A Generalized Honest-But-Curious Trust Negotiation Strategy for Harvesting Credentials

Lars E. Olson, Michael J. Rosulek, Marianne Winslett
University of Illinois at Urbana-Champaign
{leolson1,rosulek,winslett}@cs.uiuc.edu

Abstract

Need-to-know is a fundamental security concept: a party should not learn information that is irrelevant to its mission. In this paper we show that during a trust negotiation in which parties show their credentials to one another, an adversary Alice can systematically harvest information about all of a victim Bob's credentials that Alice is entitled to see, regardless of their relevance to a negotiation. We prove that it is not possible to enforce need-to-know conditions with the trust negotiation model and protocol developed by Yu, Winslett, and Seamons. We also present examples of similar need-to-know attacks with the trust negotiation approaches proposed by Bonatti and Samarati, and by Winsborough and Li. Finally, we propose possible countermeasures against need-to-know attacks, and discuss their advantages and disadvantages.

1 Introduction

Trust negotiation is an authorization approach intended for use in open systems, i.e., systems where users' identities and access privileges may not be known in advance [2, 3, 4, 6, 14, 18, 19, 20, 22]. With trust negotiation, each resource has an associated authorization policy that describes the properties of the users and software entities ("clients") that are allowed to access it. At run time, a client can learn about the policy protecting the resource it wishes to access, and present verifiable, unforgeable digital credentials to prove that it possesses the properties required by the policy. In this section, we use an extended example to present the basic concepts behind trust negotiation. Then we show how to launch a need-to-know attack against this example, illustrating how an attacker can extract information about any credential that a negotiating party might own.

1.1 Automated Trust Negotiation

Often credentials contain sensitive information, in which case we view them as sensitive resources that must also be protected by policies. For example, consider a client Bob who wants to obtain a driver's license (DL) from the Illinois Department of Motor Vehicles (DMV). To obtain his license, Bob must present either his passport with photo or a current driver's license with photo. In addition, he must present proof of Illinois residency, e.g., via an Illinois address on his current driver's license or else a power bill with a local address. Translating the DMV's policies to work with digital credentials, we have three policies:

$$\begin{aligned}&\mathit{policy\_for\_new\_DL} = \mathit{strong\_ID} \wedge \mathit{IL\_resident}\\&\mathit{strong\_ID} = \mathit{passport} \vee \mathit{DL}\\&\mathit{IL\_resident} = \mathit{IL\_DL} \vee \mathit{IL\_utility\_bill}\end{aligned}$$

Before carrying on with the example, we digress for a moment to present a few technical details. We define the body of each policy by a monotonic Boolean propositional logic formula (meaning, all symbols evaluate to either true or false, and no negations appear), preceded by a propositional symbol (the policy name) and an equals sign. The propositional symbols are typed, i.e., each symbol either denotes a policy or a credential. Each policy name can appear on the left hand side of at most one policy definition. To avoid confusion in our examples, we assume that the sets of credentials owned by Bob and the DMV are disjoint, as are their policy names; thus credential names appearing in the DMV's policies refer to credentials that Bob may possess, and vice versa. Further, we have named the subpolicies so that they are reusable in other contexts. For example, a policy that defines who is allowed to take on a particular role is likely to be widely referred to in an RBAC-based open system.

The DMV's knowledge base (KB) has four parts: its own local policies and local credentials, and the policies and credentials it has received from others. A policy is satisfied if its body evaluates to *true* under the truth assignment where

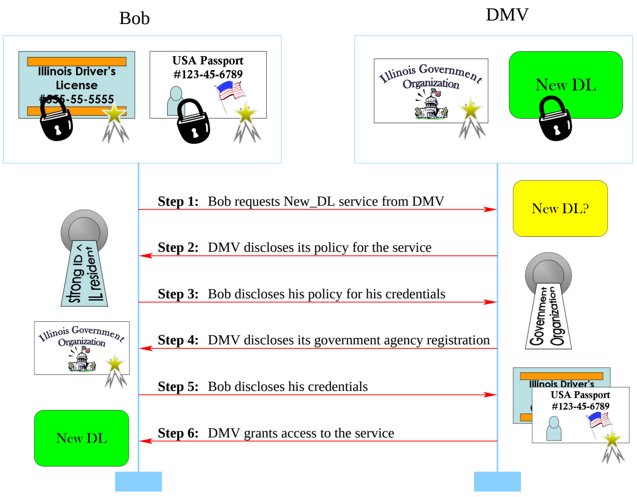
Bob
DMV
Illinois Driver's License #555-55-5555
USA Passport #123-45-6789
Illinois Government Organization
New DL
Step 1: Bob requests New_DL service from DMV
New DL?
Step 2: DMV discloses its policy for the service
strong ID ∧ IL resident
Step 3: Bob discloses his policy for his credentials
Government Organization
Step 4: DMV discloses its government agency registration
Illinois Government Organization
Step 5: Bob discloses his credentials
Illinois Driver's
USA Passport #123-45-6789
Step 6: DMV grants access to the service
New DL

Figure 1. A trust negotiation to renew a driver's license

all credentials in the local KB (i.e., all KB formulas consisting of a single propositional symbol denoting a credential) are assigned *true* and all others are assigned *false*. One can imagine using a more powerful language, or encoding these policies in datalog; the exact choice of encoding or language is not important for our purposes.

When Bob walks into the DMV to renew his license, he is not likely to open his wallet and show the DMV employees every credential that he possesses. Instead, he will study the policies and selectively choose the credentials that he presents to the DMV. He would consider the DMV to be unauthorized to view certain sensitive credentials, such as his credit cards. Other credentials such as his library card might be less sensitive, but still the DMV has no demonstrated "need to know" about them—they are irrelevant for the matter at hand. Further, he is not likely to show both his passport and his driver's license, as this is not a minimal set of credentials that satisfies the DL policy. Instead, he would prefer to show just his current license, which satisfies both the *IL_address* and *strong_ID* policies.

Now consider how to automate this process, so that Bob (actually a small agent acting on his behalf) automatically carries out a trust negotiation with (the agent for) the DMV (see Figure 1). If Bob satisfies the DMV's *policy_for_new_DL*, the DMV should immediately send him a token to access its on-line renewal service. We model this as the DMV having a *new_DL* credential in its KB, which it will show to Bob (send to his KB) once *policy_for_new_DL* is satisfied. We adopt the convention that the policy protecting object *O* will be named *policy_for_O*.

Bob should start the negotiation by explaining what he wants, namely, *new_DL*. The DMV cannot send *new_DL* to Bob yet, because *new_DL*'s protective policy *policy_for_new_DL* is not satisfied. Instead, it responds by disclosing the definition of *policy_for_new_DL*. However, that definition is insufficient for Bob's purposes. Either Bob must now request the definition of one or both subpolicies, or better yet, the DMV can volunteer that information when it discloses *policy_for_new_DL*.

Once Bob has seen *policy_for_new_DL* and at least one subpolicy definition, he can take action. If he does not have either of the credentials mentioned in a subpolicy, he will want to end the negotiation. If he does have those credentials, they are likely to be protected by access policies themselves. For example, perhaps he is willing to show his Illinois license and passport to anyone in the government. In this case his knowledge base (KB) contains the following:

policy_for_IL_DL = government
policy_for_passport = government
passport, IL_DL, library_card, patient_ID

The last line of propositions indicates that Bob possesses digital credentials representing a local license, a passport, and other credentials; he does not have a power bill in his own name. In addition, Bob's KB contains the policy definitions that the DMV just sent him.

An omniscient entity can see that Bob can satisfy *policy_for_new_DL*. Some approaches to trust negotiation allow the negotiation to proceed without either party actually showing the other its policies and/or credentials, through various clever tricks [8, 12, 15, 17]. Such approaches are appropriate when privacy and leak prevention are paramount considerations; otherwise they are rather too expensive at run time. In this paper we will consider scenarios where Bob and the DMV are willing to divulge their credentials and policies to one another (provided the appropriate access policy is satisfied). For example, Bob may send the DMV his *IL_DL* in the form of an X.509 credential, with a proof that he owns it. We will not model the details of credential formatting or proof of ownership in this paper, as they do not affect the problems that concern us.

