Programming language methods in computer security

John C. Mitchell

Department of Computer Science
Stanford University
http://www.stanford.edu/~jcm

Abstract

This invited talk will give a personal view of the field of computer security and summarize some ways that methods from the study of programming language principles can be applied to problems in computer security. Some background information is provided here in this short document.

Security and correctness

Computer security is concerned with detection and prevention of unauthorized use of computational resources. Computer security problems range from detecting potentially malicious network traffic to password systems and other access control mechanisms to mechanisms designed to prevent installed code from corrupting a computing environment.

There is some overlap between computer security and methods for ensuring software correctness. For example, web browser code that contains a Trojan (functionality to allow unauthorized access) is simply an incorrect browser implementation: the specification of a web browser does not include functionality for providing remote access to the computer on which the browser is installed. Therefore, an insecure browser could be considered an incorrect browser. For this reason, many basic security concerns can be addressed using methods designed for software assurance. At the same time, however, security properties tend to have a different flavor from other correctness properties.

One qualitative difference between security properties and other correctness properties lies in way that system input is considered. Although the following characterizations are approximate and must be taken with a grain of salt, the difference may be illustrated as follows:

- *Correctness:* A software system is *correct* if correct system input results in correct system output. To give a simple example, the specification for a function $f : A \to B$ generally says that for all inputs $x \in A$, the output $f(x) \in B$ has a certain property.

POPL '01 1/01 London, UK

- *Security:* A software system is *secure* if arbitrary input does not have undesired consequences, such as release of private information or corruption of the state of the system. To continue the function example above, the implementation of a function $f : A \to B$ is insecure if the computation of $f(y)$, for some $y \notin A$, causes overflow of some buffer allocated on the run-time stack and therefore results in some system call not related to the correct calculation of function f.

In general terms, computer security is concerned with behavior in arbitrary environments while correctness is often stated using some restrictions on the environment.

Security analysis

In general, it is only possible to prove that a system or mechanism is secure in a relative sense. More specifically, a proof of security involves some model of the behavior of the system in question and, at least as importantly, some model of the set of actions available to an attacker. This reliance on models leads to one fundamental connection between programming language methods and computer security: the kinds of programming language and system models often studied at POPL can be used to characterize the behavior of a system for the purpose of security analysis. A promising direction for POPL-style research is to characterize the actions available to an attacker within these models, and devise methods for reasoning about the possible effects of an attacker on a system.

One computer security topic that has received considerable attention in recent years is security analysis of network protocols. A number of methods have been developed, ranging from BAN logic and related approaches [BAN89, GNY90] to finite-state analysis [Ros95, MMS97] and proof methods based on higher-order logic [Pau97]. Most approaches in current use are based on enumeration or reasoning about a set of protocol traces, each trace obtained by combining protocol actions with actions of a malicious intruder.

There are several reasons why protocol analysis has attracted so much attention. One is the importance of the problem. To give one example, the Secure Sockets Layer (SSL) protocol (analyzed in [MSS98]) is used in a huge number of Internet purchases every minute. The purpose of the protocol is to establish a secret key, shared between client and server, that can be used to send a credit card number or other data under encryption. If this protocol were susceptible to practical attack, millions of Internet customers

could have their credit card numbers stolen. Another reason that protocol analysis has been popular in recent years is the relative tractability of the problem. Security protocols are typically simple distributed programs that run for three to seven communication steps and halt. Relative to other software systems, they are therefore very simple programs. Moreover, there is a standard idealized intruder model, commonly referred to as the "Dolev-Yao model," which appear to have developed from positions taken by Needham and Schroeder [NS78] and a model presented by Dolev and Yao [DY83]. In this model, the attacker can intercept messages sent on the network but cannot interfere with local protocol calculations carried out by parties to the protocol. The attacker may block network messages, decompose them into parts, remember all of the parts, decrypt parts if the key is known, and send messages composed from previous message parts to protocol participants.

More generally, there are many ways that methods from programming language analysis have been used in security protocol analysis:

- Use of process calculi and related formalisms to represent protocols in a form amenable to analysis.
- Model checking techniques to find flaws in protocols.
- Theorem proving methods to prove correctness of protocols.
- Use of concepts from logics of programs to develop specialized logics for proving protocol correctness.

