Nexus Authorization Logic (NAL): Design Rationale and Applications

FRED B. SCHNEIDER, KEVIN WALSH, and EMIN GÜN SIRER, Cornell University

Nexus Authorization Logic (NAL) provides a principled basis for specifying and reasoning about credentials and authorization policies. It extends prior access control logics that are based on "says" and "speaks for" operators. NAL enables authorization of access requests to depend on (i) the source or pedigree of the requester, (ii) the outcome of any mechanized analysis of the requester, or (iii) the use of trusted software to encapsulate or modify the requester. To illustrate the convenience and expressive power of this approach to authorization, a suite of document-viewer applications was implemented to run on the Nexus operating system. One of the viewers enforces policies that concern the integrity of excerpts that a document contains; another viewer enforces confidentiality policies specified by labels tagging blocks of text.

Categories and Subject Descriptors: D.2.0 [**Software Engineering**]: General—*Protection mechanisms*; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*

General Terms: Security

Additional Key Words and Phrases: Authorization logic, CDD, credentials-based authorization

ACM Reference Format:
Schneider, F. B., Walsh, K., and Sirer, E. G. 2011. Nexus authorization logic (NAL): Design rationale and applications. ACM Trans. Info. Syst. Sec. 14, 1, Article 8 (May 2011), 28 pages.
DOI = 10.1145/1952982.1952990 http://doi.acm.org/10.1145/1952982.1952990

1. INTRODUCTION

In *credentials-based authorization*, requests to access a resource or obtain service are accompanied by *credentials*. Each request is either authorized or denied by a *guard*. The guard is a reference monitor; it uses credentials that accompany the request, perhaps augmented with other credentials or information about system state, to make an authorization decision that enforces some given policy. Authorization decisions thus can be decentralized, with authority shared by the guard with the principals who issue credentials. Accountability for authorization decisions is made explicit through the credentials.

An untrustworthy principal might attempt accesses that violate a security policy, whereas (by definition) a trustworthy one wouldn't. So a guard would never err in authorizing requests made by trustworthy principals. However, determining whether a principal is trustworthy is rarely feasible, so guards typically substitute something that is easier to check.

Supported in part by NICECAP cooperative agreement FA8750-07-2-0037 administered by AFRL; AFOSR grant F9550-06-0019; National Science Foundation grants 0430161, 0964409, and CCF-0424422 (TRUST); ONR grants N00014-01-1-0968 and N00014-09-1-0652; and grants from Microsoft and Intel. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government.
Authors' addresses: F. B. Schneider, K. Walsh, and E. Gün Sirer, Department of Computer Science, Cornell University, Ithaca, NY 14853; email: {fbs,kwalsh,egs}@cs.cornell.edu.

DOI 10.1145/1952982.1952990 http://doi.acm.org/10.1145/1952982.1952990

Access control lists, for example, embody an *axiomatic basis* for making authorization decisions. Axioms are statements that we accept without proof. With guards that use an access control list, we accept without proof that all principals on the access control list are trustworthy, and the guard only authorizes requests made by these principals. The same rationale is applied when a system uses a principal's reputation as the basis for deciding whether that principal's requests should be authorized. An axiomatic basis is also implicit when a guard authorizes requests to run some executable only if the value of a hash indicates the executable is unaltered from a standard distribution or if a digital signature establishes the executable was endorsed by some approved software provider.

Analysis provides a way to predict whether certain behaviors by a program P are possible, so some guards employ an *analytic basis* for authorizing requests made by principals executing P. Specifically, an analysis establishes that P is incapable of certain abuses and, therefore, granting the request will not enable the security policy to be violated. Proof-carrying code [Necula 1997] is perhaps the limit case. In this approach, a program P is accompanied by a proof that its execution satisfies certain properties; a request to execute P is authorized if and only if a proof checker trusted by the guard establishes that the proof is correct and that the properties proved are sufficiently restrictive. As another example, some operating systems [Bershad et al. 1995] will authorize a request to load and execute code only if that code was type checked; type checking is a form of analysis, and programs that type check cannot exhibit certain malicious or erroneous behaviors.

Finally, a *synthetic basis* for authorization is involved whenever a program is transformed prior to execution, if that transformed program can be trusted in ways the original could not. Examples of this approach include sandboxing [Goldberg et al. 1996], software-based fault isolation (SFI) [Wahbe et al. 1993], in-lined reference monitors (IRM) [Erlingsson and Schneider 1999], and other program-rewriting methods [Hamlen et al. 2006; Sirer et al. 1999].

The discussion above suggests that a single basis for establishing trustworthiness is unlikely to suffice throughout an entire system. Different schemes are going to be useful in different settings. And schemes that combine bases are also useful—for example, type safety can be enforced by using a hybrid of program analysis (an analytic basis) and code generation that adds runtime checks (a synthetic basis). So we conjectured that substantial benefits could come from an authorization framework that incorporates and unifies axiomatic, analytic, and synthetic bases. We seem to be the first to classify authorization schemes according to this taxonomy and the first to entertain creating such a unifying framework. Our experience in designing that framework and some initial experiences in using it are the subject of this article.

—We developed a logic NAL (Nexus Authorization Logic) for specifying and reasoning about credentials and authorization policies. NAL makes minor extensions to Abadi's [2007, 2008] access control logic CDD, adding support for axiomatic, analytic, and synthetic bases[1] and adding compound principals (groups and subprincipals) that help bridge the gap from the simplifications and abstractions found in CDD to the pragmatics of actual implementations. We show that NAL, with only these two kinds of compound principals, suffices for a wide range of authorization policies in the service of some novel practical applications.

[1]In fact, any authorization logic that supports a sufficiently expressive language of beliefs should be able to enforce authorization policies according to axiomatic, analytic, and synthetic bases.

$$
\begin{array}{lll}
A ::= \{\!|v : \mathcal{F}|\!\} \;\mid\; A.\tau & & \text{compound principals} \\
\mathcal{F} ::= f(\tau, \ldots) \;\mid\; A \;\mathsf{says}\; \mathcal{F} \;\mid\; \mathsf{x} \;\mid\; (\forall \mathsf{x} : \mathcal{F}) & & \\
\quad \mid\; \mathcal{F} \wedge \mathcal{F} \;\mid\; \mathcal{F} \vee \mathcal{F} \;\mid\; \mathcal{F} \Rightarrow \mathcal{F} \;\mid\; \mathsf{true} & & \text{formulas} \\
\quad \mid\; (\forall v : \mathcal{F}) \;\mid\; (\exists v : \mathcal{F}) & &
\end{array}
$$

Fig. 1. NAL syntax.

$$
\begin{array}{rll}
\mathsf{false}: & (\forall \mathsf{x} : \mathsf{x}) & \\
\neg\mathcal{F}: & (\mathcal{F} \Rightarrow \mathsf{false}) & \\
A \to B: & (\forall \mathsf{x} : (A \;\mathsf{says}\; \mathsf{x}) \Rightarrow (B \;\mathsf{says}\; \mathsf{x})) & \\
A \xrightarrow{\overline{v}:\mathcal{F}} B: & (\forall \overline{v} : (A \;\mathsf{says}\; \mathcal{F}) \Rightarrow (B \;\mathsf{says}\; \mathcal{F})) & \text{for } \overline{v} \text{ not free in } A \text{ or } B
\end{array}
$$

Fig. 2. NAL abbreviations.

—We implemented a suite of document-viewer applications that run on Nexus, and we discuss two[2] in this article. `TruDocs` (Trustworthy Documents) controls the display of documents that contain excerpts whose use is subject to restrictions; it employs an analytic basis for authorization. `ConfDocs` (Confidential Documents) protects confidentiality of documents built from text elements that have security labels; it employs both analytic and synthetic bases for authorization.

NAL was designed for use in a new operating system, Nexus [Shieh et al. 2005], which employs a TPM (Trusted Computing Group's Trusted Platform Module (TPM), version 1.2[3]) secure coprocessor so that it has a single hardware-protected root of trust. NAL's scheme for naming principals and NAL's operators for attribution and delegation were informed by the needs of Nexus and the capabilities a TPM offers. However, we have also found NAL to be valuable in other settings (independent of Nexus) where authorization or attestation are important elements. Details about uses of NAL in Nexus and in these other system settings will be reported elsewhere; here, we focus on explaining why NAL has the features it does and why we did not include others.

2. NEXUS AUTHORIZATION LOGIC (NAL)

We start with an informal introduction to NAL, focusing on how NAL can be used to specify credentials and authorization policies. Principals with modalities says and speaks-for ($\to$) are what makes NAL different from the higher-order predicate logics often used in programming; the rest of this section discusses those differences in some detail.

The syntax of NAL formulas is given in Figure 1, and some useful NAL abbreviations appear in Figure 2. There and throughout this article, identifiers typeset in lowercase sans serif font (e.g., x) denote propositional variables; identifiers typeset in lowercase italic font (e.g., v) denote first-order variables; $\overline{v}$ abbreviates a list $v_1, v_2, \ldots, v_n$; identifiers typeset in uppercase italic font (e.g., A, B, ...) denote principals; and identifiers typeset in calligraphic font (e.g., $\mathcal{F}$, $\mathcal{G}$, ...) denote NAL formulas. The language for terms τ is left unspecified.

[2]We also have developed a third application `CertiPics` (Certified Pictures), discussed in Walsh [2011], which enforces the integrity of displayed digital images by imposing chain-of-custody restrictions on the image-editing pipeline.

[3]`http://www.trustedcomputinggroup.org/specs/TPM/`.

We write

$$\textsc{RuleName:}\frac{\mathcal{F}_1,\ \mathcal{F}_2, \ldots, \mathcal{F}_n}{\mathcal{F}}$$

to define an inference rule RULENAME that allows a *conclusion* $\mathcal{F}$ to be inferred assuming that *premises* $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n$ have been. The Appendix gives the complete list of NAL axioms and inference rules. We focus below on aspects of the logic concerned with principals and the says and speaks-for modalities.

Like CDD, NAL is a constructive logic. Constructive logics are well suited for reasoning about authorization [Garg and Pfenning 2006], because constructive proofs include all of the evidence used for reaching a conclusion and, therefore, information about accountability is not lost. Classical logics allow proofs that omit evidence. For example, we can prove $\mathcal{G}$ using a classical logic by proving $\mathcal{F} \Rightarrow \mathcal{G}$ and $\neg\mathcal{F} \Rightarrow \mathcal{G}$, since from these theorems we can conclude $(\mathcal{F} \vee \neg\mathcal{F}) \Rightarrow \mathcal{G}$, and hence $\mathsf{true} \Rightarrow \mathcal{G}$ due to Law of the Excluded Middle. This classical proof, however, does not say whether it is $\mathcal{F}$ or it is $\neg\mathcal{F}$ that is the evidence for $\mathcal{G}$, and thus the proof is arguably unsatisfactory as a rationale that $\mathcal{G}$ holds.

Beliefs and says. NAL, like its predecessors [Abadi 2007; 2008; Abadi et al. 1993; Bauer et al. 2005b; Becker and Sewell 2004; DeTreville 2002; Jim 2001; Lesniewski-Laas et al. 2007; Li et al. 2002], is a logic of beliefs. Each principal A has a *worldview* $\omega(A)$, which is a set of beliefs that A holds or, equivalently, formulas that A believes to be true. NAL formula $A\ \mathsf{says}\ \mathcal{F}$ is interpreted to mean: $\mathcal{F} \in \omega(A)$ holds.

NAL extends CDD by allowing formulas to include system- and application-defined predicates in place of propositions. Since NAL terms can include the names of principals, NAL formulas can convey information about, and hence potential reasons to trust, a principal. For example, the NAL formula

$$\mathit{Analyzer}\ \mathsf{says}\ \mathit{numChan}(P, \text{“TCP”}) = 3 \tag{1}$$

holds if and only if worldview $\omega(\mathit{Analyzer})$ contains a belief that $\mathit{numChan}(P, \text{“TCP”}) = 3$ holds. System-defined function $\mathit{numChan}(P, \text{“TCP”})$ yields the number of TCP connection open at P. An NAL formula like (1) could specify a credential or specify (part of) an authorization policy. As a credential, formula (1) asserts that *Analyzer* believes and is accountable for the truth of $\mathit{numChan}(P, \text{“TCP”}) = 3$; as a specification for an authorization policy, formula (1) requires a guard to establish that $\mathit{numChan}(P, \text{“TCP”}) = 3$ is in $\omega(\mathit{Analyzer})$.