In real life, Bob would know that he was standing in a government office; online, it is often not so obvious who one is talking to. Bob can continue the negotiation by sending the DMV his policy for his Illinois license (assuming he would rather not disclose his passport). Assume that the DMV does have a *government* credential and is willing to show it to anyone. Then the DMV can show Bob its *government* credential and prove that it owns it, which we model as having that credential appear in Bob's KB. Now Bob's Illinois license policy is satisfied, and he can show his license to the DMV, which we model as having the *IL_DL* proposition appear in the DMV's KB.

1.2 Sensitive Policies

In this particular example, neither Bob nor the DMV has any sensitive policies, i.e., policies that must themselves be protected by policies. We can represent this by adding the formula *policy_for_P = true* to the KB, for each policy P its owner possesses. Often corporate policies are sensitive, however, and personal policies can also give away undesirable hints about the resources they protect. As a real-world yet whimsical example, the Illinois DMV used to secretly sell truck driver licenses to anyone willing to pay $10,000. (This policy was uncovered after many of the licensees had accidents.) We can represent this policy by changing the definition of *policy_for_new_DL* to include a new disjunct, *bribed.* However, clearly this definition of *policy_for_new_DL* should never be shown to anyone. We can enforce this by encoding the DMV's KB as follows.

policy_for_new_DL = (*strong_ID* $\wedge$ *IL_resident*) $\vee$ *other*
other = *bribed*
policy_for_other = *false*
strong_ID = *passport* $\vee$ *DL*
IL_resident = *IL_DL* $\vee$ *IL_utility_bill*
government, new_DL

(For brevity, we will not list any KB formula of the form *policy_for_P = true*.) During a negotiation, anyone can see that there is an *other* clause in *policy_for_new_DL*, but no one can learn its definition, because the policy for disclosing the definition is always false. Someone who has learned offline about the bribery policy can push their bribe (modeled here as a *bribe* credential) to the DMV, thereby satisfying *policy_for_new_DL*.

The presence of protected policies should not be construed as an attempt to hide an unethical or illegal access policy. Any organization can use the same technique to offer special services to selected clients. For example, Sam's Club can e-mail an offer to all university employees, offering them free membership if they show their university ID card. Bob can click through the e-mail to automatically store the associated policy in his KB. Or, a policy may leak private information. For instance, suppose Bob had a resource *blood_test_results* that he will disclose to holders of an *AIDS_researcher* credential. Disclosing such a policy freely would allow anyone to infer Bob's medical condition without actually possessing the proper credential. Bob may wish to disclose such a policy only to holders of a *doctor* credential, for example.

1.3 Motivation for Need-to-Know Attack

We are now ready to discuss the need-to-know attack on trust negotiation. Suppose that the DMV wishes to collect personal information on its clients and then sell the information to marketers. The DMV can force Bob to disclose credentials during negotiation that he would never disclose in person. To automate this attack, the DMV first rewrites its policies as follows:

original_policy = (*strong_ID* $\wedge$ *IL_resident*)
policy_for_new_DL = *original_policy* $\vee$*attack*$_{\text{library}}$
attack$_{\text{library}}$ = (*original_policy*$_1$$\wedge$ *library_card*) $\vee$*attack*$_{\text{utility}}$
attack$_{\text{utility}}$ = (*original_policy*$_2$$\wedge$ *IL_utility_bill*) $\vee$*attack*$_{\text{patient}}$
attack$_{\text{patient}}$ = (*original_policy*$_3$$\wedge$ *patient_ID*)
original_policy$_1$ = *original_policy*
original_policy$_2$ = *original_policy*
original_policy$_3$ = *original_policy*
strong_ID = *passport* $\vee$ *DL*
IL_resident = *IL_DL* $\vee$ *IL_utility_bill*

While we have chosen policy names that are suggestive of their content, of course the DMV would use opaque names such as P_{753}. The rewritten policies are logically equivalent to the originals, and in fact, any client who could obtain a renewal token under the old policies will be able to obtain a token under the new policies.

When Bob asks for *new_DL*, i.e., he requests access to a token for online license renewal, the DMV sends him *policy_for_new_DL* = *original_policy* $\vee$ *attack*$_{\text{library}}$ and *attack*$_{\text{library}}$ = (*original_policy*$_1$$\wedge$ *library_card*) $\vee$ *attack*$_{\text{utility}}$.

When Bob examines these two formulas, his exact reaction will depend on the trust negotiation protocol[1] he is using. In general, the only way he can keep the negotiation moving forward is to send *library_card*, or, if his trust negotiation protocol allows it, query for the definition of some of the undefined policies. Most trust negotiation protocols would forbid him to send *policy_for_library_card = true*, as that response does not advance the state of the negotiation and is due cause for the DMV to terminate the negotiation[2]. Most protocols would also forbid him to send information unrelated to the negotiation so far, such as his patient ID; that behavior would also be due cause for termination.

Suppose that Bob sends his library card to the DMV. With that tidbit in hand, the DMV moves on to extracting utility bill information from Bob. The DMV

[1]As defined in [22], a trust negotiation protocol defines the types of messages that can be sent during a negotiation. Here we use the term in a broader sense, to also indicate any limits on the contents of a message that Bob can send. We formalize the concept later.

[2]One might argue that the DMV will learn something, so in some sense the negotiation will advance. By that reasoning, however, Bob could maliciously send an unending stream of irrelevant policy definitions to keep the DMV busy, and that would also be considered to advance the negotiation. To keep the negotiation focused, it is better to define "advancing the negotiation" as adding new detail to a partially-formed proof of authorization. We formalize this later.

sends $attack_{\text{utility}}$ = ($original\_policy_2 \wedge IL\_utility\_bill$) $\vee$ $attack_{\text{patient}}$. If his trust negotiation protocol allows it, Bob can confess that he does not have a utility bill; in that case, the DMV will move on to disclosing $attack_{\text{patient\_ID}}$ and gathering information about that aspect of his identity. Once that is complete, the DMV has finished gathering marketable information and will disclose *original_policy* = (*strong_ID* $\wedge$ *IL_resident*), *strong_ID* = *passport* $\vee$ *DL*, and *IL_resident* = *IL_DL* $\vee$ *IL_utility_bill*. The remainder of the negotiation proceeds as normal.

If Bob's trust negotiation protocol does not allow him to declare that he does not have an Illinois utility bill, or if he is sensitive about the fact that he does not possess one, he should keep the formula *policy_for_IL_utility_bill* = *false* in his KB. Then when asked for his utility bill, he can send this policy definition in response. In this case, the DMV will know that it can never learn whether Bob has an Illinois utility bill, and move on to collecting the next piece of marketable information. This does not mean that the DMV's attack on need-to-know has failed. The attack is not intended to breach confidentiality; it just makes a mockery of the need-to-know principle by gathering as much irrelevant information as it can legally have. As such, it does constitute an honest-but-curious attack. The attacker does not have to break the rules of the ATN framework, just rewrite her policies and follow an allowed (if disingenuous) negotiation strategy.

As mentioned above, if his protocol allows it, at any point Bob can query for the definition of one of the undefined policies mentioned by the DMV. For example, in the first round of the negotiation, he could request the definitions of *original_policy* and/or $attack_{\text{utility}}$. In this case, the DMV should send the definitions if the policies protecting them (*true* in this case) are satisfied. The DMV can prepare for this possibility by modifying its policies for disclosing *original_policy* and $attack_{\text{utility}}$ as follows:

policy_for_original_policy = *true*,
  policy_for_policy_for_original_policy = *false*
$policy\_for\_attack_{utility}$ = *true*,
  $policy\_for\_policy\_for\_attack_{utility}$ = *false*

(And similarly for $original\_policy_1$, $original\_policy_2$, $original\_policy_3$, $attack_{\text{patient}}$, *strong_ID*, and *IL_resident*.) These new formulas say that the definitions of *original_policy* and $attack_{\text{patient}}$ can be disclosed to anyone, but no one can learn that the formulas can be disclosed to anyone. Thus when Bob asks for the definition of *original_policy*, the DMV can send him *policy_for_policy_for_original_policy* = *false*.[3] Then if Bob wants to renew his driver's license online, he has no choice but to disclose the marketing-type information that the DMV has requested. As discussed in more detail later, under some circumstances a client could recognize that *original_policy* might be just *true*, but usually the client cannot detect this type of attack.

