Distributed System Security via Logical Frameworks

Lujo Bauer, Frank Pfenning, and Michael K. Reiter
Carnegie Mellon University

August 2005

Abstract

We describe a project to advance security in distributed systems via the application of logical frameworks. At the heart of the effort lies an *authorization logic* which plays a triple role: (1) to specify an access-control policy as a logical theory, (2) to enforce the policy by mechanically verifying proofs in the logic, and (3) to reason about the policy by characterizing the space of all possible proofs. We are deploying a security infrastructure based on these ideas using mobile phones as a universal access-control device at Carnegie Mellon University.

ACM subject classifiers: C.2.0 General—*Security and protection*; D.4.6 Security and Protection—*Access controls*; F.4.1 Mathematical Logic—*Computational Logic*; K.6.5 Security and Protection—*Authentication*

Keywords: Security, logical frameworks, authorization, access control.

1 Introduction

Our goal is to advance security in distributed systems via the application of logical frameworks. Our research targets multiple facets of the life-cycle of a distributed system, ranging from design through execution, and from sound mechanism design through sound policy enforcement. It consists of three major interconnected thrusts.

First, we use logical frameworks for encoding and enforcing access-control policies in a practical distributed system. Access-control mechanisms today, whether it be physical keys for doors or password protection for computer accounts, reflect access-control policies that are explicit only in the manual procedures of the organization that manages these resources. As such, any change in policy, e.g., creating a new computer account, or permitting a person to unlock a door, is effected through a manual process. We utilize logical frameworks to encode organizational policies within computer systems, thereby harnessing the power of these frameworks to support the management and enforcement of access-control policy, and gaining security and flexibility by doing so. We have demonstrated this capability in a ubiquitous computing test-bed that we are developing at Carnegie Mellon called Grey [6]. In this test-bed we use "smart" mobile phones as a universal access control device, for example, to open an office door or logging into a machine when approaching it.

Second, we exploit existing technologies to mechanically reason about security policies as specified in a logical framework. This closes an important security gap, helping users and managers understand the consequences of their policies. We are particularly interested in verifying non-interference properties of policies which guarantee that some principals' assertions can have no bearing on access to a a certain resource.

Third, we are developing and implementing a framework for the specification of distributed and concurrent systems and their implementations, specifically targeting the architecture outlined in the remainder of this paper. This work extends a collaboration between NRL

and Carnegie Mellon that resulted in the design of CLF, an innovative logical language for the specification of concurrent systems. CLF incorporates ideas from logical frameworks, linear logic, and monads into an expressive meta-language.[1] CLF is now fully specified and has been successfully validated on mainstream concurrency formalisms (e.g., Petri nets, the π-calculus), advanced concurrent programming languages (Concurrent ML), and security protocol specification languages (MSR). The goal of our current research is to facilitate the transition of CLF from a foundational language into an implemented tool that can be applied to the specification of complex distributed and concurrent systems. The current prototype is called LolliMon [21].

2 A Logic-Based Approach

Distributed systems are notoriously error-prone in virtually all phases of their life-cycle, including design, implementation and management. They are particularly susceptible to security breaches; since distributed systems are more complex than their centralized counterparts, modes of attack may be difficult to foresee, and defenses are often subtle and fragile. As such, distributed system security is an area that demands rigor, and there is a well-documented history of security failures resulting from informal design, implementation and management.

The last several years have witnessed substantial progress in verification methodologies based on formal logic. Nevertheless, there remain significant gaps between verifying specifications on the one hand, and translating these specifications into verified implementations and policy enforcement on the other. We have made progress toward closing this gap, through three related but complementary research thrusts. First, we are developing and deploying a system for day-to-day use in which access-control policy is expressed in and enforced via a logical framework, permitting us to utilize the sound footing of the framework to reason about the correctness of the access-control decisions it renders. Second, we are applying tools based on LF to reasoning about formally specified security policies. Third, we are developing an extended logical framework in which we can in addition formalize our distributed enforcement mechanism.

2.1 Logical Frameworks

A *logical framework* is a meta-language for specification and implementation of logical systems. Logical frameworks have a rich history and numerous applications in programming languages, logic, and automated reasoning [29, 27]. The particular logical framework most relevant to this application is LF [18] and its implementation in Twelf [31]. One of its central characteristics is that *formal proofs* with respect to a specified set of inference rules are *first-class objects*, and that checking if a given proof is well-formed is efficiently decidable.

