RC25121 (W1103-056) March 9, 2011
Computer Science

Risk-Based Access Control Decisions under Uncertainty

Ian Molloy[1], Luke Dickens[2], Charles Morisset[3], Pau-Chen Cheng[1], Jorge Lobo[1], Alessandra Russo[2]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA

[2]Imperial College London

[3]Royal Holloway University of London

Risk-Based Access Control Decisions Under Uncertainty[⋆]

Ian Molloy[1], Luke Dickens[2], Charles Morisset[3], Pau-Chen Cheng[1], Jorge Lobo[1], and Alessandra Russo[2]

[1] IBM Research
{molloyim,pau,jlobo}@us.ibm.com
[2] Imperial College London
{luke.dickens03,a.russo}@imperial.ac.uk
[3] Royal Holloway University of London
charles.morisset@rhul.ac.uk

Abstract. This paper addresses the making of access control decisions under uncertainty, when the benefit of doing so outweighs the need to absolutely guarantee these decisions are correct. For instance, when there are limited, costly, or failed communication channels to a policy-decision-point. Previously, local caching of decisions has been proposed, but when a correct decision is not available, either a policy-decision-point must be contacted, or a default decision used. We improve upon this model by using learned classifiers of access control decisions. These classifiers, trained on known decisions, infer decisions when an exact match has not been cached, and uses intuitive notions of utility, damage and uncertainty to determine when an inferred decision is preferred over contacting a remote PDP. Clearly there is uncertainty in the predicted decisions, introducing a degree of risk. Our solution proposes a mechanism to quantify the uncertainty of these decisions and allows administrators to bound the overall risk posture of the system. The learning component continuously refines its models based on inputs from a central policy server in cases where the risk is too high or there is too much uncertainty. We have validated our models by building a prototype system and evaluating it with requests from real access control policies. Our experiments show that over a range of system parameters, it is feasible to use machine learning methods to infer access control policies decisions. Thus our system yields several benefits, including reduced calls to the PDP, reducing latency and communication costs; increased net utility; and increased system survivability.

[⋆] This Research is conducted through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and U.K. Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

1 Introduction

Modern access control systems rely on the PEP/PDP architecture: a Policy Enforcement Point (PEP), defined at the user-application level, forwards each access request made within an application to a Policy Decision Point (PDP), which analyses this request and returns an authorization decision—allow or deny. In some solutions, such as [8, 11, 14], a PDP is commonly implemented as a dedicated authorization server, where a PDP is located on a different node than the PEPs. While facilitating the enforcement of a consistent policy throughout the system, this architecture relies on the PEP being able to contact the PDP to query decisions, and therefore suffers from a single point of failure. In particular, key factors that affect the performance of the PEP are latency of the communication with the PDP, reliability and survivability of this connection (and the PDP itself), as well as the aggregated impact of communication costs (which can include contacting a human to perform the authorization). For example, in several contexts, such as mobile applications, these costs may be prohibitive.

A number of approaches have been proposed to address these issues, one common theme is to *cache* access control decisions at the PEP, such that the PEP does not have to forward the same access request more than once to the PDP. This is known as *authorization recycling* [6, 21] and is exact: either the PEP finds the request-decision pair in the history, or it forwards the request to the PDP. Recently, others have been exploring the tradeoff between efficiency and accuracy in an access control context. For example, Ni et al. [15] use a machine-learning algorithm to build a classifier of policy decisions from past request-decision pairs. Once the classifier is *accurate enough*, it is used it as a local decision point. As with any machine learning approach, the classifier has a degree of uncertainty with every decision, and can be incorrect even if it has a high accuracy. Thus, every decision made with such a classifier is inherently associated with a *risk* of the decision being wrong, either incorrectly allowing or denying some requests.

This paper presents a generalisation of the machine-learning approach from [15], which quantifies and measures the uncertainty in any locally inferred decision. We propose a model where this uncertainty is translated into a measure of the risk of making this decision. This gives us a very general policy with which to make decisions: when the utility of the local decision is high with respect to its risk, enact the local decision, otherwise defer to the central PDP.

We describe a model for quantifying uncertainty and risk, and show how these inform utility and risk based access control decisions. We have built a distributed access control system to test this principle: local decision points determine whether the tradeoff of the uncertainty and utility associated with a local decision is favorable and defer to the central PDP for a binding decision if not. This is a general framework that allows us to explore the balance between avoiding potential errors in local decisions, and the desire to limit reliance on the central PDP. We present three methodologies for making decisions:

– *Expected Utility* A risk neutral approach where weight expected gains and damage equally.
– *Risk Adjusted Utility* A pessimistic version of expected utility, where we penalise the uncertainty about damage more heavily.
– *Independent Risk Constraints* Augments expected utility with independent risk thresholds, where we reject high utility decisions if it is outweighed by the risk.

To validate this system, we have used data from a large corporation used to grant entitlements and permissions to system administrators. Our classifiers are initially untrained and in the beginning always consult the central PDP. Over time, the classifiers are trained using the central PDP's decisions thus improving accuracy and reducing the uncertainty, yielding fewer queries. In our experiments, we focus on the risk adjusted utility measure due to space constraints and because it provides a more conservative view of risk. Compared with the caching algorithm, our results show that our learning based approach reduces the queries to PDP by as much as 75% and increases the system utility by eight fold when the cache hit ratio is low, and has almost identical performance to the caching algorithm when the hit ratio is high.

Our results from this dataset show that machine-learning can be viably used for informing distributed access control decisions. The accuracy and robustness of our approaches are high, and more conservative measure of risk may be applied when appropriate. We believe these techniques can generalize to other use cases and domains, including information sharing in military and coalition environments; financial applications like credit card processing; and load balancing services like Netflix or document repositories. Compared to pure machine-learning solutions, our methods estimate the uncertainty of each classification, allowing one to minimize the pitfalls of using approximate decisions. Further, our solution avoids the higher communication costs of naive caching solutions, when the access control policy is too large to fit into the cache or locality of reference is poor.

This paper is organized as follows. Section 2 briefly discusses traditional access control systems. Section 3 discusses the architecture of access control systems based on our approach. We present in Section 4 different techniques to assess the risk and uncertainty of local access control decisions. An experiment and evaluation of our approach, applied on different scenarios, is presented in Section 5. Section 6 concludes this paper.

1.1 Related Work

Caching access control decisions is not a new concept, and has been already implemented [8, 11, 14]. The framework for policy evaluation presented in [3] uses a caching mechanism and is shown to drastically decrease the evaluation time of access control requests. Clearly, caching approaches are only valid when the cache is consistent, that is, returns the same decision as the central PDP; techniques to ensure strong cache consistency are proposed in [23]. In the context of distributed systems, each node can build its own cache in collaboration with the other nodes, thus improving the accuracy of each cache [22].