The worldview of each principal is presumed to contain all NAL theorems. NAL therefore includes a *necessitation* inference rule:

$$\textsc{says-i:}\frac{\mathcal{F}}{A\ \mathsf{says}\ \mathcal{F}}.$$

We assume a constructive logical theory for reasoning about system- and application-defined predicates; SAYS-I asserts that those theorems are part of each principal's worldview.

Principals may hold beliefs that are not actually true statements and/or that are in conflict with beliefs that other principals hold. Just because $A\ \mathsf{says}\ \mathcal{F}$ holds does not necessarily mean that $\mathcal{F}$ holds or that $B\ \mathsf{says}\ \mathcal{F}$ holds for a different principal B. However, beliefs that a principal holds are presumed to be consistent with beliefs that same principal holds about its own beliefs:

$$\textsc{says-e:}\frac{A\ \mathsf{says}(A\ \mathsf{says}\ \mathcal{F})}{A\ \mathsf{says}\ \mathcal{F}}.$$

Deduction and Local-Reasoning. A principal's worldview is assumed to be *deductively closed*: for all principals A, any formula $\mathcal{G}$ that can be derived using NAL from the formulas in $\omega(A)$ is itself an $\omega(A)$. This supports having the usual implication-elimination rule

$$\text{IMP-E:}\frac{\mathcal{F},\ \mathcal{F}\Rightarrow\mathcal{G}}{\mathcal{G}}$$

along with a closure under implication rule:

$$\text{DEDUCE:}\frac{A\ \mathsf{says}(\mathcal{F}\Rightarrow\mathcal{G})}{(A\ \mathsf{says}\,\mathcal{F})\Rightarrow(A\ \mathsf{says}\,\mathcal{G})}.$$

Notice that all formulas in DEDUCE refer to the same principal. This *local-reasoning restriction* limits the impact a principal with inconsistent beliefs can have. In particular, from $A\ \mathsf{says}\ \mathsf{false}$, DEDUCE enables us to derive[4] $A\ \mathsf{says}\,\mathcal{G}$ for any $\mathcal{G}$, but DEDUCE cannot be used to derive $B\ \mathsf{says}\,\mathcal{G}$ for a different principal B. So the local-reasoning restriction causes inconsistency within $\omega(A)$ to be contained. The local-reasoning restriction also prevents mutually inconsistent beliefs held by an unrelated set of principals from being combined to derive $A\ \mathsf{says}\ \mathsf{false}$ for any principal A [Abadi 2007, 2008].

Delegation. The notation $A\rightarrow B$ (read "A speaks for B") abbreviates the NAL formula

$$(\forall \mathsf{x}:(A\ \mathsf{says}\ \mathsf{x})\Rightarrow(B\ \mathsf{says}\ \mathsf{x})). \tag{2}$$

Since propositional variable x in subformulas "$A\ \mathsf{says}\ \mathsf{x}$" and "$B\ \mathsf{says}\ \mathsf{x}$" of (2) can be instantiated by any NAL formula, we conclude that if $A\rightarrow B$ holds then all beliefs in the worldview of principal A also appear in the worldview of principal B; therefore $\omega(A)\subseteq\omega(B)$ holds. In terms of credentials, $A\rightarrow B$ characterizes the consequences of B delegating to A the task of issuing credentials. Not only would A be accountable for such credentials but so would B.

The transitivity of $\rightarrow$ follows directly from definition (2), and therefore we have the following as a derived inference rule of NAL:

$$\rightarrow\ \text{TRANS:}\frac{A\rightarrow B,\ B\rightarrow C}{A\rightarrow C}.$$

One theorem of NAL is

$$(B\ \mathsf{says}(A\rightarrow B))\Rightarrow(A\rightarrow B), \tag{3}$$

which implies that the following is a derived inference rule of NAL:

$$\text{HAND-OFF:}\frac{B\ \mathsf{says}(A\rightarrow B)}{A\rightarrow B}.$$

An interpretation of HAND-OFF (or equivalently theorem (3)) is that each principal B is an authority on its own delegations.

NAL also supports an abbreviation to assert the *restricted delegation* that only certain beliefs held by A are attributed to B. We write $A\xrightarrow{\overline{v}:\mathcal{F}}B$ (read "A speaks for B about $\mathcal{F}$"), where no identifier in $\overline{v}$ appears free in B or A, to assert

$$(\forall\overline{v}:(A\ \mathsf{says}\,\mathcal{F})\Rightarrow(B\ \mathsf{says}\,\mathcal{F})).$$

[4]Here is that proof: $\mathsf{false}\Rightarrow\mathcal{G}$ is a theorem for all $\mathcal{G}$. Therefore, by SAYS-I we conclude $A\ \mathsf{says}(\mathsf{false}\Rightarrow\mathcal{G})$ is a theorem for all $\mathcal{G}$. We then use DEDUCE and IMP-E to derive $A\ \mathsf{says}\,\mathcal{G}$.

Such a restricted delegation allows us to specify in NAL that a principal B delegates authority or trusts another principal A for only certain kinds of credentials.

For example, we should not be surprised to find a university's registrar *UnivReg* trusted by academic department *CSdept* about whether somebody is a student at that university. We specify this trust by writing the NAL restricted delegation formula

$$UnivReg \xrightarrow{v:v \in Students} CSdept, \tag{4}$$

meaning

$$(\forall v : UnivReg \text{ says } v \in Students \;\; \Rightarrow \;\; CSdept \text{ says } v \in Students).$$

Restricted delegation (4) limits which credentials issued by *UnivReg* can be attributed to *CSdept*. So credential *UnivReg* says $\mathcal{F}$ can be used with (4) to derive *CSDept* says $\mathcal{F}$ when $\mathcal{F}$ is "$Bob \in Students$" but not when $\mathcal{F}$ is "$offer(\text{CS101}, \text{Spr})$".

Restricted delegation (4) limits the abuse of privilege and spread of bogus beliefs by asserting that *CSDept* will adopt only certain kinds of beliefs held by *UnivReg*. Some care is required, though, when using this defense. A second unrestricted delegation

$$UnivReg \xrightarrow{v:v \notin Students} CSdept$$

along with (4) could allow *CSdept* says false to be derived if *UnivReg* is compromised and thus willing to issue bogus credentials

$$UnivReg \text{ says } v \in Students,$$
$$UnivReg \text{ says } v \notin Students,$$

for some student v.

As with $\rightarrow$, we have a corresponding derived inference rule for transitivity of $\xrightarrow{\overline{v}:\mathcal{F}}$:

$$\xrightarrow{\overline{v}:\mathcal{F}} \text{ TRANS}: \frac{A \xrightarrow{\overline{v}:\mathcal{F}} B,\ B \xrightarrow{\overline{v}:\mathcal{F}} C}{A \xrightarrow{\overline{v}:\mathcal{F}} C}.$$

And we have the NAL theorem

$$(B \text{ says} (A \xrightarrow{\overline{v}:\mathcal{F}} B)) \Rightarrow (A \xrightarrow{\overline{v}:\mathcal{F}} B),$$

which leads to a corresponding derived inference rule:

$$\text{REST-HAND-OFF}: \frac{B \text{ says} (A \xrightarrow{\overline{v}:\mathcal{F}} B)}{A \xrightarrow{\overline{v}:\mathcal{F}} B}.$$

2.1. Predicates and Terms in NAL

NAL is largely agnostic about how predicates and terms are implemented.[5] But an authorization mechanism that evaluates NAL formulas would be impractical unless efficient implementations are available for NAL predicate and term evaluation. The Nexus kernel, for example, provides efficient system routines for programs to read certain operating system state (abstracted in NAL formulas as terms) and to evaluate certain predefined predicates on that state. Also, any deterministic Boolean-valued routine running in Nexus can serve as an NAL predicate. So if an authorization policy can be programmed in Nexus then it can be specified using an NAL formula.

[5]Because it is a constructive logic, NAL does require that all terms and predicates be computable.

The designer of a guard in Nexus must decide what sources to trust for information about the current and past states. Presumably, a guard would trust predicate evaluations that it performs itself or that the Nexus kernel performs on its behalf. Other components might have to be trusted, too, because it is unlikely that every principal would be able to evaluate every predicate due to constraints imposed by locality and/or confidentiality. Arguably, a large part of designing a secure system is concerned with aligning what must be trusted with what can be trusted. NAL helps focus on these design choices by having each credential explicitly bind the name of principal to the belief that credential conveys, thereby surfacing what is being trusted.

NAL is agnostic about predicate and function naming, assuming only that the name is associated with a unique interpretation across all principals. One approach it to define an authoritative interpretation (including an evaluation scheme) for each name; all principals are then required to use that. Implicit in such a solution would have to be some way to determine what is the authoritative interpretation for a given name. Nexus addresses this by implementing hierarchical naming, where a name encodes the identity of the principal that is the authority for performing evaluations.

2.2. Principals in NAL

Principals model entities to which beliefs can be attributed. Examples include active entities like processors, processes, and channels, as well as passive objects like data structures and files. We require that distinct NAL principals have distinct names and that credentials attributed to a principal cannot be forged. Schemes that satisfy these requirements include the following.

—Use a public key as the name of a principal, where that principal is the only entity that can digitally sign content using the corresponding private key. A principal named by a public key K_A signifies that a belief $\mathcal{F}$ is in worldview $\omega(K_A)$ by digitally signing an encoding of $\mathcal{F}$. So a digitally signed representation of the NAL statement $\mathcal{F}$, where public key K_A verifies the signature, conveys NAL formula K_A says $\mathcal{F}$.

—Use the hash of a digital representation of an object as the name of a principal associated with that object. A principal named by hash $H(obj)$ includes a belief $\mathcal{F}$ in its worldview $\omega(H(obj))$ by having an encoding of $\mathcal{F}$ stored in obj.[6] So by having $\mathcal{F}$ be part of obj, $H(obj)$ conveys NAL formula $H(obj)$ says $\mathcal{F}$.

The benefit of using a public key K_A to name a principal is that this name then suffices for validating that a credential K_A says $\mathcal{F}$ has not been forged or corrupted. Also, credentials conveying individual beliefs or subsets of beliefs in $\omega(K_A)$ can be issued at any time. But public-private key pairs are expensive to create. Moreover, private keys can be kept secret only by certain types of principals. With a TPM, you can associate a private key with a processor and keep it secret from all software that runs on the processor; without a TPM, you can associate a private key with a processor but keep it secret only from nonprivileged software. And there is no way to associate a private key with a nonprivileged program executing on a processor yet have that key be secret from the processor or from privileged software being run.

Hashes are an attractive basis for naming principals, because hashes are relatively inexpensive to calculate and do not require secrets. However, a principal must have read-access to obj in order to generate or validate a credential $H(obj)$ says $\mathcal{F}$ for conveying beliefs that, because they are stored in obj, are part of the worldview of $H(obj)$. The use of hashes for naming principals is useful only for conveying static sets of beliefs held by objects whose state is fixed. Change the beliefs or the state of object obj and

[6]We might adopt the convention that every object obj involves two parts. The first part is a possibly empty list of the NAL formulas $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n$ in $\omega(H(obj))$; the second part is some other digital content.

name $H(obj)$ changes too, which means credentials previously issued for that object could no longer validate.

NAL is agnostic about what schemes are used to name principals. Our experience with Nexus applications has been that public keys and hashes both have uses. Nexus also implements various specialized naming schemes for some of its abstractions (e.g., processes and the kernel) that serve as principals.

