence to suggest either that existing access-control systems are insufficiently fine-grained (see such systems for relational database management systems, for example) or that having more fine-grained access-control systems would help counter the insider threat in general. While we would agree that access-control systems do need to be context-aware, we believe that any such system would still need to be governed by some pre-defined (policy) rules.

We also note the existence of some preliminary work on combining risk and access control [8] and on using trust to assign users to roles [7, 16]. However, these efforts have a rather narrow focus, respectively on Multi-Level Security and Role-Based Access Control.

In the next two sections, we formulate an alternative definition for the insider problem that is more directly related to access control. We then discuss how recent advances in trust and reputation systems, and access-control policy languages can be used to address this revised insider problem.

3 Trust, trustworthiness, and the insider problem

Let us revisit the notion of "trust". It is generally accepted that trust is an assumption about the way in which an entity will behave (in a particular situation). In the context of the design of secure systems, we must assume that certain elements of our "system", such as the hardware, will behave in a particular way. These elements are called collectively the *trusted computing base* (TCB) [11].[2] It is imperative that the TCB operates in the way in which it is assumed or intended to operate (otherwise our trust is misplaced). One of the primary goals of the designers of secure operating systems is that the code that must be included in the TCB (because it is authorized to execute privileged instructions) can be formally verified [11]. Of course, it is rare for this goal to be realized in full because of the difficulty of formally verifying programs in general.

In the context of access control, there is an assumption that authorized users will behave in a certain way: that is, authorized users are assumed – that is to say, *trusted* – not to abuse the privileges for which they are authorized. The simplest example is that it is assumed (must be assumed, in fact) that a user will not divulge his username and password to another user. Clearly, however, some authorized users are not *trustworthy*. This suggests a more abstract and formal account of the insider problem:

TW: The insider problem arises because the set of users *trusted* in some particular context is not equal to the set of *trustworthy* users for the same context.

This definition echoes the discussion of trust relationships and trust assumptions in the context of software security, made by Viega & McGraw in [27, Chapter 12]. Moreover, our scenarios and the proposed definitions of insiders given in Subsection 2.2 share a concern for resources and the interactions between users and those resources, where these interactions could modify, create or destroy resources.

Recognizing that insiders are somehow trusted with managing and transforming resources responsibly, it may be fruitful to focus on access requests and the potential threats that granting such requests may pose to the tactical or strategic aims implicit in the underlying IT system of an organization. For example, instead of asking whether a masquerading outsider is an insider or not, we may want to focus on the risk and probability that a specific inside user account has been compromised by that outsider, and let our access-control systems adapt appropriately should such a compromise be detected.

[2] Other electronic security "systems" also assume that certain entities are trusted. "Trusted third parties" are widely used in cryptographic protocols and public-key infrastructures, for example.

3.1 Insiderness

This emphasis on resources and on access rights to resources has led to the notion of *degrees of insiderness* [1], where the more privileges and skills a person has, the more risk he or she represents. However, we would argue that it is the abuse of an authorization that poses a risk and that this risk is some function of the authorized action, the trustworthiness of the user and relevant contextual information (which might include, for example, previously authorized access requests). There is no need for different levels of insiderness, just an ability to account for different levels of trustworthiness.

The degree of insiderness of a user has also been regarded as the threat level posed by the user [1]. Again, we do not regard this notion of insiderness as particularly helpful: it seems intuitively reasonable to regard a user that is known to represent a significant threat (for whatever reason) to be necessarily less trustworthy.

3.2 Trust management and risk assessment

We now briefly discuss connections between our formulation of the insider problem given in (**TW**), trust management systems, and risk-assessment frameworks. In particular, there is a natural correspondence between trustworthiness and the probability of misuse of authorizations. This suggests that trustworthiness and existing risk-assessment methodologies could be combined in a risk-aware access-control system. We will elaborate and illustrate this point in the next section. Moreover, trust-management and reputation-management systems could be regarded as tools to compute trustworthiness (not *trust*).

