A Logical Approach to Access Control, Security, and Trust

Shiu-Kai Chin and Susan Older
Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, New York 13244
http://www.ecs.syr.edu/faculty/chin http://www.cis.syr.edu/∼sueo

Abstract

Designers, auditors, and certifiers of trustworthy systems must rigorously assess compliance with security policies. Because security is best built into systems at all levels of abstraction, engineers and other practitioners who design, verify, or certify trustworthy systems need the capability to reason rigorously about security policies in general, and access decisions in particular. What is required is a logic or calculus general enough to be useful from the concrete hardware level to the abstract policy level that also captures access-control concepts such as authorization, certified statements, jurisdiction, and delegation. Ideally, this calculus should be straightforward for practitioners to use, much like the propositional logic used in hardware design by engineers. We have created an access-control logic that meets these requirements and have used this logic to account for security, trust, and access policies in hardware, software, protocols, and concepts of operations. We give an overview of the logic and its application to hardware, protocols, and policy.

Keywords: access control, security, trust, logic.

I. Introduction

The need for trusted information systems is ever growing. Concurrent with this increasing need are the challenges of assuring the *trustworthiness* of systems at a time when systems are growing ever more interconnected and complex.

The principles of building trusted systems remain the same [10][4]:

- *complete mediation:* all accesses to objects must be checked to ensure they are allowed,
- *least privilege:* grant only the capabilities necessary to compete the specified task, and
- *economy of mechanism:* security mechanisms should be as simple as possible.

Nevertheless, *rigorously* assessing that any given access request is allowed, that principals have the access rights required, and that mechanisms are specified and implemented correctly remains daunting. How will designers, verifiers, and certifiers of hardware, firmware, software, and protocols understand precisely and accurately policy statements and assure themselves of compliance?

The left side of Figure 1 shows what is expected of hardware engineers: when given a hardware design, the primary inputs, and the values in the registers, hardware engineers *derive* the value of signals anywhere in the integrated circuit using logic. The right side of Figure 1 shows what is expected of engineers designing secure systems: when given (1) a request to access a protected resource, (2) an access policy, and (3) assumptions about whose authority is trusted and their jurisdiction, engineers should be able to *derive* whether or not the reference monitor protecting the resource should allow access to the resource.

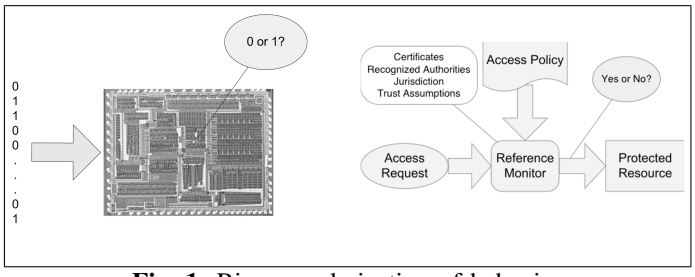

Fig. 1: Rigorous derivation of behavior

Hardware designers routinely do what is expected because logic is at the foundation of hardware design. Engineers of secure systems do not have the benefit of a similar logic. Hence, verification of security does not meet the same standard as verification of functional correctness in computer hardware.

Our objective is to enable designers, verifiers, and certifiers to rigorously reason about access control, security, and trust and to do so at the concrete hardware level up through the abstract level of security policies and concept of operations. We have devised an access-control logic that is straightforward for practitioners to learn and apply broadly. This logic is based on a logic in [1]. We have both simplified and extended [1] by substituting delegation for roles and adding the semantics for partially ordered confidentiality and integrity labels and levels. We have used this logic to describe a wide variety of access-control applications including role-based access control (RBAC) [8] and delegation in electronic retail payment systems [6].

The remainder of this paper is organized as follows. Section II gives an overview of the syntax, semantics, and inference rules of the logic. Section III is an example of mandatory access control at the physical memory level. Section IV provides a delegation example. Section V describes how the syntax and semantics of the logic are extended to accommodate security levels and labels. Section VI is an example of information confidentiality. We conclude in Section VII.

II. Overview of the Access Control Logic

A. Syntax and Semantics

Syntax of Principal Expressions: Principals are the actors in a system, such as people, processes, cryptographic keys, personal identification numbers (PINs), userid–password pairs, and so on. Principals are either *simple* or *compound*. **PName** is the collection of all simple principal names, which can be used to refer to any simple principal. For example, the following are all allowable principal names: *Alice*, *Bob*, the key K_{Alice}, the PIN 1234, and the userid–password pair $\langle \texttt{alice}, \textit{bAdPsWd!} \rangle$.