We could prevent a need-to-know attack by requiring each party to push the definition of every policy P it mentions during the negotiation, whenever *policy_for_*P is satisfied. If *policy_for_*P is not satisfied, it should instead push the definition of *policy_for_*P, unless *policy_for_policy_for_*P is not satisfied; and so on. While satisfactory in theory, this requirement is unappealing in practice. If 99% of clients gain access to a resource through a particular combination of credentials, the resource owner will be understandably reluctant to push every client the full details of the other ways to gain access. 99% of the time, sending those policies will just waste bandwidth and processing power for both client and resource owner. Similarly, one could require a resource owner to respond to a request for the definition of policy P by immediately sending P, if the policy P' that protects P is satisfied; else P', if the policy P'' that protects P' is satisfied; and so on. This suffers from the same efficiency drawbacks.

The DMV's attack will work against any client, not just Bob. However, attackers will generally require more complex encodings of their policies, for the following reason. The need-to-know attack works by ensuring that the victim has only one possible way to advance the state of the negotiation, namely, by disclosing a piece of information that the adversary wants. When the client's and/or server's policies are significantly more complex than Bob's and the DMV's, the server might lose this control while working to satisfy a policy of the client's (e.g., a policy for *library_card*). We present this more complex encoding in Section 3 and prove that it succeeds against every client in Section 4.

Suppose now that Bob is the attacker and the DMV is his unsuspecting target. Bob can encode the policy for his Illinois license in the manner that we encoded the DMV's policies earlier. With the revised encoding, Bob can find out whether the DMV owns any particular credential c, within the limits of confidentiality. Bob and the DMV can also both attack one another simultaneously, leading to a long negotiation in which eventually one party is backed into a corner and must start disclosing information about the requested credentials. The negotiation will be finite, however, as long as we ensure that client and server do not alter their policies on the fly. This can be accomplished by sending a commitment of their policy contents at the beginning of the

[3] There may be legitimate reasons for allowing policies to be *false*, but even if we disallowed such policies, this tactic could still easily be used without a simple means of detection. For instance, the policy could require some very rare credential, or even a unique credential which only belongs to the policy writer (thus guaranteeing that no one else will ever be able to satisfy the policy).

negotiation.

We formalize the need-to-know attack in Sections 2 and 3, and prove its effectiveness within the protocol described in [21] in Section 4. The reader who is willing to take those aspects on faith may wish to skip directly to Section 5, where we describe how to carry out need-to-know attacks on other trust negotiation models. In Section 6, we discuss possible remedies for the problem and conclude the paper.

2 Definitions

In this section, we give the necessary preliminaries for the need-to-know attack. We start by defining the attack formally, independently of the actual protocol being used. We then present definitions for an example model and protocol for trust negotiation, following the lead of [21]. We will use this model to demonstrate how to carry out the attack against arbitrary clients in Section 3.

2.1 General Model for Attack

For uniformity, we will use the term *resource* to refer to anything that can be protected by a policy, such as access to an online service (e.g. *new_DL*), another policy (e.g. *policy_for_new_DL*), or a credential (e.g. *IL_utility_bill*).

In order to characterize this attack, we will examine the credentials that the victim discloses to gain access to a service. Some of these credentials may be disclosed unnecessarily (conceptually, the client could have gained access without disclosing them). The following two definitions formalize which credentials are necessary.

Definition 1. *A* minimal local satisfying set *for resource R is a set S of credentials and negations of credentials with the following properties:*

- *S satisfies the policy for R, i.e., R's policy evaluates to true under the truth assignment that assigns true to each credential in S, false to each credential whose negation is in S, and false to all remaining credentials.*
- *No proper subset of S has this property.*

Clearly no credential negations will appear in minimal local satisfying sets. We include them here for technical reasons.

Definition 2. *A* minimal (global) satisfying set *for resource R is a set S of credentials and negations of credentials constructed recursively as follows:*

- *S contains a minimal local satisfying set for R.*
- *If S contains a resource protected by policy P, then S also contains a minimal local satisfying set for P.*

There are many different approaches to trust negotiation. As a generalization, though, all of them require each side to exchange credentials and policies. The following definition formalizes this.

Definition 3. *A* disclosure (message) *sent from one party to another during negotiation is a set of local credentials, local policies, and any other message elements allowed by the particular negotiation protocol in use. In particular, part of Bob's disclosure may be to indicate that he does not have a certain credential.*

Protocols can define message elements with special meanings. For example, a protocol might allow a party to declare that it would like to access resource *new_DL*, that it does not possess a particular credential *IL_utility_bill*, or to request the definition of a previously mentioned policy symbol *attack_utility*. Because it is hard to prove any properties of a protocol that permits arbitrary message elements, we will assume that the protocol uses at most the latter two types of special message elements. If a party requests the definition of a policy symbol, we assume that the definition must be provided in the next round of the negotiation (or else the definition of the policy for that definition, or of the policy for the policy for that definition, and so on to guarantee safety). We view the initial request to access a resource (e.g. to access *new_DL*) as an event that precedes and triggers the negotiation. The client can choose to push some disclosures with its request for service (e.g., *bribe*); otherwise, the first disclosures will be from the server, typically including the policy for the requested service.

Looking back at the DMV example, one can see that a need-to-know attack might not work twice on the same client. Bob could cache the DMV policies, study them before the next time he renews his license, and then push his driver's license to the DMV along with his request for renewal. The DMV would have no excuse to ask him about his other credentials before giving him the online renewal token. But since the DMV has already learned what it wanted to know about Bob, it would appreciate such a speedup in future negotiations, so this does not represent a weakness in the need-to-know attack.

With these definitions, we can now formally define the attack.

Definition 4. *Alice, the owner of a resource R, executes a* need-to-know attack *on credential c when the following two conditions hold:*

- *Whenever any client Bob successfully negotiates access to R without any prior knowledge of the policy protecting R, Alice learns all that she is authorized to know about c (either Bob discloses c, Bob reveals that he does not have c, or Alice learns that she is not authorized to access c).*

- *The set of credentials that Bob discloses throughout the negotiation is not minimal. That is, we can replace Alice's policies by a logically equivalent set of policies with the property that Bob can successfully negotiate for access to R using the same set of disclosed credentials excluding c, without any prior knowledge of the policy protecting R and without disclosing any information about c.*

With a need-to-know attack, no matter how a client gains access to R, he gives out more information than necessary during the process—his disclosures are not minimal. It is important to note that this definition does not simply mean that a client *can* disclose more than is necessary; it means that a client *must* disclose more than is necessary in order to discover the actual contents of the policy. In the DMV example from Section 1, for instance, a client possessing all of the credentials *passport*, *IL_DL*, and *IL_utility_bill* might choose to disclose all three, even though only two are necessary. In this case, a more privacy-conscious client with access to the policy can calculate a minimal disclosure set. The need-to-know attack occurs when the client is tricked into thinking it needs to disclose an unnecessary credential like *library_card*. It is irrelevant to consider that a client could discover the true policy and bypass the attack in future negotiations—the attacker only needs one negotiation to obtain the irrelevant credentials.

2.2 Example Model

The remainder of this section formalizes basic negotiation concepts for the trust negotiation model we use, in the style of [22]. These concepts will be needed for the proofs in the sections that follow.

Definition 5. *The disclosure of a resource R by its owner, or the admission that its owner does not possess R, is* safe *if R's policy is satisfied by credentials present in its owner's KB at the time of the disclosure.*

For simplicity in the definitions that follow, we do not consider the case where one party receives a policy definition or a credential from the other party offline.

Definition 6. *A* negotiation *is a sequence of safe disclosures that is triggered by a request to access a resource R. Further, each disclosure message must advance the state of the negotiation.*

To give a precise definition of what it means to advance the state of the negotiation, we rely on proof trees to represent the state of the negotiation. Intuitively, each proof tree describes a different way that a client could get access to R. The children of a node describe one way to get access to the resource represented by that node, e.g., one combination of credentials that will satisfy a policy. If a policy has not had its definition disclosed, but a policy for that policy has been disclosed, we represent that as a dotted-line child of the node for the policy.

