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 based on "says" and "speaksfor" operators, enabling within a single framework request authorization to depend on (i) the source or pedigree of the requester, (ii) the outcome of performing an analysis on the requester, or (iii) the use of trusted software to encapsulate or modify the requester. Prototype document-viewer applications that enforce integrity and confidentiality of document contents—all implemented on the Nexus operating system—illustrate the convenience and expressive power of this approach to authorization.

Categories and Subject Descriptors: D.2.0 [**General**]: Protection mechanisms; D.4.6 [**Security and Protection**]: Access controls

General Terms: Security

Additional Key Words and Phrases: Authorization Logic, Credentials-based Authorization, CDD

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*, which uses the accompanying credentials (perhaps augmented with other credentials or information about the state) to make that decision and enforce some given security policy. Authorization decisions are thus decentralized, with accountability of each element in the decision made explicit and with authority shared among the guard and the principals who issue credentials.

An untrustworthy principal might attempt accesses that violate a security policy, whereas (by definition) a trustworthy one wouldn't. So a guard ideally should authorize only those 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.

Access control lists, for example, embody an *axiomatic basis* for making authorization decisions. Axioms are statements that we accept on faith; with guards that use an access control list, we accept on faith that all principals appearing on the access control list are trustworthy, so the guard authorizes requests made by these principals. The same applies when a system uses some form of reputation to decide whether a principal's request should

Authors' addresses: Department of Computer Science, Cornell University, Ithaca, NY 14853 USA; email: {fbs,kwalsh,egs}@cs.cornell.edu.
Supported in part by NICECAP cooperative agreement FA8750-07-2-0037 administered by AFRL, AFOSR grant F9550-06-0019, National Science Foundation grants 0430161 and CCF-0424422 (TRUST), ONR grant N00014-01-1-0968, 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.

be authorized. An axiomatic basis is also implied when a guard authorizes loading and running an executable only if the value of a hash indicates that the corresponding executable is unaltered from what comes in some standard distribution or if a digital signature establishes that the executable was generated by some approved software provider.

Analysis provides a way to predict whether certain behaviors by a program P are possible, and some guards employ an *analytic basis* for authorizing requests from principals executing P. Specifically, an analysis establishes that P can be trusted not to commit certain abuses and, therefore, granting the request cannot enable P to violate the security policy. Proof carrying code [Necula 1997] is perhaps the limit case. Here, 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 property proved is 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 can be trusted not to exhibit certain malicious behaviors.

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

The discussion above suggests it is unlikely that a single basis for establishing trustworthiness would be used throughout a system. Moreover, schemes that combine bases are not unusual—for example, type correctness is often enforced by using a combination of program analysis (an analytic basis) and code generation that adds run-time checks (a synthetic basis). So we conjectured that substantial benefits could come from an authorization framework that incorporates and unifies the axiomatic, analytic, and synthetic bases for trust. We seem to be the first to classify authorization schemes through this lens and the first to entertain creating such a unifying framework. Our experience in designing and using that framework is the subject of this paper.

—We developed a language and logic NAL (Nexus Authorization Logic) for specifying and reasoning about credentials and security policies. NAL extends Abadi's CDD [Abadi 2007; 2008] access control logic by adding support for axiomatic, analytic, and synthetic bases for trust and by adding two kinds of principals (groups and sub-principals) that help bridge the gap from the simplifications and abstractions found in CDD to the pragmatics of actual software applications. We thus show that this simple logic, with only two kinds of compound principals, suffices for a wide range of authorization policies in service of some novel practical applications. Our axiomatization of groups is also an advance.

—In order to evaluate what is required for this authorization framework, we implemented an operating system, Nexus [Shieh et al. 2005], that supports encoding credentials and enforcing security policies specified using NAL. Nexus employs a TPM [Trusted Computing Group] secure co-processor in order to have a single hardware-protected cryptographic key as its root of trust. NAL's scheme for naming principals and NAL's operators for attribution and delegation were informed by the needs of such environments.

—We implemented a suite of document-viewer applications that run on Nexus, and we

$A ::= [\![v : \mathcal{S}]\!] \mid A.\tau$	compound principals
$\mathcal{S} ::= s(\tau, \ldots) \mid A \textsf{ says } \mathcal{S}$ $\mid \mathcal{S} \wedge \mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid \textsf{true}$ $\mid (\forall v : \mathcal{S}) \mid (\exists v : \mathcal{S})$	statements

Fig. 1. NAL syntax.

$\textsf{false}$:	$(\forall v : v)$	
$\neg\mathcal{S}$:	$(\mathcal{S} \Rightarrow \textsf{false})$	
$A \rightarrow B$:	$(\forall v : (A \textsf{ says } v) \Rightarrow (B \textsf{ says } v))$	speaks for
$A \xrightarrow{\overline{v}:\mathcal{S}} B$:	$(\forall \overline{v} : (A \textsf{ says } \mathcal{S}) \Rightarrow (B \textsf{ says } \mathcal{S}))$	speaks for on ...

Fig. 2. NAL abbreviations.

discuss two[1] in this paper. `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 having security labels; it employs both analytic and synthetic bases for authorization. The flexibility of NAL also enabled us to implement an authorization architecture for these applications that is novel, because documents (and not programs or human users) are the principals that make requests, which guards authorize.

2. NEXUS AUTHORIZATION LOGIC (NAL)

We start with an informal introduction to NAL, focusing on how NAL can be used to model credentials and specify security policies. A syntax for NAL formulas is given in Figure 1, and some useful abbreviations appear in Figure 2; $\mathcal{S}$ denotes a NAL formula, upper-case italic identifiers denote principals, $\overline{v}$ denotes list $v_1, v_2, \ldots, v_n$ of bound variables (which range over values, principals, or formulas), and the language for terms τ is left unspecified. Predicates and terms are discussed in §2.1, and the various kinds of NAL principals are discussed in §2.2. Appendix A gives the complete list of NAL axioms and inference rules.

Throughout, we write

$$\textsc{RuleName}\ \frac{F_1, F_2, ..., F_n}{G}$$

to define an inference rule RULENAME that allows a *conclusion* G to be inferred assuming that *hypotheses* F_1, F_2 , ..., F_n have been.

Like CDD, NAL is a constructive logic. The axioms and inference rules of NAL thus differ somewhat from what is found in classical logics. For instance, a disjunction $F \vee G$ is proved in a constructive logic by proving F alone or G alone (or both), whereas $F \vee G$

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

can be proved in a classical logic indirectly without justifying either F or G. The classical logic Law of the Excluded Middle $F \vee \neg F$, for example, supports such indirect proofs; it is not an axiom or theorem of a constructive logic.

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

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

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

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

$numChan(\cdot, \cdot)$ is a system-defined predicate, and (1) holds if and only if the worldview $\mathcal{W}(Analyzer)$ contains a belief that process P has 3 open TCP connections. A NAL formula like (1) could model a credential or specify (part of) an authorization policy. As a model for a credential, it asserts that $Analyzer$ believes and is accountable for the truth of $numChan(P, \text{“TCP”}) = 3$; as a specification for an authorization policy, it requires a guard to establish that $numChan(P, \text{“TCP”}) = 3$ is in $\mathcal{W}(Analyzer)$.

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

$$\text{SAYS-I}\ \frac{F}{A\ \mathsf{says}\ F} \tag{2}$$

We include in NAL (but do not enumerate them here) constructive logical theories for reasoning about any system- and application-defined predicates in use; SAYS-I (2) asserts that those theorems are part of each principal's worldview.

In addition, beliefs that a principal holds are presumed to be consistent with the beliefs that principal holds about its beliefs:

$$\text{SAYS-E}\ \frac{A\ \mathsf{says}\ (A\ \mathsf{says}\ F)}{A\ \mathsf{says}\ F} \tag{3}$$

Deduction and Local Reasoning. A principal's worldview is assumed to be closed under modus ponens: if F and $F \Rightarrow G$ are in $\mathcal{W}(A)$ then so is G. This supports having the usual implication-elimination rule

$$\text{IMP-E}\ \frac{F,\ F \Rightarrow G}{G} \tag{4}$$

along with a closure under implication rule:

$$\text{DEDUCE}\ \frac{A\ \textsf{says}\ (F \Rightarrow G)}{(A\ \textsf{says}\ F) \Rightarrow (A\ \textsf{says}\ G)} \tag{5}$$

Notice that the formulas in DEDUCE (5) all refer to the same principal. This *local-reasoning restriction* limits the impact a principal with inconsistent beliefs can have. In particular, from A says false, DEDUCE (5) enables us to derive[2] A says G for any G, but DEDUCE (5) cannot be used to derive B says G for a different principal B. So the local reasoning restriction causes inconsistency within $\mathcal{W}(A)$ to be contained.

Worldviews of two or more principals nevertheless can contain mutually inconsistent beliefs. Beliefs about the environment or about the state of an executing system that are formed by different means or at different times could be inconsistent. The local-reasoning restriction embodied in DEDUCE (5), however, prevents the beliefs of different principals from being combined to derive (i) false (from which any formula G could be derived) or (ii) A says false for any principal A having a consistent worldview [Abadi 2007; 2008].

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

$$(\forall v : (B\ \textsf{says}\ v) \Rightarrow (A\ \textsf{says}\ v)). \tag{6}$$

If $B \rightarrow A$ holds then all beliefs in the worldview of principal B also appear in the worldview of principal A and, therefore, $\mathcal{W}(B) \subseteq \mathcal{W}(A)$ holds. In terms of credentials, $B \rightarrow A$ characterizes the consequences of A delegating to B the task of issuing credentials. Not only would B be accountable for such credentials but so would A.

The transitivity of $\rightarrow$ follows directly from definition (6), and therefore we have the following as a derived inference rule of NAL:

$$\text{TRANS} \rightarrow \frac{C \rightarrow B, B \rightarrow A}{C \rightarrow A} \tag{7}$$

One theorem of NAL is

$$(A\ \textsf{says}\ (B \rightarrow A)) \Rightarrow (B \rightarrow A) \tag{8}$$

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

$$\text{HAND-OFF}\ \frac{A\ \textsf{says}\ (B \rightarrow A)}{B \rightarrow A} \tag{9}$$

An interpretation of HAND-OFF (9) (or equivalently theorem (8)) is that each principal A is the authority on its own delegations. That is, whether A is accountable for credentials issued by B depends on what A believes rather than depending on what B believes.