Compound principals are abstract entities that connote a combination of principals: for example, "the President in conjunction with Congress" connotes an abstract principal comprising both the President and Congress. Intuitively, such a principal makes exactly those statements that are made by *both* the President and Congress. Similarly, "the reporter quoting her source" connotes an abstract principal that comprises both the reporter and her source. Intuitively, a statement made by such a principal represents a statement that the reporter is (rightly or wrongly) attributing to his source.

The set **Princ** of *all* principal expressions is given by the following BNF specification:

$$\textbf{Princ} \quad ::= \quad \textbf{PName} \mid \textbf{Princ} \ \& \ \textbf{Princ} \mid \textbf{Princ} \mid \textbf{Princ}$$

That is, a principal expression is either a simple name, an expression of form $P \ \& \ Q$ (where P and Q are both principal expressions), or an expression of form $P \mid Q$ (where, again, P and Q are both principal expressions).

Syntax of Logical Formulas: The abstract syntax of logical formulas **Form** are constructed from the set of *principal names* and a countable set of *propositional variables* **PropVar**:

$$\begin{aligned}\textbf{Form} \quad ::= \quad & \textbf{PropVar} \mid \neg \ \textbf{Form} \mid \\ & (\textbf{Form} \vee \textbf{Form}) \mid (\textbf{Form} \wedge \textbf{Form}) \mid \\ & (\textbf{Form} \supset \textbf{Form}) \mid (\textbf{Form} \equiv \textbf{Form}) \mid \\ & (\textbf{Princ} \Rightarrow \textbf{Princ}) \mid (\textbf{Princ} \ \mathsf{says} \ \textbf{Form}) \mid \\ & (\textbf{Princ} \ \mathsf{controls} \ \textbf{Form}) \mid \textbf{Princ} \ \mathsf{reps} \ \textbf{Princ} \ \mathsf{on} \ \textbf{Form}\end{aligned}$$

The first six cases deal with standard propositional logic: propositional variables, negation, conjunction, disjunction, implication, and equivalence. The remaining four cases are specific to access control.

1) $P \ \mathsf{says} \ \varphi$ asserts that principal P made the statement φ.
2) $P \Rightarrow Q$ (pronounced "P speaks for Q") indicates that every statement made by P can also be viewed as a statement from Q.
3) $P \ \mathsf{controls} \ \varphi$ represents authority or trust. It is syntactic sugar for the implication $(P \ \mathsf{says} \ \varphi) \supset \varphi$. P is a trusted authority with respect to the statement φ.
4) $P \ \mathsf{reps} \ Q \ \mathsf{on} \ \varphi$ represents delegation. It is syntactic sugar for $(P \mid Q \ \mathsf{says} \ \varphi) \supset (Q \ \mathsf{says} \ \varphi)$. P is a trusted authority on what Q says regarding φ.

Semantics: The semantics of formulas is given via Kripke structures, as follows.

Definition: A *Kripke structure* $\mathcal{M}$ is a three-tuple $\langle W, I, J \rangle$, where:

- W is a nonempty set, whose elements are called *worlds*.
- $I : \textbf{PropVar} \to \mathcal{P}(W)$ is an *interpretation* function that maps each propositional variable p to a set of worlds.
- $J : \textbf{PName} \to \mathcal{P}(W \times W)$ is a function that maps each principal name A into a relation on worlds (i.e., a subset of $W \times W$).

Given the above, we define the extended function $\hat{J} : \textbf{Princ} \to \mathcal{P}(W \times W)$ inductively on the structure of principal expressions, where $A \in \textbf{PName}$.

$$\begin{aligned}\hat{J}(A) &= J(A) \\ \hat{J}(P \ \& \ Q) &= \hat{J}(P) \cup \hat{J}(Q) \\ \hat{J}(P \mid Q) &= \hat{J}(P) \circ \hat{J}(Q).\end{aligned}$$

Note: $R_1 \circ R_2 = \{(x, z) \mid \exists y.(x, y) \in R_1 \text{ and } (y, z) \in R_2\}$. $\diamond$

Definition: Each Kripke structure $\mathcal{M} = \langle W, I, J \rangle$ gives rise to a semantic function

$$\mathcal{E}_{\mathcal{M}}[\![-]\!] : \textbf{Form} \to \mathcal{P}(W),$$

where $\mathcal{E}_{\mathcal{M}}[\![\varphi]\!]$ is the set of worlds in which φ is considered true. $\mathcal{E}_{\mathcal{M}}[\![\varphi]\!]$ is defined inductively on the structure of φ, as follows:

$$\begin{aligned}\mathcal{E}_{\mathcal{M}}[\![p]\!] &= I(p) \\ \mathcal{E}_{\mathcal{M}}[\![\neg\varphi]\!] &= W - \mathcal{E}_{\mathcal{M}}[\![\varphi]\!] \\ \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \wedge \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\ \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \vee \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!] \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\ \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] &= (W - \mathcal{E}_{\mathcal{M}}[\![\varphi_1]\!]) \cup \mathcal{E}_{\mathcal{M}}[\![\varphi_2]\!] \\ \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \equiv \varphi_2]\!] &= \mathcal{E}_{\mathcal{M}}[\![\varphi_1 \supset \varphi_2]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\varphi_2 \supset \varphi_1]\!] \\ \mathcal{E}_{\mathcal{M}}[\![P \Rightarrow Q]\!] &= \begin{cases} W, & \text{if } \hat{J}(Q) \subseteq \hat{J}(P) \\ \emptyset, & \text{otherwise} \end{cases} \\ \mathcal{E}_{\mathcal{M}}[\![P \ \mathsf{says} \ \varphi]\!] &= \{w | \hat{J}(P)(w) \subseteq \mathcal{E}_{\mathcal{M}}[\![\varphi]\!]\} \\ \mathcal{E}_{\mathcal{M}}[\![P \ \mathsf{controls} \ \varphi]\!] &= \mathcal{E}_{\mathcal{M}}[\![(P \ \mathsf{says} \ \varphi) \supset \varphi]\!] \\ \mathcal{E}_{\mathcal{M}}[\![P \ \mathsf{reps} \ Q \ \mathsf{on} \ \varphi]\!] &= \mathcal{E}_{\mathcal{M}}[\![(P \mid Q \ \mathsf{says} \ \varphi) \supset Q \ \mathsf{says} \ \varphi]\!]\end{aligned}$$

Note that, in the definition of $\mathcal{E}_{\mathcal{M}}[\![P \ \mathsf{says} \ \varphi]\!]$, $\hat{J}(P)(w)$ is simply the image of world w under the relation $\hat{J}(P)$. $\diamond$

The semantic functions $\mathcal{E}_{\mathcal{M}}$ provide a fully defined and fully disclosed interpretation for the formulas of the logic. These functions precisely define the meaning of statements and what is described in the logic.

In practice, reasoning at the level of Kripke structures is cumbersome. Instead, we use logical rules to reason about access control.

B. Logical Rules

Logical rules in our access-control logic have the form

$$\frac{H_1 \quad \cdots \quad H_k}{C,}$$

where $H_1 \cdots H_k$ and C are formulas in the logic. $H_1 \cdots H_k$ are the *hypotheses* or *premises* and C is the *consequence* or *conclusion*. Informally, we read logical rules as "if all the hypotheses above the line are true, then the conclusion below

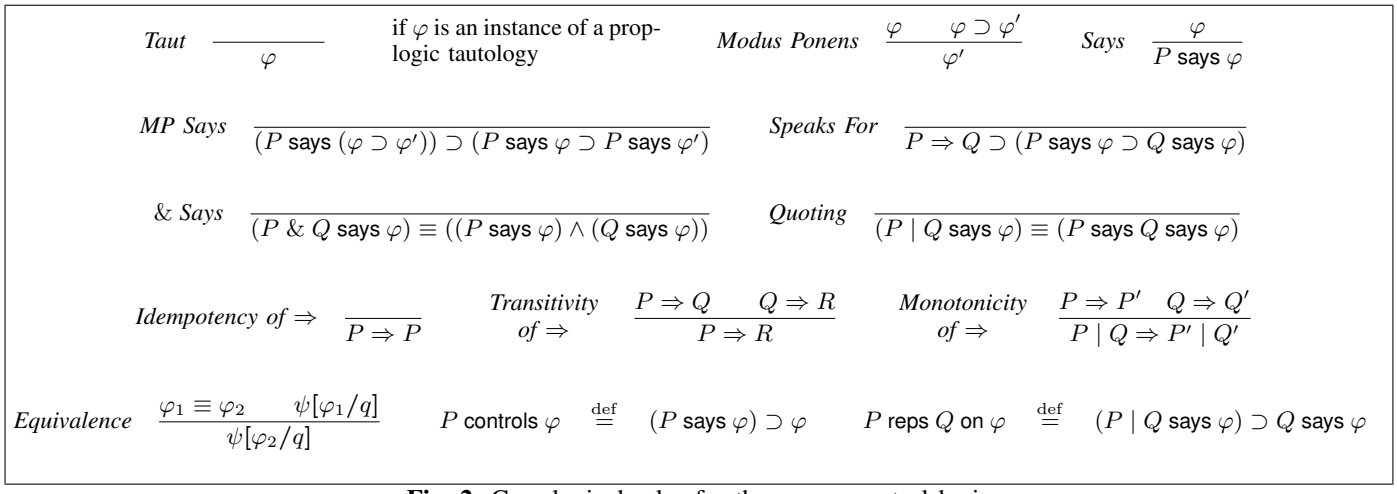

Fig. 2: Core logical rules for the access-control logic

the line is also true." If there are no hypotheses, then the logical rule is an *axiom*.