This connection is illustrated by similarities with problems and known attacks in reputation systems [12]. In a *whitewashing attack* against a reputation system, for example, the attacker exploits system vulnerabilities to repair the damage to his reputation that resulted from previous misbehavior. Similarly, a malicious insider may be able to manipulate audit logs in order to hide evidence of breaches of trust and to thus maintain his trustworthiness.

Another important aspect of risk and trust management in access control is that any relevant information about the possible detection of an insider attack needs to be reported to the access-control system so that it can adapt its behavior in an appropriate, context-aware fashion. For example, a person who received a formal reprimand for having shared the password for their user account with a co-worker would typically still have the same access-control rights and privileges as before. The damaged reputation is only reflected in their (paper) personnel records, not in the IT system through which they access sensitive electronic resources – and this mismatch may increase the risk of future insider attacks.

3.3 Pragmatics of identifying suspicious events

The key problem then is to identify (and then report) suspicious events. In the realm of social interaction, humans are quite good at identifying suspicious activities. Humans can differentiate well between anomalies that have context-aware explanations, and anomalies that can probably not be "explained away".

Machines, on the other hand, are good at recording events but less good at interpreting their significance – especially in dynamic contexts. This has long been recognized in the Artificial Intelligence community, e.g., in the so-called *Frame Problem*. The movement or destruction of large amounts of data, e.g., may just represent a yearly "housekeeping" exercise or it may indeed represent the prelude to a major attack. How would an automated policy or monitoring program know?

The challenge here is to develop automated systems that detect, record and collate all events that could indicate suspicious activity from authorized users. However, such systems must not generate too many false alarms, as such alarms will either lead to denial of access and inconvenience to well-behaved users, or to human intervention and analysis, which is costly and is not viable in large-scale systems.

The problem of having high detection rates (equivalently, few false negatives) and few false positives, is very familiar in the context of software verification. The plane manufacturer Airbus, for example, uses program-analysis techniques to detect possible memory leaks in on-board software. Since essentially all program-analysis problems are formally undecidable (meaning that no algorithm can compute accurate analysis information for all analyzed programs), only approximate solutions can be computed. Most analyses over-approximate, meaning that they catch all actual errors (zero false negatives) but also may report spurious ones. But if such an over-approximating analysis reports 900 potential memory leaks for a $100,000$ line computer program, the engineers may just abandon the use of this technique and resort to manual code inspection of what they perceive to be the critical parts of the program.

This suggests that the use of over-approximation with a low false-positive rate is attractive for the detection of insider threats. We want to catch all insider attacks, but we only want very few false alarms. The choice of the particular program analysis is informed by the program's "threat model". For example, checking that a program terminates is vital if its termination ensures the release of locks in the kernel of an operating system [9]. But for an application program, non-termination may be merely an inconvenience, whereas buffer overflows (detectable with another form of program analysis) may then compromise the entire machine on which the application runs. In short, the choice of analysis is dependent on context.

For any system that detects events that may signify an insider attack, we would expect a similar dependence on context and intent when formalizing what actually constitutes an insider attack. Risk and cost considerations, combined with the trustworthiness of users should shape these definitions. A detection system may only be concerned with worst-case threats, for example, and may not concern itself with petty insider abuse of systems. On the other hand, petty but persistent abuse may not merely be about misusing office equipment for private needs. It might indicate the

build-up of a slow but powerful insider attack. This suggests the need for the coordination of detection systems that each have their own intent and purpose and that need the ability to share their information in semantically meaningful ways across different intent boundaries.

4 Toward a context- and insider-aware policy language

We observed in the previous section that the insider problem arises because of a discrepancy between trust and trustworthiness. In the context of access control, we note that there are two points at which trust is of particular relevance.