The previous methods only return *exact* decisions, that is, a decision for a request can be made locally only if this exact request has been previously submitted and answered. In the context of the Bell-LaPadula model [12], Crampton et al. [6] present a method to calculate a previously unseen decision based on the history of submitted requests, their responses, and the formal rules governing Bell-LaPadula. A similar approach for the RBAC model [9] is presented in [21] and an extension to Bloom filters is used in [19]. These inference techniques require knowledge of the underlying access control model, and are most efficient for hierarchical and structured access control models. Similarly, inference algorithms have been proposed based on the relationship of the subjects and objects in a database [18, 17]. However, the structure of the subject/object space still needs to be known. The approach we present in this paper is valid for any space of subjects and objects, as long as they are associated with attributes, which may be unstructured.

A first experiment of using machine-learning for making access control decisions is proposed in [15], where the authors note that there is a probability of error from the classifier, that is, the classifier can return a decision different than the one that would be returned from the central PDP. The approach presented in this document quantifies such a probability of error, in order to be able to make decisions, even with a degree of uncertainty.

Finally, a notion of risk in access control is introduced in [24, 5, 13], where the notion of an access to be either allowed or denied is blurred, and where each access is instead associated with a potential utility and damage. We reuse these notions here, in order to calculate the potential utility and damage of each access.

2 Traditional Access Control

A traditional *access control system* is a mechanism that grants or denies requests made by active entities, the *subjects*, to access some passive entities, the *objects*. Such a mechanism usually consists of two parts, as shown in Figure 1: the *Policy Decision Point* (PDP), which analyzes a request and decides whether it should be granted or not; and the *Policy Enforcement Point* (PEP), to which the access request is submitted and which enforces the request based on the decision of the PDP. The PDP's decisions are assumed to be always correct. More formally, let $\mathcal{S}$ be a set of subjects, $\mathcal{O}$ a set of objects and $\mathcal{A}$ a set of access modes. An access is a triple (s, o, a), meaning that the subject s accesses the object o according to the access mode a. The PDP takes an access as input, and returns a decision to the PEP indicating whether this access violates the policy or not. Such a decision is usually **allow** or **deny**; or, as it is the case for XACML [10], can also denote that the policy is not applicable to the request or that there is not enough information available to correctly evaluate the request. The policy decision need not be restricted to **allow** or **deny** and can include mitigations and/or obligations.

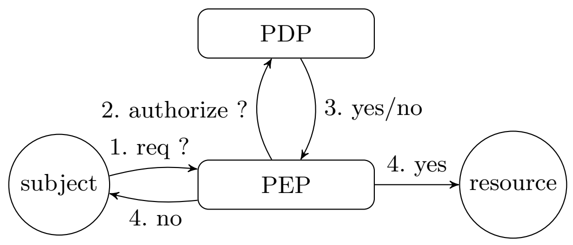


Fig. 1. Access Control System

For instance, when a user requests read access to a picture, the policy might require that the picture's resolution be downgraded before granting access. While our approach can handle any set of possible decisions, in this paper we restrict the *atomic* decisions, denoted $\mathcal{D}_{atm}$ to be the set $\{\mathbf{allow}, \mathbf{deny}\}$.

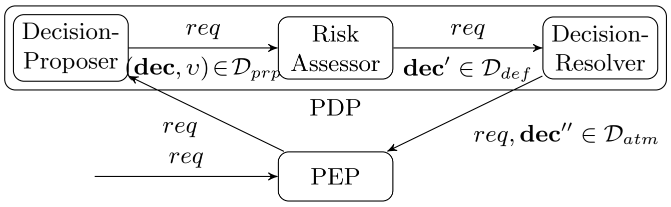
In today's networked world, access control systems are commonly distributed, meaning the PDP is not local to where decisions must be enforced: thus each node of the system requires its own PEP. As described in Section 1, it is not always possible or desirable for a PEP to contact the PDP, and to fully replicate the PDP on each node could be too difficult and costly for deployment and management. This paper describes an architecture, which can make decisions locally even when the *correct* decision is not known with certainty, and controls the level of uncertainty of local decisions—described in terms of utility and risk. There are several commonly used access control models, such as the the Bell-LaPadula model [12], the RBAC model [9], and the Chinese Wall model [4]. This approach is not tied to a particular model, but it fits well with attribute-based access control [2, 20], where each subject and object is associated with some attributes and the policy is defined using these attributes.

3 Uncertainty in Access Control

Figure 2 depicts the architecture of our system for access control under uncertainty. We assume that there is at least one *oracle* node containing the full policy, called the *central* PDP; and a *correct local decision* is one that agrees with the central policy. Each node has a local PDP consisting of three entities: a decision-proposer (or proposer), which returns a *guess* of the correct decision and a measure of the uncertainty; a risk-assessor (or assessor), which determines whether the decision is taken locally or deferred to the central PDP; and a decision-resolver (or resolver), which returns an enforceable decision. A PEP passes requests to the proposer, and enforces the decision from the resolver.

3.1 Decision-Proposer

A decision-proposer takes an access request as an input, and returns a *proposal* $(\mathbf{dec}, \upsilon)$, where $\mathbf{dec} \in \mathcal{D}_{atm}$ is a guess of the correct decision for this request, and


Decision-Proposer
req
$(\mathbf{dec}, \upsilon) \in \mathcal{D}_{prp}$
Risk Assessor
req
$\mathbf{dec}' \in \mathcal{D}_{def}$
Decision-Resolver
PDP
req
$req, \mathbf{dec}'' \in \mathcal{D}_{atm}$
req
PEP

$\mathcal{D}_{atm}$: Atomic decisions, *e.g.* $\{\mathbf{allow}, \mathbf{deny}\}$
$\mathcal{D}_{prp}$: Proposals, *e.g.* $\mathcal{D}_{atm} \times [0, 1]$
$\mathcal{D}_{def}$: Deferrable decisions, *e.g.* $\mathcal{D}_{atm} \cup \{\mathbf{defer}\}$

Fig. 2. Architecture of a local node

υ is a measure of uncertainty about **dec** being correct; υ can take a number of forms, depending on how the decision-proposer predicts decisions. In its simplest form υ is a probability, and the set of all possible outputs for the proposer is $\mathcal{D}_{prp} = \mathcal{D}_{atm} \times [0, 1]$. In this setting a proposal $(\mathbf{allow}, p)$ means that **allow** is the correct decision with probability p; $(\mathbf{allow}, p)$ is logically equivalent to $(\mathbf{deny}, 1 - p)$ if $\mathcal{D}_{atm} = \{\mathbf{allow}, \mathbf{deny}\}$.

The measure υ can also be a pair (α, β); where $\alpha, \beta \in [1, \infty]$ are parameters of a beta distribution [1], $\text{Beta}(\mu|\alpha, \beta)$; this captures our uncertainty about the value of the probability p that **dec** is correct, and μ is a variable approximating p. The beta distribution fits well with our learning based approach and gives finer control over risk estimation—see Section 4.

A proposer may be a simple caching algorithm which stores a given number of requests together with the corresponding correct decisions from the central PDP. In this setting, a proposal is a pair $(\mathbf{dec}, p) \in \mathcal{D}_{atm} \times [0, 1]$. If the policy never changes then cached decisions are always correct. So, if the decision for a request has been cached then **dec** is that decision and $p = 1$ indicating complete certainty; otherwise **dec** is randomly chosen and $p = 0.5$ indicating complete uncertainty. If the policy is periodically updated, then there is a probability that a cached decision no longer agrees with the central policy. For instance, if 10% of central decisions change every hour, then a decision cached an hour ago could be assigned a 0.9 probability of being correct.