Logical rules are used to manipulate well-formed formulas of the logic. If all the hypotheses of a rule are written down (derived) then the conclusion of the rule also can be written down (derived). All logical rules must maintain logical consistency. If all logical rules are *sound*, as defined below, then logical consistency is assured.

Definition: A logical rule

$$\frac{H_1 \quad \cdots \quad H_k}{C,}$$

is *sound* if, for all Kripke structures $\mathcal{M}$, whenever $\mathcal{M}$ satisfies all the hypotheses $H_1 \cdots H_k$, then $\mathcal{M}$ also satisfies C. $\diamond$

Figure 2 shows the core logical rules for the logic. All the rules are proved sound with respect to the Kripke semantics. Figure 3 shows three useful derived rules, which are proved using the core rules.

III. Hardware Example

Figure 4 contains a block diagram of a simple virtual machine (VM) with an instruction register IR and an accumulator ACC. Instructions from VM are monitored by the virtual machine monitor (VMM). VMM has a *memory access register* (MAR) pointing to a real address in physical memory and a *relocation register* (RR) specifying the *base* address of the memory segment being accessed in physical memory and the segment's *bound* (number of memory locations). The physical memory shown has q memory locations (0 through $q-1$). The memory has three segments, corresponding to Alice, Bob, and Carol. Alice's segment starts at physical address $base_{Alice}$ and ends at $base_{Alice} + bound_{Alice} - 1$.

Values in registers are described in the logic as statements made by registers. For example, suppose the value loaded in IR is the instruction LDA @A (i.e., load ACC with the value in virtual address A in physical memory). From an access-control perspective, the question is whether this instruction should be executed or *trapped* by VMM, which mediates all access requests to physical memory.

VMM grants access if (1) the virtual address A is less than or equal to *bound* and, (2) $base + \text{A} < q$ (i.e., the physical address corresponding to A is within the physical memory address limit). Otherwise, the instruction is *trapped* by VMM and control is turned over to the supervisor.

The decision to grant or deny (trap) request LDA @A depends on the *state* of VM and VMM and the *mandatory access control* (MAC) policy governing access to physical memory. The state of the machine (VM and VMM) is given by the register values. The registers used to decide access are IR and RR; ACC and MAR are not used. The states of IR and RR are given by:

$$IR \text{ says } \langle \text{LDA @A} \rangle \quad \text{and} \quad RR \text{ says } \langle (base_{Alice}, bound_{Alice}) \rangle,$$

where $\langle \text{LDA @A} \rangle$ and $\langle (base_{Alice}, bound_{Alice}) \rangle$ denote "it is a good idea to execute LDA @A" and "the value is in fact $(base_{Alice}, bound_{Alice})$."

The request to execute LDA @A is *trapped* when either the real address $base_{Alice} + \text{A}$ is greater or equal to q (i.e., the size of physical memory), or the virtual address A exceeds the segment bound $bound_{Alice}$. This policy is expressed as follows:

$$IR \text{ says } \langle \text{LDA @A} \rangle \supset (RR \text{ says } \langle (base_{Alice}, bound_{Alice}) \rangle \supset (((base_{Alice} + \text{A} \geq q) \vee (\text{A} \geq bound_{Alice})) \supset \langle trap \rangle)),$$

where $\langle trap \rangle$ denotes "it is a good idea to trap."

The request to execute LDA @A is granted if virtual address A falls within the bound of real memory and the segment. This is expressed as follows:

$$IR \text{ says } \langle \text{LDA @A} \rangle \supset (RR \text{ says } \langle (base_{Alice}, bound_{Alice}) \rangle \supset ((base_{Alice} + \text{A} < q) \supset ((\text{A} < bound_{Alice}) \supset \langle \text{LDA @A} \rangle))).$$

Figure 5 gives a simple proof justifying granting execution for the instruction LDA @5 (loading the accumulator with the contents of address 5 in the active segment), where the base address of the segment is 8, the segment bound is 16, and the size of physical memory is 16. Lines 1–3 are the starting assumptions: the instruction request, the state of the relocation register RR, and the mandatory access control

$$\textit{Controls}\ \frac{P\ \mathsf{controls}\ \varphi \quad P\ \mathsf{says}\ \varphi}{\varphi} \qquad \textit{Derived Speaks For}\ \frac{P \Rightarrow Q \quad P\ \mathsf{says}\ \varphi}{Q\ \mathsf{says}\ \varphi} \qquad \textit{Reps}\ \frac{Q\ \mathsf{controls}\ \varphi \quad P\ \mathsf{reps}\ Q\ \mathsf{on}\ \varphi \quad P \mid Q\ \mathsf{says}\ \varphi}{\varphi}$$

