Distributed System Security via Logical Frameworks

Frank Pfenning

Carnegie Mellon University

Invited Talk

Workshop on Issues in the Theory of Security (WITS'05)

Long Beach, California, January 10-11, 2005

Acknowledgments

- Joint work with Lujo Bauer, Mike Reiter, Kaustuv Chaudhuri, Scott Garriss, Jon McCune, Jason Rouse, Kevin Watkins
- CLF: Iliano Cervesato,Dave Walker,Kevin Watkins
- LolliMon: Pablo Lopez, Jeff Polakow
- Supported by ONR N00014-04-1-0724
- Supported by NSF Cybertrust Center "*Security through Interaction Modeling*"

- `http://www.cs.cmu.edu/~self`
- *Work in progress!*

Overview

- Access Control
- Proof-Carrying Authorization
- Logical Framework (LF)
- System Architecture
- Concurrent Logical Framework (CLF)
- Operational Semantics
- Summary

Access Control

- A plethora of mechanisms
  - Physical keys
  - Id cards (with magnetic strips)
  - Smart cards
  - Biometrics
  - Username and password
  - ...
- Limited expressiveness
- Poor cross-domain interoperability

Converged Devices ("Smartphones")

- Significant computing power (500 mHz, J2ME)
- Multiple communication channels
  - Microphone, speaker, keypad
  - Camera
  - Phone calls, GPRS
  - Bluetooth
- Becoming ubiquitous
  - ∼10,000,000 shipped in 2003
  - Set to inherit (dumb) mobile phone market (∼520,000,000 shipped in 2003, ∼670,000,000 in 2004)

Towards Universal Access Control

- Smartphones as universal access control device
  - Unlock office door (prototype working in HH, CMU)
  - Log into computer (prototype working for Windows)
  - Open building? Unlock car? ...
  - Distributed information gathering!
- Challenges
  - Unify access control mechanisms
  - Flexible, yet trustworthy policies
  - Permit formal analysis
  - Small trusted computing base

Sample Scenario

- D208 is Mike’s office, door lock equipped with a bluetooth device
- Jon is Mike’s student, carrying a smartphone
- Mike is carrying a smartphone
- Mike allows his students access to his office
- Jon would like to enter Mike’s office

Proof-Carrying Authorization (PCA)

- [Appel & Felten'99] [Bauer'03]
- Express policy in authorization logic
- Prove right to access resource within logic
- Send actual proof object
- Check proof object to grant access
- First demonstration with web browser [Bauer et al.'02]

Interaction

- Jon establishes bluetooth connection to door
- Door issues challenge $\mathsf{mike}\ \mathit{says}\ \mathsf{open}(\mathsf{jon}, \mathsf{d208})$
- Jon cannot prove this
- Jon calls Mike's phone for help, providing $\mathsf{registrar}\ \mathit{signed}\ \mathsf{student}(\mathsf{jon}, \mathsf{mike})$ asking $\mathsf{mike}\ \mathit{says}\ \mathsf{open}(\mathsf{jon}, \mathsf{d208})$
- Mike's phone replies with proof of challenge
- Jon forwards proof to door
- Door verifies proof and opens

PCA Issues

- Specification of authorization logic
  - Logical framework (LF signature)
- Proof generation
  - Distributed, certifying prover or decision procedure
- Proof representation
  - Logical framework (LF object)
- Proof checking
  - Logical framework (LF type checking)

Authorization Logic as Modal Logic

- Basic judgments
  - $P\ says\ A$ — defined as a P-indexed monad
  - $A\ true$ — defined by usual rules of intuitionistic logic
- Examples
  - $\mathsf{depthead}\ says\ \mathsf{office}(\mathsf{mike}, \mathsf{d208})$
  - $\mathsf{registrar}\ says\ \mathsf{student}(\mathsf{jon}, \mathsf{mike})$

Judgmental Definition

- Truth assumptions $\Gamma = A_1\ \mathit{true}, \ldots, A_n\ \mathit{true}$
- Defining principles for $P\ \mathit{says}\ A$

$$\frac{\Gamma \vdash A\ \mathit{true}}{\Gamma \vdash P\ \mathit{says}\ A}$$

- If $\Gamma \vdash P\ \mathit{says}\ A$ and $\Gamma, A\ \mathit{true} \vdash P\ \mathit{says}\ C$ then $\Gamma \vdash P\ \mathit{says}\ C$