In our learning based approach, the proposer is mainly a classifier produced by a machine-learning algorithm, which treats a decision as a class and assigns a request to a class based on the request's attributes. The learning process uses an input set of requests whose corresponding correct decisions are known, and relies on the assumption that two requests close to each other are likely to have the same decision. Elements of the input set are samples of past decisions made by the central PDP. The proposals are of the form $(\mathbf{dec}, \alpha, \beta)$, where α and β are computed from the numbers of correct and incorrect assignments made by the classifier. Section 5 describes and evaluates this approach in detail.

3.2 Risk Assessor

A risk-assessor (or assessor) takes a request and a proposal and returns a *deferrable decision*. The assessor must determine whether a decision can be made locally, based on the proposal, or should be deferred to the central PDP. Hence, the risk assessor can return any decision in the set of deferrable decisions, $\mathcal{D}_{def} = \mathcal{D}_{atm} \cup \{\textbf{defer}\}$, where **defer** is the decision to contact the central PDP. Note that, deferrable decisions are no longer associated with uncertainty. In other words, the role of the risk-assessor is to transform the uncertainty in a proposal into a firm decision.

A trivial definition for a risk-assessor is to fix an uncertainty threshold below which all decisions are deferred to the central policy, and above which all the decisions are taken locally. However, in general, different requests have different potential impacts on the system. Some requests have high potential utility for the system, and wrongly denying them might cause big loss of utility, and some requests have high potential damage, and wrongly allowing them is a threat to the integrity and survival of the system. Another parameter to consider is the cost of contacting the central PDP, which is important in some situations. The risk-assessor can also take a *risk appetite*, which controls the degree that the decisions are *risk-averse*; a risk-averse decision maker is willing to accept a lower expected utility if there is more certainty about the consequences of a decision. More details on the risk assessor are given in Section 4.

3.3 Decision Resolver

A decision resolver takes a request and a deferrable decision, $\mathbf{dec}_d \in \mathcal{D}_{def}$ and returns an atomic decision, $\mathbf{dec}_a \in \mathcal{D}_{atm}$. Typically, $\mathbf{dec}_a = \mathbf{dec}_d$ if $\mathbf{dec}_d \in \mathcal{D}_{atm}$; otherwise, the resolver defers the request to the central PDP.

4 Assessing Local Decisions

In this section, we consider how the risk-assessor takes a proposal from the decision-proposer, and returns a firm decision. In our system by design the proposer's suggested decision may conflict with the decision that would have been made if passed to the central policy. If a request that is allowed is denoted as a *valid* request and a request that is denied is an *invalid* request, then there can be two kinds of errors that the local PDP can make: false-allows (allowing an invalid request) and false-denies (denying a valid request). Note that there can be negative consequences resulting from either of these errors: a false-allow can result in the leakage of information, corruption of information, or privilege escalation; a false-deny can result in disruption of service, breach of service level agreements (SLA), and other consequences such as damaged reputation. Further, making a correct decision leads to gains and benefits.

In our paper the approaches that we propose will be based on the tradeoff between the gains in allowing the decision against the potential damages that may result from an incorrect decision. In order to formally describe our approach, in this section we will introduce notation to capture the gains and damages due to the correctness of the decisions to allow or deny a request. Consider an assessor which must decide whether to locally allow or deny a request, (s, o, a), or to pass this decision on to a central PDP. The assessor has a ternary choice to make: whether to allow the request locally, decision **allow**; to deny locally, decision **deny**; or to defer the decision and contact the central PDP, decision **defer**. We assume that the central PDP is an oracle, and always gives the correct decision. If the assessor decides **allow**, then either the request is valid (a true-allow) and the system makes a gain, g, because an appropriate access has been made[4]; or the request is invalid and the decision has caused some amount of damage, d_A, due to an inappropriate access being made. Likewise, if the assessor decides, **deny**, then either the request was invalid, **deny** is the correct decision and nothing additional happens (i.e. a gain of 0), or the request was valid the **deny** decision is incorrect, and a damage, d_D, is incurred for a false-deny.

Finally, the assessor may decide to **defer** to the central PDP for a decision. If the request is valid, then the central PDP will grant the request and the system will make the gain, g, but incur a cost, c (the contact cost). If the decision is **defer** and request is not valid then the central PDP will deny the request and there will be no gain and again a contact cost, c. As the central PDP is an oracle, it never makes an incorrect decision so there is no damage associated with this. These gains, costs and damages are shown in Table 1. Standard gains and costs (negative gains) are kept separate from damage, which is assumed to be associated with rare and costly negative impacts. This distinction becomes important, when we consider risk measures in Sections 4.2 and 4.3. The impacts presented in Table 1 are not the most general that could be conceived, but are sufficiently rich to be of interest. For example, there may be a small gain associated with denying an invalid request, such as an increase in system reputation, or a damage in allowing a valid request, such increasing the number of copies of a document, and thus it's exposure and risk to later compromise.

All of the approaches described in this paper will be based on the various tradeoffs between the certainty of classification and an estimation of these gains and damages. To apply our methodology to a real example, we must be able to (approximately) compute these gains and damages. We note here that there are many applications and domains where these gains and damages can be estimated. For example, in financial service transactions like credit cards, these costs may be estimated easily or provided a priori by some agreement. Other examples include cases where fined grained service level agreements are defined a priori. We will discuss further details in Section 4.4.

[4] Ideally, an access control system should enable as many appropriate accesses as possible and so our primary concern is to maximize the number and value of these accesses - subject to limiting costs and damages.

	Request Valid		Request Invalid	
	gain	damage	gain	damage
allow	g	0	0	d_A
deny	0	d_D	0	0
defer	$g - c$	0	$-c$	0

Table 1. Outcome Gains and Damage

	$U(\cdot\|\mathbf{allow}, p)$	$U(\cdot\|\mathbf{deny}, p')$
allow	$pg - (1-p)d_A$	$(1-p')g - p'd_A$
deny	$-pd_D$	$-(1-p')d_D$
defer	$pg - c$	$(1-p')g - c$

Table 2. Expected utilities for various decisions given certain proposer outputs.

4.1 Expected Utility

To aid this decision, the proposer attempts to classify request, (s, o, a), to determine whether it should be allowed or denied, and include some measure of its uncertainty in this classification. In the simplest case, the proposer can return one of two kinds of answers, e.g. $(\mathbf{allow}, p)$ or $(\mathbf{deny}, p')$. The first response, $(\mathbf{allow}, p)$ means that the proposer has classified the request as an allow, and predicts that there is a probability p that this is correct, i.e. there is a probability of p that the request is valid. Likewise, $(\mathbf{deny}, p')$ means that the proposer has classified the request as a deny, and predicts a probability p' that the request is invalid.