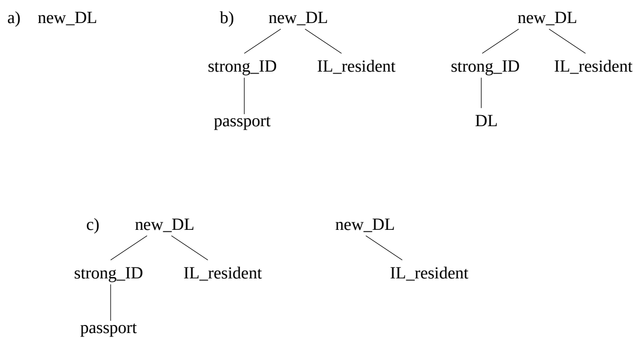

Figure 2. Example proof tree evolution

Definition 7. *An* unpruned proof tree *for the set D containing all disclosures so far in a negotiation for a resource R can be constructed as follows:*

- *The root of the tree must be a node labeled R.*
- *If P is the name of the policy that protects credential c, then every tree node labeled c has a child labeled P.*
- *If P' is the name of the policy that protects policy P, then every tree node labeled P has a single dotted-line child labeled P'. Unless otherwise stated, the term child refers only to non-dotted-line children.*
- *If the definition of policy P appears in D, then every node labeled P has the following children.*
  - *If P = false, or its logical equivalent, then the node has a single child labeled false.*
  - *Otherwise the node has children labeled c_1 through c_n, where $\{c_1, \ldots, c_n\}$ is a minimal local satisfying set for P.*

For example, *new_DL* is the root of each unpruned proof tree for Bob's negotiation with the DMV. Bob can only get access to *new_DL* (the root of each proof tree for his negotiation) by satisfying policies *strong_ID* and *IL_resident*, which are the children of the root in every unpruned proof tree for his negotiation. Below this point, the unpruned proof trees begin to differ from one another. In some, *passport* will be the sole child of *strong_ID*; in others, *DL* will be the only child of *strong_ID*. In some of the trees, *IL_DL* will be the only child of *IL_resident*; in other trees, *IL_utility_bill* will be the only child. Below this level of the trees, Bob's own disclosures determine how the trees will grow during the next round of the negotiation.

Consider Figure 2. When Bob initially requests access to the *new_DL* service, the proof tree is initialized as shown in Figure 2a. Assume, for this example, that the DMV then discloses the policy for *new_DL* (*strong_ID* $\wedge$ *IL_resident*), and also the contents of the *strong_ID* policy (*passport* $\vee$ *DL*), but not the contents of the *IL_resident* policy. Because there are two disjuncts in the *strong_ID* policy, this node is replaced by two possible children, and therefore there are two proof trees at this point as shown in Figure 2b. Assume now that the client discloses its *DL* credential, removing it from the proof trees. This results in the set of proof trees shown in Figure 2c.

Our definition of *unpruned proof tree* assumes that every resource is protected by a named policy. In the sections that follow, we will not name every policy; the effect on the trees is trivial.

An unpruned tree describes one potential way to gain access to R. However, not every potential proof can succeed. For example, Bob does not have an Illinois utility bill, so he will never renew his license by presenting his utility bill. If he discloses this fact, we can eliminate all proof trees that involve a utility bill. If he discloses his passport, we can represent this accomplishment by pruning the portion of the tree that mentions his passport. After pruning, a proof tree represents the remaining steps to take in one way to gain access to R.

Definition 8. *Given an unpruned proof tree T for a set D containing all disclosures so far in a negotiation for a resource R, the* (pruned) proof tree *T' corresponding to T is constructed as follows:*

- *Remove the tree entirely if any node is labeled c, where D contains the disclosure that a party does not possess c (written $\neg c$).*
- *For each credential c that is an element of D, remove all subtrees of T rooted at nodes labeled c.*
- *For each policy P whose definition is present in D and that is satisfied by disclosures in D, remove all subtrees rooted at nodes labeled P.*
- *If P' is the policy protecting a policy P, and P's definition is in D, then remove every subtree rooted at a node labeled P'.*
- *If a node P has the child false, remove the subtree rooted at the nearest ancestor of P that is a dotted-line child. If there is no such ancestor, then remove the entire proof tree.*

If there are no (pruned) proof trees at all, then the client cannot possibly gain access to the desired resource. If there is an empty (pruned) proof tree, then the client has just gained access to the desired resource.

To advance the state of the negotiation, a party must disclose information that changes the set of proof trees and also makes it possible for the *other* party to advance the state of the negotiation. For example, it is not sufficient for the DMV just to disclose the definition of *policy_for_new_DL*, as Bob cannot change the resulting set of proof trees: he does not know the definition of *IL_resident* or *strong_ID*. The DMV must also disclose the definition of at least one of these two policies.

Definition 9. *A disclosure message D* evolves *the proof trees if the set of proof trees immediately before D is disclosed is different from the set immediately after D is disclosed.*

Definition 10. *A set of proof trees is evolvable for a party Alice if at least one proof tree contains a leaf labeled l, which we call an* evolvable leaf*, that satisfies one of the following two properties:*

- *l is a credential possibly belonging to Alice.*
- *l is a policy of Alice's, and she has not yet disclosed its definition.*

Definition 11. *A proof tree is partially evolved if it contains an evolvable leaf for at least one party. A fully evolved tree has no evolvable leaves for either party.*

Definition 12. *A disclosure D* advances the negotiation *if one of the following three conditions is met:*

- *D evolves the proof trees and the resulting set of proof trees is evolvable for the other party.*
- *D requests the definition of a policy symbol that appears in a previous disclosure message and whose definition has not yet been disclosed.*
- *D includes the definition of a policy symbol whose definition was requested in the immediately preceding disclosure message, or the definition of the policy protecting that policy, and so on (whether or not that definition has already been disclosed). This policy definition must contain at least one non-policy symbol, to prevent unuseful disclosures like "Policy$_1$ $=$ Policy$_2$ $\wedge$ Policy$_3$."*

By contrast, we say that D is unmotivated *if it does not advance the negotiation.*

Here we skip over a fine point: Alice could request the definition of a policy symbol P, be given the definition of *policy_for_P* in the next message, and then immediately request the definition of P again in the next message. We assume that protocol includes provisions to prevent such loops, e.g., by requiring that Alice not ask again until some other state change has occurred. The details of those provisions do not matter for our purposes.

We make the following assumptions about the negotiation process:

1. Policies cannot be changed in the middle of a negotiation. This can be guaranteed by a cryptographic signature over the policies, exchanged at the beginning of the negotiation.[4]

2. Disclosures are always safe.

3. Disclosures must advance the negotiation in a way that is publicly verifiable. This assumption prevents attackers from making useless disclosures that will force arbitrary clients, who will typically not have information that is either secret or exchanged offline, to guess at how to satisfy the resource's policy.

4. A negotiation ends immediately if one party does not disclose anything that will advance the negotiation.

5. A negotiation ends immediately and access to the target resource is granted if the policy protecting the target resource has been visibly satisfied, meaning the other party knows the policy and can prove that enough credentials have been disclosed to satisfy it. Without this assumption, an attacker could drag on a negotiation long after the client has shown that it is eligible to access its desired resource.[5]

3 The Attacks

We now describe how to execute the actual need-to-know attacks, given an arbitrary set of policies and credentials. We use the following conventions:

Alice	The attacker (the DMV in the earlier example)
Bob	Alice's victim
R	Alice's resource that Bob wants to access
α	Name of the policy originally protecting R (*original_policy* in the DMV example)
α'	Name of the actual policy protecting R, rewritten for attack
a_i	A credential that Alice may own
$b_1, \ldots, b_n$	Credentials Bob may own that Alice would like to learn about
$b_{n+1}, \ldots, b_m$	Credentials Bob may own that do not interest Alice, $m \geq n$
ϕ_i	Equivalent of the *attack*$_i$ policies in the DMV example
Q_i	Policies Alice usees to prevent Bob from having more than one possible disclosure. Not needed in the DMV example.

In the following section we show how to attack a party whose policies are freely disclosable (i.e., the policy that protects their policies is *true*), in the case where the negotiation protocol does not allow one to request the definition of an undefined policy symbol. While this is perhaps an unrealistic scenario, it is the simplest case to understand. We relax these assumptions in sections 3.2 and 3.3.

3.1 Freely Disclosable Policies, No Requests for Policy Definitions

The attacker, Alice, has a resource R protected by a policy α. Alice is also interested in obtaining credentials $b_1, \ldots, b_n$ that are not mentioned in α. All the other credentials Bob could possibly own are $b_{n+1}, \ldots, b_m$; these may occur in α or may be uninteresting credentials from Alice's perspective.