The use of a logical framework in such applications as proof-carrying code [26, 2] or proof-carrying authorization [5] can be sketched broadly as followed.

1. Principal A would like to convince principal B of a certain claim C, such as his right to access a resource.

2. Principal B has announced what he is willing to accept as evidence of such a claim by publishing rules of proof for establishing claims. These rules of proof embody a certain *security policy*.

[1]This prior work was supported by ONR Grants N00014-01-1-0432 and N00173-00-C-2086 – *Efficient Logics for Reasoning about Security Protocols and Their Implementations.*

3. Principal A constructs a formal proof of C, represents this proof as an object in LF, and transmits it to B.

4. Principal B checks the proof of claim C with respect to his rules (and therefore with respect to his security policy). If the proof is correct, he accepts the claim C, otherwise he rejects it.

We will explain the particular variation of this general scenario adopted for our work in the next section, but it should already be clear that it is critical that *formal proofs* are *objects*.

This methodology is well established and there are several practically efficient implementations. However, how do we know that a given set of proof rules correctly implements an intended security policy? One of the central items of our work has been to exploit the recently developed meta-reasoning facilities of the Twelf implementation [32, 33] in order to formally reason about the policies that are specified and enforced by the access-control architecture sketched in the next section. Successes in reasoning about standard classical [28] and modal logics [25] provide some hints, but access-control logics are more complex along certain dimensions and remain a significant challenge for automated tools. Initial results in this direction have been reported in [30].

Another use for logical frameworks is explained in Section 2.3.

2.2 Implementation of Access Control via Logical Frameworks

Distributed authorization systems (e.g., [9, 15, 19]) provide a way to implement and use complex security policies that are distributed across multiple hosts. The methods for distributing and assembling pieces of these security policies can be described using formal logics [20, 17]—such formalization can dramatically increase confidence in the systems' correctness. Distributed authorization systems have been built by first designing an appropriate logic and then implementing the system around it [1, 4].

Most distributed authorization systems try to provide support for notions such as the ability to delegate privileges and aggregate principals into groups. While these systems strive to be as expressive as possible—that is, to be able to represent as many security policies as possible—they are constrained by how they choose to implement these ideas. The SPKI/SDSI [15] notion of delegation, for example, includes a Boolean flag that describes whether the delegated privilege may be redelegated by the recipient. PolicyMaker [9], on the other hand, requires any redelegation to be explicitly approved by the security policy. Each choice may be the best one for a particular situation, but no one particular choice can be the ideal one for *all* situations. The necessity of making these design choices limits the generality and expressivity of each such system. A system may be well suited for a particular environment but cannot scale to all plausible distributed-authorization scenarios, and systems developed for use in different environments may not be able to interoperate.

Proof-carrying authorization (PCA), a particularly promising, recent approach to distributed authorization, follows a different strategy to achieving generality [3, 8]. Unlike other systems, in which axioms that define ideas like delegation are part of the logic which describes the system, PCA is based on a standard, and completely general, higher-order logic (HOL) [13]. Higher-order logic is undecidable—there is no algorithm which will always be able to prove the truth of every true statement—which raises the question: how can such a logic can be used in an authorization system? A server is typically presented with a list of credentials and has to decide whether they are sufficient for access to be granted. If the logic that models this is undecidable, the server might not be able to come to the correct conclusion.

PCA solves this problem by making it the client's responsibility to prove that access should be granted. The server's task then becomes to verify that the client's proof is valid, which

can be done efficiently even if the proof is expressed in an undecidable logic. Transferring the burden of proof from the server to the client ensures that the server's task is tractable, but doesn't explain how the client is able to construct a proof. What makes the client's task possible is that any particular client doesn't need the full expressivity of the undecidable logic. Instead, a particular client is probably content to construct proofs in some decidable subset of higher-order logic—an application-specific subset that corresponds to a particular notion of delegation, a particular way of defining groups or roles, etc. This application-specific subset can be exposed to a client as a regular authorization logic, for example, a logic that models SPKI/SDSI. The client can manipulate this logic and construct proofs without knowledge of the underlying, more general (and more confusing) framework. The server verifying the proof, on the other hand, doesn't care which particular application-specific logic the client uses. As long as the application-specific logic is presented as a subset of higher-order logic, the server sees the client's proof as just another higher-order logic proof, which it knows how to verify.

