Secure Systems
Editors: Patrick McDaniel, mcdaniel@cse.psu.edu
Sean W. Smith, sws@cs.dartmouth.edu

Logical Methods in Security and Privacy

ANUPAM DATTA
Carnegie Mellon University

Computer security and privacy is concerned with the design, implementation, and analysis of mechanisms intended to guarantee that desired policies (or properties) hold in the presence of malicious adversaries. Here, I provide a partial overview of how logical methods—in particular, methods for reasoning about system security and for enforcing security and privacy policies—contribute to this discipline.

Applying logic to computer security and privacy offers long-term benefits, including the development of a scientific foundation that informs how we think about and teach security as well as design, implement, and analyze secure systems.

Reasoning about System Security

Reasoning about system security requires methods for proving that a given security mechanism, such as a network protocol or Web browser, achieves the desired security property even in the presence of a well-defined class of adversaries.

Consider the problem of enabling secure communication over an untrusted network, such as the Internet. One security mechanism for this task is the SSL protocol, which users employ when banking online, for example. SSL uses cryptographic primitives such as *public key encryption* and *digital signatures* to provide security properties such as *authentication* and *data confidentiality and integrity*. Authentication assures a user that he or she is interacting with the bank—not some adversary masquerading as the bank—while data confidentiality and integrity implies that only the user and bank can view and modify the account information. These guarantees are expected to hold even in the presence of an active adversary who can intercept, inject, and remove messages from the network.

Designing such protocols is nontrivial, and many attacks have been identified after a protocol's deployment (as in the case of the public key Kerberos, earlier versions of SSL, and the IEEE 802.11i wireless authentication protocol suite). Such attacks often arise because the protocols execute in a *concurrent environment*. For example, a bank's Web server running SSL simultaneously serves many users. A user might also be simultaneously engaged in SSL sessions with the bank and an e-commerce website. An adversary could use information acquired in one session to compromise the security of another session.

During protocol design, it's difficult to consider all possible sequences of actions that could break the desired security property and develop a sound informal argument about the protocol's security. Logical methods can aid in these tasks.

Logic

A *logic* includes a *syntax* of well-formed formulas, which we can use to express security properties. For example, let's say only Alice and Bob know a secret key used for encryption. We could express this property formally in a protocol logic as

$$\forall p.\ \texttt{Has}(p,\texttt{secretkey}) \Rightarrow (p = \texttt{Alice} \vee p = \texttt{Bob})$$

In this example, `Alice` and `Bob` are constants of type *principal*, and p is a variable of type *principal*; `secretkey` is a constant of type *key*; `Has` is a predicate with parameters of type *principal* and *key*; $\Rightarrow$ denotes logical implication; and $\vee$ is logical disjunction. The syntax of a logic precisely defines sets of variables and constants for each type (in this case, *principal* or *key*), as well as the predicates from which larger formulas are built, by repeatedly applying operators such as logical disjunction and implication. We can define the semantics (or meaning) of well-formed formulas of a logic using an appropriate model.

For example, the model for a protocol Q could be the set $\mathcal{M}_Q$ of all possible protocol executions, starting from any initial configuration. We could then define the semantics of a protocol logic using this model to capture the intuitive idea that a protocol satisfies a property expressed in the logic if and only if the property holds for all executions in $\mathcal{M}_Q$.

Model Checking

Logical methods based on *model checking* can help identify attacks on a protocol Q automatically by exhaustively checking the model $\mathcal{M}_Q$—that is, by searching for a possible execution of Q that does not satisfy the desired property. This method applies to a protocol's *finite* configurations (such as SSL with three servers and four client sessions).

Early examples of model-checking protocols include efforts with the FDR (Failure Divergences Refinement Checker) tool[1] and the Murphi tool. State-of-the art protocol model checkers, such as those in the AVISPA (Automated Validation of Internet Security Protocols and Applications) tool set, scale even better and have been successfully applied to many industrial protocols. These tools can serve as useful aids in debugging designs of complex protocols.

Protocol Logics

Although model-checking tools can automatically identify attacks on finite protocol configurations, *protocol logics* serve the dual purpose of proving the absence of attacks on a protocol for an unbounded number of concurrent sessions. An early example of a protocol logic is BAN logic;[2] a more recent example is Protocol Composition Logic.[3]

The proof system for protocol logics codify principles that protocol designers use to informally reason about protocol security. A *proof system* for a logic consists of *axioms* and *inference rules*. We prove a *theorem* by starting from axioms and repeatedly applying inference rules.

For example, an axiom in the proof system of a protocol logic might capture the difficulty of forging signatures by stating that if Alice verifies Bob's signature on a message, and Bob is the only principal who possesses his signing key, then Bob must have produced the signature. An example of an inference rule is *modus ponens*, which states that if φ and $\varphi \Rightarrow \psi$ are provable, then so is ψ. If it's provable that Alice verifies Bob's signature on a message during a protocol execution, then we could use an instance of this inference rule, together with the signature axiom, to prove the theorem that Bob produced the signature on the message.