NAL also supports an abbreviation $B \xrightarrow{\overline{v}:F} A$ (read "B speaks for A about F") to assert that only statements F by B are attributed to A:

$$(\forall \overline{v} : (B\ \textsf{says}\ F) \Rightarrow (A\ \textsf{says}\ F))$$

Such a restriction allows us to specify in NAL that one principal trusts another only on certain statements. For example, we should not be surprised to find university registrar

[2]Here is that proof: false $\Rightarrow G$ is a theorem for all G. Therefore, by SAYS-I (2) we conclude A says (false $\Rightarrow G$) is a theorem for all G. We then use DEDUCE (5) and IMP-E (4) to derive A says G for any G.

$\mathit{UnivReg}$ being trusted by academic department CSdept about whether somebody is a student at that university. In NAL we write

$$\mathit{UnivReg} \xrightarrow{v:v\in \mathit{Students}} \mathit{CSdept}, \tag{10}$$

meaning

$$(\forall v : \mathit{UnivReg}\,\textsf{says}\; v \in \mathit{Students} \quad\Rightarrow\quad \mathit{CSdept}\,\textsf{says}\; v \in \mathit{Students}).$$

Restricted delegation helps prevent statements made by a compromised principal A from causing another principal that trusts A to become compromised. For instance, suppose $\mathit{UnivReg}$ is compromised and therefore $\mathit{UnivReg}\,\textsf{says false}$ holds (which is problematic, because $\mathit{UnivReg}\,\textsf{says}\; F$ now can be derived for any statement F). From unrestricted delegation $\mathit{UnivReg} \rightarrow \mathit{CSdept}$, we could now derive $\mathit{CSdept}\,\textsf{says false}$; postulating restricted delegation (10) instead of that unrestricted delegation limits the bogus statements that could be derived and attributed to CSdept—the bogus statements must have the form $\mathit{CSdept}\,\textsf{says}\; X \in \mathit{Students}$ for X not a member of $\mathit{Students}$, and $\mathit{CSdept}\,\textsf{says false}$ is not among those.

However, adding a second unrestricted delegation

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

to (10) would again allow $\mathit{CSdept}\,\textsf{says false}$ to be derived when $\mathit{UnivReg}$ is compromised, because both $\mathit{CSdept}\,\textsf{says}\; X \in \mathit{Students}$ and $\mathit{CSdept}\,\textsf{says}\; X \notin \mathit{Students}$ could then be derived from corresponding statements made by compromised principal $\mathit{UnivReg}$; IMP-E (4) would then allow problematic formula $\mathit{CSdept}\,\textsf{says false}$ to be derived.

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

$$\textsc{trans}\xrightarrow{\overline{v}:F}\ \frac{C \xrightarrow{\overline{v}:F} B,\ B \xrightarrow{\overline{v}:F} A}{C \xrightarrow{\overline{v}:F} A} \tag{11}$$

and we have the following NAL theorem

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

which leads to a corresponding derived inference rule:

$$\textsc{rest-hand-off}\ \frac{A\,\textsf{says}\,(B \xrightarrow{\overline{v}:F} A)}{B \xrightarrow{\overline{v}:F} A} \tag{12}$$

2.1 Predicates and Terms in NAL

For NAL to be a constructive logic, all terms must be computable. Implementation realities dictate that terms and predicates not only be total but that they be efficiently computable, since only then will principals be able to issue credentials and will guards be able to evaluate authorization policies. We also assume, as is usual in logic, that terms and predicates have unique values in a given state. This does not preclude the use of non-determinism in programs that compute a term or predicate; it does imply the non-determinacy remains hidden.

In our NAL implementation for Nexus, the kernel provides system routines for programs to read certain operating system state and to evaluate certain pre-defined predicates on that state. Also, the output of any deterministic boolean-valued program can be used as the value of a NAL predicate. This flexibility is important, because it ensures that if an authorization policy can be programmed then it can be specified using predicates in NAL credentials.

When implementing an authorization mechanism, the designer of a guard must decide what sources to trust for information about state and history. Presumably, a guard would trust predicate evaluations that it performs itself or that the operating system performs on its behalf. But 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 (and therefore appears in the trusted computing base) with what can be trusted. NAL facilitates navigating this space by having credentials that bind a principal to a belief and that can be sent by one principal to another, surfacing what is being trusted.

NAL is agnostic about predicate naming, assuming only that the name of a predicate is associated with a unique interpretation. This assumption is necessary in order that a writer and all readers of a credential assign that credential the same meaning. In practice, different principals might associate different implementations with a given predicate name, inadvertently causing the meaning of a credential to change as it transits from one principal to another. One way to avoid this problem is by defining an authoritative interpretation (including an evaluation scheme) for each predicate; all principals are then required to use that. Implicit in implementing such a solution would have to be some way to determine what is the authoritative interpretation for a given predicate. Nexus addresses this by implementing hierarchical naming for predicates, where the name of the predicate encodes the name of the principal that is the authority for that predicate. This naming scheme is only a convention, and as we gain more experience using NAL we will revisit it.

2.2 Principals in NAL

Principals model entities to which statements can be attributed. Examples include active entities like processors, processes, channels, and executing programs, as well as passive objects like data structures and files.[3] We require that distinct NAL principals have distinct names and that statements attributed to a principal cannot be forged. Naming schemes that satisfy these requirements include:

—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 $\mathcal{S}$ is in worldview $\mathcal{W}(K_A)$ by digitally signing an encoding of $\mathcal{S}$. So, a digitally signed representation of the NAL statement $\mathcal{S}$, where public key K_A verifies the signature, conveys NAL formula K_A says $\mathcal{S}$.

—Use the hash of a digital representation of a principal as the name of that principal. Thus, a principal named by hash $H(Obj)$ includes a belief $\mathcal{S}$ in its worldview $\mathcal{W}(H(Obj))$ by having an encoding of $\mathcal{S}$ stored at some well known position[4] in Obj. So Obj conveys

[3]For example, in the applications of §4, principals are used to model processors, executing programs, and (passive) documents.

[4]We might adopt the convention that every object Obj involves two parts. The first part is a possibly empty list of the NAL formulas $\mathcal{S}_1$, $\mathcal{S}_2$, ..., $\mathcal{S}_n$ in $\mathcal{W}(H(Obj))$; the second part is some other digital content, such as a program or data, and it is the principal named $H(Obj)$.

the NAL formula $H(Obj)$ says $\mathcal{S}$ by having $\mathcal{S}$ be part of Obj.

Public keys are attractive as principal names because the public key K_A itself suffices for validating that a credential K_A says $\mathcal{S}$ has not been forged or corrupted. 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 non-privileged software. And there is no way to associate a private key with a non-privileged program executing on a processor yet have that key be secret from the processor or from privileged software being run.

Hashes are relatively inexpensive to calculate and do not require keeping secrets. The absence of secrets means that hashes are well suited for naming passive objects, because now any principal can generate $H(Obj)$ says $\mathcal{S}$ if indeed $\mathcal{S}$ is in worldview $\mathcal{W}(H(Obj))$. But when hashes are being used as principal names then validating that a credential conveying $H(Obj)$ says $\mathcal{S}$ has not been forged or corrupted requires access to the bits comprising Obj. Moreover, note that a principal named by a public key can revise its worldview and create corresponding credentials at any time, whereas a principal named by a hash cannot do so without changing names.

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

Sub-Principals. Systems are usually designed having some components depend on others. In hierarchically structured systems, for example, higher-level components depend on lower levels. Also, dependencies are created when one component loads and starts executing (or interpreting) another. Not infrequently, the dependency of a principal Sub on another principal Dom is so strong that Sub is being impersonated by Dom and, therefore, Sub says $\mathcal{S}$ holds if and only if Dom says $(Sub$ says $\mathcal{S})$ holds.

NAL offers *sub-principals* as a convenience for naming a principal that, by design, is impersonated by another. Given a principal A and any term τ, sub-principal $A.\tau$ is a NAL principal[5] impersonated by A. This is captured in a NAL rule:

$$\text{SUBPRIN}\ \frac{}{A \rightarrow A.\tau} \tag{13}$$

Equivalent terms define sub-principals having the same worldviews:

$$\text{EQUIV SUBPRIN}\ \frac{\tau_1 = \tau_2}{A.\tau_1 \rightarrow A.\tau_2} \tag{14}$$

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

Sub-principals are particularly useful for describing structures where a single principal is being multiplexed among various roles. For example, the principal corresponding to an executing program would be a sub-principal of the CPU (or other execution environment) that it runs in. Consider a system comprising a certification authority CA being executed

[5]Sub-principals can themselves have sub-principals, with left-associativity assumed so that $A.\tau_1.\tau_2$ abbreviates $(A.\tau_1).\tau_2$.

by an operating system OS that is running on a CPU. And suppose the hash of CA is H_{CA}, the hash of OS is H_{OS}, and the CPU's TPM stores a private key whose signatures can be verified using public key K_{CPU}. We would then define sub-principal $K_{CPU}.H_{OS}$ for the OS and define $K_{CPU}.H_{OS}.H_{CA}$ for the CA. So according to SUBPRIN (13), we have:

$$K_{CPU} \rightarrow K_{CPU}.H_{OS} \tag{15}$$

$$K_{CPU}.H_{OS} \rightarrow K_{CPU}.H_{OS}.H_{CA} \tag{16}$$

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

$$K_{CPU} \textsf{ says } (K_{CPU}.H_{OS} \textsf{ says } (K_{CPU}.H_{OS}.H_{CA} \textsf{ says } \mathcal{S}))$$

from which we can derive

$$\begin{aligned} &K_{CPU}.H_{OS}.H_{CA} \textsf{ says} \\ &\qquad (K_{CPU}.H_{OS}.H_{CA} \textsf{ says} \\ &\qquad\qquad (K_{CPU}.H_{OS}.H_{CA} \textsf{ says } \mathcal{S})), \end{aligned}$$

due to (15) and (16) and definition (6) of $A \rightarrow B$; using SAYS-E (3) twice then obtains:

$$K_{CPU}.H_{OS}.H_{CA} \textsf{ says } \mathcal{S}$$

Sub-principals 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, for example, allows the subset of credentials issued by some principal A when you trust it—that is, credentials issued under circumstances that engender trust—to be distinguished from other credentials issued by A. So instead of using a single principal $\mathit{FileSys}$, we might employ a sequence $\mathit{FileSys}.1$, $\mathit{FileSys}.2$, ..., $\mathit{FileSys}.i$, ... of sub-principals, each accountable for issuing credentials during successive intervals. Then by specifying security policies that are satisfied only by credentials attributed to a "current instance" $\mathit{FileSys}.\mathit{now}$ (for now an integer variable), a guard can reject requests accompanied by outdated credentials.