Subprincipals. System components often depend on other system components. In hierarchically structured systems, for example, higher levels depend on lower levels. Also, dependencies are created when one component loads and starts executing (or interpreting) another. The dependency of a principal *Sub* on another principal *Dom* can be so strong that *Sub* is *materialized* by *Dom*; hence *Sub* says $\mathcal{F}$ holds only if *Dom* says(*Sub* says $\mathcal{F}$) holds. For example, execution of a program *Prog* is ultimately materialized by computer hardware (say) *CPU*, and therefore *Prog* says $\mathcal{F}$ holds only if *CPU* says(*Prog* says $\mathcal{F}$) holds.

NAL offers *subprincipals* as a convenience for naming a principal that is materialized by another. Given a principal A and any term τ, subprincipal $A.\tau$ is an NAL principal[7] materialized by A. This is captured in an NAL rule:

$$\text{SUBPRIN:}\ \frac{}{A \to A.\tau}.$$

Equivalent terms define subprincipals having the same worldviews:

$$\text{EQUIV SUBPRIN:}\ \frac{\tau_1 = \tau_2}{A.\tau_1 \to A.\tau_2}.$$

Here, we assume some theory is available for proving premise $\tau_1 = \tau_2$.

Subprincipals are particularly useful for describing structures where a principal is multiplexed among various roles. For example, processes are often materialized by an operating system that multiplexes a processor. Thus the principal that corresponds to an executing program is a subprincipal of the operating system that runs that program; and the operating system itself is a subprincipal of the processor.

As an illustration, consider a system comprising a certification authority CA being executed by an operating system OS that is multiplexing a processor among a number of applications. And suppose the hash of the in-memory image for CA is H_{CA}, the hash of the in-memory image OS is H_{OS}, and the processor's TPM stores a private key whose signatures can be verified using public key K_{CPU}. In NAL, these dependencies could be characterized using a subprincipal $K_{CPU}.H_{OS}$ for the OS and a subprincipal $K_{CPU}.H_{OS}.H_{CA}$ for the CA. According to SUBPRIN, we have

$$K_{CPU} \to K_{CPU}.H_{OS}, \tag{5}$$

$$K_{CPU}.H_{OS} \to K_{CPU}.H_{OS}.H_{CA}. \tag{6}$$

A credential attributed to execution of CA would, in fact, be issued by K_{CPU}, materializing operating system OS, materializing CA. So the credential for a belief $\mathcal{F}$ held by CA would be specified by the NAL formula

$$K_{CPU} \text{ says}(K_{CPU}.H_{OS} \text{ says}(K_{CPU}.H_{OS}.H_{CA} \text{ says } \mathcal{F})),$$

[7]Subprincipals can themselves have subprincipals, with left-associativity assumed so that $A.\tau_1.\tau_2$ abbreviates $(A.\tau_1).\tau_2$.

from which we can derive

$$K_{CPU}.H_{OS}.H_{CA} \text{ says } (K_{CPU}.H_{OS}.H_{CA} \text{ says } (K_{CPU}.H_{OS}.H_{CA} \text{ says } \mathcal{F})),$$

by (5) and (6) and definition (2) of $A \rightarrow B$; using SAYS-E twice then obtains

$$K_{CPU}.H_{OS}.H_{CA} \text{ says } \mathcal{F}.$$

Subprincipals are also useful for creating different instances of a given principal, where each instance is accountable for the credentials issued during disjoint epochs or under the auspices of a different nonce or different circumstances. This allows the subset of credentials issued by some principal A at a time when you trust A to be distinguished from credentials issued by A at other times. So instead of using a single principal *FileSys*, we might employ a sequence *FileSys*.1, *FileSys*.2, ..., *FileSys*.i, ... of subprincipals, each accountable for issuing credentials during successive epochs. Then by specifying security policies that are satisfied only by credentials attributed to a "current instance" *FileSys.now* (for *now* an integer variable), a guard can reject requests accompanied by outdated credentials.

SUBPRIN allows any statement by a principal A to be attributed to any subprincipal of A. That is, from A says $\mathcal{F}$ we could derive $A.\tau$ says $\mathcal{F}$ for any subprincipal $A.\tau$. Unintended attributions can be avoided, however, by adopting a subprincipal naming convention. We might, for example, agree to attribute to subprincipal $A.\epsilon$ any belief by A that should not be attributed to any other subprincipal $A.\tau$ of A.

Groups. An NAL *group* is a principal constructed from a set of other principals, called *constituents*, and is specified *intensionally* by giving a characteristic predicate. We write $\{\!|v : \mathcal{P}|\!\}$ to denote the group formed from characteristic predicate $\mathcal{P}$; the group's constituents are those principals A for which $\mathcal{P}[v := A]$ holds.[8] As an example,

$$\{\!|v : v \rightarrow K_{CPU}.H_{OS}.H_{CA}|\!\}$$

is the group of all principals that speak for principal $K_{CPU}.H_{OS}.H_{CA}$.

The worldview of an NAL group is defined to be the union, deductively closed, of the worldviews for its constituents. Thus, if the worldview for one constituent of the group contains $\mathcal{F} \Rightarrow \mathcal{G}$ and another contains $\mathcal{F}$, then the group's worldview contains beliefs $\mathcal{F}$, $\mathcal{F} \Rightarrow \mathcal{G}$, and $\mathcal{G}$—even if the worldview for no constituent of the group contains $\mathcal{G}$.

Because the worldview of each constituent is a subset of the group's worldview, we conclude for each constituent A of group G that $A \rightarrow G$ holds. Thus, if $\mathcal{P}[v := A]$ holds then $A \rightarrow \{\!|v : \mathcal{P}|\!\}$ holds:

$$\text{MEMBER:} \frac{\mathcal{P}[v := A]}{A \rightarrow \{\!|v : \mathcal{P}|\!\}} \quad \text{free variables of } A \text{ are free for } v \text{ in } \mathcal{P}.$$

Note that $A \rightarrow \{\!|v : \mathcal{P}|\!\}$ does not necessarily imply that $\mathcal{P}[v := A]$ holds. In the absence of an NAL derivation for $\mathcal{P}[v := A]$, we could still derive $A \rightarrow \{\!|v : \mathcal{P}|\!\}$ from derivations for $A \rightarrow B$ and $B \rightarrow \{\!|v : \mathcal{P}|\!\}$.

When $v \rightarrow A$ holds for all constituents v of a group, then all beliefs in the group's worldview necessarily appear in $\omega(A)$, so the group speaks for A:

$$\rightarrow \text{ GROUP:} \frac{(\forall v : \mathcal{P} \Rightarrow (v \rightarrow A))}{\{\!|v : \mathcal{P}|\!\} \rightarrow A}.$$

[8] $\mathcal{P}[v := exp]$ denotes textual substitution of all free occurrences of v in $\mathcal{P}'$ by exp, where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by renaming bound variables to avoid capture.

This inference rule, in combination with MEMBER, allows us to justify the following derived inference rule, which asserts groups and $\rightarrow$ are monotonic relative to implication:

$$\text{GROUP MONOTONICITY:}\ \frac{(\forall v : \mathcal{P} \Rightarrow \mathcal{P}')}{\{\!|v : \mathcal{P}|\!\} \rightarrow \{\!|v : \mathcal{P}'|\!\}}.$$

Finally, note that NAL does not preclude specification of *extensionally* defined groups, wherein constituents are simply enumerated. For example, $\{\!|v : v \in \{A, B, C\}|\!\}$ is the extensionally defined group whose constituents are principals A, B, and C.

3. GUARDS: THEORY AND PRACTICE

The decision to authorize a request can be posed as a question about NAL formula derivation. We represent requests, credentials, and authorization policies as NAL formulas. A guard G that enforces an authorization policy $\mathcal{G}$ allows a request $\mathcal{R}$ to proceed by if and only if

(i) G has a set of unforged credentials $\mathtt{C}_1, \mathtt{C}_2, \ldots, \mathtt{C}_n$, where credential $\mathtt{C}_i$ conveys NAL formula $\mathcal{C}_i$, and
(ii) G establishes that NAL can be used to derive $\mathcal{G}$ from

$$\mathcal{R} \wedge \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \cdots \wedge \mathcal{C}_n.$$

Credentials could be stored at the requesting principal, at the guard, or elsewhere in the system; and they could accompany a request, be fetched when needed by the guard, or be sent periodically to the guard. Notice that if request $\mathcal{R}$ is granted, then an NAL derivation of $\mathcal{G}$ to discharge (ii) documents the rationale for this authorization decision, indicating the role each credential plays. The NAL derivation of $\mathcal{G}$ is thus a form of audit log—and a particularly descriptive one, at that.

A file system guard *FileSys* might, for example, enforce discretionary access control[9] for accessing a file (say) `foo` by employing authorization policy

$$\mathit{FileSys}\ \mathsf{says}\ \mathit{read}(\mathtt{foo}) \tag{7}$$

and issuing a restricted delegation

$$A \xrightarrow{\mathit{read}(\mathtt{foo})} \mathit{FileSys} \tag{8}$$

for any principal A whose requests $A\,\mathsf{says}\,\mathit{read}(\mathtt{foo})$ should be allowed to proceed, since (8) allows (7) to be derived from $A\,\mathsf{says}\,\mathit{read}(\mathtt{foo})$. Store the restricted delegation credential for a principal at that principal, and the result is reminiscent of capabilities; aggregate and store all of the restricted delegation credentials at the guard, and access control lists result.

Credential Distribution and Revocation. Whenever a guard G has some credential `C` that G determines is not forged, the NAL formula $\mathcal{C}$ conveyed by `C` is part of G's beliefs. G having credential `C` thus can be formalized in NAL as $G\,\mathsf{says}\,\mathcal{C}$.

One might hope that

$$(G\,\mathsf{says}\,\mathcal{C}) \Rightarrow \mathcal{C} \tag{9}$$

would hold and, therefore, beliefs acquired through credentials are sound. This property, however, is not guaranteed. A principal P might issue `C` but then change beliefs (perhaps because the state or environment has changed) in a way that invalidates $\mathcal{C}$.

If P invalidates $\mathcal{C}$ after `C` has been distributed to other principals, then an access could be granted on *false pretense* because a guard has `C` and therefore believes that

[9]This is not the only way to specify discretionary access control using NAL and guards.

$\mathcal{C}$ holds even though $\mathcal{C}$ does not. Even if guards check the truth of each credential just before use, access might still be granted on false pretense—checking a credential takes time, and concurrent actions elsewhere in the system could falsify the formula conveyed by a credential after being checked.

For example, a request A says $access(obj)$ accompanied by a credential that conveys $TimeServ$ says $clock = 900$ suffices to derive the authorization policy:

$$A \text{ says } \mathrm{access}(obj) \quad \wedge \quad TimeServ \text{ says } clock < 1000. \tag{10}$$

But

$$(G \text{ says}(TimeServ \text{ says } clock = 900)) \quad \Rightarrow \quad TimeServ \text{ says } clock = 900$$

does not hold if $TimeServ$ revises its beliefs whenever the passage of time causes $clock$ to increase. A guard that checks whether credentials are not forged and whether an NAL derivation exists for authorization policy (10) thus could grant access for requests made after $clock < 1000$ ceases to hold. So the guard could grant requests on false pretenses.

There are two general strategies for ensuring that (9) will hold and, thus, prevent accesses from being granted on false pretenses:

(i) require that antecedent G says $\mathcal{C}$ of (9) is false prior to changes in beliefs that invalidate its consequent $\mathcal{C}$;
(ii) choose for consequent $\mathcal{C}$ of (9) a formula that cannot be invalidated by principals changing their beliefs.

With strategy (i), all principals G that have a credential `C` must delete their copies (thereby falsifying G says $\mathcal{C}$) before any principal is allowed to change its beliefs in a way that invalidates $\mathcal{C}$. This credential revocation is not a new problem[10] and is generally considered infeasible when credentials propagate in unrestricted ways. But a feasible realization of strategy (i), used extensively in Nexus, exists when the propagation of credentials is restricted in a rather natural way.