The expected utility for a decision, $\mathbf{dec} \in \mathcal{D}_{def}$, is the expected gain of each outcome (request valid or invalid) minus the expected damage, and these utilities are shown in Table 2. In this approach, we assume a risk neutral posture, meaning deferrable decisions are made based purely on the expected utility, $U(\cdot)$, for each decision. The assessor simply returns the decision corresponding to the highest utility. However, with such an approach, the assessor can potentially take a decision that is *contrary* to the one returned by the proposer, e.g. the assessor returns **allow** when the proposal states **deny**, or vice versa. For instance, when the proposer returns $(\mathbf{allow}, p)$ given a request where $d_A \gg d_D$ and $g \approx c$, then it may occur that $U(\mathbf{defer}) < U(\mathbf{deny})$, but in the examples explored so far this is not the case and may even indicate a badly described system. In the rest of the paper, we assume that the gains, damages and costs are defined in such a way, as to preclude contrary decisions.

Hence, given a request *req* and a decision proposer δ returning a decision and a probability, the expected utility assessor, ρ_{eu}, is defined as:

$$\rho_{eu}(req) = \begin{cases} \textbf{allow} & \text{if } \begin{pmatrix} \delta(req) = (\textbf{allow}, p) \\ \wedge\, U(\textbf{allow}) \geq U(\textbf{defer}) \end{pmatrix} \\ \textbf{deny} & \text{if } \begin{pmatrix} \delta(req) = (\textbf{deny}, p) \\ \wedge\, U(deny) \geq U(defer) \end{pmatrix} \\ \textbf{defer} & \text{otherwise} \end{cases}$$

Thus in this first approach, we directly use the certainty of the decision to compute the expected utility and choose the decision with the highest utility. However this approach can lead to significant damages due to incorrect decisions because while the expected damage may be low, in some cases, there is a small probability of significant damage. In the following approaches we will try to explicitly take into account the risk of such significant damage.

4.2 Risk Adjusted Utility

Now let's assume that we are making the same decision, but that we want to use a risk based measure. In other words, we want to explicitly account for any uncertainty in the proposal. For this, the proposer returns a beta distribution, describing uncertain knowledge about the probability of a correct classification. If the proposer classifies the request as valid then it returns $(\textbf{allow}, \alpha, \beta)$, where $\alpha, \beta \in [0, \infty]$ are the parameters of the beta distribution [1]. Likewise, if it classifies the request as invalid then it returns $(\textbf{deny}, \alpha', \beta')$.

Again we focus just on the **allow** case, and assume that the assessor can return either **allow** and **defer**. If the proposer returns $(\textbf{allow}, \alpha, \beta)$, we denote by $p = \frac{\alpha}{\alpha+\beta}$ (the expectation of the beta distribution), as the *unbiased probability* that this decision is correct. For our risk methods, we also want to account for the uncertainty in this prediction to take a pessimistic view of the expectation of loss. Here, we achieve this with a pessimistic probability, $\tilde{p}_{\downarrow n}$, calculated using risk significance level $0 < n < 1$ and the beta distribution parameters, α and β. For example, a 5% pessimistic expectation ($n = 0.05$) considers the average damage in the worst 5% of universes—see Appendix A for details. So in effect, what this approach accomplishes is to mitigate the damage that will occur in the worst 5% of outcomes, i.e., there is less than a 5% probability there will be a worse outcome.

Our first method, called *risk adjusted utility*, adjusts the *unmodified* utility, U, to account for uncertainty at our significance level n. For this technique, we consider the unbiased probability, p, for standard gains and costs, and the pessimistic probability, $\tilde{p}_{\downarrow n}$, when considering the exceptional cases causing damage. We will assume that we are always risk averse (pessimistic) and as we do not consider contrary decisions. So, given proposal $(\textbf{allow}, \alpha, \beta)$, the probability $\tilde{p}_{\downarrow n} < p = \frac{\alpha}{\alpha+\beta}$, takes a pessimistic view of how certain the proposer is to have classified correctly when considering the impact of damage. The risk adjusted

	$\tilde{U}(\cdot\|\mathbf{allow},\alpha,\beta)$	$\tilde{U}(\cdot\|\mathbf{deny},\alpha',\beta')$
allow	$pg-(1-\tilde{p}_{\downarrow n})d_A$	–
deny	–	$-(1-\tilde{p}'_{\downarrow n})d_D$
defer	$pg-c$	$(1-p')g-c$

Table 3. Expected utilities for various decisions given certain proposer outputs.

utility, $\tilde{U}(.)$ for each deferrable decision, $\mathbf{dec} \in \mathcal{D}_{def}$, is then the *unbiased* expected gain of each outcome (request valid or invalid) conditioned on **dec**, minus the sum of the damages but weighted according to the risk adjusted probability, $\tilde{p}_{\downarrow n}$. The risk adjusted utilities for proposal $(\mathbf{allow}, \alpha, \beta)$ are shown in Table 3. Alongside are the equivalent risk adjusted utilities for proposal $(\mathbf{deny}, \alpha', \beta')$, giving unbiased probability $p' = \frac{\alpha'}{\alpha'+\beta'}$ and pessimistic probability $\tilde{p}'_{\downarrow n}$.

Hence, given a request *req* and a decision proposer δ returning a decision and a probability, the risk adjusted utility assessor, ρ_{au}, is defined as:

$$\rho_{au}(req) = \begin{cases} \mathbf{allow} & \text{if} \begin{pmatrix} \delta(req) = (\mathbf{allow}, p) \\ \wedge\ \tilde{U}(\mathbf{allow}) \geq \tilde{U}(\mathbf{defer}) \end{pmatrix} \\ \mathbf{deny} & \text{if} \begin{pmatrix} \delta(req) = (\mathbf{deny}, p) \\ \wedge\ \tilde{U}(\mathbf{deny}) \geq \tilde{U}(\mathbf{defer}) \end{pmatrix} \\ \mathbf{defer} & \text{otherwise} \end{cases}$$

This approach has shown how to modify the risk-neutral approach from the previous section to a risk averse approach where we try to mitigate the losses from the worst fraction of outcomes.

4.3 Independent Risk Constraints

The second risk method, called *independent risk constraints*, uses the standard expected utility to rank the decisions in order of risk neutral preference, but then reject any which do not satisfy our risk constraint(s). The proposer classifies the requests and returns either $(\mathbf{allow}, \alpha, \beta)$ or $(\mathbf{deny}, \alpha', \beta')$, and we calculate the expected utilities as they appear in Table 2 using $p = \frac{\alpha}{\alpha+\beta}$ for the probability of allow, and $p' = \frac{\alpha'}{\alpha'+\beta'}$ for the probability of deny. The risk, $R(\cdot)$, for each deferrable decision is calculated independently. The risk for a decision, **dec**, only takes the exceptional potential damages into account, ignoring standard gains and cost. For the risk, we take a pessimistic view, here achieved by using the risk adjusted probability, $\tilde{p}_{\downarrow n}$, and the fixed damage values, d_A and d_D. These risks are shown in Table 4. Alongside are the equivalent risks for proposal $(\mathbf{deny}, \alpha', \beta')$, giving unbiased probability p' and pessimistic probability $\tilde{p}'_{\downarrow n}$.