SUBPRIN (13) allows any statement by a principal A to be liberally (mis)interpreted and attributed to any sub-principal of A. That is, from $A \textsf{ says } \mathrm{OK}$ we could derive $A.\tau \textsf{ says } \mathrm{OK}$ for any sub-principal $A.\tau$. Such misinterpretations can be avoided by adopting a sub-principal naming convention. We might, for example, agree to attribute to sub-principal $A.\epsilon$ any statements by A that should not be attributed to any other sub-principal of A.

Groups. A NAL *group* is a principal constructed from a set of other principals, called its *constituents*, and is specified *intensionally* by giving a characteristic predicate. We write $[\![v : P]\!]$ to define the group formed from characteristic predicate P; the group is constructed using for its constituents those principals A for which $P[v := A]$ holds.[6] As an example,

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

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

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

The worldview of a NAL group is formed by taking the deductive closure of the union of the world views for the group's constituents. Thus, if the worldview for one constituent of the group contains $E \Rightarrow F$ and another contains E, then due to IMP-E (4) the group's worldview contains E, $E \Rightarrow F$, and F—even if the worldview for no constituent of the group contains F.

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 $P[v := A]$ holds then $A \rightarrow [\![v : P]\!]$ holds:

$$\text{MEMBER}\ \frac{P[v := A]}{A \rightarrow [\![v : P]\!]} \tag{17}$$

Note that $A \rightarrow [\![v : P]\!]$ does not imply $P[v := A]$, because we might have derived $A \rightarrow [\![v : P]\!]$ from $A \rightarrow B$ and $B \rightarrow [\![v : P]\!]$. The way to indicate in NAL that $P[v := A]$ holds is with a credential conveying that A satisfies P.

And when $v \rightarrow A$ holds for all constituents v of a group, then all beliefs in the group's worldview necessarily appear in $\mathcal{W}(A)$:

$$\text{GROUP} \rightarrow \frac{(\forall v : P \Rightarrow (v \rightarrow A))}{[\![v : P]\!] \rightarrow A} \tag{18}$$

This inference rule, in combination MEMBER (17), allows us to justify the following derived inference rule, which asserts groups and $\rightarrow$ are monotonic relative to implication:

$$\text{GROUP MONOTONICITY}\ \frac{(\forall v : P \Rightarrow P')}{[\![v : P]\!] \rightarrow [\![v : P']\!]} \tag{19}$$

Finally, note that NAL does not preclude specifying *extensionally* defined groups, wherein constituents are simply enumerated. For example, $[\![v : (v \rightarrow A) \vee (v \rightarrow B) \vee (v \rightarrow C)]\!]$ is the extensionally defined group constructed from constituents A, B, and C.

3. GUARDS: THEORY AND PRACTICE

The decision to authorize a request can be expressed as a question about formula derivation in NAL. An access request by a principal A is modeled using (i) NAL formula A says $\mathcal{S}$ to convey request particulars, (ii) NAL formulas C_1, C_2, ..., C_n for accompanying credentials $\hat{C}_1$, $\hat{C}_2$, ..., $\hat{C}_n$, and (iii) NAL formula P_G for the authorization policy being enforced by guard G. The request is granted if and only if G determines that P_G can be derived from

$$(A\ \textsf{says}\ \mathcal{S}) \wedge C_1 \wedge C_2 \wedge \cdots \wedge C_n \tag{20}$$

using NAL's inference rules; otherwise the request is denied. For example, a file system $\mathit{FileSys}$ implementing an authorization policy

$$\mathit{FileSys}.\texttt{foo}\ \textsf{says}\ \mathit{read}(\texttt{foo}) \tag{21}$$

for accessing a file `foo` would allow a request A says $\mathit{read}(\texttt{foo})$ to proceed if that request were accompanied by a credential conveying C that allowed $A \rightarrow \mathit{FileSys}.\texttt{foo}$ to be derived, because $C \wedge A$ says $\mathit{read}(\texttt{foo})$ can then be used to derive (21).

Not only does the guard make an authorization decision but notice that, through the derivation for P_G, the guard documents a rationale for granting the request and makes clear the role each credential has played. The derivation is thus a form of audit log—and a particularly descriptive one, at that.

The wide range of possible implementations for this derivation-based approach to authorization gives system designers considerable flexibility to make engineering trade-offs when implementing guards. Decisions the designer must make include:

—Where is each credential stored? Credentials could be stored at the requesting principal, at the guard, or elsewhere in the system.

—How is each credential obtained by the guard? Credentials could accompany a request, be fetched when needed by the guard, or be sent periodically to the guard.

—Where and how is the derivation of the guard's authorization policy P_G performed? This could be done by the requesting principal [Appel and Felten 1999; Bauer 2003], it could be done by the guard (perhaps by coordinating a distributed computation based on subgoals in the proof [Bauer et al. 2005]), or it could be a service provided by some trusted third party.

—When is each credential generated? If a credential $\hat{C}_i$ corresponding to NAL formula C_i is issued, then we might expect C_i to hold thereafter. But changes to the system state could cause a principal to change its beliefs, falsifying C_i. Guards and other principals with access to $\hat{C}_i$ but lacking independent means for validating C_i must be implemented with this reality in mind.

The remainder of this section explores these matters in more detail.

Sources of Derivations. Constructing a NAL derivation for some arbitrary given formula is an undecidable problem, because NAL terms include integers and rich data structures whose axiomatizations are undecidable. However, NAL formulas P_G for authorization policies found in practice are often easily derived when accompanying credentials have a prescribed form. For example, we might specify discretionary access control[7] for requests from a principal A to access an object obj by writing the following NAL formula for P_G:

$$(A \textsf{ says } access(obj)) \quad \wedge \quad (A \rightarrow owner(obj)) \tag{22}$$

Derivation of (22) is trivial if we prescribe that requests $A \textsf{ says } access(obj)$ are accompanied by a credential that attests

$$owner(obj) \textsf{ says } A \rightarrow owner(obj). \tag{23}$$

Or, the guard for obj might itself store an access control list ACL_{obj}, which is interpreted as attesting (23) for every principal A appearing in ACL_{obj}.

An alternative to having a guard G perform a derivation of P_G would be to have G check a derivation supplied with the request. This is a decidable task because, by definition, inference rule applications are mechanically checkable. To illustrate, we return to the discretionary access control given above. Instead of accompanying a request with a credential that attests to (23), a principal A making a request might provide a set of credentials and a derivation from those credentials of what is needed for establishing conjunct $A \rightarrow owner(obj)$ of (22).

The idea of having derivations accompany requests is not a panacea. In order for a principal to produce a derivation of P_G, that principal must know what P_G is, which requires

[7]Discretionary access control policies are authorization polices where access privileges for an object are set by the owner of that object.

divulging the criteria for authorizing requests. And sometimes we may want to keep principals unaware that different criteria apply to each. Also, having each different requester independently derive P_G makes changing P_G difficult, since all principals that submit requests to guard G would have to be identified, notified, and updated.

Credential and Policy Dynamics. Possession by guard G of a credential $\hat{C}$ puts the NAL formula C conveyed by $\hat{C}$ among G's beliefs. Therefore, NAL models G's possession of a credential $\hat{C}$ as: $G\,\textsf{says}\ C$. One might hope that

$$(G\,\textsf{says}\ C) \Rightarrow C \tag{24}$$

would hold as well, although this is by no means guaranteed. The principal issuing a credential might subsequently change its beliefs (perhaps because the state has changed) but after a credential has been sent elsewhere, that credential is no longer available to the issuer for update or deletion.[8] Yet if (24) does not hold for one or more credentials that a guard G has received, then the guard could have a NAL derivation of authorization policy P_G from those credentials even though P_G does not actually hold.

For example, a request $A\,\textsf{says}\ access(obj)$ accompanied by a credential $\hat{C}_{tme}$ conveying $TimeServ\,\textsf{says}\ clock = 0900$ suffices to derive:

$$A\,\textsf{says}\ \mathrm{access}(obj) \quad \wedge \quad TimeServ\,\textsf{says}\ clock < 1000 \tag{25}$$

But the instance of (24) corresponding to $\hat{C}_{tme}$ does not hold if $clock$ increases with the passage of time. The NAL derivation of (25), however, is not invalidated by this change to the worldview of $TimeServ$. So authorization policy (25) does not hold despite the guard having a NAL derivation from the request and accompanying credentials.

Even if a guard G could check the truth of NAL formula C conveyed by each credential $\hat{C}$ that G uses, there is no guarantee that P_G would hold after the checks. The process of checking the truth of a credential takes time, and concurrent actions elsewhere in the system could falsify the formula conveyed by one credential while the others are being checked. However, by restricting system execution, guard construction, and/or what NAL formulas credentials may convey, we can ensure that P_G will hold whenever it can be derived in NAL from a request and accompanying credentials.

System execution generally satisfies certain restrictions—time never decreases and the past is immutable—not to mention restrictions coupled to the semantics of system and application functionality. This means that some truths do not change as execution proceeds, which can be leveraged for defining credentials that cannot be falsified by future execution. For example, once $clock > 1000$ holds, it cannot be later falsified. And a credential for communicating that some predicate P held on a past system state cannot be falsified if P holds when that credential is issued.