Fig. 3: Derived rules

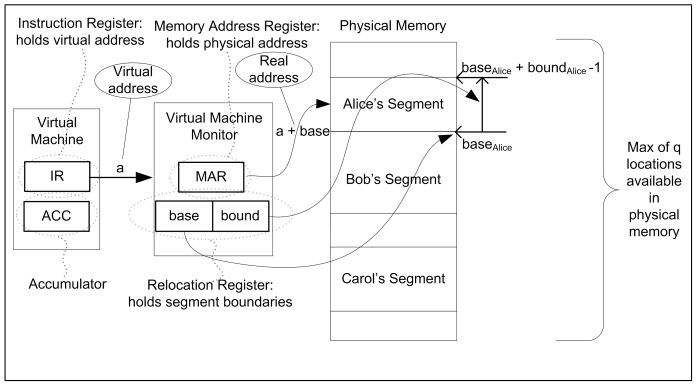

Fig. 4: Virtual Machine and Virtual Machine Monitor

1.	IR says $\langle$LDA @5$\rangle$	request
2.	RR says $\langle(8,16)\rangle$	RR value
3.	IR says $\langle$LDA @5$\rangle \supset$ $(RR$ says $\langle(8,16)\rangle \supset$ $((8+5<32) \supset$ $((5<16) \supset \langle$LDA @5$\rangle))))$	mandatory access policy
4.	RR says $\langle(8,16)\rangle \supset$ $((8+5<32) \supset$ $((5<16) \supset \langle$LDA @5$\rangle)))$	1,3 Modus Ponens
5.	$(8+5<32) \supset$ $((5<16) \supset \langle$LDA @5$\rangle))$	2,4 Modus Ponens
6.	$8+5<32$	Taut
7.	$(5<16) \supset \langle$LDA @5$\rangle$	6, 5 Modus Ponens
8.	$5<16$	Taut
9.	$\langle$LDA @5$\rangle$	8, 7 Modus Ponens

Fig. 5: Hardware proof

policy for the specific memory segment and physical memory. The remaining lines are *derived* by applying the *Modus Ponens* and *Taut* inference rules in Figure 2. The proof gives us a theorem in the form of a *derived* inference rule:

$$\frac{\begin{array}{c} IR\ \mathsf{says}\ \langle\text{LDA @5}\rangle \qquad RR\ \mathsf{says}\ \langle(8,16)\rangle \\ IR\ \mathsf{says}\ \langle\text{LDA @5}\rangle \supset (RR\ \mathsf{says}\ \langle(8,16)\rangle \supset ((8+5<32) \supset \\ ((5<16) \supset \langle\text{LDA @5}\rangle)))\end{array}}{\langle\text{LDA @5}\rangle.}$$

The above rule describes the behavior of VMM as a sound inference rule.

IV. Delegation Example

Delegation, whereby representatives are given authority to act on behalf of others, is a crucial capability to capture in the logic. Delegation is used extensively in electronic networks as the principals originating transactions rarely, if ever, interact directly with resources they are seeking to access. Usually, access is accomplished using an intermediary process or delegation.

To illustrate how delegation works, consider a simplified health-care proxy where Bob is Alice's delegate in case Alice is in a coma. Alice's wishes are to not be resuscitated if she

1.	S_{Alice} says $(Bob$ reps $Alice$ on $(coma \supset dnr))$	Medical proxy
2.	$Alice$ controls $(coma \supset dnr)$	Policy
3.	$Alice$ controls $(Bob$ reps $Alice$ on $(coma \supset dnr))$	Policy
4.	$S_{Alice} \Rightarrow Alice$	Trust assumption
5.	$coma$	*Alice*'s medical condition
6.	Bob says $(Alice$ says $(coma \supset dnr))$	Bob's statement
7.	$Alice$ says $(Bob$ reps $Alice$ on $(coma \supset dnr))$	4,1 Derived Speaks For
8.	Bob reps $Alice$ on $(coma \supset dnr)$	3, 7 Controls
9.	Bob says $(Alice$ says $(coma \supset dnr))$ $\equiv Bob \mid Alice$ says $(coma \supset dnr)$	Quoting
10.	$Bob \mid Alice$ says $(coma \supset dnr)$	9, 6 Equivalence
11.	$coma \supset dnr$	2, 8, 10 reps
12.	dnr	5, 11 Modus Ponens

Fig. 6: Delegation proof

is in a coma, which is represented by the formula

$$coma \supset dnr,$$

where $coma$ and dnr respectively denote Alice being in a coma and her desire not to be resuscitated.