Internalize Modality

- P says A — proposition "P says A"
- Introduction

$$\frac{\Gamma \vdash P \; says \; A}{\Gamma \vdash (P \text{ says } A) \; true} \; says I$$

- Elimination

$$\frac{\Gamma \vdash (P \text{ says } A) \; true \quad \Gamma, A \; true \vdash P \; says \; C}{\Gamma \vdash P \; says \; C} \; says E$$

- Interplay between judgments of propositions critical for *reasoning about* authorization logic

Example

- Mike gives his students access to his office

$$\textsf{mike}\ says$$
$$\forall O.\ \forall S.\ (\textsf{depthead says office}(\textsf{mike}, O))$$
$$\supset (\textsf{registrar says student}(S, \textsf{mike}))$$
$$\supset (\textsf{mike says open}(S, O))$$

Rule Specification

- Use LF Logical Framework [Harper et al.'93]
  - Meta-language representing deductive systems
  - Judgments as types
  - Proofs as objects
  - Proof checking as type checking
  - Tested in the battlefield (PCC, FPCC, FTAL, PCA)
- Minimalistic
  - Types $A ::= a\, M_1\, \ldots\, M_n \mid A_1 \to A_2 \mid \Pi x{:}A_1.\, A_2$
  - Atomic Objects $R ::= c \mid x \mid R\, N$
  - Normal Objects $N ::= \lambda x.N \mid R$

Rule Examples in LF

```
princ : type.
prop : type.

saysj : princ -> prop -> type.
true : prop -> type.

st : true A -> saysj P A.
says_i : saysj P A -> true (says P A).
says_e : true (says P A) ->
           (true A -> saysj P C) -> saysj P C.
```

Signed Statements

- Basic judgment $P\ signed\ A$ without rules
- Represented as X.509 certificate
- Include in proofs

$$\frac{P\ signed\ A}{\Gamma \vdash P\ says\ A}\ X.509$$

Proof Search

- Usually, logically shallow (decidable)
- Prover produces proof object
- Distributed information gathering, abduction
- Caching

Derived Rules

- Inference rules as constructors for proof terms
- Definitions for derived rules of inferences

```
idem : saysj P (says P A) -> saysj P A
     = [u] says_e (says_i u)
          [u1] says_e u1 [u2] st u2.
```

$$\dfrac{\dfrac{P \; says \; (P \text{ says } A)}{(P \text{ says } P \text{ says } A) \; true} \qquad \dfrac{\ldots \quad \dfrac{\dfrac{}{A \; true \vdash A \; true}}{A \; true \vdash P \; says \; A}}{(P \text{ says } A) \; true \vdash P \; says \; A}}{P \; says \; A}$$

Proof Representation

- Proofs refer to derived rules `idem`
- Proofs refer to signed certificates `(x509 _)`
- Example

```
ex3 : saysj mike (open jon d208)
  = idem (says_e (says_i (x509 x3)) [u3] st
      (imp_e (imp_e (all_e (all_e u3 d208) jon)
        (says_i ex1)) (says_i ex2))).
```

Proof Checking

- Receive proof, including X.509 certificates
- Validate certificates (including expiration)
- Check resulting LF proof object by LF type checking
- Inherent extensibility
    - Any proposition can be signed
    - Definitions at the LF level

PCA Summary

- Formalize authorization logic in LF
- Express policy in authorization logic
- Sample interaction
  - Resource challenges with proposition
  - Client constructs proof in LF by distributed certifying theorem proving
  - Resource checks LF proof by type-checking
- Flexible, extensible
- Small trusted computing base

Current Status and Plans

- Reasoning about policies
  - Closed-world assumption
  - Use meta-logical framework Twelf [Schürmann et al.’99]
  - Basic tool: cut elimination theorem for authorization logic
  - Need deeper logical properties (focusing)
- Implementation still uses higher-order logic in LF
  - Easier to extend?
  - Impossible to reason about
- Richer distributed theorem proving

Interaction Scenario Revisited

- Jon establishes bluetooth connection to door
- Door issues challenge $\mathsf{mike}\ \mathit{says}\ \mathsf{open}(\mathsf{jon}, \mathsf{d208})$
- Jon cannot prove this
- Jon calls Mike's phone for help, providing $\mathsf{registrar}\ \mathit{signed}\ \mathsf{student}(\mathsf{jon}, \mathsf{mike})$ asking $\mathsf{mike}\ \mathit{says}\ \mathsf{open}(\mathsf{jon}, \mathsf{d208})$
- Mike's phone replies with proof of challenge
- Jon forwards proof to door
- Door verifies proof and opens

System Architecture

- Several interaction protocols
  - Jon–Door, Jon–Mike, Mike–Computer, ...
- Multiple communication channels
  - Bluetooth
  - Camera (read bar code)
  - Screen and keypad (choose resource)
  - GPRS and text messaging
- Multiple concurrent sessions
- Time stamps, certificate revocation, ...