The decision making process involves either a fixed threshold $t > 0$, or more generally for each deferrable decision, $\mathbf{dec} \in \mathcal{D}_{def}$, with utility, $u_{\mathbf{dec}}$, a threshold that depends on the utility, i.e. $t(u_{\mathbf{dec}})$. We then take the following steps

	$R(\cdot \mid \textbf{allow}, \alpha, \beta)$	$R(\cdot \mid \textbf{deny}, \alpha', \beta')$
allow	$(1 - \tilde{p}_{\downarrow n})d_A$	0
deny	0	$(1 - \tilde{p}'_{\downarrow n})d_D$
defer	0	0

Table 4. Risks for various decisions given certain proposer outputs.

1. Evaluate the expected utilities and risks as shown respectively in Tables 2 and 4.
2. Rank the decisions according to their utilities.
3. Check each decision, $\textbf{dec} \in \mathcal{D}_{def}$ in turn (highest utility first), and choose the first decision whose risk is below $t(u_{\textbf{dec}})$.

For example, let's assume that the probability p is sufficiently close to 1 that **allow** is preferable to **defer**. Notice also that **defer** carries zero risk and will always be risk acceptable. Hence, given a request *req* and a decision proposer δ returning a decision and a probability, the independent risk assessor, ρ_{ic}, is defined as:

$$\rho_{ic}(req) = \begin{cases} \textbf{allow} & \text{if} \begin{pmatrix} \delta(req) = (\textbf{allow}, p) \\ \wedge\, U(\textbf{allow}) \geq U(\textbf{defer}) \\ \wedge\, R(\textbf{allow}) \leq t(U(\textbf{allow})) \end{pmatrix} \\ \textbf{deny} & \text{if} \begin{pmatrix} \delta(req) = (\textbf{deny}, p) \\ \wedge\, U(\textbf{deny}) \geq U(\textbf{defer}) \\ \wedge\, R(\textbf{deny}) \leq t(U(\textbf{deny})) \end{pmatrix} \\ \textbf{defer} & \text{otherwise} \end{cases}$$

More generally still we may not always have a decision of zero risk, and hence we may need some way to resolve a decision when all decisions exceed the risk threshold. There are a number of ways of doing this, and the most appropriate technique is a subject of future research.

Both risk methods have their advantages and disadvantages. For instance, risk adjusted utility behaves more like standard expected utilities, and there is always a clearly defined decision for the assessor. Conversely, the independent risk assessor keeps the two measures utility and risk separate, and hence is more flexible and we could define multiple risk thresholds at different significance levels. However, we do not necessarily know how to deal with situations where all decisions violate a risk constraint.

4.4 Estimating Gain and Damage

In the previous discussion, we implicitly assumed all utility, damage, and costs are given in comparable units such as currency. There are many such applications where this is true. For example, in credit card processing the card issuer levies an *Authorization Fee* to evaluate a request and may charge a *Charge Back Fee*, in addition to the transaction amount, if the transaction was fraudulent. In all instances, the gains, damages, losses, and communication costs are in a common currency.

In many scenarios the damage, gains, losses, and costs may be given in different units, e.g., gain in currency, damage in reputation, and loss of physical assets. Consider a military operation where a soldier requests access to sensitive resources, e.g., SECRET and NOFORN classified documents. Failure to grant access to the resources can jeopardize the mission, possibly resulting in a successful attack that damages military assets, e.g., weapon depots. Granting the request risks potential leakage of sensitive information, possibly harming the reputation of the military and the country, and risking future cooperation of informants. However, granting the request may also lead to successfully completed missions, reducing the loss of life and ending conflicts early. The utility measures discussed previously requires these measures to be converted into a common set of units, and there is uncertainty in such a conversion rate.

We suggest the same techniques used to calculate the risk of making an incorrect decision described in Section 4.2 can be applied to selecting conservative, risk-adjusted conversion rates. For example, if one obtains a probability density function (PDF) over conversion rates, say human life to dollars, the expected value of this PDF can be used for gains, while a pessimistic value can be used for potential losses. If we take household income as one potential conversion rate, then the median US household income, around \$34–55K, could be used for expected value, while the top 5%, or \$157K, might be used for the risk.

Probability distributions for conversion rates can be obtained for each potential measure, and can be estimated from a variety of sources. For example, reputational damage has been estimated by a company's stock price after the announcement of a security breach[5], and the cost to recover from the incident[6]. Estimates for human life can be found from actuarials, law suits and settlements, liability limits, health coverage costs, and financial contributions, among other sources. By taking a pessimistic view of these conversions, we aim to avoid the sorts of mistakes made on the infamous Ford Pinto[7] (which estimated a human life to \$US 200,725).

5 Experiment

In this section, we describe our prototype system and experimental results. To evaluate the usefulness of risk-based decision making, we devise a simple decentralized access control system consisting of a single local risk-based policy

[5] `http://www.informationweek.com/news/security/showArticle.jhtml?articleID=171200494`

[6] PGP provides an annual estimate of the costs of a security breach: `http://www.pgp.com/insight/research_reports/`

[7] `http://motherjones.com/politics/1977/09/pinto-madness`

enforcement and decision point, and a central policy decision point that called the oracle. In our experiments, users submit requests, which are locally evaluated and an estimate of the uncertainty calculated as described in Section 4.2 and Appendix A. By evaluating potential risks and gains, the system determines when decisions can be made locally, and when the oracle must be queried at a cost. We run the system over a large number of requests, e.g., 100,000, logging the request, decisions, and uncertainty. After which, we compare these decisions with the oracle policy, and aggregate the net utility of the system, including any gains, costs and damages incurred during operation. We first describe our experimental system in more detail. For clarity, we distinguish between *standard* damage (for a false-allow) with loss (damage of a false-deny).

5.1 Experiments

The oracle is considered infallible, and always returns correct results. The oracle contains a real access control policy from a system that provisions administrative access to a commercial system obtained from a corporation. The policy consists of 881 permissions and 852 users with 25 attributes describing each user; there are 6691 allowed requests. We do not have access to actual permission usage logs, e.g., requests or frequency of permission use, and instead simulate incoming requests by sampling from the full policy, obtaining both valid (allow) and invalid (deny) requests. Users are selected uniformly randomly, and permissions are selected from a multinomial distribution where all allowed permissions are independently and identically distributed, as are all denied permissions, but the probability of selecting an allowed permission is a system parameter of our choosing. For example, while a uniform sampling of single permissions at random may return a valid transaction 0.9% of the time (the density of the user-permission relation), we can generate sample requests with an arbitrary frequency of valid transactions, e.g. 30%, 50% or 80%. This allows us to model different scenarios, such as a system under attack (mostly invalid requests), or normal usage (mostly valid requests).

5.2 Classifiers

Our experimental prototype is implemented in Python, and uses PyML for support vector machines[8], which we use for our classifier. A support vector machine (SVM) is a supervised learning method for classification and regression. Given labeled points in a k-dimensional space, an SVM finds a hyperplane (normal vector $\mathbf{w}$ and offset b from the origin) that separates the two classes and maximizes the minimal distance between any point and the hyperplane. The class of an unseen point $\mathbf{x}$ is determined by which side of the hyperplane it is on, i.e., $\mathrm{sign}(\mathbf{w} \cdot \mathbf{x} - b)$. See Figure 3.