First, a human user must authenticate to a computer, thereby generating some binding of the human user to security-related information, such as a user ID, security group identifiers, role identifiers, security labels, etc. We say that (successful) authentication gives rise to a *security context* that is associated with all programs executed by the authenticated user, and it is this context that will be used to evaluate whether subsequent user actions are authorized or not.

One method of authentication may be deemed to be more "secure" or "trustworthy" than other methods, so we can have greater confidence in the trustworthiness of the processes that are spawned as a result of a successful authentication event that employs this particular method. One natural approach would be to make the security context a function of both the user identity and the method of authentication, so that a "stronger" (whatever that might mean) authentication method means (ultimately) that the authenticated user will be authorized to perform a greater variety of actions. Of course, we must now consider what makes one method of authentication more secure than another. This may be determined by the way in which the user authenticates (password, biometric, knowledge of some secret key, etc.), or it may be determined by the user's location (local, remote, etc.), or indeed by many other possible factors.

The second point at which trust becomes relevant is when this security context is used to determine the actions for which the associated user is authorized. In many situations, the security context comprises identifiers, such as roles, that are directly associated with authorizations. (For our purposes, a *principal* is an entity or identifier that is directly authorized for certain actions by an authorization policy.) But the security context need not necessarily consist of principals: in Unix, for example, the security context (comprising security identifiers for user and group accounts) is mapped at request evaluation time to exactly one principal from the set $\{\texttt{owner}, \texttt{group}, \texttt{world}\}$.

With regard to this aspect of trust, we need to examine the justification for binding a user (or some part of a user's security context) to a principal, since it is this binding that ultimately decides the actions for which the user is authorized. This binding may be delayed, as in Unix, until request time. Unlike Unix, this binding may also be dependent on conditions and information not traditionally considered when evaluating access requests. (We may not wish to bind a user to certain princi-

pals if the location of the user is anomalous, for example.) A policy-based approach may well be useful here, and would be rather similar to existing *context-aware* access control systems.

We will concentrate on this approach in the remainder of this paper. In this section, we consider how we might develop a general-purpose language for specifying access-control policies that take factors such as risk and the trustworthiness of authorized users into account.

As we have said, we believe that the central problem is that trusted users may not be trustworthy. (That is, we need to be able to deal with a user for whom an erroneous, unjustified or obsolete assumption has been made about his reliability.) The question, then, is how to define access-control policies in such a way that the privileges for which users are authorized can be modified automatically (ie, without human intervention) when those users are revealed, or suspected, to be untrustworthy.[3] We therefore list additional requirements for addressing the insider problem within policy-based, access-control systems.

4.1 Context and request predicates

To understand these requirements, it is useful to think of access requests as tuples of the form (s,o,a,c) where s ranges over subjects (i.e. user proxies), o ranges over objects (i.e. protected resources), a ranges over actions (such as *read*, *write*, etc.), and c ranges over contextual information (such as whether or not the requestor is accessing the system remotely). Sets of such tuples are thus subsets X of a space $S \times O \times A \times C$ of access requests. Such subsets can therefore be interpreted as predicates, which we refer to as *request predicates*. Such predicates may depend only on context, so they could be seen as describing subsets of C. We then call such predicates *context predicates*.

Given $X \subseteq S \times O \times A \times C$, we write p_X^+ to denote the policy that authorizes all requests in X and denies all others. [4] Dually, policy p_X^- authorizes a request if and only if it is not in X.

4.2 Requirements

Then our requirements for policy-based access control that can address insider threats can be described as follows:

[3] Of course, a preliminary problem is how to reliably identify and report to the access-control system those users whose trustworthiness is questionable. We do not consider this problem here.

[4] Strictly speaking, it is the policy-decision point that decides whether or not a request is authorized by a policy or not. However, we don't make this distinction explicit henceforth.