In Nexus, a principal is called an *authority* if it is both the sole source of certain credentials and the only principal that can invalidate an NAL formula conveyed by those credentials. We ensure that the only way that a principal can obtain a credential issued by an authority is directly from that authority. The authority thus knows which principals have its credentials. Prior to invalidating the belief conveyed in such a credential, the authority requests that these principals delete their copies and awaits confirmations (either from the principal or from Nexus asserting that the principal has been terminated).

To ensure that principals obtain credentials A says $\mathcal{F}$ directly from the issuing authority A, such credentials are represented in a way that conveys $\mathcal{F}$, allows the recipient to attribute $\mathcal{F}$ to A, but does not allow any other principal to validate that attribution. For example, the Nexus IPC mechanism implements authenticated, integrity-protected channels. Such a channel can speak for an authority A and transmit a formula $\mathcal{F}$ that only the single recipient at the end point of that channel can attribute to A. Authorities in Nexus distribute credentials using such channels. However, the same effect also could be achieved on an ordinary channel by using cryptography; if the two endpoints share a secret, for example, then message authentication codes suffice.

Strategy (ii) for ensuring that (9) will hold requires restricting the execution of principals and/or choosing for $\mathcal{C}$ a sufficiently weak formula. System developers thus are made responsible for supporting revocation. Fortunately, system execution generally

[10]It arises, for example, in connection with capabilities and with public-key certificates that describe name-key bindings.

does satisfy certain restrictions—time never decreases and the past is immutable, for example—not to mention restrictions coupled to the semantics of system and application functionality. So some truths do not change as execution proceeds, and this can be leveraged for defining NAL formulas $\mathcal{C}$ that cannot be falsified by future execution. For instance, once $clock > 1000$ holds, it cannot be later falsified by time advancing. And a credential attesting that some predicate p once held cannot subsequently be falsified if p holds when that credential is issued.

Imposing additional execution restrictions on principals is the other way to instantiate strategy (ii) for ensuring that (9) continues to hold. Suppose that, in order to authorize some request, a guard G requires A says p for p a state predicate, but that A says p could be invalidated from time to time.

—One solution is to prevent principals from invalidating p until some time in the future, in effect using a credential that conveys a form of lease [Gray and Cheriton 1989]. For example,

$$A\text{says}(clock < 1000 \;\Rightarrow\; p)$$

is not falsified when *clock* advances, so (9) will now hold. And if credentials

$$TimeServ \text{ says } clock < 1000, \tag{11}$$

$$TimeServ \xrightarrow{v:clock<v} G \tag{12}$$

are available, then G can still conclude that A says p is satisfied. Moreover, if *TimeServ* is implemented as an authority then we can ensure that (11) satisfies (9); and if delegation (12) is never disseminated outside of G, then it too will satisfy (9).

—An alternative to using leases is to have principals follow a locking protocol before invalidating p. For example, we might postulate a lock ℓ_p with two modes of access. Any number of principals can concurrently hold `shared` access, and a principal can hold `exclusive` access only if no other principal holds `shared` or `exclusive` access, with the following restrictions on execution:

(i) a guard acquires `shared` access to ℓ_p before authorizing a decision using a credential involving p, and the guard releases the lock afterward;
(ii) a principal acquires `exclusive` access to ℓ_p before falsifying p and must reestablish p prior to releasing ℓ_p.

Then a credential conveying

$$A\text{says}(locked(\texttt{shared}, \ell_p) \Rightarrow p)$$

is never falsified even though p might be, so (9) holds. Moreover, if a guard G acquires ℓ_p with `shared` access before making an authorization decision, so G has credentials

$$LockMngr \text{ says } locked(\texttt{shared}, \ell_p), \tag{13}$$

$$LockMngr \xrightarrow{locked(\texttt{shared},\ell_p)} G \tag{14}$$

attesting to $locked(\texttt{shared}, \ell_P)$, then G guard can conclude that A says p is satisfied at that time. Credentials (13) and (14) can be made to satisfy (9) if *LockMngr* is implemented as an authority and (14) is never disseminated outside of G.

Sources of Derivations. If NAL includes terms whose axiomatization is undecidable (e.g., integers or rich data structures), as will often be the case, then we cannot hope to build a universal guard—a guard that, for any choice of authorization policy $\mathcal{G}$ and set of credentials $\{\mathtt{C}_1, \mathtt{C}_2, \ldots, \mathtt{C}_n\}$, derives $\mathcal{G}$ from $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ if and only if such an NAL derivation exists. This undecidability result, however, does not preclude building

guards that automatically generate an NAL derivation for some particular authorization policy when given credentials in some pre-specified form. The file system example above illustrates this, because authorization policy (7) can be derived automatically from a request A says $read(\texttt{foo})$ when given a credential that conveys (8).

An alternative to having a guard perform the derivation of an authorization policy $\mathcal{G}$ would be to accompany each request with an NAL derivation of $\mathcal{G}$ [Appel and Felten 1999; Bauer 2003] or for the guard to solicit the derivation from trusted third parties [Bauer et al. 2005b]. In either case, the guard checks an NAL derivation rather than generating its own. This check is a decidable task, because NAL derivations are finite in length and inference rule applications are mechanically checkable. But having each request be accompanied by a derivation is not a panacea. For a principal to produce a derivation of an authorization policy $\mathcal{G}$, that principal must know what $\mathcal{G}$ is. Yet sometimes $\mathcal{G}$ must be kept secret so that, for example, various principals do not learn that different authorization criteria apply to each. Also, having each requester independently derive $\mathcal{G}$ makes changing $\mathcal{G}$ difficult, since principals that will submit requests to the guard must either be programmed to accommodate such changes (which might not be possible for the same reason that universal guards cannot exist) or must be found and manually updated whenever $\mathcal{G}$ is altered.

4. EXAMPLE APPLICATIONS: A DOCUMENT-VIEWER SUITE

To gain confidence in the utility of our authorization framework, we used it and prototyped a suite of document-viewer applications that run on Nexus. This required formulating authorization policies in NAL, implementing an NAL proof checker (see appendix B of Schneider et al. [2009] for details about the proof checker), and building Nexus support for creating credentials and guards.

In each of the viewer applications, we define a one-to-one correspondence between documents and principals. And the principal for the document to be displayed—not the human user viewing the document—is the principal whose requests are authorized by a guard. This unconventional architecture allows us to benefit from employing analytic and synthetic bases for authorization. Had the system instead been designed to process requests from human users wishing to view documents, then we would have been limited to employing an axiomatic basis for authorization, since humans are hard to analyze and do not take kindly to transformations.

4.1. `TruDocs`: Analytic and Axiomatic Bases for Authorization

`TruDocs` is a document-viewer application that ensures excerpts attributed to a document are consistent with policies that document specifies. We start with the observations that documents convey beliefs and that excerpts derived from a document also convey beliefs. So for d_i some document, NAL provides a natural way for formalizing which beliefs d_i conveys. We identify d_i with a principal $Prin(d_i)$ and write an NAL formula $Prin(d_i)$ says $\mathcal{P}$ for each belief $\mathcal{P}$ that d_i conveys.

We represent an excerpt e appearing in a document d as a 4-tuple $e = \langle \chi, d, l, d' \rangle$, where χ is the text of the excerpt, d' is a *source* document to which the excerpt is being attributed, and l is the location where the excerpt appears in d. Notice, distinct appearances of text χ in d are considered to be different excerpts. As with documents, each excerpt e_i can be identified with an NAL principal $Prin(e_i)$, where $Prin(e_i)$ says $\mathcal{P}$ holds for every belief $\mathcal{P}$ that except e_i conveys. Define $src(e)$ to be the source document (i.e., d' above) from which e was purportedly derived; $Prin(src(e))$ is therefore the principal corresponding to $src(e)$.

The reader of an excerpt e and the author of source document $src(e)$ would expect that beliefs conveyed by e are also conveyed by $src(e)$: $\omega(Prin(e)) \subseteq \omega(Prin(src(e)))$ or equivalently $Prin(e) \rightarrow Prin(src(e))$ would therefore hold. But whether $Prin(e) \rightarrow$

$Prin(src(e))$ actually holds will depend on how e was derived from $src(e)$. Quoting too few words, quoting out of context, redaction, elision of words and clauses, all can produce an "excerpt" that conveys different beliefs than are conveyed in the source. We define a document d to have *integrity* if and only if, for every excerpt e appearing in d, the beliefs e conveys are also conveyed by $src(e)$. This property can be formalized in NAL as a credential

$$\texttt{TruDocs says } (\forall e\colon\ e \in d\ \Rightarrow\ Prin(e) \rightarrow Prin(src(e))) \tag{15}$$

that TruDocs issues about d, where relation $e \in d$ holds if and only if document d contains excerpt e.[11]

The author of a document d' cannot be expected to enumerate all possible excerpts e that convey beliefs found in d'. So authors (or the organizations they work for) associate *use policies* with documents they produce. To be eligible for inclusion in another document d, an excerpt e must comply with the use policy associated with $src(e)$. TruDocs limits use policies to those that can be specified as syntactic criteria or as other computable checks whose compliance implies $Prin(e) \rightarrow Prin(d')$, meaning the beliefs expressed by excerpt e are from document d'.

We can associate a use policy with a source document d' by issuing a credential that conveys the NAL formula

$$Prin(d')\,\mathsf{says}(\forall e, d\colon (d' = src(e) \wedge usePol_{d'}(e,d))\ \Rightarrow\ (Prin(e) \rightarrow Prin(d'))), \tag{16}$$

where $usePol_{d'}(e,d)$ is a predicate satisfied if excerpt e appearing in d is consistent with the use policy associated with d'. Credentials like (16) enable (15) to be derived by checking each excerpt e against the use policy for $src(e)$:

$$\texttt{TruDocs says } (\forall e\colon\ e \in d\ \Rightarrow\ usePol_{src(e)}(e,d)). \tag{17}$$

Thus a guard handling a request for the display of a document d can mechanically derive (15) or, conversely, deny a display request if d does not have integrity. Note that (17) is discharged using an analytic basis for trust because authorization depends on a form of analysis: checking use policies.

TruDocs can also handle copyright's "fair use" and other noncomputable use policies by employing an axiomatic basis for trust. One or more human authorities H_i for which TruDocs has issued a credential that conveys

$$\texttt{TruDocs}\,\mathsf{says}(H_i \rightarrow \texttt{TruDocs}) \tag{18}$$

are solicited to check the use policy. H_i in turn provides credentials that convey

$$H_i\,\mathsf{says}(Prin(e) \rightarrow Prin(src(e))) \tag{19}$$

for excerpts e, such that $e \in d$ holds and the use policy is satisfied. Receipt of such a credential for each excerpt e in d is all that is needed for TruDocs to derive (15). So this approach corresponds to deriving (17), where $usePol_{src(e)}(e,d)$ is satisfied if and only if TruDocs has credentials (18) and (19).

Implementation Details. TruDocs comprises an editor TDed for use by document authors, a viewer TDview for displaying documents, and some additional support software.

—TDed allows a document d that contains excerpts to be created, enables a use policy to be defined and associated with that document, and constructs a unique name $nme(d)$ for the document. By construction, $nme(d)$ embodies a validated set of *document particulars*, such as title, author, publication venue, publication date, etc.

[11]Definition (15) treats nested excerpts as if each appears directly in d. Other treatments are possible.

—TDview implements a guard to authorize display requests from documents; a display request for d is granted only if (17) can be derived, since (15) can then be derived from that. Whenever TDview displays a document, it displays at the end of each excerpt e the document particulars embodied in $nme(src(e))$, thereby giving the reader a human-intelligible description for the source document from which e was derived.

TDed and TDview were obtained by modifying the OpenOffice software suite.[12] We added 739 lines of Visual Basic code and 5066 lines of C code to OpenOffice. TDed and TDview also use the NAL proof checker library (an additional 4677 lines of C code) and third-party libraries: OpenSSL (for hashing, signature generation, and signature verification) and XOM (for XML manipulation and canonicalization). Because Nexus does not yet support sophisticated user interfaces, TDview was implemented as two separate components. One component is trusted and runs on the Nexus kernel; it executes the NAL proof checker and an analysis engine. The other component is not trusted and runs on a Linux platform; it displays information and implements the user interface. TDed runs on an untrusted Linux platform.