By imposing additional restrictions on execution by principals that falsify certain predicates, we obtain a second general approach for constructing credentials that satisfy (24). Suppose that, in order to authorize some request, a guard requires that $A\,\textsf{says}\ P$ hold, for P a state predicate.

—We might restrict principals from invalidating P until some time in the future, and use a

[8]This revocation problem is well known in connection with capabilities that are used to access a resource and with public key certificates that describe name-key bindings.

credential conveying a form of lease [Gray and Cheriton 1989]

$$A\,\textsf{says}\,(\mathit{clock} < 1000 \ \Rightarrow\ P)$$

instead of $A\,\textsf{says}\ P$. Not only is this alternative credential not falsified, but if $\mathit{clock} < 1000$ holds then a guard can conclude that $A\,\textsf{says}\ P$ is satisfied.

—We might stipulate that principals follow some sort of 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 relinquishes the lock afterward,
  (ii) a principal acquires exclusive access to ℓ_P before falsifying P, and
  (iii) a lock holder must reestablish P prior to releasing ℓ_P.
  Then a credential conveying

$$A\,\textsf{says}\,(\mathit{locked}(\text{shared}, \ell_P, A) \Rightarrow P)$$

is never falsified even though P might be. Moreover, if a guard acquires ℓ_P with shared access while making an authorization decision then that guard can conclude that $A\,\textsf{says}\ P$ is satisfied at that time.

The way that a guard uses the above kinds credentials to derive $A\,\textsf{says}\ P$ warrants amplification. Both cases involved a NAL formula $A\,\textsf{says}\,(L \Rightarrow P)$. According to DEDUCE (5), we can derive from that

$$(A\,\textsf{says}\ L) \ \Rightarrow\ (A\,\textsf{says}\ P).$$

So with a credential $A\,\textsf{says}\ L$, we can use IMP-E (4) to derive $A\,\textsf{says}\ P$. Since A is unlikely to be an authority on clock or on lock acquisitions, A would delegate to the appropriate authorities by issuing credentials like

$$A\,\textsf{says}\,(\mathit{TimeServ} \rightarrow A) \quad \text{or} \quad A\,\textsf{says}\,(\mathit{LockMngr} \rightarrow A)$$

from which $\mathit{TimeServ} \rightarrow A$ and $\mathit{TimeServ} \rightarrow A$ are derived due to HAND-OFF (8). This then allows

$$A\,\textsf{says}\ \mathit{clock} < 1000 \quad \text{or} \quad A\,\textsf{says}\ \mathit{locked}(\text{shared}, \ell_P, A)$$

to be derived from credentials conveying

$$\mathit{TimeServ}\,\textsf{says}\ \mathit{clock} < 1000 \tag{26}$$
$$\mathit{LockMngr}\,\textsf{says}\ \mathit{locked}(\text{shared}, \ell_P, A) \tag{27}$$

which authorities might issue.

Observe that credentials (26) and (27) have a special property: a guard G can check whether each satisfies (24) for the interval needed by G. In the case of credential (26) from $\mathit{TimeServ}$, if G can read clock, then G can calculate how much time remains until $\mathit{clock} < 1000$ is falsified and the credential expires; (24) remains satisfied if the certificate conveying (26) is deleted by the guard before that expiration time is reached. For credential (27) from $\mathit{LockMngr}$, G acquires a shared lock on ℓ_P before augmenting its worldview based on the credential, and G deletes that credential before releasing the lock.

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 a NAL proof checker (described in Appendix B), and building Nexus support for creating credentials and guards.

In each of the viewer applications, documents are considered principals. And a document to be displayed—not the human user viewing the document—is the principal whose requests are authorized by a guard. This unconventional architecture allowed us to benefit from employing analytic and/or synthetic bases for authorization. Had the system instead been designed to process requests from human users wishing to view documents, 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

Documents convey beliefs. If D is a name for some document, then we can use NAL to formalize which beliefs D conveys: Identify D with a principal $\overline{D}$ and write NAL formula $\overline{D}$ says P for each belief P that D conveys.

Excerpts derived from a document also convey beliefs. We represent an excerpt E appearing in a document D as a 4-tuple $E = \langle \chi, D, \ell, D' \rangle$, 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 can be identified with a NAL principal $\overline{E}$, where $\overline{E}$ says P holds for every belief P that E conveys. Define $Src(E)$ to be the source document (i.e., D' above) from which E was purportedly derived, and define $\overline{Src}(E)$ to be the principal corresponding to $Src(E)$.

The reader of a document that contains an excerpt E and the author of source document $Src(E)$ would expect that beliefs conveyed by E are conveyed in $Src(E)$ and, therefore, $\overline{E} \rightarrow \overline{Src}(E)$ holds. But whether $\overline{E} \rightarrow \overline{Src}(E)$ 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 our `TruDocs` system issues about D

$$\texttt{TruDocs}\ \textsf{says}\ (\forall E\colon\ E \prec D \ \Rightarrow\ \overline{E} \rightarrow \overline{Src}(E)) \tag{28}$$

where relation $E \prec D$ holds if and only if document D contains excerpt E.[9]

The author of a document D' cannot be expected to enumerate all possible excerpts E that convey beliefs found in D' or, equivalently, list all instances of $\overline{E} \rightarrow \overline{D'}$ that hold. 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 $\overline{E} \rightarrow \overline{D'}$.

[9]Definition (28) treats nested excerpts as if each appears directly in D. Other treatments are possible.

The Wesleyan Cinema archive [Wesleyan Cinema Archive], for example, stipulates that an individual excerpt be verbatim, not exceed 75 words, and that any given document contain at most 20 such excerpts; the New International Version of the Bible [New International Version Bible] stipulates that excerpts contain at most 500 verses, that they be copied verbatim, and that the excerpts account for at most 25% of the document; and W3C [World Wide Web Consortium (W3C)] allows inclusion of their logo unmodified in a document provided an automatic validator accepts the resulting document and the logo image has a link to the W3C URL.

The role of a use policy that has been associated with a source document D' can be formalized abstractly in NAL as a credential conveying

$$\overline{D'}\ \textsf{says}\,(\forall E, D\colon (D' = \mathit{Src}(E) \wedge \mathit{usePol}_{D'}(E,D)) \;\Rightarrow\; (\overline{E} \rightarrow \overline{D'})) \tag{29}$$

where $\mathit{usePol}_{D'}(E,D)$ is a predicate satisfied if excerpt E appearing in D is consistent with the use policy associated with D'. Credentials like (29) enable (28) to be derived by checking each excerpt E against the use policy for $\mathit{Src}(E)$:

$$\texttt{TruDocs}\ \textsf{says}\ (\forall E\colon\ E \prec D \;\Rightarrow\; \mathit{usePol}_{\mathit{Src}(E)}(E,D)) \tag{30}$$

Thus, a guard that employs (30) as the authorization policy for display requests from a document D can mechanically derive (28) or, conversely, deny a display request if D does not have integrity. So this authorization policy embodies an analytic basis for trust, because authorization depends on checking use policies, a form of analysis.

`TruDocs` can also handle Copyright's "fair use" and other non-computable use policies by employing an axiomatic basis for trust. One or more human authorities H_i for which `TruDocs` has issued a credential conveying

$$\texttt{TruDocs}\,\textsf{says}\,(H_i \rightarrow \texttt{TruDocs}) \tag{31}$$

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

$$H_i\,\textsf{says}\,(\overline{E} \rightarrow \overline{\mathit{Src}}(E)) \tag{32}$$

for excerpts E, such that $E \prec D$ 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 (28). So this approach corresponds to deriving (30) where $\mathit{usePol}_{\mathit{Src}(E)}(E,D)$ is satisfied if and only if `TruDocs` has credentials (31) and (32).

Implementation Details. `TruDocs` comprises an editor `TDed` for use by document authors, a viewer `TDview` for displaying documents, and some additional support software. `TDed` and `TDview` were obtained by modifying the OpenOffice software suite [OpenOffice].

—`TDed` allows a document D containing excerpts to be created, enables a use policy to be defined and associated with that document, and constructs a unique name $\mathit{Nme}(D)$ for the document. $\mathit{Nme}(D)$ is constructed so that it embodies a validated set of *document particulars*, such as title, author, publication venue, publication date, etc.

—`TDview` implements a guard to authorize display requests from documents; a display request for D is granted only if (30) can be derived, since (28) 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 $\mathit{Nme}(\mathit{Src}(E))$, thereby giving the reader a human-intelligible description for the source document from which E was derived.

In order to derive (30) for a document D, the `TDview` guard enumerates the excerpts in D and processes each excerpt E as follows.

(1) Determine $usePol_{Src(E)}(E, D)$.

(2) Derive

$$\texttt{TruDocs}\ \textsf{says}\ usePol_{Src(E)}(E, D) \tag{33}$$

from the credentials accompanying the display request and/or by performing local checking. Specifically, the derivation of (33) is done by `TDview` in conjunction with the NAL proof checker and builtin support for text matching:

—`TDview` checks to see if the display request was accompanied by credentials and/or a NAL proof that discharges (33), and if so, `TDview` checks that proof (which includes validating the credentials), issuing an instance of a credential conveying (33) if the proof is sound;

—if not, `TDview` determines if it has builtin support to validate $usePol_{Src(E)}(E, D)$, attempts that validation, and if successful `TDview` issues an instance of a credential conveying (33);

—otherwise, `TDview` displays an error message that details the use policy that it could not satisfy, requesting additional credentials and/or a NAL proof be provided.

Note that some trust assumptions are required, due to 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 (33), a statement being attributed to `TruDocs`. Second, for each credential EA_i says $\mathcal{S}$ provided by an external authority EA_i and used in step 2, there must be a credential `TDview` says $(EA_i \rightarrow \texttt{TDview})$ signifying that EA_i is trusted by `TDview` and, therefore, `TDview` can derive `TDview` says $\mathcal{S}$ from EA_i says $\mathcal{S}$. The name of each such trusted external authority EA_i is communicated to `TDview` at startup.