1. The ability to support richer types of policy decisions beyond the usual binary "deny" and "grant", so that inconsistencies, definitional gaps, error conditions, etc., can be articulated, accommodated, and propagated.
2. The ability to declare context predicates and request predicates, where the latter express sets of access requests of interest and the former capture contextual state; and the ability to transform such declarations into access-control policies, for example by turning predicates X into policies p_X^+ or p_X^-.
3. The ability to transform an access-control policy p into a set of access requests X that captures some aspect of the behavior of policy p. For example, we might want to identify all requests authorized by policy p in a specific context. However, this is by no means the only possibility. We may, for example, wish to compute those requests for which the policy p is over-defined (conflicts) or under-defined (gaps).
4. Support for typed, modular programming of access-control policies, with a rich set of declarative coordination patterns. This would facilitate the creation of robust, composed authorization policies and would leverage the abilities of the previous two requirements.

4.3 Policy transformations via declarative programming

Given these requirements, we propose that there is value in transferring the principles of declarative programming into the domain of context- and trustworthiness-aware access control. Declarative programming (by which we loosely mean extensions of functional programming, logic programming; or modeling languages based on logic such as Alloy [14]) focuses on the "what" and not on the "how" of computation. So declarative access-control policies are concerned with what requests will be granted under which contextual circumstances, but they don't have to specify how such policies are then enforced.[5]

This separation of concerns between policy specification and implementation brings additional benefits. One can design declarative languages that have typed modules and so support robust policy composition that facilitates reuse and offers a policy authoring tool that is hopefully descriptive and intuitive enough for policy writers.[6] Indeed, we believe that it is this support of modular, declarative programming that will be able to accommodate context- and trustworthiness-aware access control that has a flexible range of granularity and is not necessarily identity-based.

[5] We assume that declarations will have intuitive operational or denotational semantics and will therefore be implementable.

[6] It is of interest to note that the financial trading community has now recognized the need to be able to program as close to the user domain (e.g. financial traders) as possible. Declarative programming, mostly in the form of functional programming, is becoming an important tool in that sector, even for back-office aspects such as specifying and analyzing contracts [15].

4.4 Discussion of requirements

We now discuss each of these requirements in more detail and illustrate what their realizations would and will provide. The first requirement allows us to treat a wide variety of functions and inputs as "policies". For example, a policy might be composed of sub-policies that return quantitative decisions, e.g. about trustworthiness or risk, and these non-binary decisions could then be converted into appropriate binary policy decisions.

The second requirement suggests the use of abstract request predicates and propositional-logic connectives so that one can form expressions such as

$$\texttt{Manager} \wedge \texttt{OnDuty} \wedge \neg \texttt{Weekend} \tag{1}$$

The access request is left implicit. Indeed, an abstract request predicate may not even depend on the access request. For example, if a subject is requesting to edit a certain PDF document, the evaluation of the above expression will not depend on the type of action or type of document, but will depend on whether said subject presently has role `Manager`, is currently on duty, and whether the request occurs not during a week-end. So `Weekend` would better be seen as a context predicate.

The expression in (1) is declarative since it does not elaborate on how the connections between, say, `Manager` and subjects are being implemented. In particular, it does not necessarily commit to an underlying RBAC model nor explicitly depend on the handling of secure attributes, etc. The provision of abstract interfaces for specifying access-control policies was suggested and developed in the work of Bruns & Huth in [6, 5] already.

One can now promote expressions such as the one in (1) to genuine policies, e.g.

$$\texttt{Grant if Manager} \wedge \texttt{OnDuty} \wedge \neg \texttt{Weekend} \tag{2}$$

which is the declarative way of encoding policies of form p_X^+ discussed above. We adopt here the interpretation of [5, 6], where the policy in (2) grants all requests that satisfy the composed predicate in (1), and where said policy has a gap (meaning it is undefined) at all other requests.