Alice prepares for the attack by rewriting her policy to be $\alpha' = \alpha \vee \phi_1 \vee \nu$, where ϕ_1 comes from the following definition of ϕ_i:

- $\phi_i = (P_i \wedge b_i) \vee \phi_{i+1}$ for $i < n$
- $\phi_n = (P_n \wedge b_n)$
- each $P_i = \alpha$

and ν is defined as follows:

- $\nu = \bigvee_{i=1}^{m} (Q_0 \wedge Q_i)$
- $Q_0 = \alpha$
- $Q_i = b_i$, for $1 \leq i \leq m$

Notice that ν also covers Bob's other credentials that Alice doesn't necessarily want. Figure 3 shows an example where Alice wants credentials b_1, b_2, and b_3, and Bob could possibly also have credential b_4. The policies for ϕ_2 and ϕ_3

[4] There may be legitimate reasons why a policy may change based on information discovered during a negotiation; however, if we allow changes to take place, then a need-to-know attack is quite simple. For every disclosure, rewrite it to contain a disjunct with a single protected policy. On the next turn, this protected policy can be rewritten to request any arbitrary credential not yet received. Our work shows that a need-to-know attack still exists under more strict conditions.

[5] It could legitimately be argued that when a policy is satisfied, the resource *must* be disclosed, regardless of whether it is visibly satisfied. Indeed, one of our proposed solutions to this problem discusses how to enforce such a rule without violating the privacy of either party. However, if we apply to all resources in general (including credentials), an agent is likely to satisfy a lot of policies during the course of the negotiation, Such a rule would impinge on the other agent's autonomy in determining what to disclose, effectually resulting in a need-to-know attack in the other direction: a client could send all of its credentials and then wait to receive a list of resources for which it has satisfied the policies.

$$R \leftarrow \alpha \vee \phi_1 \vee \nu$$

$$\phi_1:\quad (P_1 \wedge b_1) \vee \overbrace{((P_2 \wedge b_2) \vee \overbrace{((P_3 \wedge b_3)}^{\phi_3}))}^{\phi_2})$$

$$\nu:\quad (Q_0 \wedge Q_1) \vee (Q_0 \wedge Q_2) \vee (Q_0 \wedge Q_3) \vee (Q_0 \wedge Q_4)$$

Figure 3. Example attack against freely disclosable policies

are shown expanded in this figure; of course, Alice will not immediately reveal the contents of these named policies.

Lemma 1. *The rewritten policy α' is logically equivalent to α. Thus, rewriting the policy will not allow unauthorized access, nor will it disallow authorized access.*

Proof. Straightforward case analysis. □

Alice also needs to rewrite the policies for each of her credentials in such a way that only one credential at a time is requested. She can do this by replacing each credential in the policy with a unique named policy containing that credential, and revealing the contents of each named policy one at a time (a step to which we will refer later). For instance, if the policy for a_1 is $b_1 \wedge (b_2 \vee b_3)$, Alice rewrites it to the following (for some unused symbols Π_1, Π_2, and Π_3):

*policy_for_*$a_1 = \Pi_1 \wedge (\Pi_2 \vee \Pi_3)$
$\Pi_1 = b_1$
$\Pi_2 = b_2$
$\Pi_3 = b_3$

Alice commits to the rewritten policies, rather than the original policies, and can compute a cryptographic commitment on them if requested. We are now ready to define Alice's strategy.

Alice's strategy: When Bob asks for access to R, Alice discloses her rewritten policy, as well as the contents of ϕ_1 and ν. Thereafter, on each of her turns to make a disclosure, she uses the following strategy:

1. If Bob has one or more evolvable leaves in the current proof tree, reveal the definition of a policy $Q_i = b_i$, where Alice already knows all that she is authorized to know about b_i, or where the policy for b_i is satisfied. Since such a disclosure technically evolves the proof tree, but practically gives Bob nothing new to work with, we will hereafter call these Q_j policies *stalling policies*.

2. Otherwise, perform the first of these disclosures which is possible:
   - Disclose a minimum satisfying set for one of Bob's credentials.
   - Reveal pieces of relevant policies for one of Alice's resources (not including the Q_j's and ϕ_j's) until Bob has an evolvable leaf.
   - Reveal as many of the ϕ_j policies as needed to give Bob an evolvable leaf.

3. When Alice can make no more such disclosures, she has finished her need-to-know attack. She can disclose the definition of α and carry out an ordinary trust negotiation with Bob.

3.2 Freely Disclosable Policies, Requests for Policy Definitions

We now modify the protocol to allow Bob to request that Alice disclose the definition of a policy symbol that has been mentioned but not yet defined. We require Alice to answer this request immediately by either revealing the contents, or showing the access policy for the named policy (or for its policy, and so on).

Unfortunately, Alice can still execute the attack by adding unsatisfiable policies to protect the policies that were created as described in Section 3.1. She cannot use the unsatisfiable policies to protect them directly, because they would prevent her from ever revealing (for example) α or any of the stalling policies to anyone. Instead, we take an approach shown in Figure 4. Each named policy (including those protecting Alice's credentials, which are not shown) is protected by another named policy τ_i. Each τ_i contains the policy *true*, but is in turn protected by the policy *false*. In other words, Alice is free to disclose each original policy, but she is not free to disclose the fact that she is free to disclose them. Because each τ_i is a different named policy, revealing one of the protected named policies yields no new information about any of the other τ_i policies, and thus Bob gets no benefit from this new protocol.

3.3 Protected Policies

A natural question that follows from these attack examples is whether the attack arises due to the mismatch in expressiveness between the attacker's protected policies and the victim's unprotected policies, or whether it is more fundamental to trust negotiation. Consider the policies under ν. Disclosing each subpolicy Q_i effectively allows the attacker to stall the negotiation, since although by strict definition they do advance the negotiation, they do not request anything new or allow the victim any new evolvable leaves.

Nothing precludes either party from duplicating these stalling policies, and there is no limit to how many policies

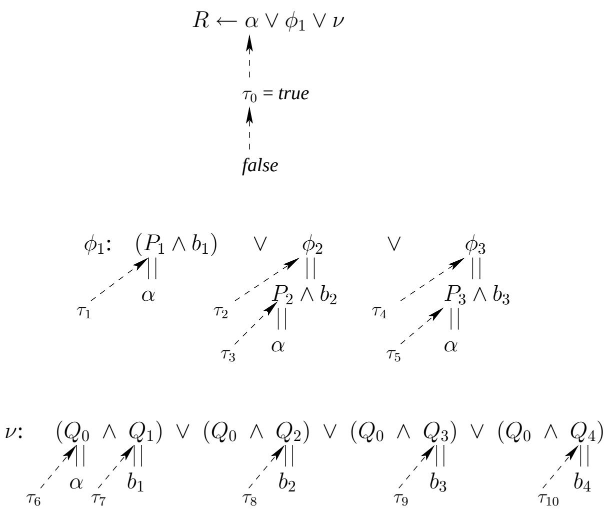
Figure 4. Example attack under the modified protocol

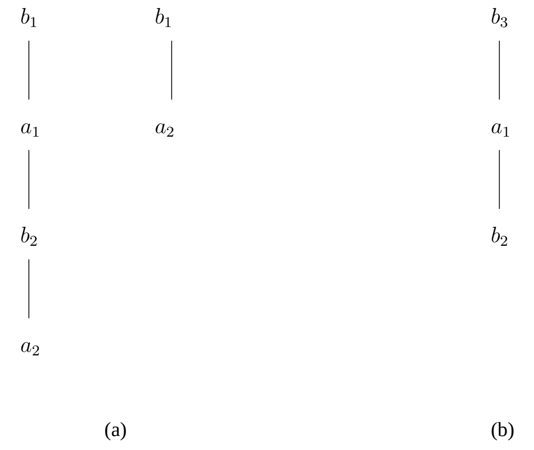
Figure 5. (a) Example proof tree sections leading to reinstated nodes, (b) Example proof tree section containing reinstated nodes

they may have, other than the physical memory of the machine and the bandwidth of the connection. Thus, against a victim with protected policies, a well-prepared attacker can add enough stalling policies to force the victim to exhaust all evolvable leaves until he is forced to start disclosing his policies, and the attacker can proceed as described in Section 3.2. In the worst case, against a victim who also has stalling policies, it becomes an arms race where the party with the most stalling policies can eventually attack the other party.

