Guest Editors’ Introduction

Engineering Secure Systems

Our computers help us manage and process information across a wide variety of systems and platforms. We can gather, analyze, and store massive amounts of data, streamline global operations with just-in-time deliveries of goods and equipment, control physical infrastructures, operate worldwide financial systems, and, on a more individual level, create art and music, download movies, chat with friends, shop, bank, and more. Most of us assume that our systems are going to do what they’re supposed to do, but what do we mean by “supposed to do,” especially when it comes to security? For the most part, security is a behind-the-scenes service that we don’t concern ourselves with until something goes wrong. Most of us probably don’t expect our systems to allow criminals to obtain or manipulate our valuable information, nor do we expect catastrophic failures of large-scale systems due to manipulation by adversaries.

A typical developer might think that a system is acceptable if it provides the customer’s requested functionality; a wise developer might also ensure that the system isn’t a danger to the user’s health or safety. The result can be a carefully constructed system that also provides the intended services. But wait! What if the system does its job, but still leaves an entryway so that cyber miscreants can slip in and steal or modify valuable information? What if these miscreants wreak havoc by causing systems to go off kilter? Even if our wise developer could construct the system carefully, many such systems are used in ways that were neither intended nor anticipated—for example, systems designed for the enterprise but used in multi-tenant settings such as the cloud. This scenario highlights the problem of misplaced trust: the system we trust isn’t as trustworthy as we had imagined and now exhibits some mixture of both expected and unexpected functionality. A disconnect exists between user assumptions regarding what the system was supposed to do and what it ended up doing. How did this happen? The answer is that the system does something unexpected because it contains unspecified or misused functionality in the form of flaws or, worse, clandestine artifices.

Unspecified functionality is rampant in many of today’s systems. It’s caused by choices made by developers ranging from sloppy design and coding practices to unintended interactions in system composition. It would be a gross mischaracterization to say that all unspecified functionality is exploitable or bad, although you might argue that even the most benign-seeming extra functionality doesn’t conform with expectations regarding the utilization of processor resources. However, it’s correct to state that unspecified or misused functionality often results in exploitable vulnerabilities. Thus, as the value of the information stored in a given system increases, the risk of vulnerability exploitation likewise increases. The economic model can vary. Some attacks are valuable only if they have a large number of victims—for example, those that steal a little bit from a lot of people.
Guest Editors’ Introduction

Others go for the gold and exfiltrate valuable intellectual property or cause infrastructure systems to malfunction, as in the case of the Stuxnet worm.\(^1\)

Historical Review

The history of computer security is the story of concerted efforts to address the problem of unspecified functionality. In the late 1960s and early 1970s, when computers weren’t commodity products and were owned only by large enterprises and major government entities, military security experts regarded the intentional insertion of clandestine code into systems as a palpable threat.\(^2\) To address this challenge, they identified technologies, standards, and processes that allowed system builders to state that their systems were trustworthy.\(^5\)

The systems weren’t perfect, but users could be reasonably confident that exploitation by adversaries had not taken place during system development and that exploitation during system operation would be very difficult. In the intervening years, however, computers became commodity products, product life cycles decreased dramatically, users demanded functionality, and rigorous security engineering suffered.

Construction of highly trustworthy systems was and continues to be highly challenging and borders on an artisanal process: it requires leaders with considerable experience to head development teams, similar to the master masons who built cathedrals with a team of journeymen and apprentices during the Middle Ages. It can also entail a perilous path through political and bureaucratic minefields.

In This Issue

Gone are the days, if they ever existed, when a system could be built and everyone could head off into the sunset confident that it was finished and that maintenance would be minimal. A system must be designed so that its security claims remain valid from inception through retirement. This means that security engineering must start at the earliest stages of development, when wise choices can have a major impact on system trustworthiness and vulnerabilities are relatively inexpensive to fix. In this sense, the system must be designed for extension and upgrades.

We chose three illustrative papers for this special issue on engineering secure systems. They’re intended to offer insights into the world of secure system development. Two come from people in aerospace and industry who have spent lifetimes working on highly trustworthy systems, and the third is from a youthful, energetic academic group.