This approach gives PCA great flexibility. Unlike traditional distributed authorization systems, a PCA-based system can be customized to describe many different sets of authorization scenarios. At the same time, because the different scenarios are expressed in the same underlying framework, all of these components can be made to inter-operate.

Logical frameworks, such as LF and CLF, are ideal tools for describing and reasoning about security logics including PCA and the higher-order logic on which it is based, and thus for providing assurance that certain security properties are achieved. However, inconsistencies between the model of an authorization system and its implementation can negate some of the conclusions of such reasoning. It is thus worthwhile to consider integrating logical frameworks directly into the implementation of a system, i.e., so that the implementation of a distributed system explicitly uses the data structures and proof-generation and proof-checking techniques of logical frameworks. An earlier proof-of-concept experiment [5, 8] suggested this to be feasible, and we are in the process of extending the use of logical frameworks in the implementation of access-control mechanisms in real systems.

Technology has evolved to the point where it is no longer necessary for access control in the physical world and access control on computers to be separate domains. The development and proliferation of high-powered and relatively inexpensive mobile computing devices (PDAs and "smart" mobile phones) and abundance of local communications options (Bluetooth and WiFi) have extended the reach of computers into everyday life. At the same time, recent advances in logic-based access-control suggest that it is possible to build practical access-control systems with nearly unlimited flexibility. The convergence of these developments makes it possible to introduce powerful new paradigms in which mobile devices take the place of both physical keys and computer passwords, eliminating the need for many separate access-control systems and introducing into the world of physical access-control drastic advances in flexibility, convenience, and security.

In particular, we envision an environment in which a Bluetooth-enabled, "smart" mobile phone will be the sole access-control token that a person will need to carry, replacing keys, smart cards, and passwords. The mobile phone will enable its bearer to enter his car, unlock his office door, and log onto his computer. To make this possible the mobile phone will generate PCA proofs that demonstrate that the bearer of the phone is authorized to access his office, computer, etc. The office door and computer logon program will contain a proof checker that will verify proofs prior to allowing access. The policy itself, which must be reflected in the proof of access, may remain distributed until the moment of proof construction, with each piece housed on a different device or provided by a different entity. In addition to the convenience of having to carry only one device, such a system greatly increases security by permitting the use of flexible, distributed, and precise policies. In practice, for example, the

policy authorizing an employee access to his office often culminates in handing to the employee a key, which he can then use as he sees fit; in our system, on the other hand, the policy could be evaluated at the time of access, perhaps permitting accesses that a physical key could not and denying others, in response to pieces of the policy that have been dynamically changed.

Such uses of mobile devices will vastly increase the importance of preventing their misuse, as is particularly of concern if the device is physically captured. Ultimately the utility of a PCA proof rests on the protection of cryptographic keys from unintended disclosure, and those stored on a mobile device are especially vulnerable to an attacker who physically reverse-engineers this device. An aspect of the system currently under development are "capture protection" services to render devices far less susceptible to misuse if captured, by utilizing a remote "capture protection server" to confirm that a device remains in its proper owner's possession before permitting its cryptographic key to be used. This can be accomplished without disclosing the device's key to the servers; while permitting the device to move the capture protection server it is utilizing as convenient; and in the face of arbitrary efforts to obtain the key from the device by reverse-engineering [23, 22].

We have deployed this access-control system for our own daily use, with a limited set of resources (e.g., our own office doors, computer logins) and users. As the system matures, we intend to broaden its use to a larger user base and a range of resources and activities on the CMU campus (e.g., purchases). Deploying such a system for everyday use involved solving or making progress on a range of issues that had not been fully addressed in such applications before, some specific to our logic-based approach and others merely exacerbated by it:

- Utilizing such a general approach for access control requires that the unintended consequences of this generality can be constrained. For example, in such a logic-based approach, authority is proved in a formal framework to which a variety of parties (users, computers) contribute "statements" (credentials). This raises the question of what unintended consequences result from participants who behave unexpectedly, e.g., by uttering contradictory statements. This issue is fundamental to the viability of this approach, but remained largely unexplored until recently [30], as sketched in Section 2.1.