Limits in on-line 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 requests display. For example, when creating a document D, `TDed` has access to all documents from which excerpts appearing in D are being derived. `TDed` is therefore an obvious place to perform some analysis and issue credentials that later aid `TDview` in deriving (33). This, however, does require an additional trust assumption: $\texttt{TDed} \rightarrow \texttt{TDview}$.

`TDview` directly implements the kinds of analysis commonly found in use policies; but being only a prototype, it is far from complete. This builtin support currently includes 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, limiting the length of individual excerpts, the aggregate length or number of the excerpts from a given document, and so on. `TDview` also can validate compliance with a use policy that stipulates excerpts derived from D' not appear in documents having certain document particulars—for example, that excerpts from D' 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 $\overline{HNme}(D)$, as follows. Each document

D is represented in XML and we define $\overline{HNme}(D) = H(X_D)$, where X_D is the XML representation (using the DocBook [DocBook] standard) for D and where $H(\cdot)$ is a SHA1 hash. $\overline{HNme}(D)$, because it is relatively short, can serve as the name for a file storing X_D in a file system or web server. And, in X_D, for each excerpt E, `TruDocs` not only stores $Nme(Src(E))$, which provides the document particulars for $Src(E)$, but `TruDocs` also stores name $\overline{HNme}(Src(E))$, which provides direct access to the file storing $X_{Src(E)}$.[10]

A binding between principals $\overline{HNme}(D)$ (i.e., $H(X_D)$) and $Nme(D)$ is made by a `TruDocs` principal `Reg` (named by public key $K_{\texttt{Reg}}$); `Reg` runs on a separate machine from `TDed` and `TDview`. `Reg` not only creates bindings but it also validates document particulars and disseminates the existence of $\overline{HNme}(D)$–$Nme(D)$ bindings by issuing credentials. In particular, a document D being created with `TDed` becomes eligible for viewing 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

$$K_{\texttt{Reg}} \textsf{ says } (\overline{HNme}(D) \rightarrow K_{\texttt{Reg}}.Nme(D)) \tag{34}$$

which is returned by `Reg` to `TDed`, where it is piggybacked[11] on X_D. Notice that if we define $\overline{D}$ to be $K_{\texttt{Reg}}.Nme(D)$, we can derive

$$\overline{HNme}(D) \rightarrow \overline{D}, \tag{35}$$

a binding between $\overline{HNme}(D)$ and $\overline{D}$: SUBPRIN (13) derives $\overline{HNme}(D) \rightarrow K_{\texttt{Reg}}.Nme(D)$ from (34) and then use the above definition of $\overline{D}$ 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 (29), now that two different principals are being associated with each document:

$$\begin{aligned}\overline{HNme}(D') \textsf{ says } (\forall E, D\colon\ & (D' = Src(E) \wedge usePol_{D'}(E, D)) \\ & \Rightarrow (\overline{E} \rightarrow \overline{HNme}(D')))\end{aligned} \tag{36}$$

But $\overline{E} \rightarrow \overline{HNme}(D')$ derives $\overline{E} \rightarrow \overline{D'}$, since (35) can be derived from the instance of (34) piggybacked on $X_{D'}$. This means that from (34) and (36), `TDview` can always automatically derive:

$$\begin{aligned}H(X_{D'}) \textsf{ says } (\forall E, D\colon\ & (D' = Src(E) \wedge usePol_{D'}(E, D)) \\ & \Rightarrow (\overline{E} \rightarrow \overline{D'}))\end{aligned} \tag{37}$$

[10]If only file name $\overline{HNme}(Src(E))$ were stored in X_D, then after D has been created, an attacker could change what is stored in file $\overline{HNme}(Src(E))$, thereby invalidating the consistency of the information from $Nme(Src(E))$ that gets displayed at the end of E with the document particulars for $Src(E)$.

[11]Credential (34) cannot be stored in X_D, because that would change name $H(X_D)$ for that principal, rending credential (34) useless.

And the NAL derivation of (28) from (37) is virtually the same as the derivation of (28) from (29), 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 multi-level security [Department of Defense 1985; Weissman 1969] for accessing documents comprising *text elements*. Each text element $\chi = \langle D, \ell, len\rangle$ in document D and starting at location ℓ for len characters is assigned a *classification label* $\mathcal{L}_T(\chi)$ by some trusted *classification authority*; each user H is assigned a *clearance* $\mathcal{L}_U(H)$ by some trusted *clearance authority*.

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

$$D \textsf{ says } (\forall \chi \in Txt(D) \colon \;\; \mathcal{L}_T(\chi) \trianglelefteq \mathcal{L}_U(H)) \tag{38}$$

holds. This 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 $\mathcal{L}_T(\chi)$ and the choice of clearance authority determines $\mathcal{L}_U(H)$, so these choices albeit indirectly effect whether H satisfies (38).

`ConfDocs` is agnostic about the set of security labels and partial order relation $\trianglelefteq$. The system simply requires the means (internally builtin or by appeal to an external authority) to determine whether $\mathtt{L} \trianglelefteq \mathtt{L}'$ holds for any pair of security labels `L` and $\mathtt{L}'$. `ConfDocs` has builtin 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 a pair designates a set of compartments constructed from descriptors, such as `crypto`, `nuclear`, etc. There is a total order $\blacktriangleleft$ on the levels: $\mathtt{U} \blacktriangleleft \mathtt{C}$, $\mathtt{C} \blacktriangleleft \mathtt{S}$, and $\mathtt{S} \blacktriangleleft \mathtt{TS}$; the usual partial order $\subseteq$ on sets of compartments; and

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

holds if and only if $lvl \blacktriangleleft lvl' \vee lvl = lvl'$ and $cmpt \subseteq cmpt'$ hold.

If a document D does not satisfy authorization policy (38) for a given user H, then it is often possible by modifying D to derive a document that does.

—Deleting text from D narrows the scope of the universal quantification in (38) by removing a text element χ from $Txt(D)$, thereby eliminating an obligation $\mathcal{L}_T(\chi) \trianglelefteq \mathcal{L}_U(H)$ that cannot be derived.

—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 $\mathcal{L}_T(\chi) \trianglelefteq \mathcal{L}_U(H)$ that cannot be derived into one $\mathcal{L}_T(\chi') \trianglelefteq \mathcal{L}_U(H)$ that can be.

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

Implementation Details. 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 convey

$$D \textsf{ says } (\mathcal{L}_T(\chi) = \texttt{L}_\chi) \quad \text{or} \quad CA_T \textsf{ says } (\mathcal{L}_T(\chi) = \texttt{L}_\chi)$$

to give a classification label $\texttt{L}_\chi$ for each text element $\chi \in Txt(D)$. The latter would be accompanied by a restricted delegation

$$D \textsf{ says } CA_T \xrightarrow{v_1, v_2 \,:\, \mathcal{L}_T(v_1) = v_2} D \tag{39}$$

for classification authority CA_T, 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*

$$San \textsf{ says } (\mathcal{L}_T(Edit(\chi, s)) = \texttt{L}_{Edit(\chi, s)}) \tag{40}$$

that give a classification label for the text element produced by executing a builtin edit function to modify χ according to script s. Here, San is either D or some classification authority CA_T for which restricted delegation (39) appears in the representation of D. And script s may include commands like $\texttt{Overprint}(\chi)$, which overprints boxes $\square$ on the characters of χ, as well as standard text editor commands like $\texttt{Replace}(x, y)$, which replaces all instances of character string x with string y, and so on.

Credentials like (40) define a *sanitization policy* for D. Such a policy characterizes ways to transform a document containing information that is too highly classified given the clearance of some intended readers into a document those readers can access. The hard part is resolving the tension between hiding too much and indirectly leaking classified information. Sanitization of paper documents, for example, often involves crossing-out fragments of text (and `ConfDocs` supports this functionality with its $\texttt{Overprint}(\chi)$ command) but a document sanitized in this manner might still leak information to a reader by revealing the length of a crossed-out name or the existence of an explanatory note. Yet, deleting large segments of text when sanitizing a document could raise unwarranted questions in a reader's mind.

A user H attempting to view a document D invokes the `ConfDocs` program `CDview`, furnishing a credential signed by some clearance authority CA_U that attests to $\mathcal{L}_U(H)$:

$$CA_U \textsf{ says } \mathcal{L}_U(H) = \texttt{L}_\texttt{H} \tag{41}$$

All clearance authorities are not 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

$$D \textsf{ says } CA_U \xrightarrow{v_1, v_2 \,:\, \mathcal{L}_U(v_1) = v_2} D \tag{42}$$

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

`CDview` authorizes a request by a document D for display to a user H provided (38) holds, where classification label $\mathcal{L}_T(\chi)$ for each text element $\chi \in Txt(D)$ and clearance $\mathcal{L}_U(H)$ come from authorities that the publisher of D is willing to trust. This is achieved by enforcing

$$(\exists v : D \textsf{ says } \mathcal{L}_U(H) = v \;\;\wedge\;\; D \textsf{ says } (\forall \chi \in Txt(D) \colon\; \mathcal{L}_T(\chi) \trianglelefteq v)) \tag{43}$$

which is easily seen to imply (38).

The first conjunct of (43) can be derived by the guard in `CDview` from user-provided credential (41) if D contains a suitable restricted delegation (42). This means that the publisher of D can include restricted delegation credentials in D to authorize any specific user to view D. For settings where that power resides elsewhere, we would change the first conjunct of (43). For example, if by statute a user H must be assigned a clearance by some authority CS_{NSA} for purposes of viewing D, then the first conjunct of (43) would be replaced by CS_{NSA} says $\mathcal{L}_U(H) = v$; and in invoking `CDview`, user H would have to provide credentials from which this new first conjunct of authorization policy (43) can be derived.

The second conjunct of (43) is discharged by `CDview` as follows. For each element χ in $Txt(D)$, `CDview` checks whether $\mathcal{L}_T(\chi) \trianglelefteq \mathrm{L}$ holds. If it does not hold, then `CDview` attempts corrective action, thereby engaging in a synthetic basis for authorizing a display request for a modified document, D':

(1) `CDview` checks to see whether any of the sanitization credentials for D allow construction of a modified text element χ' such that $\mathcal{L}_T(\chi') \trianglelefteq \mathrm{H}$ holds. If it does, then `CDview` obtains D' by replacing χ with χ'.