If Alice is in a coma, she is unable to speak for herself. Alice designates Bob to be her delegate to say $coma \supset dnr$, when she cannot. She puts it in writing and signs it with her signature S_{Alice}. The signed statement is formulated as:

$$S_{Alice}\ \mathsf{says}\ (Bob\ \mathsf{reps}\ Alice\ \mathsf{on}\ (coma \supset dnr)).$$

Hospital policies must be aligned with Alice's if her wishes are to be honored. Also, her signature must be recognized as hers. Two policy statements regarding Alice's authority or jurisdiction are required along with a trust assumption. They are:

1) Alice's wishes when she is in a coma count (i.e., she has jurisdiction in this case): $Alice$ controls $(coma \supset dnr)$.
2) Alice has authority to designate Bob as her delegate to say on her behalf not to be resuscitated if she is in a coma: $Alice$ controls $(Bob$ reps $Alice$ on $(coma \supset dnr))$.
3) Alice's signature is recognized: $S_{Alice} \Rightarrow Alice$.

In the unfortunate event that Alice lapses into a coma and Bob says Alice does not wish to be revived in this event, the behavior of Alice's hospital is expressed by the following derived inference rule and theorem:

$$\frac{\begin{array}{c} S_{Alice}\ \mathsf{says}\ (Bob\ \mathsf{reps}\ Alice\ \mathsf{on}\ (coma \supset dnr)) \\ Alice\ \mathsf{controls}\ (coma \supset dnr) \\ Alice\ \mathsf{controls}\ (Bob\ \mathsf{reps}\ Alice\ \mathsf{on}\ (coma \supset dnr)) \\ S_{Alice} \Rightarrow Alice \\ coma \qquad Bob\ \mathsf{says}\ (Alice\ \mathsf{says}\ (coma \supset dnr))\end{array}}{dnr.}$$

The soundness of the above behavior is proved in Figure 6.

V. Adding Security Levels

Confidentiality access policies are common in the military and also in industry. Most models use a partial ordering of confidentiality levels to which security labels and and clearances are assigned. A common set of confidentiality levels and associated labels are $\{unclassified, confidential, secret, top\ secret\}$ and $\{\text{U,C,S,TS}\}$. The labels are what is stamped on documents, their interpretation is the mapping to confidentiality levels. The partial ordering of levels typically forms a lattice. In this section we outline how confidentiality levels are added to the syntax and semantics of the access-control logic. The same approach is use to add other kinds of levels and partial orders, such as those for grading *integrity* or *availability*.

Syntax: The first step is to introduce syntax for describing and comparing security levels. **SecLabel** is the collection of *simple security labels*, which are used as names for the confidentiality levels (e.g., TS, S, C, and U).

Often, we refer abstractly to a principal P's security level. We define the larger set **SecLevel** of *all* possible security-level expressions:

$$\textbf{SecLevel} \quad ::= \quad \textbf{SecLabel} \ / \ \mathsf{slev}(\textbf{PName}).$$

A security-level expression is either a simple security label or an expression of the form $\mathsf{slev}(A)$, where A is a simple principal name. Informally, $\mathsf{slev}(A)$ refers to the security level of principal A.

Finally, we extend our definition of well-formed formulas to support comparisons of security levels:

$$\textbf{Form} \quad ::= \quad \textbf{SecLevel} \leq_s \textbf{SecLevel} \ / \ \textbf{SecLevel} =_s \textbf{SecLevel}$$

Informally, a formula such as $\text{C} \leq_s \mathsf{slev}(\textit{Kate})$ states that Kate's security level is greater than or equal to the security level C. Similarly, a formula such as $\mathsf{slev}(\textit{Barry}) =_s \mathsf{slev}(\textit{Joe})$ states that Barry and Joe have been assigned the same security level.

Semantics: Providing formal and precise meanings for the newly added syntax requires us to first extend our Kripke structures with additional components that describe security classification levels. Specifically, we introduce extended Kripke structures of the form

$$\mathcal{M} = \langle W, I, J, K, L, \preceq \rangle,$$

where:

- W, I, and J are as defined earlier.
- K is a non-empty set, which serves as the universe of *security levels*.
- $L : (\textbf{SecLabel} \cup \textbf{PName}) \to K$ is a function that maps each security label and each simple principal name to a security level. L is extended to work over arbitrary security-level expressions, as follows:

  $$L(\ \mathsf{slev}(A)) = L(A),$$

  for every simple principal name A.
- $\preceq \subseteq K \times K$ is a partial order on K: that is, $\preceq$ is *reflexive* (for all $k \in K$, $k \preceq k$), *transitive* (for all $k_1, k_2, k_3 \in K$, if $k_1 \preceq k_2$ and $k_2 \preceq k_3$, then $k_1 \preceq k_3$), and *anti-symmetric* (for all $k_1, k_2 \in K$, if $k_1 \preceq k_2$ and $k_2 \preceq k_1$, then $k_1 = k_2$).