The third requirement allows us to identify sets of access requests that stem from policies themselves. For sake of illustration, consider two primitives

$$p\texttt{@Denies} \qquad\qquad p\texttt{@Grants}$$

that, given an access-control policy p, declare request predicates that capture that set of access requests that policy p denies, respectively grants. It should be noted that policy p may well not be equal to policy `Grant if` p`@Grants`. The policy `Grant if` p`@Grants` only has two possible decisions: grants and gaps. But policy p may have a number of possible decisions, such as conflicts or errors.

Meeting the fourth requirement means that we can wrap policy composition patterns into parameterized modules, and then instantiate or reuse such modules in policy composition or orchestration. For example,

```
pol insiderThreat(abnormalBeh : reqs, insider: reqs) {
  (grant if !abnormalBeh & insider) >
  (deny  if abnormalBeh | !insider)
}
```

declares such a module with two types, `pol` for policies and `reqs` for request predicates. This module takes as input two request predicates and outputs a policy, where > declares a priority composition. The output policy therefore grants a request if it comes from an insider and if no abnormal behavior has been flagged; and it denies a request if either abnormal behavior has been flagged or the request does not come from an insider. The module itself does not state how insiders and abnormal behavior are defined. But this separation of concern creates flexibility, since different circumstances may require different notions of insiders or abnormality.

For example, `abnormalBeh` could indicate whether a request wants to copy unusually large amounts of data from one medium or location to another. Or `abnormalBeh` could encapsulate a risk-based or statistical analysis of the requestor's behavior, and so reduce such quantitative results to a binary authorization "decision".

In another context, `insider` may declare those sets of requests in which either a manager assigns pay increments to non-managers or in which non-managers rate the performance of their managers, as in

$$\begin{aligned}\texttt{insider} = {} & (\texttt{manager} \wedge \texttt{assign} \wedge \texttt{payIncrease}) \vee \\ & (\neg\texttt{manager} \wedge \texttt{rate} \wedge \texttt{managerPerformance})\end{aligned} \tag{3}$$

We point out that requestors that satisfy the predicate `insider` defined above are not defined in mere terms of roles, but in terms of the combination of role, object, and action. In particular, managers are considered as "outsiders" (i.e. not authorized or trusted) when it comes to rating the performance of managers. Of course, such combinations could be enriched with additional context predicates.

4.5 Policy transformations

Request predicates also leverage policy analysis. Since p@`Grants` and `insider` are both request predicates, we can examine whether the implication

$$p@\texttt{Grants} \rightarrow \texttt{insider} \tag{4}$$

is valid. If so, then policy p will not grant requests unless they stem from those who are declared to be insiders. This would show that policy p cannot be persuaded to grant access to outsiders, assuming that outsiders are implicitly defined by $\neg$`insider`.

This approach does not impose the exclusive use of static-analysis methods though. A dynamic version of (4) would test its validity at run-time, and adapt policy behavior accordingly, if needed. We can express this in a parameterized module:

```
pol grantOnlyInsiders(P : pol, insider : reqs) {
  (grant if P@grants & insider) > deny
}
```

The idea here is that the module takes a policy `P` and a request predicate `insider` as input, and retrofits policy `P` such that the retrofitted policy will grant only if the request is in the "set" `insider` of interest and the policy `P` would grant that request. The retrofitted policy denies all other requests, including those that don't satisfy `insider` but would be granted by `P`. This ensures that (4) holds in all executions of the access-control system, but now for the invocation

```
grantOnlyInsiders(P,insider)
```

instead of for `P`.

Finally, we give a flavor of how policy languages that meet our requirements could realize the approach we had suggested in Section 3, which uses trustworthiness to inform a risk-evaluation strategy for access control.