(2) `CDview` otherwise obtains D' by deleting χ.

And `CDview` then proceeds anew to discharge the second conjunct of (43) for document D'. In the worst case, this procedure terminates with (43) satisfied for a document D' where $Txt(D') = \emptyset$, and `CDview` will display an empty document.

Nexus Sealed Bundles. To ensure that invoking `CDview` is the only way to view documents, they are stored and transmitted in encrypted form. Nexus, in conjunction with a TPM secure co-processor, implements a storage abstraction that is ideal for this task. A Nexus *sealed bundle* B stores its payload $payload(B)$ in encrypted form and also specifies a NAL group $group(B)$ of principals authorized to decrypt $payload(B)$.

By invoking the Nexus $\texttt{Decrypt}(B)$ 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 materialized 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

$$group(B) \text{ says } \texttt{Decrypt}(B)$$

holds. To allow an access thus requires the kernel to validate a proof of $A \rightarrow group(B)$, thereby establishing that A is a constituent of $group(B)$. This proof validation is discharged by the kernel checking whether A satisfies predicate $P_B[v := A]$, where P_B was originally used in defining $group(B)$ and saved in the bundle; $A \rightarrow group(B)$ then follows due to MEMBER (17). Our implementation also allows A to provide a proof of $A \rightarrow C$; the kernel would check that proof and then check that $P_B[v := C]$ is satisfied. Notice, the set of principals satisfying P_B is not necessarily static if P_B depends on state, so the constituents of $group(B)$ 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`: programs whose hash

$H_{\texttt{CDview}}$ equals the hash of some pre-determined correct object code for `CDview`, where that object code was loaded and is being executed by a Nexus process running on a valid Nexus kernel, which itself is being executed on a trusted processor with associated TPM. Such a principal is specified using NAL sub-principals as $K_{CPU}.H_{Nexus}.process_{23}.H_{\texttt{CDview}}$ because `CDview` is actually being impersonated by some Nexus process (here, $process_{23}$), which in turn is being impersonated by the Nexus, which itself is impersonated by the a hardware processor. (Only to simplify the exposition above, have we been naming principals by program names rather than giving the fully qualified list of sub-principals (hardware key, kernel, process) that actually defines the principal's name.) 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

NAL was born of necessity. Our original plan for Nexus was to adopt 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 only implemented a decidable subset of the logic. But Appel and Felten's [1999] proof carrying authentication addressed that, suggesting that all requests be accompanied by proofs and that guards only perform proof checking. Moreover, proof carrying authentication employed a higher-order logic supporting 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. So 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 versus accessing a majority versus accessing all of the content). A system designer would presumably mark such an access 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 turns out to be subtle, because it can lead to inconsistencies (allowing false to be derived, hence any authorization policy to be satisfied).

In particular, principals evaluate state predicates, but (as discussed in §2.1) it is unrealistic to expect that any principal could evaluate any state predicate or that different principals evaluating the same state predicate at different times would compute the same value. We thus needed a way for one principal to include in its worldview a state predicate F evaluated by some other principal.

—One approach [Abadi et al. 1993; Appel and Felten 1999; Howell 2000] is to use SAYS-I

(2) along with a new inference rule

$$\frac{A \,\textsf{says}\; F, \;\; \textsf{controls}(A, F)}{F}$$

where $\textsf{controls}(A, F)$ holds when A is a trusted authority on the truth of F.

—The other approach [Abadi 2007; 2008; Lesniewski-Laas et al. 2007] is to postulate a local-reasoning restriction and require that a principal use delegation to import and reason about beliefs from others.

We adopted this second approach because an inconsistency in a principal or among a set cannot then be propagated to a principal that does not speak-for or receive credentials from these. The first approach offers no such guarantees about propagation of inconsistencies, and it also requires characterizing sets of beliefs F' covered by $\textsf{controls}(A, F)$: If $\textsf{controls}(A, F)$ holds and $F \Rightarrow F'$ is valid then is A necessarily also trusted on F'? Is A necessarily trusted on $\neg F$?

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 has a very simple language, in part, because details about principals and beliefs are left unspecified. So CDD offered us the freedom to define principals that would match what Nexus provided and to use state predicates in beliefs (importing theories for giving these state predicates suitable interpretations).

NAL's sub-principals are derived from Alpaca [Lesniewski-Laas et al. 2007] named roles. Prior proposals (e.g., SDSI/SPKI [Rivest and Lampson 1996] and Taos [Wobber et al. 1994]) restricted the term τ used in defining a sub-principal $A.\tau$ to being a fixed string, which meant that only static roles could be supported.

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, and with inference rules MEMBER (17) and GROUP $\rightarrow$ (18), 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, too. 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[12] group member; or-groups are not sound with respect to IMP-E (4) and therefore require different proof rules from 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. Credentials about intentionally specified conjunctional groups could easily be supported in

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

NAL as an abbreviation:

$$\langle\langle v : P\rangle\rangle \textsf{ says } F: \quad (\forall v : P \Rightarrow (v \textsf{ says } S))$$

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 involving 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 duplicate in the logic state that already exists in a system. In doing so, 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.

5.1 Other Related Work

Any authorization scheme involves trade-offs. What is the class of authorization policies that can be enforced? How much computation, communication, and storage is necessary to decide whether a request should be allowed to proceed? PolicyMaker [Blaze et al. 1999; Blaze et al. 1996; Blaze et al. 1998] makes one set of choices, and it was the first authorization scheme to focus on considerations of trust as an input to authorization decisions. [13] Policies, credentials, and trust relationships are expressed in PolicyMaker as imperative programs in a safe language; a generic policy compliance checker interprets these programs in order to check whether a policy is satisfied for some given credentials under given trust assumptions. REFEREE [Chu et al. 1997] extends this approach to include policies that govern actions by programs representing policies, credentials, and trust relationships; 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 which has declarative semantics and can be compiled into ordinary logic programs (including, for example, Prolog).

Binder [DeTreville 2002] was actually the first authentication scheme to employ a language based on Datalog; it embodies 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. Subsequent work based on Datalog includes SD3 [Jim 2001], the RT family of logics [Li et al. 2002], Cassandra [Becker and Sewell 2004], Soutei [Pimlott and Kselyov 2006], and then SecPAL [Becker et al. 2007]. DKAL [Gurevich and Neeman 2008] introduces a new dimension to credentials-based authorization by proposing extensions to SecPAL that prevent sensitive information

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

carried in credentials and authorization policies from leaking, even when users having different clearances share the same underlying authorization policies, database of credentials, and implementation.

SecPAL, which targeted 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 constraints 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 $\neg(A\ \textsf{says}\ F)$ 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 might 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).

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*, for example, provide a structure for localizing 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 for authentication, which are dependent on signatures and hashes for attributing statements to principals, do not provide support for reasoning about these operations within the logic. Nor does NAL or other prior work take the local reasoning restriction this next step to authorities, although doing so should be straightforward. As another important point of difference, 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, and none do so in anything approaching the generality needed for using analytic or synthetic bases in authorization. Taos and SecPAL 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 (W3C) 2007]) is also rooted in this general approach, which could bode well for the future. Bauer [2003] used proof carrying authorization for implementing access control to web pages. The Grey Project [Bauer et al. 2008; Bauer et al. 2005] 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].

5.2 Least Privilege versus Minimal Identity

Credentials used for authorizing a request reveal attributes of the requester. This can impinge on privacy, where *privacy* is defined as the right of an individual to control the dissemination and use of information about that individual. To protect privacy, we should strive to employ authorization policies that minimize the information needed about individuals.

Credentials-based authorization offers the flexibility to define policies and credentials involving only limited information about individuals. So credentials-based authorization has the potential to protect privacy by supporting system designs that instantiate the following principle, where an *identity* is a set of attributes.

Principle of Minimal Identity. Employ identities that embody the smallest set of attributes needed for the task at hand.

By way of comparison, in *identifier-based authorization*, access decisions depend only on a label associated with the principal making the request—there is no flexibility to disclose only certain attributes of the requester (although one can employ labels whose connection to an individual is impossible discern). Privacy with identifier-based authorization is, consequently, coarser-grained. Identifier-based authorization also admits privacy compromises from *linking attacks*, whereby labels used in different requests are correlated; credentials-based authorization can be less prone to linking attacks, because correlated credentials do not imply correlated requesters (although credentials that contain identifiers or contain sets of unique attributes can be traced back to specific requesters).

The contents of a credential can be seen as a kind of privilege, since it provides a basis for authorizing requests. Logical implication defines a partial ordering on these privileges: if $C \Rightarrow C'$ holds, then a credential conveying $A\,\mathsf{says}\,C$ is considered stronger than one that conveys $A\,\mathsf{says}\,C'$. We might then view credentials-based authorization through the lens of the well known Saltzer-Schroeder [1975] mandate:

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

And, in so doing, we discover this mandate offers the same guidance as the Principle of Minimal Identity! So with credentials-based authorization, security and privacy are both well served by expecting weaker credentials from principals.

This account also clarifies that merits of preferring analytic and synthetic bases for authorization to an axiomatic basis. The analytic and synthetic bases allow credentials to embody exactly what is needed for authorization, and are thus consistent with the Principles of Minimal Identity and Least Privilege. The axiomatic basis, however, is a form of identifier-based authorization and, therefore, it is going to be coarse-grained, subject to linking attacks, and unlikely to satisfy either the Principle of Minimal Identity or the Principle of Least Privilege. There are thus some strong, principled arguments in favor of the authorization architecture proposed in this paper.

APPENDIX

A. NAL INFERENCE RULES

NAL's axiomatization is equivalent to CDD [Abadi 2007; 2008], augmented with rules for existential quantification, sub-principals, and groups. The SAYS-I rule of NAL is called UNITM in CDD; and Abadi shows 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). Variable substitution and capture avoidance restrictions apply.