An SVM is a linear classifier, however a kernel can be applied first to allow non-linear classification. Kernels may result in over fitting when the number of

[8] PyML provides a wrapper for libSVM

input datapoints is too small, e.g., less than or equal to the degree of a polynomial kernel. To prevent this, we do not train the SVM until a sufficient number of data points are specified. We use both polynomial and Gaussian kernels.

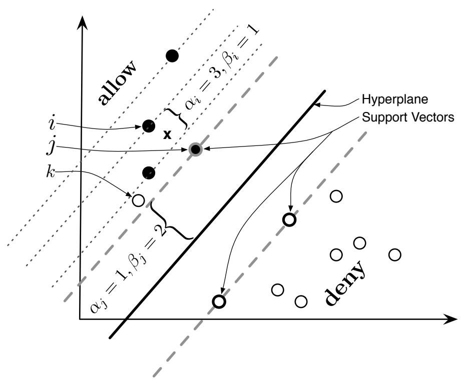


Fig. 3. A Support Vector Machine classifier and calculation of α, and β parameters for uncertainty.

Intuitively there is less uncertainty in a prediction the farther away from the hyperplane the point is projected, $|\,\mathbf{w} \cdot \mathbf{x} - b\,|$, and the uncertainty is high between the support vectors (the points closest to the hyperplane). We define an $(\alpha_i, \beta_i$ value for each band corresponding to the point i in our cached training set. Any point correctly classified closer to the hyperplane increments α_i, while each point farther away incorrectly classified increments β_i. For example, in Figure 3, the incorrectly predicted point k will increment β_j, while correctly classifying a new point x would increment α_i (and points farther away). The α and β values are only valid within their respective bands, inclusively. We use the points in the training set as our initial sample, and further update these quantities when we defer to the oracle if the uncertainty is too high. When we retrain, all α, β values are reset. The hyperplane itself has $\alpha = 1, \beta = 1$.

In our setting, the data points represent users. We map a user to a k-dimensional space by converting their attributes into binary vectors. For example, consider the *Department* attribute that has three values: R&D, Sales, and Formula One. These are represented by the binary vectors $\langle 0, 0, 1\rangle$, $\langle 0, 1, 0\rangle$, and $\langle 1, 0, 0\rangle$, respectively. The final k-bit vector for a user is the concatenation of all attributes. In the remainder of this paper when we refer to a subject $s \in \mathcal{S}$, we are referring to a point in some k-dimensional space.

Finally, there are $|\,\mathcal{O} \times \mathcal{A}\,|$ SVM classifiers, one per permission, trained on subjects $s \in \mathcal{S}$. An alternative implementation is to define a single classifier whose input is $\mathcal{S} \times \mathcal{O} \times \mathcal{A}$, however, based on the results from [15], some object-right requests have more uncertainty than others. To simplify calculating the uncertainty of the classified requests, we choose to define a classifier per permission.

Each PDP based on an SVM will store a small number of data points (subjects and decisions) in a buffer for later retraining. Before predicting the decision using an SVM, we first check to see if the point is in the cached buffer. When it is, we return the known decision with zero uncertainty (assuming a static policy). When a new input and decision is specified by querying the oracle, it is provided to the SVM so that it may be retrained. When the number of samples exceeds the size of the buffer a new SVM is trained and the point correctly classified and project farthest from the hyperplane is discarded. Thus the buffer contains the points that most closely define the support vectors. We use polynomial (degree four) and radial bias function (RBF) kernels in our experiments.

We implement several baseline PDPs and learning-based PDPs and strategies to compare the risk-based against:

- **Default PDP** This local PDP will always defer to the central PDP.
- **FIFO Cache** A first-in-first-out cache. If the request is not in the cache, the oracle is queried. We use two FIFO cache sizes: (E) the size of the FIFO cache is equal to the total memory of the SVMs; (Inf) the FIFO cache is unbounded.

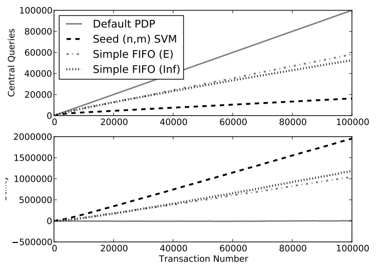
And several learning-based PDPs:

- **SACMAT'09** Consistent with the approach taken by Ni et al., we train an SVM for each permission and accept any decision they return, i.e., we assume there is zero uncertainty.
- **Seeded SVM** An SVM is constructed for each permission and seeded with n random users granted the permission, and m random users not granted the permission. We train and test on the $n + m$ users. We use $n = m = 10$.
- **Unseeded SVM** An untrained SVM is created for each permission. Before the SVMs can be trained a minimum number of allow and deny decisions must be specified by querying the oracle. The buffer of an unseeded SVM is the same as a seeded SVM.

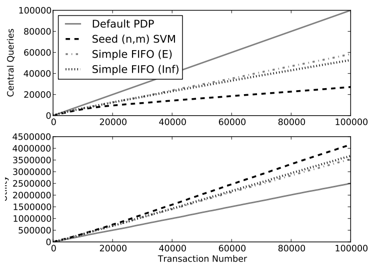
5.3 Parameters

We experiment with a wide range of system parameters for the damage resulting from a false-allow, the loss associated with a false-deny, the gains from a true-allow, and the cost to contact the oracle. We assume there is no gain or loss from a true-deny (besides the possible communication costs). In all experiments, the absolute value of the system parameters is unimportant, but rather their relative values.

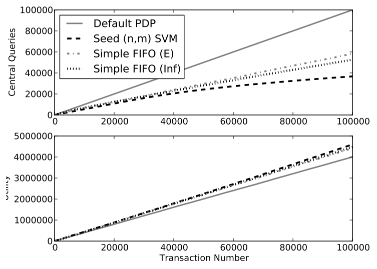
- *Military Scenario* There is a small cost to contact the oracle. The gains are twice the contact costs, and damage and loss are twice the gains.
- *Financial Application* The loss is zero. There is a small contact cost, and gains are four times the contact costs. Damage is ten times the gains.



(a) Military Scenario



(b) Financial Application



(c) Service Provider

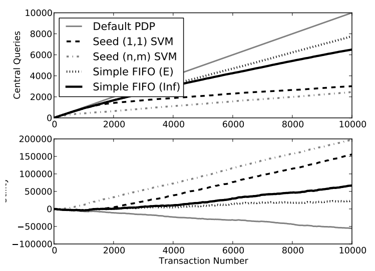
Fig. 4. Evaluation under different system parameters.

– *Service Provider* There is a small contact cost and the damage is only twice the contact costs while the gains are ten times the contact costs, and losses are ten times the gains.