4.6 Risk- and trustworthiness-aware policy composition

Our first requirement stated that policy decisions need not be binary, and could even be continuous. We also argued that trustworthiness is an appropriate and useful notion for dealing with the insider problem. We now illustrate how these things can be brought together within our policy language: The parameterized module

```
bool tooRisky(R: req; riskThreshold : double) {
  return (cost(R) / trustworthiness(R.subject)
         > riskThreshold)
}
```

returns a Boolean and takes as input single request `R` and a threshold `riskThreshold` for the risk the access-control system is willing to accept when granting this request. The risk is perceived as being too high if the estimated cost (that will be incurred if the request is authorized and subsequently abused) divided by the requestor's trustworthiness (the probability of the requestor not abusing the authorization) is strictly above a specified risk threshold.

We illustrate how this function could be used to make a policy risk-aware. For example

```
pol riskFilter(P : pol) {
  (deny if tooRisky(this)) > P
}
```

is a module that takes policy `P` as input and then produces a policy that denies if the presented request `this` is judged to be too risky; otherwise, it will behave as `P`.

We will now briefly consider the architectural ramifications of supporting the evaluation policies written in our context- and trustworthiness-aware policy language.

5 Access-control architectures and the insider problem

In considering an architecture that accommodates the awareness of risk and trustworthiness, there are good reasons for reusing existing access-control architectures such as the standard one depicted in Figure 1. For one, it may simply be infeasible to rebuild an entire access-control system and so one might be committed to its underlying architecture. For another, adapting an existing and already implemented architecture may be much more cost-effective. Moreover, such reuse readily accommodates legacy systems that don't yet support or indeed don't require support for such notions of awareness.

Assuming we adopt the core architecture shown in Figure 1, we need to add hardware and software components that can provide the measures of trustworthiness, the observations of context, and the assessment of risk. In particular, we will need the following:

1. trust-management software for evaluating user trustworthiness
2. event-monitoring systems (both hardware and software), including actuators and sensory networks (if applicable)
3. risk-evaluation systems (that may combine stochastic with worst-case uncertainty)
4. auditing systems and audit-processing software

Existing research in each of these areas will certainly be of use here, as well as recent work on usage control in access control [21] and on intrusion-prevention systems [17].

The generic architecture modified in this manner is depicted in Fig. 2. What is not shown in detail, though, are the necessary programming interfaces for determining the values of context and request predicates that occur in the access-control policy. But our language viewed such predicates already as abstract interfaces, i.e. as methods with an implicit parameter (a contextual request that we exposed with construct

`this` in the body of method `riskFilter` above). Request predicate `manager`, e.g., could be seen as a method that returns a set of requests:

```
reqs manager() { implementation code }
```

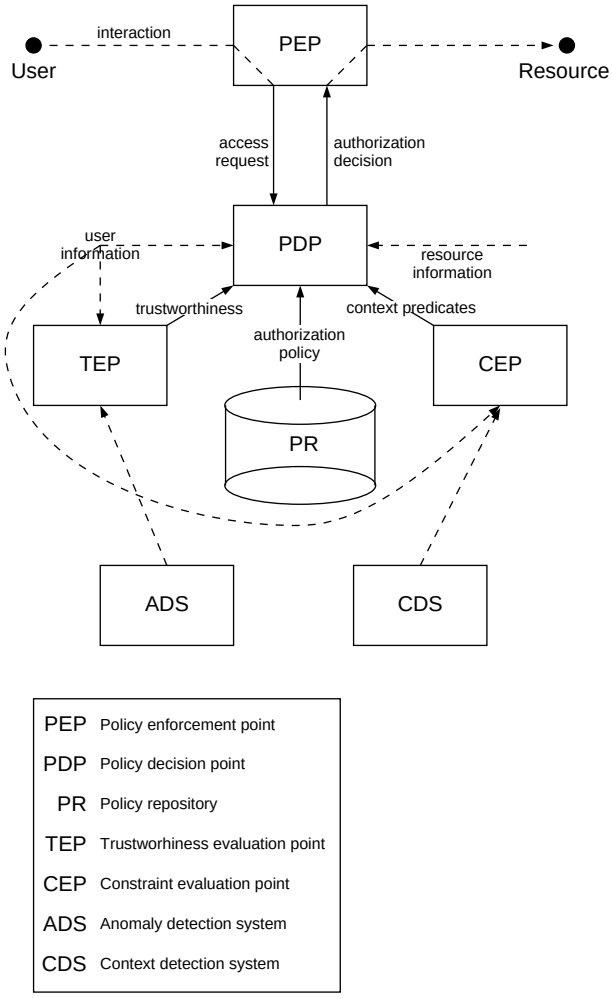

Fig. 2 An extended access-control architecture that can provide support for risk- and trustworthiness-aware access control