$$\ell_1 =_s \ell_2 \stackrel{\text{def}}{=} (\ell_1 \leq_s \ell_2) \wedge (\ell_2 \leq_s \ell_1)$$

$$\textit{Reflexivity of } \leq_s \quad \frac{}{\ell \leq_s \ell}$$

$$\textit{Transitivity of } \leq_s \quad \frac{\ell_1 \leq_s \ell_2 \qquad \ell_2 \leq_s \ell_3}{\ell_1 \leq_s \ell_3}$$

$$sl \leq_s \quad \frac{\mathsf{slev}(P) =_s \ell_1 \quad \mathsf{slev}(Q) =_s \ell_s \quad \ell_1 \leq_s \ell_2}{\mathsf{slev}(P) \leq_s \ \mathsf{slev}(Q)}$$

Fig. 7: Inference rules for relating security levels

Using these extended Kripke structures, we extend the semantics for our new well-formed expressions as follows:

$$\mathcal{E}_{\mathcal{M}}[\![\ell_1 \leq_s \ell_2]\!] \quad = \quad \begin{cases} W, & \text{if } L(\ell_1) \preceq L(\ell_2) \\ \emptyset, & \text{otherwise} \end{cases}$$

$$\mathcal{E}_{\mathcal{M}}[\![\ell_1 =_s \ell_2]\!] \quad = \quad \mathcal{E}_{\mathcal{M}}[\![\ell_1 \leq_s \ell_2]\!] \cap \mathcal{E}_{\mathcal{M}}[\![\ell_2 \leq_s \ell_1]\!].$$

As these definitions suggest, the expression $\ell_1 =_s \ell_2$ is simply syntactic sugar for $(\ell_1 \leq_s \ell_2) \wedge (\ell_2 \leq_s \ell_1)$.

Logical Rules: Based on the extended Kripke semantics we introduce logical rules that support the use of security levels to reason about access requests. Specifically, the definition, reflexivity, and transitivity rules in Figure 7 reflect that $\leq_s$ is a partial order. The fourth rule is derived and convenient to have.

VI. Information Security Example

The example confidentiality policy we describe is a simplified form of the Bell-LaPadula policy [2], [3]. The policy is based on the notion of controlling information flow based on classification level. For example, information classified as *secret* should only be *read* by principals whose security clearance is at least at the *secret* level. This restriction, known as the *simple security condition*, prevents leaking of information by preventing information at one classification level being read by people and processes at lower levels.

Another way information is leaked is when principals at one classification level *write* files at lower classification levels. For example, principals at the *secret* level are only allowed to write files at the *secret* level or *higher*. This is known as the **-property*. We describe both these restrictions below and formalize them in the logic.

Simple Security Condition: P can read O if and only if:

1) P's security level is at least as high as O's (i.e., $\mathsf{slev}(O) \leq_s \mathsf{slev}(P)$), and
2) P has discretionary read access to O (i.e., $P \ \mathsf{controls} \ \langle read, O \rangle$).

This policy reflects the thinking that people or processes should have read access to information at their security level or below, provided they need to know. The first condition, $\mathsf{slev}(O) \leq_s \mathsf{slev}(P)$ reflects the condition that P can read O provided it is at P's security level or below. The second

1.	$\text{S} \leq_s \text{TS}$	Ordering on labels
2.	$\mathsf{slev}(\mathit{Alice}) =_s \text{TS}$	Alice's clearance
3.	$\mathsf{slev}(\mathit{foo}) =_s \text{S}$	foo's classification
4.	$(\ \mathsf{slev}(\mathit{foo}) \leq_s\ \mathsf{slev}(\mathit{Alice})) \supset$ $(\mathit{Alice}\ \mathsf{controls}\ \langle \mathit{read}, \mathit{foo}\rangle)$	Simple security condition
5.	$\mathit{Alice}\ \mathsf{says}\ \langle \mathit{read}, \mathit{foo}\rangle$	Alice's request
6.	$\mathsf{slev}(\mathit{foo}) \leq_s\ \mathsf{slev}(\mathit{Alice})$	2, 3, 1 $sl \leq_s$
7.	$\mathit{Alice}\ \mathsf{controls}\ \langle \mathit{read}, \mathit{foo}\rangle)$	6, 4 Modus Ponens
8.	$\langle \mathit{read}, \mathit{foo}\rangle$	7, 5 Controls

Fig. 8: Security proof

condition—that P has discretionary read access to O—reflects P's need to know (i.e., some controlling authority has granted P read access to O, provided P has sufficient clearance). We express this policy as:

$$(\ \mathsf{slev}(O) \leq_s\ \mathsf{slev}(P)) \supset (P\ \mathsf{controls}\ \langle \mathit{read}, O\rangle).$$

**-Property:* P can *write* file O if and only if:

1) O's security level is at least as high as P's (i.e., $\mathsf{slev}(P) \leq_s\ \mathsf{slev}(O)$), and
2) P has discretionary write access to O (i.e., $P\ \mathsf{controls}\ \langle \mathit{write}, O\rangle$).