- Logic-based approaches have previously been demonstrated in narrow contexts, where proof-generation strategies were neatly laid out with human assistance. This does not scale, and we further conjecture that different proof strategies may be appropriate for different environments. We have developed an approach in which the task of generating a proof is automatically distributed among the set of entities best qualified to construct it. Additionally, we are designing mechanisms that permit the proof-generation infrastructure to learn from experience, i.e., in which proof generation strategies are discovered and refined automatically, over time, and made available for use by other devices as needed. Initial results are reported in [7].

- Previous work on implementing distributed authorization systems has typically been in the context of networks of well-connected machines with plentiful computational abilities. A practical system such as the one we are deploying must explicitly account for the limitations of severely resource-constrained devices (e.g., smartphones) that communicate via a range of protocols with highly variable capacity, latency, and availability (e.g., GPRS, SMS). We have developed several strategies for coping with these difficulties, including designing a modular and lightweight communications framework well suited to such a heterogeneous environment, and intuitive and streamlined user interfaces that facilitate interaction between users and smartphones. We report on the design and describe the initial implementation of our system in [6].

- Keeping authorization credentials within a device obviously raises concerns surrounding the device's capture. We are experimenting with the aforementioned techniques for "capture resilience" in our setting, i.e., techniques that render devices largely invulnerable to capture and misuse, despite their not being physically tamper-resistant [23, 22].

2.3 Distributed Architecture Specification with a Framework Extension

With the techniques sketched in the previous section we can formally specify security policies, reason about their correctness, and enforce them in a distributed implementation. However, we cannot reason about the distributed implementation itself.

We are currently taking the first critical step towards formally reasoning about the distributed implementation by designing and implementing an extension of the logic framework LF that directly supports the specification of distributed and concurrent systems. This builds on prior research on the Concurrent Logical Framework (CLF) [34, 10]. CLF allows specifying a distributed system at a high level of abstraction. It is based on a novel combination of linear logic [16], type theory [14], and monads [24]. Our current work follows two related threads in pursuing this goal:

- We have resolved some implementation challenges of CLF, resulting in a prototype called LolliMon [21], but more remain. Specifically, the efficiency of concurrent simulation does not yet allow larger-scale experiments. Nonetheless, LolliMon has been very useful for describing implementations in CLF, as the resulting programs are too large for visual inspection, while type-checking and simulation would provide at least partial assurance.

- While the LolliMon implementation allows simulation of a distributed system, it is not suited to testing reachability. This requires a theorem prover or, in some fragments, a model checker. The initial design and implementation of a theorem prover for linear logic is described in [11, 12]; current research is aimed at making it more easily applicable to CLF.

We are in the process of formally specifying the evolving PCA architecture described in the previous section in CLF. A future direction of research would be to also formalize the meta-reasoning about the architecture itself (and not just specific policies as proposed in Section 2.1). The above items of current research are promising pre-requisites for this planned future work.

3 Conclusion

We have described a distributed security architecture based on a logical framework. The logical framework plays several roles: it serves to specify the security policy as a logical theory in an authorization logic, enforce it via checking of formal proofs, and reason about it by proof-theoretic analysis. We have realized a prototype for our framework at Carnegie Mellon University, exploiting the computational power and communication capabilities of "smart" cell phones for flexible distributed access control. Our experience so far has been positive: while the logical machinery provides a sound, uniform, and inherently extensible foundation, typical situations do not require to understand this underlying machinery for day-to-day tasks.

Acknowledgments. We gratefully acknowledge contributions by Kevin Bowers, Kaustuv Chaudhuri, Deepak Garg, Scott Garriss, Jonathan M. McCune, Jason Rouse, Peter Rutenbar, and Kevin Watkins to both the theory and implementation of the system.

This ongoing research is supported by the Office of Naval Research under grant N00014-04-1-0724: *Distributed System Security via Logical Frameworks*, the National Science Foundation grant number CNS-0433540, and the U.S. Army Research Office contract number DAAD19-02-1-0389.

References

[1] M. Abadi, E. Wobber, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pages 256–269. ACM Press, Dec. 1993.

[2] A. Appel. Foundational proof-carrying code. In J. Halpern, editor, *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*, pages 247–256. IEEE Computer Society Press, June 2001. Invited Talk.