Similarly, as we have already demonstrated, quantitative predicates based on risk and trustworthiness are written as parameterized methods. So at the policy-decision point the policy can successfully evaluate all these methods provided that they have been implemented in the policy repository.

6 Concluding remarks

The insider problem is extremely difficult to define, much less to solve. In this chapter we have identified the contribution that access control could make to the insider problem in general, and proposed a policy-based framework for such access control. Our key observation was that problems of insiderness could be rephrased in terms of problems of trust and trustworthiness.

Based on that observation, we then demonstrated how principles from declarative programming could be combined with established policy-composition principles to obtain a policy language in which the management and coordination of risk, trustworthiness, and conventional request processing could be accommodated. We illustrated how such coordinations could support the prevention or mitigation of insider attacks – by dynamically adjusting levels of trustworthiness on the bases of information received from the context handlers of the access-control system. In future work we mean to examine existing case studies of insider attacks to determine whether our approach could have detected, prevented or otherwise mitigated such attacks if policies had been made risk-aware.

We conclude by discussing related work that is relevant for extending the approach that we have described in this chapter.

Bruns and Huth have shown that primitives such as p@`Grants` and p@`Denies` can be defined as request predicates if policies are composed out of request predicates with operators similar to those discussed in this chapter [5]. In their work, request predicates were atomic, without structure. But it is easy to extend policy composition and the above primitives to request predicates that support composition operators from propositional logic, as demonstrated in the talk [13] at the Dagstuhl Seminar [2].

The ideas put forward in this chapter are perhaps closest to the work by Probst *et al.* in [24]. The authors point out that the predominant approach to addressing the insider problem is to maintain audit logs for "post mortem" analysis. They suggest a formalism for high-level access-control models, where models can then be mapped into terms of a process algebra. Those terms are then subjected to static analyses that yield safe overapproximations of users' capabilities and restrictions. Our proposal extracts expressions that state such capabilities and restrictions, but we can not only use them for static analysis but also as wrappers that can extend or restrict given policies at run-time.