In order to derive (17) for a document d, the TDview guard enumerates the excerpts in d and processes each excerpt e as follows.

(i) Determine the predicate $usePol_{src(e)}(e, d)$ that applies for each excerpt e in d.
(ii) Check $usePol_{src(e)}(e, d)$ and, if it holds, issue

$$\texttt{TruDocs}\ \mathsf{says}\ usePol_{src(e)}(e, d). \tag{20}$$

Step (ii) is implemented with assistance from the NAL proof checker and built-in support for text matching, as follows:

—TDview checks to see if the display request was accompanied by credentials and/or an NAL proof that discharges (20), and if so, TDview checks that proof, issuing a credential conveying (20) if the proof is sound;
—if not, TDview determines if it has built-in support to validate $usePol_{src(e)}(e, d)$, attempts that validation, and if successful TDview issues a credential conveying (20);
—otherwise, TDview displays an error message that details the use policy that it could not satisfy, requesting additional credentials and/or an NAL proof be provided.

Note that some trust assumptions are required, because of NAL's local reasoning restriction. First, $\texttt{TDview} \rightarrow \texttt{TruDocs}$ must be assumed so that credentials issued by TDview can contribute to the derivation of (20), a statement being attributed to TruDocs. This assumption can be discharged if we take TruDocs to be $H_{CPU}.H_{OS}.H_{\texttt{TDview}}$, making TruDocs synonymous with TDview. Alternatively, we could take TruDocs to be a public key $K_{\texttt{TruDocs}}$ chosen for this purpose by the user or an administrator; we would then have to arrange for the distribution of signed credentials that convey

$$K_{\texttt{TruDocs}}\ \mathsf{says}\ \texttt{TDview} \rightarrow K_{\texttt{TruDocs}}.$$

A second trust assumption we require is that, for each credential $EA_i\ \mathsf{says}\ \mathcal{F}$ provided by an external authority EA_i and used in step (ii), there must be a credential

$$\texttt{TDview}\ \mathsf{says}(EA_i \rightarrow \texttt{TDview})$$

that signifies EA_i is trusted by TDview and, therefore, $\texttt{TDview}\ \mathsf{says}\ \mathcal{F}$ can be derived by TDview from $EA_i\ \mathsf{says}\ \mathcal{F}$. The name of each such trusted external authority EA_i is communicated to TDview at startup.

[12] http://www.openoffice.org/.

Limits in online storage or concerns about confidentiality are just two reasons TDview might not have access to certain source documents. So TDview is not always able to validate $usePol_{src(e)}(e, d)$ directly and might instead have to import credentials from human or external authorities. Moreover, having TDview import credentials can improve performance by undertaking an expensive analysis once rather than each time a document display is requested. For example, when creating a document d, TDed has access to all documents from which excerpts appearing in d are derived. TDed is therefore an obvious place to perform some analysis and issue credentials that later aid TDview in deriving (20). This, however, does require an additional trust assumption: TDed $\rightarrow$ TDview.

TDview currently supports matching an excerpt and source text verbatim or allowing for change of case, replacing fragments of text by ellipses, inserting editorial comments enclosed within square brackets, and limiting the length of individual excerpts or the aggregate length or number of the excerpts from a given document. TDview also can validate compliance with a use policy that stipulates excerpts not appear in documents having certain document particulars—for example, that excerpts not appear in documents authored by a given individual or published in a given venue.

A name $nme(d)$ that lists document particulars would prove problematic if we want to use an ordinary file system and store d as a file named $nme(d)$. So TruDocs associates with each document d a principal named $Hnme(d)$, as follows. Each document d is represented in XML, and we define $Hnme(d) = H(x_d)$, where x_d is the XML representation (using the DocBook[13] standard) for d and where $H(\cdot)$ is a SHA1 hash. $Hnme(d)$, because it is relatively short, can serve as the name for a file storing x_d in a file system or web server. For each excerpt e, TruDocs stores in x_d name $nme(src(e))$, which provides the document particulars for $src(e)$, and name $Hnme(src(e))$, which provides direct access to the file storing $x_{src(e)}$.[14]

A binding between principals $Hnme(d)$ (i.e., $H(x_d)$) and $nme(d)$ is made by TruDocs principal Reg (named by public key K_{Reg}); Reg runs on a separate machine from TDed and TDview. Reg creates bindings, validates document particulars, and disseminates the existence of $Hnme(d)$ to $nme(d)$ bindings by issuing credentials. In particular, a document d created with TDed becomes eligible for view only after the user invokes the publish operation; publish causes pair $\langle x_d, nme(d)\rangle$ to be forwarded to Reg, which checks that

(i) $nme(d)$ is unique,
(ii) $nme(d)$ is consistent with document particulars (e.g., author, title, publication venue, publication date) conveyed in x_d, and
(iii) each document particular in $nme(d)$ is valid according to relevant external authorities (e.g., the authoritative reprints repository maintained by the journal where d is purported to have been published).

If (i)–(iii) hold, then $nme(d)$ is considered validated and Reg generates a credential

$$K_{\text{Reg}} \text{ says } (Hnme(d) \rightarrow K_{\text{Reg}}.nme(d)), \tag{21}$$

which is returned by Reg to TDed, where it is piggybacked[15] on x_d. Notice that, if we define $Prin(d)$ to be $K_{\text{Reg}}.nme(d)$, we can derive

$$Hnme(d) \rightarrow Prin(d), \tag{22}$$

[13] http://www.docbook.org/.

[14] If only name $Hnme(src(e))$ were stored in x_d, then after d has been created, an attacker could change what is stored in file $Hnme(src(e))$, thereby invalidating the consistency of the information from $nme(src(e))$ that gets displayed at the end of e with the document particulars for $src(e)$.

[15] Credential (21) cannot be stored in x_d, because that would change name $H(x_d)$ for that principal, rendering credential (21) useless.

a binding between $Hnme(d)$ and $Prin(d)$: SUBPRIN derives $Hnme(d) \rightarrow K_{\texttt{Reg}}.nme(d)$ from (21) and then use the above definition of $Prin(d)$ to substitute for $K_{\texttt{Reg}}.nme(d)$.

Finally, as noted above, when `TDed` creates a document d', it stores a use-policy credential as part of $x_{d'}$. The credential stored is actually a variant of (16), now that two different principals are associated with each document:

$$Hnme(d')\,\mathsf{says}(\forall e, d\colon\ (d' = src(e) \wedge usePol_{d'}(e,d)) \Rightarrow\ (Prin(e) \rightarrow Hnme(d'))). \tag{23}$$

But $Prin(e) \rightarrow Hnme(d')$ derives $Prin(e) \rightarrow Prin(d')$, since (22) can be derived from the instance of (21) piggybacked on $x_{d'}$. This means that from (21) and (23), `TDview` can always automatically derive

$$H(x_{d'})\,\mathsf{says}(\forall e, d\colon\ (d' = src(e) \wedge usePol_{d'}(e,d)) \Rightarrow\ (Prin(e) \rightarrow Prin(d'))), \tag{24}$$

and the NAL derivation of (15) from (24) is virtually the same as the derivation of (15) from (16), again remaining independent of document d and thus not something the guard of `TDview` must regenerate to authorize each display request.

4.2. `ConfDocs`: A Synthetic Basis for Authorization

`ConfDocs` implements multilevel security [Department of Defense 1985; Weissman 1969] for accessing documents comprising *text elements*. Each text element χ in a document is assigned a *classification label* $\lambda_T(\chi)$ by some trusted *classification authority* T; each human user H is assigned a *clearance* $\lambda_U(H)$ by some trusted *clearance authority* U. And each document d is identified with with a unique principal $Prin(d)$.

Classification labels and clearances are selected from a set of *security labels* on which a partial order relation $\preceq$ has been defined. A document d comprising a set $txt(d)$ of text elements is authorized for display to a user H if and only if

$$Prin(d)\,\mathsf{says}(\forall \chi \in txt(d)\colon\ \lambda_T(\chi) \preceq \lambda_U(H)) \tag{25}$$

holds. Policy (25) makes d—or, rather, the publisher of d—the ultimate authority on which users can read d, by leaving the choice of classification authority and clearance authority with d. In particular, the choice of classification authority determines the value of $\lambda_T(\chi)$ and the choice of clearance authority determines the value of $\lambda_U(H)$, so these choices (albeit indirectly) effect whether H satisfies (25).

`ConfDocs` is agnostic about the set of security labels and partial order relation $\preceq$. The system simply requires the means (internally built-in or by appeal to an external authority) to determine whether $\texttt{L} \preceq \texttt{L}'$ holds for any pair of security labels `L` and `L'`. `ConfDocs` has built-in support for security labels structured as pairs [Denning 1976; Sandhu 1993], where the first element of the pair designates a sensitivity level `U` (unclassified), `C` (confidential), `S` (secret), or `TS` (top-secret), and the second element of the pair designates a set of compartments constructed from descriptors, such as `crypto`, `nuclear`, etc. There is a strict total order $\sqsubset$ on the levels ($\texttt{U} \sqsubset \texttt{C}$, $\texttt{C} \sqsubset \texttt{S}$, and $\texttt{S} \sqsubset \texttt{TS}$), the usual partial order $\subseteq$ on sets of compartments, and

$$\langle lvl, cmpt\rangle \preceq \langle lvl', cmpt'\rangle$$

holds if and only if $lvl \sqsubseteq lvl'$ and $cmpt \subseteq cmpt'$ hold.

If a document d does not satisfy authorization policy (25) for a given user H, then it is often possible to derive a document that does.

—Deleting text from d narrows the scope of the universal quantification in (25) by removing a text element χ from $txt(d)$, thereby eliminating an obligation $\lambda_T(\chi) \preceq \lambda_U(H)$ that could not be discharged.

—Modifying d (say, by changing certain prose in a text element χ to obtain χ') could change the contents of $txt(d)$ in a way that transforms an obligation $\lambda_T(\chi) \preceq \lambda_U(H)$ that could not be discharged into one $\lambda_T(\chi') \preceq \lambda_U(H)$ that can be.

Each implements a synthetic basis for authorization, and our `ConfDocs` prototype supports both.

Implementation Details. `ConfDocs` provides a program `CDview` for viewing documents and provides some shell scripts for creating documents. `CDview` is 5787 of C code that runs on Nexus and uses of the NAL proof checker library. Shell scripts (175 lines of Bash) that invoke the OpenSSL library to perform encryption allow a user (as detailed below) to attach policies to documents and then encrypt the result for subsequent use by `CDview`.

Each `ConfDocs` document d is represented using XML according to the DocBook standard. The representation for a document d includes set $txt(d)$ of text elements, as well as credentials that give a classification label $\mathtt{L}_\chi$ for each text element $\chi \in txt(d)$:

$$Prin(d)\ \mathsf{says}\ (\lambda_T(\chi) = \mathtt{L}_\chi) \quad \text{or} \quad CA_T\ \mathsf{says}\ (\lambda_T(\chi) = \mathtt{L}_\chi).$$

Here CA_T is a classification authority; credentials it issues must be accompanied by a suitable restricted delegation

$$Prin(d)\ \mathsf{says}\ CA_T \xrightarrow{v_1, v_2 : \lambda_T(v_1) = v_2} Prin(d), \tag{26}$$

attesting that the publisher of d trusts CA_T to assign classification labels to text elements in d.

The representation of d optionally may include *sanitization credentials*

$$San\ \mathsf{says}\ (\lambda_T(edit(\chi, s)) = \mathtt{L}_{edit(\chi, s)}) \tag{27}$$

that give a classification label for the text element produced by executing a built-in edit function to modify χ according to script s. Here, San is either $Prin(d)$ or some classification authority CA_T for which restricted delegation (26) appears in the representation of d. Script s comprises standard text editor commands like `Replace`(x, y), which replaces all instances of character string x with string y, and so on.