[3] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, Singapore, Nov. 1999.

[4] D. Balfanz, D. Dean, and M. Spreitzer. A security infrastructure for distributed Java applications. In *21th IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May 2000.

[5] L. Bauer. *Access Control for the Web via Proof-carrying Authorization*. PhD thesis, Princeton University, Nov. 2003.

[6] L. Bauer, S. Garriss, J. M. McCune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of the 8th Information Security Conference (ISC'05)*, Sept. 2005. An extended version of this paper appears as CMU Computer Science Department Tech Report 05-111.

[7] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security & Privacy*, May 2005.

[8] L. Bauer, M. A. Schneider, and E. W. Felten. A general and flexible access-control system for the web. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, Aug. 2002.

[9] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust-management system. In *Proceedings of the 2nd Financial Crypto Conference*, volume 1465 of *Lecture Notes in Computer Science*, Berlin, 1998. Springer.

[10] I. Cervesato, F. Pfenning, D. Walker, and K. Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

[11] K. Chaudhuri and F. Pfenning. A focusing inverse method prover for first-order linear logic. In R.Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, pages 69–83, Tallinn, Estonia, July 2005. Springer Verlag LNCS 3632.

[12] K. Chaudhuri and F. Pfenning. Focusing the inverse method for linear logic. In L.Ong, editor, *Proceedings of the 14th Annual Conference on Computer Science Logic (CSL'05)*, pages 200–215, Oxford, England, Aug. 2005. Springer Verlag LNCS 3634.

[13] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[14] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[15] C. M. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. M. Thomas, and T. Ylonen. *SPKI Certificate Theory*, Sept. 1999. RFC2693.

[16] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[17] J. Y. Halpern and R. van der Meyden. A logic for SDSI's linked local name spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 111–122, Mordano, Italy, June 1999.

[18] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, Jan. 1993.

[19] R. Housley, W. Polk, W. Ford, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, Apr. 2002. RFC3280.

[20] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Trans. Comp. Sys.*, 10(4):265–310, Nov. 1992.

[21] P. López, F. Pfenning, J. Polakow, and K. Watkins. Monadic concurrent linear logic programming. In A.Felty, editor, *Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming (PPDP'05)*, pages 35–46, Lisbon, Portugal, July 2005. ACM Press.

[22] P. MacKenzie and M. K. Reiter. Delegation of cryptographic servers for capture-resilient devices. *Distributed Computing*, 16(4):307–327, December 2003.

[23] P. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. *International Journal of Information Security*, 2(1):1–20, November 2003.

[24] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[25] T. Murphy VII, K. Crary, R. Harper, and F. Pfenning. A symmetric modal lambda calculus for distributed computing. Technical Report CMU-CS-04-105, Carnegie Mellon University, Feb. 2004.

[26] G. C. Necula. Proof-carrying code. In N. D. Jones, editor, *Conference Record of the 24th Symposium on Principles of Programming Languages (POPL'97)*, pages 106–119, Paris, France, Jan. 1997. ACM Press.

[27] F. Pfenning. The practice of logical frameworks. In H. Kirchner, editor, *Proceedings of the Colloquium on Trees in Algebra and Programming*, pages 119–134, Linköping, Sweden, Apr. 1996. Springer-Verlag LNCS 1059. Invited talk.

[28] F. Pfenning. Structural cut elimination I. intuitionistic and classical logic. *Information and Computation*, 157(1/2):84–141, Mar. 2000.

[29] F. Pfenning. Logical frameworks. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 17, pages 1063–1147. Elsevier Science and MIT Press, 2001.

[30] F. Pfenning. Constructive authorization logics. 4th Workshop on Foundations of Computer Security (FCS'05), Chicago, Illinois, July 2005. Invited Talk.

[31] F. Pfenning and C. Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[32] C. Schürmann. *Automating the Meta Theory of Deductive Systems*. PhD thesis, Department of Computer Science, Carnegie Mellon University, Aug. 2000. Available as Technical Report CMU-CS-00-146.

[33] C. Schürmann and F. Pfenning. A coverage checking algorithm for LF. In D. Basin and B. Wolff, editors, *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2003)*, pages 120–135, Rome, Italy, Sept. 2003. Springer-Verlag LNCS 2758.

[34] K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.