Formal Specification

- Should formally specify architecture and protocols!
  - Good software engineering
  - Simulation
  - Reason informally
  - Model-check abstraction
  - Reason formally
- Varying levels of abstraction

Modeling Requirements

- Important for faithful simulation
  - Expressive (e.g., LF proofs, nonces)
  - Sequential (e.g., proving, proof checking)
  - Distributed (e.g., resources, theorem proving)
  - Concurrent (e.g., multiple sessions)
- Critical for reasoning
  - As high-level as possible
- Significant, but not addressed
  - Timing
  - Probabilities

The Concurrent Logical Framework

- Conservative extension of LF
- Representation principles
  - Judgments as types, proofs as objects (as for LF)
  - Concurrent computations as monadic objects
- Underlying type theory
  - $A \to B$, $\Pi x{:}A.\, B$ as for LF
  - $A \multimap B$, $A \mathbin{\&} B$, $\top$ as in linear logic
  - $\{-\}$ monad as in lax logic, functional programming
  - $A \otimes B$, 1, $!A$, $\exists x{:}A.\, B$ as in linear logic encapsulated in the monad

CLF

- Well-understood theory [Cervesato,Pfenning,Walker,Watkins'03,'04]
- Current work
  - Operational semantics [Lopez,Pfenning,Polakow,Watkins]
  - Fragment implemented in O'CAML [Polakow]
  - Theorem proving [Chaudhuri]
- Future work
  - Reasoning about specifications
  - Abstraction and model-checking

Representation Methodology

- State of the world as *linear context*
- Rules in unrestricted context (elide here)
- Linear assumptions can be consumed and added during logical reasoning
- For example, a state transition r consuming a and b while adding c and d, is represented by

$$r : a \otimes b \multimap \{c \otimes d\}$$

- Computations as proofs (omit in this talk)
- Computation as proof search

Role of Monad

- Monad ensures that *proofs* take the structure of a *concurrent computation*
- Without the monad
  - Unclear how to obtain a compositional bijection between proofs and computation (too many proofs)
  - Unclear how to endow (all of) linear logic with an operational semantics adequate for simulation

The Concurrency Monad

- Judgment $A\ lax$, derived with

$$\frac{\Gamma \vdash A\ true}{\Gamma \vdash A\ lax}$$

- Substitution principle

  If $\Delta_1 \vdash A\ lax$ and $\Delta_2, A\ true \vdash C\ lax$ then $\Delta_1, \Delta_2 \vdash C\ lax$

- Corresponds to composing two computations:
  - First from Δ_1 to obtain A
  - Second from the new state Δ_2, A to C
  - Results in computation from Δ_1, Δ_2 to C

Monadic Type Constructor

- Type $\{A\}$ — computation returning an A

$$\frac{\Delta \vdash A\ lax}{\Delta \vdash \{A\}\ true}\{\,\}I \qquad \frac{\Delta_1 \vdash \{A\}\ true \quad \Delta_2, A\ true \vdash C\ lax}{\Delta_1, \Delta_2 \vdash C\ lax}\{\,\}E$$

- $\{\,\}I$ initiates computation
- $\{\,\}E$ corresponds to one step
- Can take a step only if we are in concurrent computation

Operational Semantics

- Logic programming: *computation as proof search*
- Novel combination of forward and backward reasoning
  - Backchaining search outside monad (Prolog)
  - Forward chaining don't-care non-determinism inside monad
- Shown here only by example

Starting a Computation

- Clause $A \circ\!\!- B$ — to solve A solve subgoal B
- Goal $A \multimap \{B\}$
  - Add A to state
  - Start computation
  - Solve B when no further steps are possible (quiescence)
- Example:

$$\textsf{simulate} \circ\!\!- (\textsf{listen jon} \multimap \{\textsf{done}\})$$