NAL rules derived from CDD:

$$\text{TRUE}\ \frac{}{\textsf{true}}$$

$$\text{IMP-E}\ \frac{F\ ,\ F \Rightarrow G}{G} \qquad \text{IMP-I(U)}\ \frac{\begin{array}{c}\text{U}\ \overline{F}\\ \vdots \\ \overline{G}\end{array}}{F \Rightarrow G}$$

$$\text{DEDUCE}\ \frac{A\,\textsf{says}\,(F \Rightarrow G)}{(A\,\textsf{says}\ F) \Rightarrow (A\,\textsf{says}\ G)}$$

$$\text{SAYS-I}\ \frac{F}{A\,\textsf{says}\ F} \qquad \text{SAYS-E}\ \frac{A\,\textsf{says}\ A\,\textsf{says}\ F}{A\,\textsf{says}\ F}$$

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

$$\text{OR-LEFT-I}\ \frac{F}{F \vee G} \qquad \text{OR-RIGHT-I}\ \frac{G}{F \vee G} \qquad \text{CASES}\ \frac{F \Rightarrow H\ ,\ G \Rightarrow H}{(F \vee G) \Rightarrow H}$$

$$\text{FORALL-I}\ \frac{F}{(\forall v : F)} \qquad \text{FORALL-E}\ \frac{(\forall v : F)}{F[v := \tau]}$$

NAL extensions to CDD:

$$\text{EXISTS-I}\ \frac{F[v := \tau]}{(\exists v : F)}$$

$$\text{EXISTS-E}\ \frac{F \Rightarrow G\ ,\ (\exists v : F)}{G}\ v \text{ not free in } G$$

$$\text{SUBPRIN}\ \frac{}{A \rightarrow A.\tau} \tag{13}$$

$$\text{EQUIV SUBPRIN}\ \frac{\tau_1 = \tau_2}{A.\tau_1 \rightarrow A.\tau_2} \tag{14}$$

$$\text{MEMBER}\ \frac{P[v := A]}{A \rightarrow [\![v : P]\!]} \tag{17}$$

$$\text{GROUP} \rightarrow \frac{(\forall v : P \Rightarrow (v \rightarrow A))}{[\![v : P]\!] \rightarrow A} \tag{18}$$

NAL derived inference rules:

$$\text{TRANS} \rightarrow \frac{C \rightarrow B\ ,\ B \rightarrow A}{C \rightarrow A} \tag{7}$$

$$\text{HAND-OFF} \frac{A \textsf{ says } (B \rightarrow A)}{B \rightarrow A} \tag{9}$$

$$\text{TRANS} \xrightarrow{\overline{v}:F} \frac{C \xrightarrow{\overline{v}:F} B\ ,\ B \xrightarrow{\overline{v}:F} A}{C \xrightarrow{\overline{v}:F} A} \tag{11}$$

$$\text{REST-HAND-OFF} \frac{A \textsf{ says } (B \xrightarrow{\overline{v}:F} A)}{B \xrightarrow{\overline{v}:F} A} \tag{12}$$

$$\text{GROUP MONOTONICITY} \frac{(\forall v : P \Rightarrow P')}{[\![v : P]\!] \rightarrow [\![v : P']\!]} \tag{19}$$

B. NAL PROOF CHECKER

We implemented a NAL proof checker in C.[14] Our goal was not to build a production-quality system but rather to ascertain whether a NAL proof checker could be fast enough to be on the critical path for accesses. In the interest of expedience, we axiomatized only the one theory (portions of arithmetic) needed for reasoning about objects used by our examples, and we paid no attention to low-level optimization of the proof checker's performance.

Preliminary measurements suggest that the NAL proof checker achieved our goals. A typical authorization in ConfDocs requires proofs involving roughly 1000 inference rules, and our proof checker validates these proofs in approximately 300 milliseconds—fast enough so that a user of ConfDocs senses no delay. We do ultimately plan to use NAL for authorizing accesses to all Nexus kernel resources, where significantly faster proof checking would be required. However, the power of a general purpose proof checker is unlikely to be needed for that setting, because policies, credentials, and proofs there will assume only a few simple forms. So special-purpose proof checkers, which can be considerably faster, will be implemented.

Stack-based Proof Checking

NAL proofs have natural representations as trees, with each node corresponding to a NAL formula: The root of the tree corresponds to the formula being proved; each leaf corresponds to an axiom or valid credential; and whenever a NAL formula P is derived using an inference rule $\mathcal{R}$ from n premises P_1, P_2, ..., P_n, then the node corresponding to P has label $\mathcal{R}$ and has a successor node corresponding to P_1, one corresponding to P_2, etc.

Proofs are conveyed to our NAL proof checker using a postfix-like representation of proof trees. The formula corresponding to the root of the tree is preceded by a list of inference rule names, axiom names, identifiers for credentials, and meta-rules (which allow terser encodings for some constructions). We adopted this representation because it promised performance advantages over the usual Hilbert-style proof representation as a

[14]C is the only high level language Nexus supports, so we had little choice here. Admittedly, other languages would have been better suited to the task.

sequence of formulas, each accompanied by a justification (itself the name of a credential, axiom, or an inference rule and list of previous line numbers that identify premises). In particular:

—Our postfix-like representation tends to be shorter than the Hilbert-style representation, because formulas can be long and the postfix-like representation does not contain intermediate formulas or references to lines in the proof. Our postfix-like representation only contains the formulas at the root and leaves of the tree, plus names (which are short) of the inference rules used to derive intermediate formulas.

—Our postfix-like representation allows proof-checking with a smaller footprint for its working memory. In the Hilbert-style representation, checking line i requires that lines 1 through $i-1$ be available in working memory; later lines can be stored on disk. In our postfix-like representation, the working memory need only contain a stack that stores premises for the next inference to be checked. Since each NAL inference rule involves only a few premises, this means the stack depth need only store a few elements (where each element stores an entire NAL formula).

A NAL proof in our postfix-like representation is checked by reading one line at a time and performing a series of reductions. Initially, there is an empty stack of *judgments*. Each line of the proof is read, pushed on to the stack, and a reduction initiated according to a rule (given below) selected based on the line. Roughly speaking, a line corresponding to an inference rule[15] requiring n premises, is processed as follows. The proof checker:

(1) pops n judgments from the stack for use as the rule's premises,

(2) checks any side conditions for the rule,

(3) constructs the rule's conclusion based on the premises, and

(4) pushes the rule's conclusion on to the stack.

Once all input lines have been processed, the proof is deemed correct if the stack contains a single element that matches the root of the proof tree being checked.

Our NAL proof checker has a built-in, hard-coded routine for each NAL inference rule that involves checking a side condition or modifying a sequent. Each such routine $\mathcal{R}$ can characterized in terms of (i) the contents of the stack top when $\mathcal{R}$ is activated, and (ii) how executing $\mathcal{R}$ changes the stack. Here is that list:

—`assume` $\mathcal{S}$.
Push judgment $\{\mathcal{S}\} \vdash \mathcal{S}$ on to stack.

—`impi` $\mathcal{S}$.
Pop $\Sigma \vdash \mathcal{S}'$ then push $\Sigma \setminus \{\mathcal{S}\} \vdash \mathcal{S} \Rightarrow \mathcal{S}'$.

—`rename` $\mathcal{S}$.
Pop $\Sigma \vdash \mathcal{S}'$ then push $\Sigma \vdash \mathcal{S}$, if $\mathcal{S} =_\alpha \mathcal{S}'$.

—`foralli` v.
Pop $\Sigma \vdash \mathcal{S}$ then push $\Sigma \vdash (\forall v : \mathcal{S})$.

[15]The NAL axiomatization in Appendix A does not use sequents; the proof checker does, because that is more convenient. But there is a trivial translation from proofs in terms of the axiomatization of Appendix A into the sequent-based logic used by our NAL proof checker.

—`foralle` τ.
Pop $\Sigma \vdash (\forall v : \mathcal{S})$ then push $\Sigma \vdash \mathcal{S}[v := \tau]$, provided no free variables in τ are captured during the substitution.

—`existsi` $(\exists v : \mathcal{S})$.
Pop $\Sigma \vdash \mathcal{S}[v := \tau]$ then push $\Sigma \vdash (\exists v : \mathcal{S})$, provided no free variables in τ are captured during the substitution.

—`existse`.
Pop $\Sigma \vdash \mathcal{S} \Rightarrow \mathcal{S}'$, and $\Sigma' \vdash (\exists v : \mathcal{S})$, then push $\Sigma \cup \Sigma' \vdash \mathcal{S}'$, provided v is not free in $\mathcal{S}'$.

—`member` $[\![v : \mathcal{S}]\!]$.
Pop $\Sigma \vdash \mathcal{S}[v := A]$, then push $\Sigma \vdash A \rightarrow [\![v : \mathcal{S}]\!]$, provided no free variables in A are captured during the substitution.

—`saysforall`.
Pop $\Sigma \vdash A \,\textsf{says}\, (\forall v : \mathcal{S})$ then push $\Sigma \vdash (\forall v : A \,\textsf{says}\, \mathcal{S})$, provided v is not free in A.

—`forallsays`.
Pop $\Sigma \vdash (\forall v : A \,\textsf{says}\, \mathcal{S})$ then push $\Sigma \vdash A \,\textsf{says}\, (\forall v : \mathcal{S})$, provided v is not free in A.

For the other NAL inference rules, because they do not involve checking side conditions or modifying sequents, we can invoke a proof checker routine that augments the proof checker's logic with new inference rules and derived inference rules. So as part of initialization for the NAL proof checker, this routine is used and all those remaining NAL inference rules are added.

Finally, the NAL proof checker supports a few additional rules that, by manipulating the stack in simple ways, allow some proofs to be represented in a more compact form:

—`pushdown` n.
Moves the top element on stack down n elements.

—`pullup` n.
Move the n^{th} element down stack up, to be at the top of the stack.

—`dup`.
Duplicates the top element of stack.

Implementation Details

The NAL proof checker is approximately 5000 lines of code, including about 2500 lines concerned with manipulating and parsing NAL formulas. The code for the proof checker can be instantiated in several ways:

Stand-alone service. A process (IPD, in Nexus terminology) monitors a well-known port for requests from clients. Three kinds of client requests are supported:

check. Checks correctness of a proof in the client request.

add_derived_inference_rule. Checks soundness and adds a new named inference rule to a locally maintained per-client list of inference rules. The rules on this list may be used by this client in subsequent proofs; proofs from other clients are unaffected.

add_inference_rule. Adds an arbitrary new inference rule to a locally maintained per-client list of inference rules. The proof checker makes no effort to check soundness of the new rule, so care must be taken when making this kind of request—unsound inference rules might allow arbitrary requests to satisfy any authorization policy.