4 Proofs

In this section, we prove that Alice's strategy does indeed yield a need-to-know attack on all of the credentials $b_1, \ldots, b_n$. For brevity, we refer to pruned proof trees as simply "proof trees" unless otherwise indicated.

Observe that during a negotiation, a party may evolve a leaf by disclosing its policy. However, not all of the resources mentioned in that policy may appear as leaves in the resulting proof tree. Indeed, some of the resources mentioned in the policy may have been mentioned before, and appear with a subtree below them. Some of these subtrees may have been irrelevant and are now once again relevant. For instance, consider Figure 5a. This represents a section of a possible proof tree where the policy for b_1 is $a_1 \vee a_2$, the policy for a_1 is b_2, the policy for b_2 is a_2, and a_2 is freely disclosable. If Alice discloses a_2, she satisfies both the policy for b_2 and the policy for b_1. If Bob chooses to disclose b_1, then the subtree rooted at a_1 is no longer relevant. If, however, Bob introduces another credential b_3 with a policy a_1, a_1 is not an evolvable leaf for Alice because she has already disclosed its policy, but is still waiting to receive b_2.

Definition 13. *A* reinstated *node in a proof tree is one that appears due to one of its ancestors being mentioned in the previous disclosure.*

For instance, both a_1 and b_2 in Figure 5b are reinstated nodes, since a_1 was mentioned in a previous disclosure.

Lemma 2. *During the need-to-know portion of Alice's strategy, any evolvable leaf owned by Bob that is reinstated by some policy disclosure must itself have a visibly satisfied policy.*

For instance, this lemma guarantees that the policies protecting credentials such as b_2 in Figure 5b have already been satisfied. We will use this lemma in the following main theorem, and prove it later, being careful to avoid circular logic.

Theorem 1. *After each of Alice's disclosures (described in steps 1 and 2 of the strategy), Bob either has a single evolvable leaf representing a credential whose policy is not yet satisfied, or he has several evolvable leaves representing credentials whose policies are each satisfied.*

Proof. We proceed by induction on the number of disclosures. After Alice's first disclosure, Bob's only evolvable leaf is b_1, so the claim holds in the base case.

Now suppose Bob makes a disclosure, after which he still has an evolvable leaf. Then there are two cases:

- The leaf was already evolvable before he made the disclosure. Then by the inductive hypothesis, *all* of his evolvable leaves from the beginning of his disclosure had satisfied policies.

- The leaf was reinstated when Bob revealed a policy. By Lemma 2, this leaf's policy is satisfied.

In either case, if there are evolvable leaves for Bob when Alice's turn to make a disclosure starts, then they all have satisfied policies. Now, if Alice performs a stalling disclosure $Q_j = b_j$, there are several possibilities for her choice of b_j:

- b_j has been disclosed (or Bob does not have it). In this case, b_j will be pruned from the resulting proof tree (or the entire tree mentioning b_j will be deleted in the pruning process). No new evolvable leaves are added.

- b_j has a satisfied policy. But then b_j must be one of the existing leaves, not a new one.

In both cases, the invariant is preserved. It suffices to show that Alice has a valid $Q_j = b_j$ stalling disclosure available to her. But note that each time she must make a stalling disclosure, Bob's evolvable leaves all have satisfied policies. Each time a policy for one of Bob's credentials b_j becomes satisfied, Q_j becomes a valid stalling disclosure for Alice, and remains valid until Alice uses it. Thus there are at least as many unused Q_j policies as there are satisfied leaves, so there will always be a valid choice of Q_j for Alice in this step.

Now we consider the case where Bob has no remaining evolvable leaves after his turn to make a disclosure. We consider one of the three possible kinds of disclosures that Alice's strategy prescribes:

- She may disclose a minimum satisfying set for one of Bob's resources. If one of Bob's resources becomes a leaf in this step, it must be because Alice's disclosure satisfied its policy, so the invariant holds.

- If she discloses one of her own policies, adding a node to the tree representing a credential b_j (owned by Bob), two things can happen: If b_j was never mentioned before, then it is the only leaf added to the pruned proof trees. Otherwise, there is a tree of reinstated nodes below b_j. By Lemma 2, all the reinstated leaves (if any) must have satisfied policies. (Also, disclosing a policy might yield no evolvable leaves for Bob, if for instance it mentions a credential that Bob has already disclosed. In this case, Alice reveals as many policies as necessary until Bob has an evolvable leaf.)

- Disclosing the contents of one of the ϕ_i policies that has not yet been disclosed adds at most one evolvable leaf for Bob.

  Another important observation for this type of disclosure is the following: Suppose the first two types of disclosures are not possible. One possibility is that each proof tree containing another ϕ_i has been removed entirely in the pruning process (by becoming unsatisfiable), or its ϕ_i has been pruned away (by b_i being disclosed). Otherwise, the contents all of the policies in the pruned proof trees (apart from the ϕ_i's and Q_i's) are fully disclosed, and Alice cannot disclose a minimum satisfying set for any of Bob's credentials. If her credentials cannot satisfy any policy in any tree, then she can never make progress on any of these trees and can conclude that she will never satisfy any of Bob's policies mentioned within those trees.

In all cases, the invariant holds. □

Corollary 1. *For any agent with freely disclosable policies, Alice's strategy is a need-to-know attack on* $b_1, \ldots, b_n$.

Proof. The need-to-know portion of Alice's strategy finishes when all ϕ_i policies are exhausted. As observed above, she only moves on to the next ϕ_i when the she learns she can never obtain any of the credentials in any proof trees that contain ϕ_i, or when each of these trees are sufficiently pruned. If *all possible* proof trees containing ϕ_i have been pruned, then either b_i was disclosed, Bob does not have b_i, or Alice can never satisfy the policies needed to obtain b_i. In each case, Alice learns all that she is authorized to learn about b_i.

Furthermore, after the need-to-know portion of the strategy, the "true policy" α is revealed and the standard negotiation follows, from which Bob can gain access to R. □

It now suffices to prove Lemma 2.

Proof (of Lemma 2). To avoid circular logic, we will consider the first time where the statement of the lemma is violated, and show a contradiction. Up until this point in the negotiation, the statement of the lemma held, and so the invariant given in Theorem 1 has also held.

Consider any leaf b_i which was reinstated by some policy disclosure, and whose policy is not satisfied. Since it appears as a leaf, Bob must not yet have disclosed the policy for b_i (otherwise its policy has been completely pruned, and therefore satisfied). As it was reinstated, it must have been mentioned (and relevant) at some previous time during the negotiation. Since its policy has never been given, it must have appeared as a leaf at that time. But by the invariant of Theorem 1, this leaf with an unsatisfied policy must have been Bob's only evolvable leaf! His only possible disclosures could have been to disclose b_i, admit that he did not have b_i, disclose its policy, or abort the negotiation. Since Bob did not disclose its policy, and the negotiation is continuing, he did not take the latter two choices. But the former two choices would have pruned b_i from all future proof trees, a contradiction.

By this contradiction, the condition of the lemma must always hold. □

Theorem 2. *If Bob can request policy definitions, the need-to-know attack still either limits Bob to one evolvable leaf after each disclosure, or multiple evolvable leaves representing credentials for which the policies have been satisfied.*

Proof. (Sketch) Any time Bob requests the definition of a policy, Alice can reveal the policy protecting that policy. If Bob requests the definition of one of the P_i, Q_i, or ϕ_i policies, Alice will send one of the τ_i policies. Each τ_i is protected by a policy that cannot be satisfied, so these will never be revealed. Thus the only part of these policies that could give Bob a new evolvable leaf is b_i. Thus Alice can still follow the same strategy as in Theorem 1. □

What if Alice attacks Bob, and Bob immediately turns around and attacks Alice? Both parties can stall the negotiation for quite a while, by greeting each request for a credential by a need-to-know attack. However, recall that both parties committed to their policies at the beginning of the negotiation. Since each party can have only finitely many policies protecting his policies and credentials, after a finite number of messages, one party (say, Alice) will not be able to stall any longer. From that point onward, Bob will control the negotiation, giving Alice only one evolvable leaf after each of his disclosures. However, once he finishes his attack, Alice can continue with hers, giving Bob only one evolvable leaf until the negotiation ends.