Credentials like (27) define a *sanitization policy*. Such a policy characterizes ways to transform a document containing information that readers are not authorized to access into a document those readers are. The hard part is resolving the tension between hiding too much and indirectly leaking classified information. Sanitization of paper documents, for example, often involves replacing fragments of text with white space but a document sanitized in this manner might still leak information to a reader by revealing the length of a replaced name or the existence of an explanatory note.

A user H attempting to view a document d invokes `CDview`, furnishing a credential signed by some clearance authority CA_U that attests to $\lambda_U(H)$:

$$CA_U\ \mathsf{says}\ \lambda_U(H) = \mathtt{L}_\mathtt{H}.$$

Not all clearance authorities are equivalent. The publisher of d controls whether a clearance authority CA_U is trusted to assign clearances and, thus, can participate in determining which users have access to d. Specifically, the publisher includes a credential

$$Prin(d)\ \mathsf{says}\ CA_U \xrightarrow{v_1, v_2 : \lambda_U(v_1) = v_2} Prin(d)$$

in the `ConfDocs` representation of d for each clearance authority CA_U that is trusted.

Nexus Sealed Bundles. To ensure that `CDview` is the only way to view documents, they are stored and transmitted in encrypted form. Nexus, in conjunction with a TPM secure coprocessor, implements a storage abstraction that is ideal for this task. A Nexus *sealed bundle b* comprises (i) a payload $payload(b)$ stored in encrypted form and (ii) an NAL group $Group(b)$ of constituents authorized to decrypt $payload(b)$.

By invoking the Nexus `decrypt`(b) kernel operation, a principal A is seen by the Nexus kernel (which, by design, knows the identity of the process it is executing) to be providing the credential

$$A \text{ says } \texttt{decrypt}(b),$$

although this credential is never actually generated and thus cannot be stolen for use by some other principal. Nexus responds by decrypting and returning $payload(b)$ to A if and only if authorization policy

$$Group(b) \text{ says } \texttt{decrypt}(b)$$

can be derived. To allow an access thus requires the kernel to verify a proof of $A \rightarrow Group(b)$, thereby establishing that A is among $Group(b)$ constituents. The kernel discharges this obligation by checking whether A satisfies an NAL formula $\mathcal{P}_b[v := A]$, where $\mathcal{P}_b$ was originally provided for defining $Group(b)$ and saved in the bundle; $A \rightarrow Group(b)$ then follows due to MEMBER. Our implementation also allows A to provide a proof of $A \rightarrow C$, where C is some other principal; the kernel would validate that proof and then check that $\mathcal{P}_b[v := C]$ is satisfied. Notice, the set of principals satisfying $\mathcal{P}_b$ is not necessarily static if $\mathcal{P}_b$ depends on state, and therefore the $Group(b)$ constituents may be dynamic.

Each `ConfDocs` document d is stored using a Nexus bundle b_d, where $Group(b_d)$ is a fixed set of principals corresponding to valid instances of `CDview`. A program is considered a valid copy of `CDview` if and only if it's hash equals the hash $H_{\texttt{CDview}}$ of some predetermined correct object code for `CDview`, that object code was loaded and is being executed by a Nexus process running on a valid Nexus kernel, and the Nexus kernel is itself executing on a trusted processor with associated TPM. Such a principal is specified using NAL subprincipals as $K_{CPU}.H_{Nexus}.process_{23}.H_{\texttt{CDview}}$ because `CDview` is actually being materialized by some Nexus process (here, $process_{23}$), which in turn is being materialized by the Nexus, which itself is materialized by the hardware processor.[16] And the group $Group(b_d)$ of principals for each document d is defined in NAL as

$$\{\!|v : (\exists i : \ \ v \rightarrow K_{CPU}.H_{Nexus}.process_i.H_{\texttt{CDview}})|\!\}.$$

5. DISCUSSION

Genesis of NAL. Our original plan for Nexus was to adopt—not adapt—prior work in credentials-based authorization. The Lampson et al. [1992] account (which introduced says and $\rightarrow$ operators) seemed to offer a compelling framework for the kinds of authorization Nexus was going to support, had been formalized by Abadi et al. [1993] as a logic, and was used in the Taos operating system [Wobber et al. 1994]. There was the matter of generating proofs and checking them—Taos had implemented only a decidable subset of the logic. Appel and Felten's [1999] proof-carrying authentication addressed that, suggesting that all requests be accompanied by proofs and that guards perform only proof checking. Moreover, proof-carrying authentication employed

[16]To simplify the exposition, above we name principals using only program names rather than giving the fully qualified list of subprincipals (hardware key, kernel, process) that actually defines the principal's name.

a higher-order logic, so it supported application-specific predicates; and it was implemented in Twelf [Pfenning and Schürmann 1999], so a proof checker was available.

A clear focus of this prior work was authentication for the varied and nuanced principals found in distributed systems. Operators to construct new principals (e.g., roles, quoting, etc.) were central to that enterprise. In Nexus, system state and properties of principals were going to be important inputs to authorization, too. We embarked on a series of design exercises to see how well those needs would be served by the prior work.

Our attempt to design a simple digital rights management (DRM) system was particularly instructive. We sought flexibility in what should count as an access to the managed content (e.g., accessing any part vs. accessing a majority vs. accessing all of the content). A system designer would presumably record accesses by changing the system's state. So we concluded that a logic for credentials and authorization policies ought to include state predicates.

However, adding arbitrary state predicates to an authentication logic is subtle. If stand-alone state predicates can be formulas then an inconsistency would have a far-reaching effect by allowing false to be derived, and hence any authorization policy to be satisfied. We thus restricted state predicates to appearing only in worldviews of principals. Since it is unrealistic to expect that every principal could evaluate every state predicate or that different principals evaluating the same state predicate at different times would compute the same value, we needed a way for one principal to include in its worldview a state predicate p evaluated by some other principal.

—One approach [Abadi et al. 1993; Appel and Felten 1999; Howell 2000] is to use SAYS-I along with a new inference rule

$$\text{CNTRL:}\ \frac{A\ \mathsf{says}\ p,\quad \mathsf{controls}(A,\ p)}{p},$$

where $\mathsf{controls}(A,\ p)$ holds if A is considered a trusted source regarding the truth of p.

—The other [Abadi 2007, 2008] is to postulate a local-reasoning restriction and require that principals use delegation to import and reason about beliefs from others.

We rejected the first approach because it offers fewer guarantees about the propagation of inconsistencies, and it also requires characterizing sets of state predicates p' covered by $\mathsf{controls}(A,\ p)$: if $\mathsf{controls}(A,\ p)$ holds and $p \Rightarrow p'$ is valid then is A necessarily also trusted on p'? Is A necessarily trusted on $\neg p$?

CDD [Abadi 2007, 2008], which had been subject to careful analysis and embraced a local-reasoning restriction, then became an obvious candidate for the foundation of NAL. Moreover, CDD left unspecified details about principals and beliefs, so it offered us freedom to define principals that would match what Nexus provided and to use state predicates in beliefs (with theories that interpret these state predicates).

NAL subprincipals are derived from named roles in Alpaca [Lesniewski-Laas et al. 2007]. Prior proposals (e.g., SDSI/SPKI [Rivest and Lampson 1996] and Taos [Wobber et al. 1994]) had restricted the term τ used in defining a subprincipal $A.\tau$ to being a fixed string, which meant that only static roles could be supported. By allowing τ to be any term, the identity of an NAL subprincipal can be state-dependent.

Groups in NAL are a special case of the *dynamic unweighted threshold structures* defined by Delegation Logic [Li et al. 2003]. And Delegation Logic was the first to suggest that group membership be specified intensionally, although no proof rules were given (nor were they needed) there. Our approach to authorization requires proof rules for satisfying authorization policies from credentials; with inference rules MEMBER and $\rightarrow$ GROUP, NAL appears to be the first logic for reasoning about such groups. The deductive

closure semantics we selected for NAL groups was first proposed in Abadi et al. [1993] along with an axiomatization for extensionally defined instances of such groups.

Other semantics for groups have been proposed. With the *or-groups* of Syverson and Stubblebine [1999], which are also supported in proof-carrying authentication [Appel and Felten 1999], a belief is considered to be in the worldview of a group if and only if that belief is in the worldview of some[17] group member; or-groups are not sound with respect to IMP-E and therefore would require different proof rules from other NAL principals. In groups with *conjunctive* semantics (sometimes called *conjunctive principals* [Abadi et al. 1993; DeTreville 2002; Ellison et al. 1999; Li et al. 2003] or *and-groups* [Syverson and Stubblebine 1999]), a belief appears in the worldview of a group if and only if that belief appears in the deductive closure of the intersection of the worldviews for all members. We conjecture that conjunctional groups could be supported in NAL as the following abbreviation:

$$\langle\!\langle v : \mathcal{P} \rangle\!\rangle \text{ says } \mathcal{F}\text{:} \quad (\forall v : \mathcal{P} \Rightarrow (v \text{ says } \mathcal{F})).$$

Finally, various proposals (e.g., Ellison et al. [1999] and Li et al. [2003]) have been made for groups that exhibit *k threshold* semantics, whereby a belief is in the worldview of the group if and only if that belief is in the worldviews of at least k group members. This construct is quite expressive, difficult to axiomatize, and (fortunately) has not been needed for the applications we explored.

We were not the first to see a need for state in an authentication logic. As soon as support for revocation or expiration of credentials is contemplated, the need for state-dependent credentials and policies becomes apparent. In Becker and Nanz [2007], credentials and policies can have side effects that involve the addition or removal of assertions from the local rule base; Cassandra [Becker and Sewell 2004] represents state in terms of the activation and deactivation of roles; and linear logics [Garg et al. 2006; Bowers et al. 2007] encode state information in terms of how many times an axiom can be used. These encodings all duplicate in the logic state that already exists in a system. Expressiveness is often lost in the translation, preventing certain policies from being formalized. Moreover, in this prior work, either some sort of globally available state is being assumed, which becomes difficult to implement in a distributed system, or the state is local to a guard, which limits what authorization policies could be implemented.

Other Related Work. PolicyMaker [Blaze et al. 1996, 1998, 1999] was the first authorization scheme to focus on considerations of trust as an input to authorization decisions.[18] Policies, credentials, and trust relationships are expressed in PolicyMaker as imperative programs in a safe language; a generic compliance checker interprets these programs to determine whether a policy is satisfied given the provided credentials and trust assumptions. REFEREE [Chu et al. 1997], designed to support Web applications, extends this approach by supporting policies about credential-fetching and signature verification; KeyNote [Blaze et al. 1998] adds restrictions to make compliance checking efficient; and Delegation Logic [Li et al. 2003] replaces PolicyMaker's imperative programs with D1LP, a monotonic version of Datalog that has declarative semantics and can be compiled into ordinary logic programs (e.g., Prolog).

SD3 [Jim 2001], Binder [DeTreville 2002], the RT family of logics [Li et al. 2002], Cassandra [Becker and Sewell 2004], Soutei [Pimlott and Kselyov 2006], and

[17]Some authors have referred to such as groups as implementing *disjunctive* semantics, but this term has been used by other authors to describe groups that have the semantics defined by NAL, which also requires a deductive closure.

[18]However, considerations about trust are the basis for the definitions of groups and roles in prior work on access control.

SecPAL [Becker et al. 2007] all employ languages based on Datalog; the result is a tasteful compromise between the efficient decision procedures that come with PolicyMaker's imperative programs and the declarative elegance of the Abadi et al. [1993] access control calculus.

SecPAL, which targets grid computing environments and has also been used for authorization in a weakly consistent peer-to-peer setting [Wobber et al. 2009], is quite expressive despite limitations inherent in Datalog. It supports delegation credentials that are contingent on the evaluation of predicates over a guard's local state. And, unlike other authorization schemes based on logic programming, SecPAL allows negations of the form $\neg(A \text{ says } \mathcal{F})$ to appear within policies (but not credentials); syntactic constraints on credentials and policies nevertheless guarantee policy checking is sound, complete, and always terminates, under the assumption (which unfortunately can be violated by a denial-of-service attack) that all credentials are available whenever a policy is evaluated. A tractable decision procedure for authorization was obtained by translating from SecPAL into a Datalog variant (viz. Datalog with Constraints).