Library Routines. The proof-checker library routines can be compiled and linked into a program. The entry points are: check, add_derived_inference_rule, and add_inference_rule.

Interactive Command. The proof checker command is invoked from the shell by a human user. The command reads an input file containing zero or more inference rules (which are temporarily added to the list of inference rules), zero or more derived inference rules (which are checked and then added to the list of rules), and one or more proofs (which are checked).

Guard Library. This is a wrapper around the proof-checker library routines. The interface allows a Nexus guard object to be created. Each guard object maintains a separate rule list and is configured with an authorization policy. A program processing a request invokes guard_check, which causes the guard object to
(i) invoke the proof checker on the provided proof,
(ii) validate that what is proved matches the authorization policy, and
(iii) check that each premise of the proof is backed by a valid credential (provided along with the proof).
A guard object includes helper routines for validating credentials (e.g., cryptographic signatures checks as well as methods to read the system state, time of day, dates, etc.); this set of routines is easily extended to handle new kinds of credentials.

The proof checker does not directly check proofs of claims about arithmetic or other objects that appear in beliefs. These are handled by invoking external routines.[16] The invocation occurs whenever, in the process of checking a proof, the NAL proof checker encounters a line

$$\texttt{extern}\ proof\_checker\_name :\ string$$

where $proof\_checker\_name$ is the name of some pre-installed proof checker and $string$ contains a formula F (using a syntax known to $proof\_checker\_name$) to be proved followed by a purported proof. F is then returned to the NAL proof checker if and only if the proof provided for F checks.

ACKNOWLEDGMENTS

Martin Abadi has been an extremely helpful sounding board throughout the evolution of this work, and he brought GROUP MONOTONICITY (19) to our attention. In addition, he, Lujo Bauer, Willem de Bruijn, Robbert Constable, Andrew Myers, and Ted Wobber gave us useful feedback on an early draft of this paper. We are very grateful to them all.

REFERENCES

ABADI, M. 2007. Access Control in a Core Calculus of Dependency. *Electronic Notes in Theoretical Computer Science 172*, 5–31.

ABADI, M. 2008. Variations in Access Control Logic. In *Deontic Logic in Computer Science*. 96–109.

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A Calculus for Access Control in Distributed Systems. *ACM Programming Languages and Systems 15,* 4, 706–734.

APPEL, A. W. AND FELTEN, E. W. 1999. Proof-Carrying Authentication. In *ACM Computer and Communications Security*. ACM, New York, NY, USA, 52–62.

[16]An alternative is to use add_inference_rule and extend the NAL proof checker by adding inference rules for each new theory. We rejected this approach both because it is clumsy and because it precludes using decision procedures and model checkers where they exist.

BAUER, L. 2003. Access Control for the Web via Proof-Carrying Authorization. Ph.D. thesis, Princeton University, Princeton, NJ, USA.

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 *Human Factors in Computing Systems*. 543–552.

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

BAUER, L., GARRISS, S., AND REITER, M. K. 2005. Distributed Proving in Access-Control Systems. In *IEEE Security and Privacy*. IEEE Computer Society Press, Washington, DC, USA, 81–95.

BECKER, M., FOURNET, C., AND GORDON, A. 2007. Design and Semantics of a Decentralized Authorization Language. In *IEEE Computer Security Foundations*. IEEE Computer Society Press, Washington, DC, USA, 3–15.

BECKER, M. Y. AND NANZ, S. 2007. A Logic for State-Modifying Authorization Policies. In *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 *IEEE Computer Security Foundations*. IEEE Computer Society Press, Washington, DC, USA, 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 *SOSP '95: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 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*, 185–210.

BLAZE, M., FEIGENBAUM, J., AND KEROMYTIS, A. D. 1998. KeyNote: Trust Management for Public-Key Infrastructures. In *Security Protocols Workshop*. 59–63.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized Trust Management. In *IEEE Security and Privacy*. IEEE Computer Society Press, Washington, DC, USA, 164–173.

BLAZE, M., FEIGENBAUM, J., AND STRAUSS, M. 1998. Compliance Checking in the PolicyMaker Trust Management System. In *Financial Cryptography*. Springer-Verlag, 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 *Network and Distributed System Security Symposium*. Internet Society, San Diego, California.

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

DENNING, D. E. 1976. A Lattice Model of Secure Information Flow. *Communications of the ACM 19,* 5 (May), 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 *IEEE Security and Privacy*. IEEE Computer Society Press, Washington, DC, USA, 105–113.

DOCBOOK. `http://www.docbook.org/`.

ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. 1999. SPKI Certificate Theory. *IETF RFC 2693*.

ERLINGSSON, Ú. AND SCHNEIDER, F. B. 1999. SASI Enforcement of Security Policies: A Retrospective. In *New Security Paradigms Workshop*. ACM Press, New York, NY, USA, 87–95.

GARG, D., BAUER, L., BOWERS, K., PFENNING, F., AND REITER, M. 2006. A Linear Logic of Authorization and Knowledge. In *European Symposium on Research in Computer Security*. Springer-Verlag, 297–312.

GARG, D. AND PFENNING, F. 2006. Non-Interference in Constructive Authorization Logic. In *IEEE Computer Security Foundations*. IEEE Computer Society Press, Washington, DC, USA, 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 *Usenix Security Symposium*.

GRAY, C. AND CHERITON, D. 1989. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *SOSP '89: Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 202–210.

GUREVICH, Y. AND NEEMAN, I. 2008. Dkal: Distributed-knowledge authorization language. In *CSF '08: Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*. IEEE Computer Society, Washington, DC, USA, 149–162.

HAMLEN, K. W., MORRISETT, G., AND SCHNEIDER, F. B. 2006. Certified In-lined Reference Monitoring on .NET. In *ACM Workshop on Programming Languages and Analysis for Security*. ACM, New York, NY, USA, 7–16.

HOWELL, J. 2000. Naming and Sharing Resources Across Administrative Boundaries. Ph.D. thesis, Dartmouth College, Hanover, New Hampshire, USA.

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

JIM, T. 2001. SD3: A Trust Management System with Certified Evaluation. In *IEEE Security and Privacy*. IEEE Computer Society Press, Washington, DC, USA, 106–115.

LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. 1992. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems 10*, 265–310.

LESNIEWSKI-LAAS, C., FORD, B., STRAUSS, J., MORRIS, R., AND KAASHOEK, M. F. 2007. Alpaca: Extensible Authorization for Distributed Services. In *ACM Computer and Communications Security*. ACM, New York, NY, USA, 432–444.

LI, N., GROSOF, B. N., AND FEIGENBAUM, J. 2003. Delegation Logic: A Logic-Based Approach to Distributed Authorization. *ACM Transactions on Information and System Security 6*, 128–171.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a Role-Based Trust-Management Framework. In *IEEE Security and Privacy*. IEEE Computer Society Press, Washington, DC, USA, 114–130.

NECULA, G. C. 1997. Proof-Carrying Code. In *ACM Principles of Programming Languages*. 106–119.

NEW INTERNATIONAL VERSION BIBLE. Terms of Use. `http://www.ibs.org/bibles/termsofuse.php`.

OPENOFFICE. `http://www.openoffice.org/`.

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-security-1.0.pdf`.

PFENNING, F. AND SCHÜRMANN, C. 1999. System Description: Twelf—A Meta-Logical Framework for Deductive Systems. In *International Conference on Automated Deduction*. Springer-Verlag.

PIMLOTT, A. AND KSELYOV, O. 2006. Soutei, a Logic-Based Trust Management System, System Description. In *FLOPS 2006*, M. Hagiya and P. Wadler, Eds. Lecture Notes in Computer Science, vol. 3945. 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. *IEEE 63*, 9 (Sept.), 1278–1308.

SANDHU, R. S. 1993. Lattice-Based Access Control Models. *IEEE Computer 26*, 11, 9–19.

SHIEH, A., WILLIAMS, D., SIRER, E. G., AND SCHNEIDER, F. B. 2005. Nexus: A New Operating System for Trustworthy Computing. In *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 *SOSP '99: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*. ACM, New York, NY, USA, 202–216.

SYVERSON, P. F. AND STUBBLEBINE, S. G. 1999. Group Principals and the Formalization of Anonymity. In *World Congress on Formal Methods in the Development of Computing Systems*. Springer-Verlag, London, UK, 814–833.

TRUSTED COMPUTING GROUP. Trusted Platform Module (TPM) Specification, version 1.2. `https://www.trustedcomputinggroup.org/specs/TPM/`.

WAHBE, R., LUCCO, S., ANDERSON, T. E., AND GRAHAM, S. L. 1993. Efficient Software-Based Fault Isolation. In *Symposium on Operating Systems Principles*. 203–216.

WALSH, K. Nexus Authorization Logic. Ph.D. dissertation (in preparation), Cornell University, Ithaca, New York, USA.

WEISSMAN, C. 1969. Security Controls in the ADEPT-50 Time-Sharing System. In *AFIPS Fall Joint Computer Conference (FJCC)*. Vol. 35. 119–133.

WESLEYAN CINEMA ARCHIVE. Guidelines for Direct Quotations from Material Held by the Wesleyan Cinema Archives. `http://www.wesleyan.edu/cinema/guidelines.htm`.

WOBBER, E., ABADI, M., BURROWS, M., AND LAMPSON, B. 1994. Authentication in the TAOS Operating System. *ACM Transactions on Computer Systems 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. July.

WORLD WIDE WEB CONSORTIUM (W3C). Markup Validation Service. `http://validator.w3.org/`.

WORLD WIDE WEB CONSORTIUM (W3C). 2007. Web Services Policy 1.5 - Framework (WS-Policy). `http://www.w3.org/TR/ws-policy/`.

Submitted September 13, 2009