Restricting write access to files at or above a principal's clearance level means that information flow that occurs by virtue of writing files can only occur at the same level or go upwards. We express the *-property as follows:

$$(\ \mathsf{slev}(P) \leq_s\ \mathsf{slev}(O)) \supset (P\ \mathsf{controls}\ \langle \mathit{write}, O\rangle).$$

Example: Suppose Alice's security level is TS, file *foo*'s level is S, and the ordering on security labels is $\text{U} \leq_s \text{C} \leq_s \text{S} \leq_s \text{TS}$. Under the Bell-LaPadula policy, if Alice has discretionary access to *foo* then she should be able to read *foo*, which we denote by $\langle \mathit{read}, \mathit{foo}\rangle$. The desired behavior is given by the following derived inference rule or theorem.

$$\frac{\begin{array}{c}\text{S} \leq_s \text{TS} \quad \mathsf{slev}(\mathit{Alice}) =_s \text{TS} \quad \mathsf{slev}(\mathit{foo}) =_s \text{S} \\ (\ \mathsf{slev}(\mathit{foo}) \leq_s\ \mathsf{slev}(\mathit{Alice})) \supset (\mathit{Alice}\ \mathsf{controls}\ \langle \mathit{read}, \mathit{foo}\rangle) \\ \mathit{Alice}\ \mathsf{says}\ \langle \mathit{read}, \mathit{foo}\rangle\end{array}}{\langle \mathit{read}, \mathit{foo}\rangle.}$$

The soundness of the above behavior is proved in Figure 8.

VII. Conclusion

The logic we described here captures access-control concepts such as authorization, delegation, jurisdiction, trust, and partial orderings on confidentiality, integrity, and availability labels. Our examples show that the logic is straightforward to apply at a variety of abstraction levels spanning the lowest level of hardware (e.g., control of physical memory) to the abstract level of security policies (e.g., Bell-LaPadula). This logic enables engineers in the roles of designers, verifiers, and certifiers to reason rigorously about access control, security, delegation, and trust in a straightforward fashion.

It is reasonable to ask how accessible the logic is to practitioners and students. Over the last seven years, we have taught this logic to US Air Force Reserve Officer Training Corps students from over forty US universities. Our experience is that our logic is learned and successfully applied by rising juniors and seniors in engineering and computer science who have a sophomore level of understanding of discrete mathematics and logic. Our assessment results are detailed in [5].

In addition to what is described in this paper, we have implemented this logic [9] as a conservative extension to the Higher Order Logic (HOL) theorem prover [7]. The HOL implementation provides a rigorous, automated, and easy means for the independent verification of security claims, and the re-use of designs and policies.

We are currently developing and extending a new implementation of the logic in HOL. We are using the logic in a wide variety of applications including secure hardware design and commercial banking protocols.

References

[1] Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report Technical Report MTR-2547, Vol. I, MITRE Corporation, Bedford, MA, March 1973.

[3] D. Bell and L. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997 Rev. 1, MITRE Corporation, Bedford, MA, March 1975.

[4] Matt Bishop. *Computer Security: Art and Science*. Addison Wesley Professional, 2003.

[5] Shiu-Kai Chin and Susan Older. A rigorous approach to teaching access control. In *Proceedings of the First Annual Conference on Education in Information Security*. ACM, 2006.

[6] Shiu-Kai Chin and Susan Older. Reasoning about delegation and account access in retail payment systems. In Vladimir Gorodetsky, Igor V. Kotenko, and Victor A. Skormin, editors, *MMM-ACNS*, volume 1 of *Communications in Computer and Information Science*, pages 99–114. Springer, 2007.

[7] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, New York, 1993.

[8] Thumrongsak Kosiyatrakul, Susan Older, and Shiu-Kai Chin. A modal logic for role-based access control. In Vladimir Gorodetsky, Igor V. Kotenko, and Victor A. Skormin, editors, *MMM-ACNS*, volume 3685 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2005.

[9] Thumrongsak Kosiyatrakul, Susan Older, Polar R. Humenn, and Shiu-Kai Chin. Implementing a calculus for distributed access control in higher order logic and hol. In Vladimir Gorodetsky, Leonard J. Popyack, and Victor A. Skormin, editors, *MMM-ACNS*, volume 2776 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2003.

[10] Jerome Saltzer and Michael Schroeder. The Protection of Information in Computer Systems. *Proceedings IEEE*, 1975.