References

1. Bishop, M., S. Engle, S. Peisert, S. Whalen, and C. Gates, *Case Studies of an Insider Framework*, Proc. of Hawaii International Conference on System Sciences, pp. 1–10, IEEE Computer Society Press, 2009.
2. Bishop, M., D. Gollmann, J. Hunker, and C. W. Probst, *Countering Insider Threats*, Dagstuhl Seminar 08302, Leibnitz Center for Informatics, 18 pp., Dagstuhl Seminar Proceedings, ISSN 1862 - 4405, July 2008.
3. Bishop, M., *Panel: The Insider Problem Revisited*, Proc. of NSPW 2005, ACM Press, 2006.
4. Brackney, R., and R. Anderson, *Understanding the Insider Threat*, Proc. of a March 2004 Workshop, RAND Corp., Santa Monica, California, March 2004.
5. Bruns, G., and M. Huth, *Access-Control Policies via Belnap Logic: Effective and Efficient Composition and Analysis*, Proc. of CSF 2008, pp. 163–178, IEEE Computer Society Press, 2008.
6. Bruns, G., D. S. Dantas, and M. Huth, *A simple and expressive semantic framework for policy composition in access control*, Proc. of FMSE 2007, pp. 12–21, ACM Press, 2007.
7. Chakraborty, S. and I. Ray, *TrustBAC: integrating trust relationships into the RBAC model for access control in open systems*, Proc. of SACMAT '06, pp. 49–58, ACM Press, 2006.
8. Cheng, P.-C., P. Rohatgi, C. Keser, P. A. Karger, and G. M. Wagner, *Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control*, IBM Research Report, RC24190 (W0702-085), Computer Science, February 2007.
9. Cook, B., A. Podelski, and A. Rybalchenko, *Terminator: Beyond safety*, Proc. of CAV'06, LNCS 4144, pp. 415–418. Springer, (2006.
10. Cranor, L. F. and S. Garfinkel (editors), *Security and Usability – Designing Secure Systems That People Can Use*, O'Reilly, California, August 2005.
11. *Department of Defense Trusted Computer System Evaluation Criteria*, Technical Report DoD 5200.28-STD, US Department of Defense, 1985.
12. Hoffman, K., D. Zage, and C. Nita-Rotaru, *A Survey of Attack and Defense Techniques for Reputation Systems*, To appear in ACM Computing Surveys, Volume 41, Issue 4, December 2009.
13. Huth, M., *A Simple Language for Policy Composition and Analysis*, Talk given at [2]. `www.doc.ic.ac.uk/~mrh/talks/Dagstuhl08.pdf`
14. Jackson, D., *Software Abstractions: Logic, Language, and Analysis*, MIT Press, 2006.
15. Jones, S. P., J.-M. Eber, and J. Seward, *Composing contracts: an adventure in financial engineering (functional pearl)*, ACM SIGPLAN Notices 35(9): 280–292, ACM Press, 2000.
16. Lee, A. and T. Yu, *Towards a dynamic and composable model of trust*, Proc. of SACMAT'09, pp. 217–226, ACM Press.
17. Locasto, M. E., K. Wang, A. D. Keromytis, and S. J. Stolfo, *FLIPS: Hybrid Adaptive Intrusion Prevention* , in: Recent Advances in Intrusion Detection, LNCS 3858, pp. 82–101, Springer, 2006.
18. Moore, A. P., D. M. Cappelli, and R. F. Trzeciak, *The "Big Picture" of Insider IT Sabotage Across U.S. Critical Infrastructures*, Technical Report CMU/SEI-2008-TR-009, ESC-TR-2008-009, Carnegie Mellon University, May 2008.
19. The New York Times, *French Bank Says Rogue Trader Lost $7 Billion* , 25 January, 2008.
20. Patzakis, J., *New Incident Response Best Practice: Patch and Proceed is No Longer Acceptable Incident Response Procedure*, Guidance Software, Pasadena, California, September 2003.
21. Park, J., and R. S. Sandhu, *The $UCON_{ABC}$ usage control model*, ACM Trans. Inf. Syst. Secur. 7(1): 128–174, ACM Press, 2004.
22. Park, J. S. and J. Giordano, *Role-Based Profile Analysis for Scalable and Accurate Insider-Anomaly Detection*, Proc. IPCCC'06, 2006.
23. Park, J. S. and J. Giordano, *Access Control Requirements for Preventing Insider Threats* , Proc. ISI'06 LNCS 3975, pp. 529–534, Springer, 2006.

24. Probst, Ch. W., R. R. Hansen, and F. Nielson, *Where Can an Insider Attack?*, Proc. of FAST'06, LNCS 4691, pp. 127–142, Springer, 2006.
25. Probst, Ch. W. and J. Hunker, *The Risk of Risk Analysis-And its relation to the Economics of Insider Threats*, Proc. of the Eighth Workshop on the Economics of Information Security (WEIS 2009), June 2009.
26. Sandhu, R. S., E. J. Coyne, H. L. Feinstein, and C. E. Youman, *Role-Based Access Control Models*, IEEE Computer 29(2): 38–47, 1996.
27. Viega, J. and G. McGraw, *Building Secure Software*, Addison-Wesley Professional Computing Series, 2002.