Research Statement

Timothy L. Hinrichs

Background and Overview

Computer security is primarily concerned with improving the protection of computer systems against abuse while minimizing the degradation of their usability. Perfect protection is difficult because the system must permit people to carry out mission-critical tasks, and perfect usability is difficult because the system must prevent people from carrying out abusive tasks. A perfectly secure system is one that allows people to use the system as it was intended and stops people from using the system in ways it was not.

At the heart of security is therefore the idea that a system ought to be used in some ways and ought not be used in others. That is, the security of a system can only be evaluated in the context of the semantics for that system—the definition for how the system and its users are supposed to interact. For example, the access control semantics of a system dictates which individuals should be given access to which resources. The privacy semantics of a system describes how much information the system is allowed to infer about the individuals who contribute information to the system. The accountability semantics of a system states the extent to which individuals can be held responsible for the events that occur in the system. Each aspect of a system's semantics allows us to assess how well defended the system is with respect to attacks that attempt to circumvent those semantics.

One differentiator in security research is how formal the representation of a system's semantics is. Some work utilizes only the semantics of the programming language to define incorrect behavior (*e.g.*, buffer overflow). Other work combines the semantics of the programming language with the apparent intent of a program to identify problems (*e.g.*, SQL injection and Cross-Site Scripting). Still other work assumes the semantics of the system is written formally and separately from the program (*e.g.*, in a type system or in first-order or temporal logic).

Generally speaking, my research focuses on **leveraging a separate, formal representation of a system's semantics to improve that system's security** in two ways. The first is precisely *assessing* how well a system meets its security objectives, a process that can be automated to some extent if the semantics is written in a machine-processable language. The second is more ambitious: *enforcing* the semantics of the system automatically. A key difficulty in both tasks is balancing the expressiveness of the language for writing semantics and the computational complexity of analyzing those semantics, problems that I am well-suited to solve given my background in logic and automated reasoning.

Ongoing Work

Web Security In web security, many research problems are concerned with the three basic components of a web application: the client (always implemented in a web browser), the web server (implemented in a myriad of ways), and the HTTP requests that allow the client and server to communicate. Most of the time we are concerned with the security of the web server, since it is responsible for storing large amounts of sensitive information, such as the credit card numbers of a company's clients. According to OWASP's Web Hacking Incident Database, the most prevalent vulnerability in web servers today is failing to adequately check if inputs received over HTTP obey the appropriate semantics (the input validation semantics). In fact, some of the recent web-based

attacks on high-profile organizations like Sony, PBS, and Fox were achieved via SQL injection, an attack made possible by improper input validation.

Improper input validation sometimes occurs because a web developer wishing users to have feedback about improper inputs builds a client that performs sophisticated input validation and a web server that neglects to perform that same validation. The developer's rationale is that any input reaching the server will have already been validated by the client, a faulty assumption because an attacker can compromise the web browser or even submit HTTP requests manually, thereby circumventing the client-side validation entirely.

Two of my projects (NoTamper [1] and WAPTEC [2]) aim to **identify potential input validation attacks** on web servers. By extracting the input validation semantics from the web browser using program analysis and testing the web server against those semantics, we identified a number of parameter tampering vulnerabilities in real-world web applications. For example, using an input validation attack on a bank's web server, an attacker could transfer money between two arbitrary accounts at that bank. Using an input validation attack at a computer equipment e-commerce site, an attacker could purchase any amount of computer equipment for free by additionally "purchasing" negative quantities of other products. The two tools we developed to identify such vulnerabilities differ in how they test the server against the input validation semantics extracted from the browser. NoTamper treats the web server as a black box, only analyzing the web pages returned by the web server, whereas WAPTEC treats the server as a white box, using program analysis to analyze PHP code. NoTamper is therefore more widely applicable since the server need not be written in PHP, but WAPTEC is more accurate and complete.

My third project on web security (Plato [3]) **prevents input validation attacks** by synthesizing both the browser and server validation code from a logical representation of the input validation semantics. Instead of independently writing the client and server validation code, the developer writes the input validation semantics once in logic and invokes Plato to synthesize the necessary code. This simple approach not only reduces the developer's workload but also avoids the problem of maintaining two separate code bases that implement the same functionality, a notoriously difficult problem in practice. Plato is also useful for developers because with a little additional information about how the client should be displayed, it can construct a fully functional, highly interactive web client. A web client constructed by Plato performs two tasks for a user: identifying errors in her data (input validation) and automatically filling in values implied by her data. An error means the user data is *logically inconsistent* with the input validation semantics, and implication means that the user data together with the validation semantics *logically entails* some value. Implication is especially difficult because it requires a version of paraconsistent entailment to ensure that an inconsistent premise set does not imply all possible values. Plato utilizes a form of paraconsistent entailment designed for the web and novel synthesis algorithms to construct code that implements error detection and implied value computation for a given input validation semantics.

Access Control The access control semantics for a computer system dictates which users should be given access to which resources. Unlike many areas of security, it is common for the access control semantics of a system to be stated explicitly and enforced directly. Two of my projects involve designing languages for expressing the access control semantics of a particular domain. A third project aims at *evaluating* how well-suited an access control solution is for a particular domain.

FML (Flow Management Language) [6] is a declarative language for managing small and medium-sized networks that use the commercially-fielded network operating system NOX (a.k.a.