In addition to protocol analysis, here are some other computer security topics can be addressed using techniques developed or used in POPL-style research:

- *Information flow and noninterference:* the study of how information may be transferred from one user (or process) to another in a multi-user system and how such transfer of information can be prevented.
- *System security flaws.* An astonishing number of computer security advisories stem from buffer-overflow errors in system programs. Such flaws are amenable to source-code static analysis methods and dynamic program-monitoring methods.
- *Mobile code security:* When code is transferred and executed dynamically, program analysis methods (such as Java bytecode verification) can be used to examine code before it is installed. Proof-carrying code [NL96] is a popular approach that has received significant attention at POPL and related conferences.

There are many additional topics represented in current security conferences such the IEEE Symposium on Security and Privacy and the IEEE Computer Security Foundations Workshop, both listed at http://www.ieee-security.org/, the ACM Conference on Computer and Communications Security, listed at http://www.acm.org/sigsac/, and the Crypto and Eurocrypt conferences organized by the International Association for Cryptologic Research, www.iacr.org.

Compositionality and observational congruence

One particular folk belief that may interest the POPL audience is the belief in the security community that security properties do not compose. A general problem with composition is that when two mechanisms are combined, one may inadvertently reveal information related to the security of the other. Here is a simplified example to illustrate the point.

- *Specification:* Any party *Alice* must be able to send any message m to any other party *Bob* in such a way that no passive eavesdropper listening on the network can determine the identity of message m.
- *Implementation:* We assume a public key infrastructure so that *Alice* knows the public encryption key KB of *Bob*, *Bob* knows *Alice*'s public key KA, and the corresponding decryption keys KB^{-1} and KA^{-1} are initially known only to *Bob* and *Alice*, respectively.

  To send message m, *Alice* computes the encryption $\{\!|m|\!\}_{KB}$ of message m with *Bob*'s public key and sends two values to *Bob*: the encrypted message $\{\!|m|\!\}_{KB}$ *and* *Alice*'s private decryption key KA^{-1}.

Assuming that a good encryption function is used, the implementation above meets its specification. A passive eavesdropper will obtain two values from the network: the encryption of m and a private decryption key not related to the encryption of m. Since the private decryption key is not related to the encryption of m, the eavesdropper cannot learn the message m.

Consider what happens if we compose the secure protocol above with the same protocol used in reverse to send a message from Bob to Alice. Using the notation commonly found in the literature, here is the resulting protocol:

$$\begin{array}{lcl} Alice \rightarrow Bob & : & \{\!|m|\!\}_{KB}, KA^{-1} \\ Bob \rightarrow Alice & : & \{\!|m'|\!\}_{KA}, KB^{-1} \end{array}$$

The symbols mean that Alice sends the first pair of values to Bob and Bob sends the second pair of values to Alice. This protocol clearly does not satisfy the composition of the two specifications: after seeing both messages, a passive eavesdropper can learn both messages, m and m', since each transmission contains the decryption key needed to decrypt the message contained in the other transmission.

A promising approach for developing compositional security properties is to use *observational equivalence,* a standard and well-studied relation in programming language and concurrency theory. For those not familiar with the concept, two programs or systems, P and Q, are observationally equivalent if they give rise to the same observable behavior in all contexts. In symbols,

$$P \cong Q \text{ iff for all contexts } C[\,] \text{ we have } C[P] = C[Q]$$

where $C[P]$ is the result of placing P in context $C[\,]$ and $=$ is some basic equality defined using some primitive form of observations, such as printing a number or sending a boolean value on some predetermined channel. The important fact about observational equivalence is that it is provably a congruence relation. Therefore, if we specify security properties as equivalences between systems and their specifications, compositionality will follow.

To the best of my knowledge, the potential for using observational equivalence in security specifications was first realized by Abadi and Gordon and described in their paper on the Spi-calculus [AG99]. The idea is very general and seems promising for a variety of formalisms, including some simpler that Spi-calculus and some that are more complex (e.g., [LMMS98] and related papers).

References and further information

Copies of the slides for this talk and additional references will be available at the web site listed below the author's address above.

References

[AG99] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society, Series A*, 426(1871):233–271, 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36.

[DY83] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[LMMS98] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Conf. Computer and Communication Security*, 1998.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symp. Security and Privacy*, pages 141–151, 1997.

[MSS98] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In *Proceedings of the 7th USENIX Security Symposium*, pages 201–216, San Antonio, TX, 1998.

[NL96] G. Necula and P. Lee. Safe kernel extensions without run-time checking. In *Second Symposium on Operating Systems Design and Implementation*, 1996.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pau97] L.C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Soc Press, 1995.