Typically, the proof system for a logic is *sound*, implying that every theorem about a protocol Q proved using the proof system actually holds for all executions in $\mathcal{M}_Q$. This soundness result implies that if we can construct a formal proof of protocol security, then the informal argument that the proof codifies doesn't contain the kind of fallacious reasoning that is at the heart of many errors in protocol designs.

Using soundness results to connect proof systems to protocol execution semantics is one difference between some of the early protocol logics and more recent ones. Other differences include broader coverage of protocols and properties and the development of modular reasoning principles that have enabled application of these methods to large industrial security protocol suites, such as the IEEE 802.11i suite for wireless local area networks and the IEEE 802.11s suite for wireless mesh networks.

Deployment: Limitations and Challenges

It's important to note that the security guarantees assured by these methods are not absolute; they hold only in the underlying model of protocol execution and attack. An adversary could break the security property using capabilities that the model doesn't consider.

For example, we typically assume that cryptographic primitives are perfect (for example, a signature is unforgeable with probability of 1). We also often assume that the programming language for modeling protocols doesn't include features such as pointers that an implementation in a language like C will have. Nevertheless, the results of such an analysis have value, because they demonstrate the absence of a large class of design-level attacks on protocols. Indeed, researchers are actively working on logical methods for more refined models that consider, for example, complexity-theoretic models of cryptography and features of implementation languages such as C and F#.

The study of logical methods for reasoning about security protocols has produced a significant scientific foundation for this area and resulted in methods and tools that have impacted deployed protocols. Although my earlier discussion focused on protocols, related logical methods have also been developed for reasoning about security properties of other classes of systems, including operating system kernels, trusted computing systems, and Web platforms. Several institutions have successfully used these methods and tools in teaching the process of secure protocol and system design.

Specifically, before implementing a completed protocol or system design, it's worthwhile to subject the design to a careful analysis using, for example, model-checking tools. Some of these tools (such as the Murphi model checker) are easy to use, even without any background in formal logic. This approach can help identify and correct design-level flaws. However, if the model checker doesn't find any attacks, it doesn't mean that the system is secure, because the model checker can only check small finite configurations (usually less than 10 concurrent sessions of security protocols).

It's then useful to try to prove that the protocol or system is secure using the proof system of a logic. This step typically requires more expertise in logic; graduate

students without significant prior experience can usually learn to use such logics with a reasonable amount of effort.

In addition to reasoning about system security, logical methods can also be used to design and implement mechanisms for enforcement of policies—for example, for access control and privacy.

Enforcing Security and Privacy Policies

In addition to reasoning about system security, logical methods can also be used to design and implement mechanisms for enforcement of policies—for example, for access control and privacy.

Consider the problem of protecting patient privacy in a hospital. The hospital's privacy policy, following the Health Insurance Portability and Accountability Act (HIPAA), could specify conditions under which the hospital can use a patient's personal health information (such as for diagnosis, treatment, or billing) and share it with entities outside the hospital (such as the patient or his or her insurance company). The hospital might use a combination of security mechanisms involving *access control* and *audits* to ensure that it respects this policy, even in the presence of external adversaries and insiders (hospital employees) who might try to violate the policy.

Access Control

Access control mechanisms could restrict access to a patient's record to only the set of nurses and doctors who are involved in caring for the patient (the care team). However, a care-team doctor could delegate access rights to another doctor standing in for him or her on a particular day. So, doctors could gain access to a patient's record by providing evidence to the reference monitor (which mediates access) that reveals he or she is either part of that patient's care team or was granted access rights by a doctor in that care team.

One insight that Bulter Lampson and his colleagues pointed out is that a logic could represent access control policy rules, such as the ones I've outlined.[4] A logical proof that shows why policy rules authorize access could constitute sufficient evidence for the reference monitor to grant access. Furthermore, logical inference and automated proof search could implement the policy rules directly.

There's now a substantial body of work on logic-based access control methods and their use in operating systems, Web security, physical access control, file system access control, and distributed trust management systems. Hospitals could, in principle, use similar mechanisms to restrict access to patient records.

In practice, many hospitals currently use coarse-grained access control, for example, to restrict access to patient records to hospital employees based on their roles (doctor, administrative staff, and so on) or physical location (for example, nurses working for a large hospital chain might have access to records of patients only at the location where they work). Indeed, it's important to understand what access control policies work best in a healthcare setting, given the need to balance easy access for healthcare providers and patient privacy. Once we determine the appropriate access control policies, we can enforce them using logic-based access control mechanisms (in particular, we could easily codify role-based access control).

A remaining challenge, which researchers are currently actively working on, is to make logic-based access control more usable—for example, by developing policy languages with easily understandable syntax as well as tools with user-friendly interfaces for authoring and updating policies.