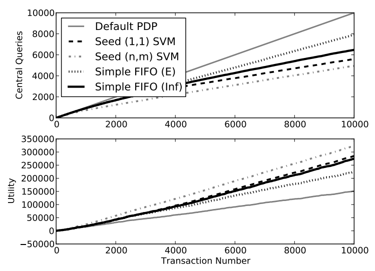
5.4 Evaluation Results

We now present our experimental results. First, we compare our general approach with a default policy that always contacts the oracle, and two FIFO caches: one with a bounded queue size (the same as all SVMs combined); and one unbounded. For these experiments we restrict ourselves to a seeded SVM where we provide it with random allow and deny decision a priori required to train, and evaluate the results against our three parameter scenarios: military, financial, and service provider. The results are shown in Figure 4. It is clear from the figures that both the caching and risk-based classifier approaches reduce the number of calls to the oracle, by as much as 75% in the case of the military scenario, while maintaining equal or better utility than the default or caching-based mechanisms (almost twice the utility in the military scenario). In only the service provider example was the increase in utility not statistically significant.

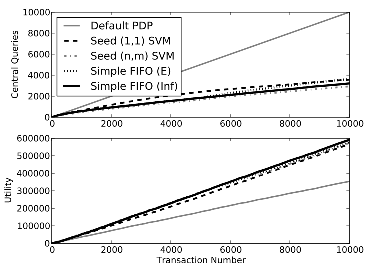
Next, we consider the impact of the rate of invalid versus valid requests has on the performance of the classifiers. We evaluate this for two reasons. First, it is unknown that the ratio is in a typical deployment. Second, it has been found



(a) 30% Valid Requests



(b) 50% Valid Requests



(c) 70% Valid Requests

Fig. 5. When the cache-miss rate is high, risk-based classifiers significantly outperform traditional caching.

that SVMs result in more false-denies than false-allows [15], so varying the ratio may reveal useful insights. We found that the sampling rate of valid versus invalid requests impacts the rate of cache hits for the FIFO implementations greatly impacted their performance. When the number of cache hits was low, for example, by sampling more invalid requests, we explore a larger amount of the possible request space, resulting in more cache misses before a hit is returned. Conversely, when the rate of valid requests was high, a larger fraction of the valid sampled requests reside in the cache (in part due to the smaller sampling size because the fraction of allowed requests is so low). The higher number of false-denies also results in increased uncertainty for many of these requests, causing the SVMs to be retrained more frequently, and thus deferring more requests. The results are shown in Figure 5.

Next, we consider the potential impact of trusting the SVM classifiers without first evaluating their uncertainty, we is done in [15]. Here, the false-allows and false-denies can result in devastating losses. To illustrate this, we implement a naive decision resolver (SACMAT'09) that accepts any predicted decision, regardless of the risk. Figure 6 illustrates the potential decrease in utility from a small number of false-positives the classifier.

Finally, we evaluate how well the systems perform when the communication costs become prohibitive. Here, we set the cost to contact the oracle to be

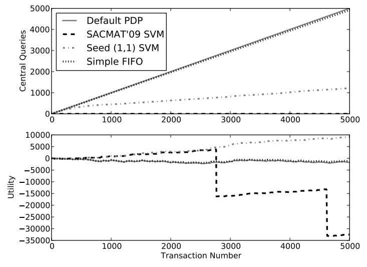


Fig. 6. The impact of not evaluating the classifier risk.

five-times the gains from allowing a valid request, and set the damage to be four-times the communication costs. This type of scenario may correspond to a military setting where communication is expensive, e.g., requiring a satellite link, or where radio usage should be minimized to avoid triangulation by the enemy. The results, shown in Figure 7, illustrate that while the risk-based outperforms the FIFO caches and reduces queries by 65%, it cannot keep the utility positive. However, it should be noted that the FIFO caches loose over three-times more utility due to the communication costs.

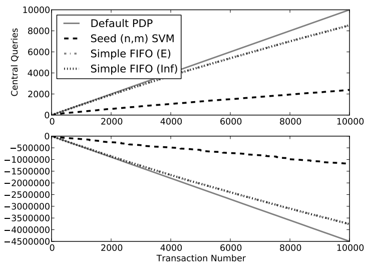


Fig. 7. Communication costs is five times the gains, and damages are ten times the gains.

6 Conclusion

This paper presents a new risk-based architecture for a local PDP, where a decision is first proposed with a known level of uncertainty, for instance when using a machine-learning algorithm; the tradeoff of the uncertainty and utility associated with this decision is then assessed, determining whether the decision can be taken locally or the central PDP should be contacted. We have defined three different risk methodologies, namely *Expected Utility, Risk Adjusted Utility*, and *Independent Risk Constraints*, each representing a different risk perspective.

Our approach has been validated using data from a large corporation, and consistently performed better in experiments than a naive caching mechanism, in a number of different scenarios. In the best case, we reduce the number of queries to the central PDP by as much as 75% and saw an eight-fold increase in the system utility. These profound improvements occured when the cache hit rate is low.

A future direction for our work would be to compare it with approximate authorization recycling [6, 21], however, these approaches are fitted for particular access control models, e.g. the Bell-LaPadula and RBAC models, respectively, and therefore we would need to adapt our system appropriately and find large enough datasets.

We also need to consider dynamic access control policies, where a decision for a given request might change over time. In this context, the uncertainty returned by the decision-proposer must combine the uncertainty due to classifier error with the uncertainty due to possible updates to the policy. Finally, additional research is needed to better determine when a classifier should be retrained and when a given sample should be discarded, particularly challenging when dealing with dynamic policy changes.

References

1. C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007.
2. P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 134–143, New York, NY, USA, 2000. ACM.
3. K. Borders, X. Zhao, and A. Prakash. Cpol: high-performance policy evaluation. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 147–157, New York, NY, USA, 2005. ACM.
4. D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 329–339, May 1989.
5. P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *IEEE Symposium on Security and Privacy*, pages 222–230. IEEE Computer Society, 2007.
6. J. Crampton, W. Leung, and K. Beznosov. The secondary and approximate authorization model and its application to Bell-LaPadula policies. In *Proceedings of 11th ACM Symposium in Access Control Models and Technologies*, pages 111–120, 2006.
7. E. Eberlein, R. Frey, M. Kalkbrener, and L. Overbeck. Mathematics in Financial Risk Management. *Jahresbericht der Deutschen Mathematiker Vereinigung*, 109:165–193, 2007.
8. Entrust. Getaccess design and administration guide, September 1999.
9. D. F. Ferraiolo and D. R. Kuhn. Role-based access control. In *Proceedings of the 15th National Computer Security Conference*, pages 554–563, 1992.