DKAL [Gurevich and Neeman 2008] introduces a new dimension to credentials-based authorization by extending SecPAL to prevent any sensitive information carried in credentials and authorization policies from leaking, even when users that have different clearances share the same underlying authorization policies, database of credentials, and implementation.

Alpaca [Lesniewski-Laas et al. 2007], like NAL, builds on proof-carrying authentication [Appel and Felten 1999]. However, the domain of applications for Alpaca—unifying and generalizing public-key infrastructures (PKIs) to support authentication—is quite different from NAL's goal of supporting authorization. And that explains differences in focus and function. Alpaca *authorities* (different from NAL authorities), for example, provide a structure to localize reasoning associated with a given logical theory; this turns out to be convenient in Alpaca for dealing with the mathematical operations and coercions used in authentication protocols. NAL and other logics that are dependent on signatures and hashes for attributing beliefs to principals do not provide support for reasoning about these operations within the logic. Another important point of difference is that Alpaca—unlike NAL—has only limited support for stateful protocols. Nonces can be used in Alpaca to achieve one-use or limited-use credentials; there is no way, however, to use Alpaca for protocols that depend in general on history, as would be required (and is supported in NAL) for DRM or even as needed for implementing many authentication protocols.

Relatively few systems—most, research prototypes—support credentials-based authorization, but none do so in anything that approaches the generality needed for using analytic or synthetic bases in authorization. This prior work includes Taos and SecPAL, which were already mentioned; the W3C Web Services WS-Security [Organization for the Advancement of Structured Information Standards (OASIS) 2004] standard (in particular, WS-Policy [World Wide Web Consortium 2007]) is also rooted in this general approach, and that could bode well for the future. Bauer [2003] used proof-carrying authorization for access control to web pages. The Grey Project [Bauer et al. 2005a, 2008] integrates a linear logic and proof-carrying authentication on a smart phone platform, and it has been used for authorizing access to offices and shared labs. And Howell and Kotz [2000] implemented a credentials-based approach for use within and between applications running in Java virtual machines; that logic is an extension of SPKI [Ellison et al. 1999].

6. CONCLUDING REMARKS

This article describes NAL, a logic for specifying credentials and authorization policies. Novelty was not a design goal of NAL—simplicity was. So instead of designing NAL

from scratch, we started with an existing bare-bones authorization logic (CDD) that abstracts the essence of such logics, and we instantiated its notion of principals and its underlying predicate logic. Then, by building a suite of document-viewer applications, we demonstrated that NAL, despite its simplicity, is expressive and convenient enough to be a practical basis for implementing authorization in real systems.

NAL also provided a vehicle for us to understand and bridge the gap between what authorization logics provide and real systems. The implementation of credentials and of principal names is one area where such a gap often exists. There are, for example, significant practical differences between credentials implemented by digital signatures, by hashes, and by ordinary messages on authenticated channels. These differences include the cost to create and validate credentials, whether secrets must be stored or shared, whether certain memory must be accessible to the credential holders, and whether the credential can be forwarded. Only in contemplating authorization for real applications did these differences become apparent and did the design tradeoffs they enable become clear.

An imperative in the design of NAL and in our approach to supporting authorization was to empower system designers with flexibility for defining policies and implementing guards. This caused us to resist adding to NAL special-purpose constructs that shape policy by directly supporting revocation of credentials or by enforcing bounds on credential usage. Such constructs are only one way to create an authorization logic that is inherently nontemporal for use in a setting, like a computer system, where principals' beliefs actually do change over time. A system designer—informed by the semantics of an application—should know the best means for handling changes in principals' beliefs and, therefore, should be given the flexibility to implement that means. The state predicates that can appear in NAL formulas provide that flexibility, because changes in principals' beliefs are necessarily correlated with changes in system state.

NAL's flexibility also led us to take a very different view about the role of guards (or reference monitors) in systems. We proposed in this article that guards be seen as testing requester trustworthiness, where the authorization policy enforced by a guard defines the criteria by which requester trustworthiness is evaluated. Identity-based and reputation-based authorization illustrate an axiomatic basis for deciding whether a requester is trustworthy, and this basis is widely used in practice. We argued in this article, however, that analytic and synthetic bases are also worth supporting; and our document-viewer applications illustrated the power and convenience of these. Note, our thesis that authorization be viewed not as an end to itself but rather as a proxy for a trustworthiness test is not limited to NAL or even to authorization logics.

Since credentials convey attributes of principals, any approach to authorization that makes decisions based on credentials could impinge on the privacy of individuals.[19] This risk is formulated succinctly by Cameron [2005] in his second law, as stated here.

> *Minimal Disclosure for a Constrained Use.* The solution that discloses the least amount of identifying information and best limits its use is the most stable long-term solution.

Viewed though this lens, identity-based authorization can be problematic when a requester does not have the flexibility to create different identities for different kinds of requests. The easy route—a single identity for each individual—creates the greatest opportunity for privacy compromise. With the analytic and synthetic bases, the focus changes to determining what attributes must be known for an authorization decision. The easy route here is to ask for less, and that bias helps with privacy.

[19]We define *privacy* to be the right of an individual to control the dissemination and use of information about that individual.

Authorization that favors privacy is not only good for people but also turns out to be good for system security. A *privilege* imparts a special right to perform some task. Hence, a credential can be seen as conveying a kind of privilege, and logical implication defines a partial order on these privileges: if $\mathcal{C} \Rightarrow \mathcal{C}'$ holds, then a credential that conveys $\mathcal{C}$ is considered stronger than one that conveys $\mathcal{C}'$. Notice that disclosure of stronger credentials is more likely to violate privacy. And recall the well-known Saltzer–Schroeder [Saltzer and Schroeder 1975] guidelines for building secure systems, as stated here.

Principle of Least Privilege. Assign each principal the minimum privileges it needs to accomplish its task.

The Saltzer–Schroeder mandate thus offers the same guidance as Cameron's second law, above; security and privacy are both well served by using weaker credentials.

APPENDIX: NAL INFERENCE RULES

NAL's axiomatization is similar to CDD [Abadi 2007, 2008], augmented with rules for subprincipals and groups, and with standard rules for quantification in a predicate calculus [van Dalen 2004; Troelstra and van Dalen 1988]. The SAYS-I rule of NAL is called UNIT in CDD; and Abadi has shown that CDD's BINDM axiom is equivalent to NAL's SAYS-E (also known as C4) in the presence of SAYS-I and DEDUCE (both of which are present in NAL). We assume, but do not show, rules for variable renaming and substitution.

The derivation of any NAL formula $\mathcal{F}$ can be represented as a proof tree whose nodes correspond to NAL formulas.

—Leaves correspond to axioms and assumptions. Each assumption has a unique label Λ_i.
—Each internal node in the tree corresponds to the conclusion $\mathcal{G}$ of some NAL inference rule. The formulas that correspond to the node's immediate predecessors are the premises needed to conclude $\mathcal{G}$ using the rule.
—The root of the tree corresponds to $\mathcal{F}$.

Rules PROP-FORALL-I, FORALL-I, and EXISTS-E below involve side conditions that refer to "uncanceled assumptions." In a proof tree that derives $\mathcal{F}$, an assumption $\mathcal{A}$ with label Λ_i is defined to be *canceled* if and only if any path from the node that corresponds to $\mathcal{F}$ to the node that corresponds to $\mathcal{A}$ passes through a node derived by applying inference rule IMP-I(Λ_i); otherwise $\mathcal{A}$ is considered *uncanceled*.

A.1. NAL Rules from Constructive Predicate Logic.

$$\text{TRUE:}\ \frac{}{\mathsf{true}} \qquad \text{IMP-E:}\ \frac{\mathcal{F}\ ,\ \mathcal{F} \Rightarrow \mathcal{G}}{\mathcal{G}} \qquad \text{IMP-I}(\Lambda_i)\text{:}\ \frac{\begin{array}{c}\Lambda_i:\ \overline{\mathcal{F}} \\ \vdots \\ \mathcal{G}\end{array}}{\mathcal{F} \Rightarrow \mathcal{G}}$$

$$\text{AND-I:}\ \frac{\mathcal{F}\ ,\ \mathcal{G}}{\mathcal{F} \wedge \mathcal{G}} \qquad \text{AND-LEFT-E:}\ \frac{\mathcal{F} \wedge \mathcal{G}}{\mathcal{F}} \qquad \text{AND-RIGHT-E:}\ \frac{\mathcal{F} \wedge \mathcal{G}}{\mathcal{G}}$$

$$\text{OR-LEFT-I:}\ \frac{\mathcal{F}}{\mathcal{F} \vee \mathcal{G}} \qquad \text{OR-RIGHT-I:}\ \frac{\mathcal{G}}{\mathcal{F} \vee \mathcal{G}} \qquad \text{OR-E:}\ \frac{\mathcal{F} \Rightarrow \mathcal{H}\ ,\ \mathcal{G} \Rightarrow \mathcal{H}\ ,\ \mathcal{F} \vee \mathcal{G}}{\mathcal{H}}$$

$$\text{PROP-FORALL-I:}\ \frac{\mathcal{F}}{(\forall \mathsf{x} : \mathcal{F})}$$ x is not free in any uncanceled assumptions in the derivation of $\mathcal{F}$

$$\text{PROP-FORALL-E:}\ \frac{(\forall \mathsf{x} : \mathcal{F})}{\mathcal{F}[\mathsf{x} := \mathcal{G}]}$$ free variables of formula $\mathcal{G}$ are free for x in $\mathcal{F}$

$$\text{FORALL-I:}\ \frac{\mathcal{F}}{(\forall v : \mathcal{F})}$$ v is not free in any uncanceled assumptions in the derivation of $\mathcal{F}$

$$\text{FORALL-E:}\ \frac{(\forall v : \mathcal{F})}{\mathcal{F}[v := \tau]}$$ free variables of term τ are free for v in $\mathcal{F}$

$$\text{EXISTS-I:}\ \frac{\mathcal{F}[v := \tau]}{(\exists v : \mathcal{F})}$$ free variables of term τ are free for v in $\mathcal{F}$

$$\text{EXISTS-E:}\ \frac{\mathcal{F} \Rightarrow \mathcal{G}\ ,\ (\exists v : \mathcal{F})}{\mathcal{G}}$$ v is not free in $\mathcal{G}$ or in any uncanceled assumptions in the derivation of $\mathcal{F} \Rightarrow \mathcal{G}$

A.2. NAL Rules Derived from CDD.

$$\text{DEDUCE:}\ \frac{A\,\mathsf{says}(\mathcal{F} \Rightarrow \mathcal{G})}{(A\,\mathsf{says}\,\mathcal{F}) \Rightarrow (A\,\mathsf{says}\,\mathcal{G})}$$

$$\text{SAYS-I:}\ \frac{\mathcal{F}}{A\,\mathsf{says}\,\mathcal{F}} \qquad \text{SAYS-E:}\ \frac{A\,\mathsf{says}(A\,\mathsf{says}\,\mathcal{F})}{A\,\mathsf{says}\,\mathcal{F}}$$

A.3. NAL Extensions.

$$\text{SUBPRIN:}\ \frac{}{A \to A.\tau}$$

$$\text{EQUIV SUBPRIN:}\ \frac{\tau_1 = \tau_2}{A.\tau_1 \to A.\tau_2}$$

$$\text{MEMBER:}\ \frac{\mathcal{P}[v := A]}{A \to \{\!|v : \mathcal{P}|\!\}}$$ free variables of A are free for v in $\mathcal{P}$

$$\to\ \text{GROUP:}\ \frac{(\forall v : \mathcal{P} \Rightarrow (v \to A))}{\{\!|v : \mathcal{P}|\!\} \to A}$$ v is not free in A

A.4. NAL Derived Inference Rules.

$$\text{FALSE:}\ \frac{\mathsf{false}}{\mathcal{F}}$$

$$\rightarrow\ \text{TRANS:}\ \frac{A \rightarrow B\ ,\ B \rightarrow C}{A \rightarrow C} \qquad \xrightarrow{\overline{v}:\mathcal{F}}\ \text{TRANS:}\ \frac{A \xrightarrow{\overline{v}:\mathcal{F}} B\ ,\ B \xrightarrow{\overline{v}:\mathcal{F}} C}{A \xrightarrow{\overline{v}:\mathcal{F}} C}$$

$$\text{HAND-OFF:}\ \frac{B\ \mathsf{says}(A \rightarrow B)}{A \rightarrow B} \qquad \text{REST-HAND-OFF:}\ \frac{B\ \mathsf{says}(A \xrightarrow{\overline{v}:\mathcal{F}} B)}{A \xrightarrow{\overline{v}:\mathcal{F}} B}$$

$$\text{GROUP MONOTONICITY:}\ \frac{(\forall v : \mathcal{P} \Rightarrow \mathcal{P}')}{\{\!|v : \mathcal{P}|\!\} \rightarrow \{\!|v : \mathcal{P}'|\!\}}$$

ACKNOWLEDGMENTS

Martin Abadi has been an extremely helpful sounding board throughout the evolution of this work, and he brought GROUP MONOTONICITY to our attention. In addition, he, Lujo Bauer, Willem de Bruijn, Michael Clarkson, Robert Constable, Joe Halpern, Andrew Myers, and Ted Wobber gave us useful feedback on an early draft of this article. The three TISSEC reviewers provided prompt and very thoughtful feedback on our initial submission. We are very grateful to them all.