Audits

Note that access control mechanisms aren't sufficient for enforcing all privacy policies, such as clauses in the HIPAA Privacy Rule that

- impose conditions on transmission of various types of personal information based on subjective beliefs (for example, letting hospital officials share protected health information with law enforcement agencies if a death is suspicious);
- restrict information use to specified purposes (such as treatment or payment); and
- associate future obligations with transmissions (for example, requiring hospitals to respond in a timely manner to patient requests pertaining to the patient's protected health information).

The current practice is to record all accesses to patient records in an audit log and rely on manual audits to identify violations and to hold individuals accountable for such violations. A significant open problem is to develop foundations for accountability and computational methods for structuring audits that reduce human involvement in the process.

Adam Barth and his colleagues' recent work provides a representative example of the use of logical methods for representing and enforcing privacy policies.[5] This effort formalizes concepts from *contextual integrity*, a philosophical account of privacy in terms of the transmission of personal information.

A central concept in this theo-

ry is that of a *context*, which captures the idea that people transact and act in society as individuals in certain capacities or *roles* in distinctive social contexts, such as healthcare, education, and banking. A communication action *transmits* some type of personal information about a *subject* in some role (such as personal health information about a patient) from a *sender* in some role (such as Bob, a doctor) to a *recipient* in some role (such as Alice, the patient). Such transmissions are governed by *norms of transmission* that prescribe or proscribe certain transmissions based on the context, roles, and type of information.

The logic of privacy that Barth and his colleagues developed is interpreted using a semantic model of interacting agents in roles that transmit personal information to each other. This model includes a record of transmission actions. The syntax of this logic can express social norms of transmission or privacy policies codified in laws. For example, a formula in the logic can represent the norm that hospitals can transmit personal health information about a patient to a third party only if the patient has consented to such disclosures. This norm would be violated in an execution that records a transmission action from a hospital to a third party of a patient's personal health information, but doesn't record a transmission action from the patient to the hospital consenting to such disclosures. In this way, the logic captures the ideas of contextual integrity discussed earlier.

Subsequent work builds on this logic to identify the logical structure of a rich class of privacy policies, formalizing all transmission-related clauses in the HIPAA privacy rule. These formalizations also serve as the starting point for automated policy enforcement based on a combination of logical methods that support access control and limited forms of auditing. These are useful steps toward the development of practical tools for enforcing privacy policies, although much work remains to effectively enforce the entire HIPAA rule.

Remaining challenges for computer-assisted audits include dealing with future obligations, determining whether information was used for a specified purpose, and addressing issues stemming from distributed audit logs (for example, in a health information exchange scenario involving a network of hospitals). In addition to enforcement, we also need to develop tools for policy analysis that could help individuals understand a complex privacy policy, such as HIPAA. The tools could help answer questions about whether a hospital can transmit a certain type of information to a patient and, if so, could produce a list of conditions under which the information could be transmitted.

Although I've used healthcare as a motivating example, these methods and remaining challenges apply to many other sectors in which privacy is a concern, including finance, Web services, education, and government.

Looking forward, logic should continue to contribute toward computer security and privacy by providing general methods for designing, implementing, and analyzing security mechanisms. Several existing methods and tools for security analysis of system designs are ready for wider adoption in industry; other methods require a certain level of background in formal logic that could be provided by ensuring that topics like propositional and first-order logic receive attention in the undergraduate computer science curriculum. These steps will help produce secure systems with properties that are better understood.

Logical methods of analysis won't guarantee absolute security, but they'll rule out large classes of attacks and make explicit the assumptions about adversary capabilities and system characteristics. Indeed, a challenge moving forward is to expand the scope of these methods to cover more detailed models of systems and adversaries.

Logical methods will also contribute toward the design and implementation of new mechanisms for enforcing privacy policies—a major societal challenge. Specifically, we should see significant progress in terms of foundations and methods for audits and accountability in the near future. □

References

1. P. Ryan et al., *Modelling and Analysis of Security Protocols*, Addison-Wesley, 2001.
2. M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *ACM Trans. Computer Systems*, vol. 8, no. 1, 1990, pp. 18–36.
3. A. Datta et al., "Protocol Composition Logic (PCL)," *Electronic Notes in Theoretical Computer Science*, vol. 172, Apr. 2007, pp. 311–358.
4. B.W. Lampson et al., "Authentication in Distributed Systems: Theory and Practice," *ACM Trans. Computer Systems*, vol. 10, no. 4, 1992, pp. 265–310.
5. A. Barth et al., "Privacy and Contextual Integrity: Framework and Applications," *IEEE Symp. Security and Privacy*, 2006, pp. 184–198.

Anupam Datta *is an assistant research professor at Carnegie Mellon University. His research interests include developing logical methods to address problems associated with security protocols, trustworthy systems, and data privacy. Datta has a PhD in computer science from Stanford University. He is a member of IEEE and the ACM. Contact him at danupam@cmu.edu.*