10. S. Godik and T. Moses. Oasis extensible access control markup language (XACML). OASIS Committee Secification cs-xacml-specification-1.0, November 2002, http://www.oasis-open.org/committees/xacml/, 2002.
11. G. Karjoth. Access control with IBM Tivoli Access Manager. *ACM Trans. Inf. Syst. Secur.*, 6:232–257, May 2003.
12. L. LaPadula and D. Bell. Secure Computer Systems: A Mathematical Model. *Journal of Computer Security*, 4:239–263, 1996.
13. I. Molloy, P.-C. Cheng, and P. Rohatgi. Trading in Risk: Using Markets to Improve Access Control. In *New Security Paradigms Workshop (NSPW)*. Applied Computer Security Associates (ACSA), September 2008.
14. Netegrity. Siteminder concepts guide, 2000.
15. Q. Ni, J. Lobo, S. B. Calo, P. Rohatgi, and E. Bertino. Automating role-based provisioning by learning from examples. In B. Carminati and J. Joshi, editors, *SACMAT*, pages 75–84. ACM, 2009.
16. F. W. J. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark, editors. *NIST Handbook of Mathematical Functions*. Cambridge University Press, 2010.
17. S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 551–562, New York, NY, USA, 2004. ACM.
18. A. Rosenthal and E. Sciore. Administering permissions for distributed data: factoring and automated inference. In *Proceedings of the fifteenth annual working conference on Database and application security*, DAS'01, pages 91–104, Norwell, MA, USA, 2002. Kluwer Academic Publishers.
19. M. V. Tripunitara and B. Carbunar. Efficient access enforcement in distributed role-based access control (RBAC) deployments. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 155–164, New York, NY, USA, 2009. ACM.
20. L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, FMSE '04, pages 45–55, New York, NY, USA, 2004. ACM.
21. Q. Wei, J. Crampton, K. Beznosov, and M. Ripeanu. Authorization recycling in rbac systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 63–72, New York, NY, USA, 2008. ACM.
22. Q. Wei, M. Ripeanu, and K. Beznosov. Cooperative secondary authorization recycling. In *Proceedings of the 16th international symposium on High performance distributed computing*, HPDC '07, pages 65–74, New York, NY, USA, 2007. ACM.
23. M. Wimmer and A. Kemper. An authorization framework for sharing data in web service federations. In W. Jonker and M. Petkovic, editors, *Secure Data Management*, volume 3674 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2005.
24. L. Zhang, A. Brodsky, and S. Jajodia. Toward Information Sharing: Benefit And Risk Access Control (BARAC). *Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, 0:45–53, 2006.

A Pessimistic Probabilities

Sections 4.2 and 4.3 make use of pessimistic probabilities, which are derived from a risk significance level n and the parameters of a beta distribution α and

β. In this section, we demonstrate way in which we can define these pessimistic probabilities.

To begin, we must know which side of the unbiased probability corresponds to pessimism and which to optimism, which depends on the semantics of our prediction. For instance, consider that the proposer returns $(\mathbf{allow}, \alpha, \beta)$, giving an unbiased probability of $p = \frac{\alpha}{\alpha+\beta}$ that **allow** is correct. Under these circumstances, an optimistic view point is that the true probability of **allow** is greater than p, and conversely a pessimistic viewpoint is that it is less than p. So here, a pessimistic probability is represented by a probability adjusted downward with some significance level of n, i.e. $\tilde{p}_{\downarrow n}$. Considering a contrary decision, e.g. **deny**, will switch this viewpoint, giving, in our example, a pessimistic probability greater than p, and is represented by a probability adjusted upwards with some significance level n, i.e. $\tilde{p}_{\uparrow n}$ [9]. The other cases are dealt with similarly. The meaning of n depends on how we define the risk.

To construct the risk adjusted probability, we use a method derived from a common risk metric called Expected Shortfall (ES) [7]. In essence, the ES considers average negative impact in the worst $100n\%$ of universes. Therefore, if $n = 1$, the pessimistic probability $\tilde{p}_{\downarrow 1}$ is simply the unbiased probability. To find $\tilde{p}_{\downarrow n}$ for other values of n, we first consider the probability density function for the beta distribution with parameters α and β given by

$$\mathrm{Beta}(\mu|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha-1}(1-\mu)^{\beta-1} \tag{1}$$

where $\Gamma(x)$ is the gamma function [1] and μ represents the possible values for the probability that a local **allow** is correct. The unbiased probability given this distribution is derived from the expected value of μ given α and β, i.e.

$$\mathbf{E}(\mu|\alpha,\beta) = \int_0^1 \mu\mathrm{Beta}(\mu|\alpha,\beta)d\mu = \frac{\alpha}{\alpha+\beta} = p \tag{2}$$

For the downward adjusted probability at significance n, we must evaluated the integral

$$\tilde{p}_{\downarrow n} = \mathbf{ES}_n(\mu|\alpha,\beta) = \int_0^C \mu\mathrm{Beta}(\mu|\alpha,\beta)d\mu \tag{3}$$

where $\mathbf{ES}_n(\mu|\alpha,\beta)$ is the Expected Shortfall value of μ given α, β and a significance level n; and the upper bound of the interval, C, satisfies

$$n = \int_0^C \mathrm{Beta}(\mu|\alpha,\beta)d\mu = \mathrm{I}_C(\alpha,\beta) \tag{4}$$

where $\mathrm{I}_C(\alpha,\beta)$ is the regularised incomplete beta function [16] and can be solved to give,

$$\mathrm{I}_C(\alpha,\beta) = \sum_{j=\alpha}^{\alpha+\beta-1} \frac{(\alpha+\beta+1)!}{j!(\alpha+\beta-1-j)!}C^j(1-C)^{\alpha+\beta-1-j} \tag{5}$$

[9] This switch occurs, because given the proposer returns $(\mathbf{allow}, \alpha, \beta)$ and considering **deny**, means that we would prefer it if the proposer has misclassified the decision.

To find C given our value n, we must invert Equation (4). This does not have a general analytic solution, but can be found numerically for appropriate values of α, β and n.

Equation (3) is simplified using Equations (1) and (5) to

$$\begin{aligned}\tilde{p}_{\downarrow n} &= \int_0^C \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha}(1-\mu)^{\beta-1}d\mu \\ &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(\alpha+1)\Gamma(\beta)}{\Gamma(\alpha+\beta+1)}\int_0^C \mathrm{Beta}(\mu|\alpha+1,\beta)d\mu \\ &= \frac{(\alpha+1)}{(\alpha+\beta+1)}\mathrm{I}_C(\alpha+1,\beta) \qquad (6)\end{aligned}$$

To adjust the probability upwards, i.e. for $\tilde{p}_{\uparrow n}$, the approach is similar, but instead of replacing the upper bound on the integral's interval, we replace the lower bound. So

$$\tilde{p}_{\uparrow n} = \int_D^1 \mu\mathrm{Beta}(\mu|\alpha,\beta)d\mu \qquad (7)$$

where D, satisfies

$$n = 1 - \int_0^D \mathrm{Beta}(\mu|\alpha,\beta)d\mu = 1 - \mathrm{I}_D(\alpha+1,\beta) \qquad (8)$$

Notice that the closer the significance value n gets to 1, the closer the adjusted probabilities are to p, and as $n \to 0$, $\tilde{p}_{\downarrow n} \to 0$ and $\tilde{p}_{\uparrow n} \to 1$. So for us, a small value for n represents a high degree of pessimism, while a high value for n gives an unbiased view. There are a number of ways that pessimistic viewpoints can be incorporated into risk based decision making [7], here we focus on two that use these pessimistic probabilities.