REFERENCES

ABADI, M. 2007. Access control in a core calculus of dependency. *Electron. Notes Theoret. Comp. Sci. 172*, 5–31.

ABADI, M. 2008. Variations in access control logic. In *Deontic Logic in Computer Science*. Lecture Notes in Computer Science, vol. 5076, Springer, Berlin, Germany, 96–109.

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst. 15,* 4, 706–734.

APPEL, A. W. AND FELTEN, E. W. 1999. Proof-carrying authentication. In *Proceedings of the Annual ACM Computer and Communications Security*. ACM Press, New York, NY, 52–62.

BAUER, L. 2003. Access control for the Web via proof-carrying authorization. Ph.D. dissertation. Princeton University, Princeton, NJ.

BAUER, L., CRANOR, L., REEDER, R. W., REITER, M. K., AND VANIEA, K. 2008. A user study of policy creation in a flexible access-control system. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 543–552.

BAUER, L., GARRISS, S., MCCUNE, J. M., REITER, M. K., ROUSE, J., AND RUTENBAR, P. 2005a. Device-enabled authorization in the Grey system. In *Proceedings of the Information Security Conference*. 431–445.

BAUER, L., GARRISS, S., AND REITER, M. K. 2005b. Distributed proving in access-control systems. In *Proceedings of the IEEE Conference on Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA, 81–95.

BECKER, M., FOURNET, C., AND GORDON, A. 2007. Design and semantics of a decentralized authorization language. In *Proceedings of the IEEE Conference on Computer Security Foundations*. IEEE Computer Society Press, Los Alamitos, CA, 3–15.

BECKER, M. Y. AND NANZ, S. 2007. A logic for state-modifying authorization policies. In *Proceedings of the European Symposium on Research in Computer Security*. 203–218.

BECKER, M. Y. AND SEWELL, P. 2004. Cassandra: Flexible trust management, applied to electronic health records. In *Proceedings of the IEEE Conference on Computer Security Foundations*. IEEE Computer Society Press, Los Alamitos, CA, 139–154.

BERSHAD, B. N., SAVAGE, S., PARDYAK, P., SIRER, E. G., FIUCZYNSKI, M. E., BECKER, D., CHAMBERS, C., AND EGGERS, S. 1995. Extensibility, safety, and performance in the SPIN operating system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. ACM Press, New York, NY, 267–283.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. D. 1999. The role of trust management in distributed systems security. *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*. Lecture Notes in Computer Science, vol. 1603. Springer, Berlin, Germany, 185–210.

BLAZE, M., FEIGENBAUM, J., AND KEROMYTIS, A. D. 1998. KeyNote: Trust management for public-key infrastructures. In *Proceedings of the Security Protocols Workshop*. 59–63.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized trust management. In *Proceedings of the IEEE Conference on Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA, 164–173.

BLAZE, M., FEIGENBAUM, J., AND STRAUSS, M. 1998. Compliance checking in the PolicyMaker trust management system. In *Financial Cryptography*. Springer-Verlag, Berlin, Germany, 254–274.

BOWERS, K. D., BAUER, L., GARG, D., PFENNING, F., AND REITER, M. K. 2007. Consumable credentials in logic-based access-control systems. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, Reston, VA, 143–157.

CAMERON, K. 2005. The laws of identity. `http://www.identitybloc.com/`.

CHU, Y.-H., FEIGENBAUM, J., LAMACCHIA, B., RESNICK, P., AND STRAUSS, M. 1997. REFEREE: Trust management for Web applications. *World Wide Web J. 2,* 3, 127–139.

DENNING, D. E. 1976. A lattice model of secure information flow. *Commun. ACM 19,* 5, 236–243.

DEPARTMENT OF DEFENSE. 1985. Trusted computer security evaluation criteria (TCSEC), DoD 5200.28-STD. `http://csrc.nist.gov/publications/history/dod85.pdf`.

DETREVILLE, J. 2002. Binder, a logic-based security language. In *Proceedings of the IEEE Conference on Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA, 105–113.

ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. 1999. SPKI certificate theory. Internet Engineering Task Force. RFC 2693. `www.ietf.org`.

ERLINGSSON, Ú. AND SCHNEIDER, F. B. 1999. SASI enforcement of security policies: A retrospective. In *Proceedings of the New Security Paradigms Workshop*. ACM Press, New York, NY, 87–95.

GARG, D., BAUER, L., BOWERS, K., PFENNING, F., AND REITER, M. 2006. A linear logic of authorization and knowledge. In *Proceedings of the European Symposium on Research in Computer Security*. Springer-Verlag, Berlin, Germany, 297–312.

GARG, D. AND PFENNING, F. 2006. Non-interference in constructive authorization logic. In *Proceedings of the IEEE Conference on Computer Security Foundations*. IEEE Computer Society Press, Los Alamitos, CA, 283–296.

GOLDBERG, I., WAGNER, D., THOMAS, R., AND BREWER, E. A. 1996. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the Usenix Security Symposium*.

GRAY, C. AND CHERITON, D. 1989. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*. ACM Press, New York, NY, 202–210.

GUREVICH, Y. AND NEEMAN, I. 2008. DKAL: Distributed-knowledge authorization language. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*. IEEE Computer Society Press, Los Alamitos, CA, 149–162.

HAMLEN, K. W., MORRISETT, G., AND SCHNEIDER, F. B. 2006. Certified in-lined reference monitoring on .NET. In *Proceedings of the ACM Workshop on Programming Languages and Analysis for Security*. ACM Press, New York, NY, 7–16.

HOWELL, J. 2000. Naming and sharing resources across administrative boundaries. Ph.D. dissertation. Dartmouth College, Hanover, NH.

HOWELL, J. AND KOTZ, D. 2000. End-to-end authorization. In *Operating System Design & Implementation*. USENIX Association, Berkeley, CA, 151–164.

JIM, T. 2001. SD3: A trust management system with certified evaluation. In *Proceedings of the IEEE Conference on Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA, 106–115.

LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. 1992. Authentication in distributed systems: Theory and practice. *ACM Trans. Comp. Syst. 10*, 265–310.

LESNIEWSKI-LAAS, C., FORD, B., STRAUSS, J., MORRIS, R., AND KAASHOEK, M. F. 2007. Alpaca: Extensible authorization for distributed services. In *Proceedings of the ACM Conference on Computer and Communications Security*. ACM Press, New York, NY, 432–444.

LI, N., GROSOF, B. N., AND FEIGENBAUM, J. 2003. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inform. Syst. Sec. 6*, 128–171.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a role-based trust-management framework. In *Proceedings of the IEEE Conference on Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA, 114–130.

NECULA, G. C. 1997. Proof-carrying code. In *Proceedings of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, 106–119.

ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS). 2004. Web services security: SOAP message security 1.0 (WS-Security 2004). `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-se%curity-1.0.pdf`.

PFENNING, F. AND SCHÜRMANN, C. 1999. System description: Twelf—a meta-logical framework for deductive systems. In *Proceedings of the International Conference on Automated Deduction*. Springer-Verlag, Berlin, Germany, 202–206.

PIMLOTT, A. AND KSELYOV, O. 2006. Soutei, a logic-based trust management system, system description. In *Proceedings of the 8th International Symposium on Functional and Logic Programming* (FLOPS), M. Hagiya and P. Wadler, Eds. Lecture Notes in Computer Science, vol. 3945. Springer, Berlin, Germany, 130–145.

RIVEST, R. AND LAMPSON, B. 1996. SDSI—a simple distributed security infrastructure. `http://theory.lcs.mit.edu/cis/sdsi.html`.

SALTZER, J. H. AND SCHROEDER, M. D. 1975. The protection of information in computer systems. *Proc. IEEE 63,* 9, 1278–1308.

SANDHU, R. S. 1993. Lattice-based access control models. *IEEE Comp. 26,* 11, 9–19.

SCHNEIDER, F. B., WALSH, K., AND SIRER, E. G. 2009. Nexus authorization logic (NAL): Design rationale and applications. Tech. rep. Cornell University, Ithaca, NY, `http://hdl.handle.net/1813/13679`.

SHIEH, A., WILLIAMS, D., SIRER, E. G., AND SCHNEIDER, F. B. 2005. Nexus: A new operating system for trustworthy computing. In *Proceedings of the Symposium on Operating Systems Principles* Work-in-Progress Session.

SIRER, E. G., GRIMM, R., GREGORY, A. J., AND BERSHAD, B. N. 1999. Design and implementation of a distributed virtual machine for networked computers. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*. ACM Press, New York, NY, 202–216.

SYVERSON, P. F. AND STUBBLEBINE, S. G. 1999. Group principals and the formalization of anonymity. In *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems*. Springer-Verlag, Berlin, Germany, 814–833.

TROELSTRA, A. S. AND VAN DALEN, D. 1988. *Constructivism in Mathematics*. Studies in Logic and the Foundations of Mathematics Series, vol. 121, J. Barwise et al., Eds. Elsevier, Amsterdam, The Netherlands.

VAN DALEN, D. 2004. *Logic and Structure*, 4th ed. Springer, Berlin, Germany.

WAHBE, R., LUCCO, S., ANDERSON, T. E., AND GRAHAM, S. L. 1993. Efficient software-based fault isolation. In *Proceedings of the Symposium on Operating Systems Principles*. 203–216.

WALSH, K. 2011. Support for mutually suspicious subsystems. Ph.D. dissertation, Cornell University, Ithaca, NY.

WEISSMAN, C. 1969. Security controls in the ADEPT-50 time-sharing system. In *Proceedings of the Fall American Federation of Information Processing Societies National SemiAnnual Computer Conference*. Vol. 35. 119–133.

WOBBER, E., ABADI, M., BURROWS, M., AND LAMPSON, B. 1994. Authentication in the TAOS operating system. *ACM Trans. Comp. Syst. 12,* 1, 3–32.

WOBBER, T., RODEHEFFER, T. L., AND TERRY, D. B. 2009. Policy-based access control for weakly consistent replication. Tech. rep. MSR–TR–2009–15. Microsoft Research, Redmonds, WA.

WORLD WIDE WEB CONSORTIUM. 2007. Web services policy 1.5 - framework (WS-Policy). `http://www.w3.org/TR/ws-policy/`.

Received September 2009; accepted December 2009