5 Attacking Other Protocols

The concept of protecting sensitive policies with other policies is described and formally analyzed for interoperability properties with other parties in [21]. Analysis of this added expressiveness led to the formulation of the need-to-know attack. It is important to understand, however, that such an attack is not simply a weakness of the particular model we chose, but rather a problem with trust negotiation in general. We show how a need-to-know attack can be carried out in two other models. In both examples, we demonstrate a simple attack in which only one unneeded credential is requested. While not as well generalized and formalized as the attacks described in Section 3, it demonstrates the existence of the attack with an easy-to-understand example. In both cases, we are also able to omit the stalling policies. Conceptually, these are only necessary when there are clearly other ways for the attacker to advance the negotiation (the protected policies) and neither model requires the attacker to state the existence of protected policies.

The first model, proposed in [6], allows access policies to be sensitive; however, unlike [21], it requires the negotiating party to be authorized to see the contents of a protection policy before the protected resource can be revealed. The first part of a policy lists attributes that the client must satisfy, and the second part lists conditions that the server will evaluate on the attributes before granting access. Because these conditions may be complicated, the server is not required to specify exactly how to evaluate these conditions. We can formulate a similar attack in this model as follows, using the same scenario as in Section 1, where Alice attempts to contact a service called new_DL hosted by DMV. The actual policy for new_DL is (strong_ID $\wedge$ IL_resident). DMV wishes to see Alice's library_card credential, if she has one. DMV can write its policies as shown in Figure 6.

Because the second prerequisite policy contains nothing that a client Bob can disclose, Bob must disclose his library card if he owns one. If he doesn't own one, the server can disclose the resource's policy anyway since the second prerequisite policy is satisfied. Thus, even though the *library_card* credential is not logically required, there is no way a client can access the service without disclosing this credential (provided the server can satisfy its policy). This constitutes a need-to-know attack on the *library_card* credential. The second service_prereqs policy may seem a little far-fetched, but consider a policy like "allow open access during normal business hours," which doesn't depend on any client credentials, just some internal server condition. Such a policy would be indistinguishable from this policy, from the client's point of view. It would not be desirable to disallow policies containing an empty credential list simply to prevent a need-to-know attack from occurring, as such a benefit would come at the expense of expressibility of some legitimate policies.

A similar model was proposed in [19]. This paper describes a trust negotiation model that allows "Ack policies" that must first be satisfied before the actual policy for a credential can be disclosed, even if the client does not possess the credential. A similar attack can be defined for this model, as shown in Figure 7. The paper also describes a protocol for exchanging policies and credentials by defining a structure called a Trust-Target Graph, and rules for which each party can add edges to the graph. When Alice requests access to the new_DL service, DMV is allowed to advance the protocol by adding the control edge $\langle$DMV : DMV.new_DL $\overset{?}{\twoheadleftarrow}$ Alice$\rangle \leftarrowtail \langle$Alice : ?X.library_card $\overset{?}{\twoheadleftarrow}$ DMV$\rangle$. Similar to the previous example, Alice has no choice but to disclose her policy for her library card, if she has one.

Other frameworks for trust negotiation, such as [3, 4, 5, 11], do not support sensitive policies; however many potential applications of trust negotiation do include the need to protect sensitive policies. We conjecture that adding a

service_prereqs(new_DL()) $\leftarrow$ credential(library_card(user=U, ...)).
service_prereqs(new_DL()) $\leftarrow$ *(empty)* | unknown_predicate().
service_reqs(new_DL()) $\leftarrow$ strong_ID(U), IL_resident(U).
strong_ID and IL_resident defined as needed
unknown_predicate() $\leftarrow$ true.

Figure 6. Example attack using model from [6].

AC_{DMV}[new_DL] = DMV.strong_ID, DMV.IL_resident.
Ack_{DMV}[new_DL] = (Library.library_card) $\vee$
(DMV.strong_ID, DMV.IL_resident).

Figure 7. Example attack using model from [19].

layer of policy protection to these systems without carefully considering possible countermeasures would similarly introduce vulnerability to this type of attack.

The attacks described in this paper apply to protocols in which credentials are actually disclosed to each other. This opens the question of whether other protocols and models that do not directly disclose credentials, such as *idemix* [8], OSBE [17], OACerts [15], and hidden credentials [12], might also be vulnerable to such an attack. A variation of the need-to-know attack could be carried out through repeated negotiations, by augmenting the true policy with extra unneeded requirements. This attack is considerably weaker, as it does not maintain logical equivalence to the original policy, and requires the victim to cooperate by restarting the negotiation. However, such an attack would still be indistinguishable from legitimate policy changes. It is not known whether a full need-to-know attack could be performed on these models.

A proposed protocol for exchanging protected credentials cryptographically is described in [1]. Each party receives the other party's credential if and only if each policy is satisfied; otherwise, neither party gains any knowledge about the other, including what the policy was. While such a protocol would seemingly eliminate the need for sensitive policies, their model assumes the existence of single credentials containing all relevant attributes, issued from a central authority. This would not be appropriate for open systems. The weaker variation of the attack using repeated negotiations could also be applied to this protocol.

6 Discussion and Conclusion

In this paper, we have formalized the need-to-know attack and proven its effectiveness. The need-to-know attack relies on an attacker Alice's ability to hide portions of her policy and limit the possible responses of her victim Bob during trust negotiation, thereby harvesting information irrelevant to the negotiation. Alice can rewrite her policies so that Bob repeatedly finds himself with only two options: send Alice information about a credential c that she would like to harvest, or end the negotiation. Our study suggests that protocol designers for trust negotiation protocols may need to consider this type of vulnerability when allowing protected policies.

There are several potential countermeasures to mitigate the need-to-know attack, which we sketch below:

- The most important is to allow parties to ask for the definitions of policy names that have been mentioned during the negotiation, but not yet defined. As we have already shown, this does not prevent or stop a need-to-know attack, but it is a useful component of a larger set of countermeasures that build upon this capability. A second useful step is to have parties commit to their policy definitions at the start of a negotiation. Similarly, our proofs are effective even with this countermeasure, but at least it can prevent on-the-fly attacks against selected clients (e.g., ones with certain zip codes) and has other advantages, discussed below.
- Enforce fairness in the protocol. One way to do this is to require each party to automatically push the appropriate policy definition as soon as a policy name is mentioned. Similarly, one could give highest priority to requests for policy definitions (i.e., a requested policy definition must be sent unless the policy protecting it is not satisfied). However, this makes it easy for denial-of-service attackers to request many policy definitions. As discussed earlier, it can also be highly inefficient when most clients can only gain access to a resource through one particular combination of credentials.

  It is also a difficult problem to *enforce* that players follow a non-cheating strategy, when cheating is undetectable. One solution is to incorporate zero-knowledge proofs [10]. In conjunction with providing commitments of policies up-front (to thwart policy rewriting), Bob could request that Alice prove to him (in zero-knowledge) that she is withholding a certain named policy because Bob has not yet satisfied it (and not because she is "cheating"). In the case of our need-to-know attack, this would affect the τ_i policies, which are true but are never revealed as being true. However, if the parties are willing to accept the computational overhead of zero-knowledge proofs, other protocols with similar efficiency drawbacks (discussed below) might be more appealing.

- A solution involving societal pressure is for an auditing agency to examine the policies of a resource owner to look for evidences of inappropriate information-harvesting practices. If no such evidence is found, the agency can sign a certificate that includes a commitment of the policy contents. A client can look for such a certificate before starting a negotiation with an unfamiliar resource owner, check that the certificate was issued by an auditor that it trusts, check that the commitment in the certificate matches the commitment that the owner sent at the beginning of the negotiation, and have the resource owner prove that it is the entity mentioned in the certificate.

  Even without policy certification, a party might recognize that it is under attack. For example, suppose that Bob does not have a utility bill, patient ID, or library card when he renews his license. He will disclose no new credentials between the time he first asks for the definition of *original_policy* and when he receives it, which suggests that the policy for *original_policy* was already satisfied when he requested that definition. However, in practice policies may contain references to external conditions, such as the time of day. Since Bob (more precisely, his agent) never sees the definition of *original_policy*, or the definitions of the other protected policies that were equivalent to *original_policy*, it will be hard for his agent to argue that it was already satisfied. He could report his suspicions to an auditor, though. He could also pool his experiences with those of other clients, to form a clearer view of Alice's policies and behavior. Similarly, if a human inspects the list of credentials that a particular resource owner requested from his agent, the human will know whether the requests were really appropriate for the resource being accessed. For example, Bob-the-human will know that the DMV should not request his patient ID, and report it to an auditor.