Broadcast

- $!A$ — A is unrestricted
- In words:

  d208 continuously broadcasts that it is a door
- In symbols:

$$!\mathsf{broadcast}\ \mathsf{d208}\ \mathsf{door}$$

Creating Nonces

- In words:

  If principal P is listening
  and principal Q broadcasts that it is a door
  then create a fresh session identifier s
  and P sends a hello message to the door
  and awaits the challenge from Q with nonce s

- In symbols:

  $$\mathsf{listen}\ P \otimes\ !\mathsf{broadcast}\ Q\ \mathsf{door}$$

  $$\multimap \{\exists s.\ \mathsf{send}\ P\ Q\ \mathsf{hello}\ s \otimes \mathsf{receive\_challenge}\ P\ Q\ s\}$$

- After transition, P no longer listens for broadcast

Integrating Sequential Computation

- Given a clause $A \otimes B \multimap \{C\}$, we first solve A, then B as subgoals before taking a forward step.
- Mostly, A and B are atomic, but can involve arbitrary (Prolog-like) computation!
- Example:

$$\begin{array}{l} \mathsf{receive\_challenge}\ P\ Q\ Sid \\ \otimes\ \mathsf{send}\ Q\ P\ (\mathsf{challenge}\ J)\ Sid \\ \otimes\ \mathsf{find\_proof}\ D\ J \\ \multimap \{\mathsf{send}\ P\ Q\ (\mathsf{proof}\ D\ J)\ Sid \otimes \mathsf{finish\_session}\ P\ Sid\} \end{array}$$

Running Sessions Concurrently

- Computation in the monad is don't-care non-deterministic
- Proof terms representing computations differing in the order of independent steps are identified (true concurrency)
- Example: one session

  $\mathsf{simulate} \multimapinv (\mathsf{listen\ jon} \multimap \{\mathsf{done}\})$
- Example: two concurrent sessions, interleaved

  $\mathsf{simulate2} \multimapinv (\mathsf{listen\ jon} \multimap \mathsf{listen\ mike} \multimap \{\mathsf{done} \otimes \mathsf{done}\})$

Summary of Operational Semantics

- Novel combination of forward and backward proof search
- Outside monad $\Delta \vdash A\ true$
  - Backward chaining search (Prolog, λProlog, Twelf)
- Transition to concurrent computation

$$\frac{\Delta \vdash A\ lax}{\Delta \vdash \{A\}\ true}$$

- Inside monad $\Delta \vdash A\ lax$
  - Don't-care non-deterministic forward chaining

Quiescence

- Goal $\Delta \vdash C\ lax$
- Non-deterministically select clause with monadic head, e.g., $A \multimap \{B\}$
- Solve subgoal $\Delta \vdash A\ true$ (usually atom or $\otimes$)
- Commit, if successful, consuming some resources, leaving Δ'
- Continue with $\Delta' \vdash C\ lax$
- Try other clause if $\Delta \vdash A\ true$ not provable
- Transition to goal $\Delta \vdash C\ true$ is no clause applies

Saturation

- Unrestricted assumptions cannot be consumed
- Inside monad
  - $A \multimap \{!B\}$ adds unrestricted assumption B if new
  - Saturate if no clauses that apply would add a new assumption
- Useful for specifying decision procedures and theorem proving at very high level of abstraction

Current Work

- Prototype implementation (LolliMon) [Polakow]
  - No proof terms, only partial dependencies
  - Adds affine resources, choice $\oplus$ and 0
  - Adds polymorphism, output, some arithmetic
- Executable specification of architecture
  - No principal obstacle to complete model
  - Currently partial specification

Future Work

- Theory
  - Full definition of operational semantics
  - Properties of operational semantics
- Implementation
  - Improve robustness and efficiency
  - Add proof terms
  - Support richer constraints
- Architecture specification
  - Distributed theorem proving
  - Multiple levels of abstraction

Project Summary

- Distributed system security via logical frameworks
- Towards universal access control
- Smartphones as enabling hardware
- Proof-carrying authorization / LF
- Formal system specification / CLF

Some Future Work

- Deployment in new building ($\sim$70 doors)
- Policy engineering, user interfaces
- Phone upgrades, multiple usage patterns
- Reasoning about policies in authorization logic
- Verifying architecture properties
  - Model-checking abstractions of CLF specification
  - Full meta-theorem proving
- Probabilistic reasoning and timing constraints