Ethane). NOX is used widely around the globe to enable research (well over 50 academic and research projects) and serves as the basis for such high profile projects as the Open Source Routing project[1], which is backed by Google. FML allows administrators to declaratively express network security policies in terms of high-level names (*e.g.*, users and hosts) instead of individually configuring possibly hundreds of low-level components (*e.g.*, firewalls and router ACLs). The semantics expressed in FML are enforced directly by NOX via a FML interpreter. Conceptually, each time a new flow is initialized, NOX consults the FML policy to decide how to react, *e.g.*, drop the flow, impose waypointing restrictions, or limit its communication rate. The crucial technical problem in this project was designing a language sufficiently expressive to capture a wide range of network security semantics while admitting an implementation that works at network speeds (about 10^5 flows per second). **FML serves as the basis for the policy engine used within Citrix's distributed virtual switch/controller[2], which has been shipping for over a year**.

Another project on access control (TBA—Tag-Based Authorization [5]) was motivated by two MITRE reports detailing the access control problems experienced by the U.S. government when it joins other countries in coalitions to address issues of world-wide significance such as the recent tsunami in Japan. Each time a country joins a coalition, the U.S. must grant access to pertinent information for that country's operatives, and each time a country leaves the coalition, those rights must be revoked. Unlike traditional access control settings, each coalition event can affect thousands of users and millions of resources—the sheer scale of the changes is problematic. TBA confronts this problem by breaking the representation of a country's access control semantics into two pieces. Each country's users and sensitive resources are assigned tags representing their important characteristics, and access control decisions, which are codified in a declarative policy, are made based entirely on the tags of the user and resource in question. When a country joins the coalition, its operatives' tags are added to the U.S. system, and the U.S. modularly augments its policy for the new country; when a country leaves the coalition, tags are removed and the country's module is deleted from the policy. Either the tags or the declarative policy can be changed independently of one another; hence, as users and resources change, relatively untrained users can change their tags, and when countries join or leave the coalition, security analysts can carefully change the logical policy. **In addition to the military, TBA is also well-suited for access control on many of today's social media web sites like Facebook, Twitter, and Flickr** where tagging is commonplace.

FML and TBA are access control solutions designed for particular domains: networks and dynamic coalitions. ACEL [4] is a mathematical framework that allows us to make formal claims about how well-suited a particular access control solution is for a given domain. To use the framework, an analyst formalizes the domain as an *access control workload* and scores each candidate system using a cost metric chosen by the analyst. To demonstrate ACEL, we evaluated several well-known access control systems against the dynamic coalition workload and found that **the model underlying the U.S. government's access control system is fundamentally ill-suited for coping with coalition operations**.

Future Work

In the future, I plan to expand my efforts to understand how to enforce security semantics directly in computer systems. Two of my efforts are based on existing lines of work (web security and access

[1] `http://opensourcerouting.org/`

[2] `http://support.citrix.com/article/CTX129669`

control), and one is based on a new line of work: privacy.

First, web development practices today can and should be enhanced so that instead of a programmer writing a single, monolithic code base that simultaneously encodes all aspects of the application (*i.e.*, the core functionality and all of the pertinent security semantics), the programmer writes several code bases encoding different aspects of the application. A web development framework would then generate the corresponding monolithic code base so that by construction it achieves the desired functionality while enforcing the requisite security semantics. This separation of the intended security semantics from the application functionality simplifies the maintenance and understandability of the application, and it allows the development framework to offer guarantees about the security properties of the overall application.

Second, ACEL is currently limited in several ways. It does not provide an infrastructure for realistically comparing and contrasting logic-based access control solutions like FML and TBA; it does not address workloads with temporal events; it does not allow for automated analyses. In the future, I will expand the infrastructure to address logic-based, non-logic-based, and hybrid solutions; I will expand the workload language to capture a broader class of events; and I will investigate automation through model checking and theorem proving.

Third, recent efforts within the medical community to build HIPAA-compliant computer systems have spurred researchers into developing a logical encoding of HIPAA—a formal privacy semantics for medical applications. I plan to investigate how we might augment existing computer systems and design new ones that are HIPAA-compliant by construction. This particular endeavor is complex because parts of HIPAA rely on information not contained in computer systems (such as whether or not the doctor *believes* an action is in the best interest of the patient). HIPAA therefore serves as a real-world example of a security semantics where only part can be enforced directly by the computer system; the remainder can at best be supported by the system. Building HIPAA-compliant systems will require a careful blend of code synthesis, activity monitoring and alerts, and postmortem activity auditing.

References

[1] P. Bisht, T. L. Hinrichs, N. Skrupsky, R. Bobrowicz, and V. N. Venkatakrishnan. NoTamper: Automatic blackbox detection of parameter tampering opportunities in web applications. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 607–618, 2010.

[2] P. Bisht, T. L. Hinrichs, N. Skrupsky, and V. N. Venkatakrishnan. WAPTEC: Whitebox analysis of web applications for parameter tampering exploit construction. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 445–456, 2011.

[3] T. L. Hinrichs. Plato: A compiler for interactive web forms. In *Proceedings of the Symposium on Practical Aspects of Declarative Languages*, 2011.

[4] T. L. Hinrichs, W. C. Garrison, A. J. Lee, and J. C. Mitchell. Application-sensitive access control evaluation: Logical foundations. (Under review).

[5] T. L. Hinrichs, W. C. Garrison, A. J. Lee, S. Saunders, and J. C. Mitchell. TBA: A hybrid of logic and extensional access control systems. In *Proceedings of the International Workshop on Formal Aspects of Security and Trust*, 2011.

[6] T. L. Hinrichs, N. Gude, M. Casado, J. C. Mitchell, and S. Shenker. Practical declarative network management. In *Proceedings of the ACM SIGCOMM Workshop on Research on Enterprise Networking*, pages 1–10, 2009.