- Restrict policy definitions to at most two layers. More precisely, consider the set of resources that Alice possesses and that are *not* policies. Alice can define policies that protect those resources (layer 1) and policies that protect those policies (layer 2), but we no longer allow Alice to define policies that protect layer-2-policies. That prevents the DMV from using the policies defined in Section 3.2. We conjecture that it also suffices to prevent *all* need-to-know attacks using this protocol.

- Require that a policy not be stronger than the policy it protects. That is, anyone that can satisfy a policy can also satisfy the policy that protects it. This is an approach discussed in [13] and would certainly prevent the usage of the *false* policies that we used in Section 3.2. This solution would, however, come at the expense of autonomy in defining legitimate policies for VIP treatment or unadvertised specials—for instance, one might allow Sam's Club employees to view the policy for the special offer for university employees described in Section 1, even if they themselves are not university employees.

- Have resource owners explicitly delimit the scope of the credentials that they are willing to disclose in order to gain access to a particular resource—a "need-to-know policy" for that resource. For example, Bob can predefine a need-to-know policy for license renewal: he will present no credentials other than his current license. When checking out an audiobook, he will only present his library card. However, this approach has drawbacks in practice. It takes time and effort to write and debug a policy; most credential owners would not be willing to expend that energy. While a helpful third party, such as an auditing firm, could issue standard packages of suggested need-to-know policies for widely accessed online services (such as all DMVs in the US and Canada), these policies will only cover a fraction of the total number of services available online. For example, it is unrealistic to expect such a package to explicitly list all the libraries offering audiobooks in North America. While the auditor could offer a suggested need-to-know policy for libraries in general, Bob would either have to manually identify a particular site as a library, which would be too much work for him; rely on the site to self-categorize itself correctly, which is unreliable; or rely on a third-party categorization service, which would be hard-pressed to keep up with the fast-changing world of internet service offerings. Thus we prefer to avoid explicit encodings of need-to-know policies, if other countermeasures suffice.

- Use an initial negotiation in which an agent does not disclose anything, allowing him to discover as much about the other agent's policies as possible. While this countermeasure would certainly work against the strategies described in this paper, it would not be difficult to imagine an attacker Alice "calling the bluff" of a victim Bob by surmising that Bob may be intentionally hiding his credentials, and in turn refusing to disclose any of her policies in the hopes that Bob would relax his policies and repeat the negotiation later. Additionally, this may be construed as an attempt to cause a denial-of-service attack.

- Use a negotiation method that does not directly disclose credentials, such as the kinds of techniques suggested in work on *idemix* [8], OSBE [17], OACerts [15], hidden credentials [12], and other cryptographic approaches to trust negotiation [7, 9]. These techniques can be integrated into a system that also supports direct disclosure of credentials [16]. For very sensitive information, these techniques can be well worth the extra runtime costs of employing them.

The measures described above will not prevent Alice from harvesting information from "stupid" clients. For example, Bob might not request any policy definitions from Alice, even though he is allowed to. From a theoretical viewpoint, it is tempting to think that such clients deserve what they may get. However, in practice, there may be good reasons to build clients with sharply limited reasoning abilities. For example, to save power Bob might not analyze policies at all, but instead send all information that he can about every credential mentioned so far in the negotiation. To harvest information about credential c from Bob, Alice can just rewrite her policy α to $\alpha \vee (\alpha \wedge c)$ without hiding anything. To prevent such attacks, policy auditors could issue special certificates to resource owners whose policies meet the special needs of "stupid" clients.

Acknowledgements This research was supported by the NSF under grants IIS-0331707, CNS-0325951, and CNS-0524695; and by Sandia National Laboratories under grant number DOE SNL 541065. We also wish to thank the anonymous referees from the Workshop on Privacy in the Electronic Society (WPES 2007) for their suggestions on other countermeasures to the need-to-know attack.

References

[1] G. Ateniese, M. Blanton, and J. Kirsch, "Secret Handshakes with Dynamic and Fuzzy Matching," *Network and Distributed System Security Symposuim (NDSS)*, San Diego, CA, Feb. 2007.

[2] L. Bauer, S. Garriss, and M. K. Reiter, "Efficient Proving for Practical Distributed Access-Control Systems," *12th European Symposium on Research in Computer Security (ESORICS)*, Dresden, Germany, Sep. 2007.

[3] M. Becker and P. Sewell, "Cassandra: Distributed Access Control Policies with Tunable Expressiveness," *IEEE Workshop on Policies for Distributed Systems and Networks*, Jun. 2004.

[4] E. Bertino, E. Ferrari, and A. Squicciarini, "Trust-X: A Peer-to-Peer Framework for Trust Establishment," *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 16(7): 827-842, 2004.

[5] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The KeyNote Trust Management System," IETF RFC 2704, Sept. 1999.

[6] P. Bonatti and P. Samarati, "Regulating Service Access and Information Release on the Web," *ACM Conference on Computer and Communications Security (CCS)*, Athens, Greece, Nov. 2000.

[7] S. Brands, *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*, MIT Press, 2000.

[8] J. Camenisch and E. Van Herreweghen, "Design and Implementation of the *idemix* Anonymous Credential System," *ACM Conference on Computer and Communications Security (CCS)*, Washington, DC, Nov. 2002.

[9] K. Frikken, M. Atallah, and J. Li, "Attribute-Based Access Control with Hidden Policies and Hidden Credentials," *IEEE Transactions on Computers (TC)*, 55(10):1259-1270, 2006.

[10] O. Goldreich, *Foundations of Cryptography: Basic Tools*, Cambridge University Press, Cambridge, United Kingdom, 2001.

[11] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, and B. Smith, "Advanced Client/Server Authentication in TLS," *Network and Distributed System Security Symposium*, San Diego, CA, Feb. 2002.

[12] J. Holt, R. Bradshaw, K. E. Seamons, and H. Orman, "Hidden Credentials," *2nd ACM Workshop on Privacy in the Electronic Society (WPES)*, Washington, DC, Oct. 2003.

[13] K. Irwin and T. Yu, "Preventing Attribute Information Leakage in Automated Trust Negotiation," *ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, Nov. 2005.

[14] H. Koshutanski and F. Massacci, "An Interactive Trust Management and Negotiation Scheme," in *Formal Aspects of Security and Trust*, ed. T. Dimitrakos and F. Martinelli, pp. 139-152, 2004.

[15] J. Li and N. Li, "OACerts: Oblivious Attribute Certificates," *3rd Conference on Applied Cryptography and Network Security (ACNS)*, New York, NY, Jun. 2005.

[16] J. Li, N. Li, and W. H. Winsborough, "Automated Trust Negotiation Using Cryptographic Credentials," to appear in *ACM Transactions on Information and System Security (TISSEC)*, 2007.

[17] N. Li, W. Du, and D. Boneh, "Oblivious Signature-Based Envelope," *ACM Symposium on Principles of Distributed Computing (PODC)*, Boston, MA, Jul. 2003.

[18] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, and K. E. Seamons, "Adaptive Trust Negotiation and Access Control," *ACM Symposium on Access Control Models and Technologies (SACMAT)*, Stockholm, Sweden, Jun. 2005.

[19] W. H. Winsborough and N. Li, "Towards Practical Automated Trust Negotiation," *IEEE Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, Monterrey, CA, Jun. 2002.

[20] W. H. Winsborough, K. E. Seamons, and V. E. Jones, "Automated Trust Negotiation," In *DARPA Information Survivability Conference and Exposition*, volume I, pages 88-102. IEEE Press, Jan. 2000.

[21] T. Yu, M. Winslett, and K. E. Seamons, "Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies in Automated Trust Negotiation," *ACM Transactions on Information and System Security (TISSEC)* 6(1): 1-42, Feb. 2003.

[22] T. Yu, M. Winslett, and K. E. Seamons, "Interoperable Strategies in Automated Trust Negotiation," *ACM Conference on Computer and Communications Security (CCS)*, Philadelphia, PA, Nov. 2001.