Used in conjunction with a sound public-key infrastructure, smart cards can facilitate a broad range of activities: identification and access control, shopping, banking, transportation, healthcare, and so on. Recognition of this potential led a team at IBM to develop a highly trustworthy smart-card operating system. In “Lessons Learned Building the Caernarvon High-Assurance Operating System,” Paul Karger, Suzanne McIntosh, Elaine Palmer, David Toll, and Samuel Weber describe their path to a Common Criteria evaluation at Evaluation Assurance Level 7 for their system. EAL7 is the highest assurance rating possible under the international standard and is intended to provide confidence that the system will resist highly sophisticated attacks as well as the usual spectrum of digital nastiness. Their article reveals that technology is only part of the story on the road to high assurance. Alliances, corporate support, and lots of persistence can be decisive factors in successful system development. (Note: Paul Karger died suddenly last fall. He was always ready for a lively discussion and possessed an encyclopedic knowledge of the security literature. He is both loved and missed. The November/December 2010 issue of this magazine contains a retrospective of Paul’s life and contributions to our field.)

Clark Weissman and Timothy Levin provide a perspective on the development of a focused high-assurance distributed system. In “Lessons Learned from Building a High-Assurance Crypto Gateway,” they show that even with a head start of decades of systems security engineering experience, old lessons repeat themselves in new contexts even as new challenges present themselves. The focus of their lessons learned is the Encryption-box Security System, which uses gateways with policy orchestrated by a network security controller to create an encrypted network that connects hosts with various security attributes.

A vexing problem in deploying secure systems is to provide a way to ensure the trustworthy distribution of the systems and their updates.
Guest Editors’ Introduction

...at various points over an extended period. How can an enterprise system’s owners know that the code being booted and executed by clients hasn’t been corrupted at some time? In their article, “Network-Based Root of Trust for Installation,” Joshua Schiffman, Thomas Moyer, Trent Jager, and Patrick McDaniel describe a way to demonstrate that a client file system in an enterprise setting can be cryptographically traced back to its origins: the installer and disk image used to produce it. The article illustrates the use of a hardware extension—in this case, the Trusted Platform Module (TPM)—to provide a chain of evidence that allows a verifier to decide whether the system is acceptable.

Reflections on Security Engineering

Are well-engineered secure systems even possible? Well, not if we don’t have people building them. A lesson we learned as editors of this special issue was that there seems to be considerably less serious security engineering taking place than one would expect, based on the hype about cyberattacks and security breaches that seem to flood the mainstream media. It might seem thrilling to engage in a game of penetrate and patch with cyber adversaries when the battlefield is comprised of systems with easily exploitable foundations that weren’t designed to be secure from the outset, but where’s the work that demonstrates systems are largely free of vulnerabilities? We were a bit surprised at the paucity of activity in this area. This is especially of concern from the perspective of building up the body of knowledge for training future generations of researchers by ensuring that both the mindset and the lessons learned in the past are reflected in future security designs. Rigorous security engineering is hard—no one doubts that—but it doesn’t mean that it isn’t worth doing. Several challenges make the study of security engineering a fruitful area for research and development. Let’s review some topics that appeal to us, and, hopefully, to you as well.

Composition of components so that the whole isn’t less than the sum of its parts has and continues to be a major security engineering challenge. How should elements be interconnected? How should their dependencies be organized? How can the guarantees provided by the engineering of components such as a high-assurance operating system for a smart card be inherited by the larger system—say, the authentication and authorization system that inherits them? There’s no formula to guide us here, just a set of design principles and years of practice. Additional science is needed to address this problem in a way that will be useful to those constructing systems.

Highly trustworthy systems depend on a high degree of modeling and formal verification, both of which are time-consuming, difficult, and require specialized skills. Many of the tools available for this work provide system representations that are incomprehensible to the engineers who must move forward with detailed designs and implementations. Even at the architectural level, we can encounter dissonance when combining formalisms with concrete plans. Evaluation at the highest levels of the Common Criteria (among other requirements) calls for a functional specification, a high-level design, a low-level design, and a formal mapping between them. Considerable work is needed on tools to help engineering teams efficiently construct trustworthy systems.

In addition to security, systems must satisfy many nonfunctional properties to be useful in practice, including the so-called RAS properties (reliability, availability, and serviceability). More recently, many properties are being unified in the concept of resiliency, which captures the idea that a system should continue to operate even in the presence of an external force, perhaps with a reduced level of service. Practical mechanisms for ensuring resiliency (including security) and graceful degradation merit further investigation. Sometimes, there can be a tension between security and other system properties such as survivability and resilience: How can these properties be balanced? Should users be able to choose the balance point? What externalities might drive dynamic adaptation of these properties?

Economic factors can determine whether a system is built using a rigorous methodology. This is arguably the single most important factor determining the viability of secure systems. The cost incurred for engineering them must be balanced against the benefit accrued from the prevention of potential attacks. This is always a difficult argument to make, especially because good security should be invisible. So how is a customer to know whether the system provides a good return on investment?

User acceptability is another major challenge, especially when security seems to be an obstacle to functionality. Furthermore, systems that boast considerable security engineering can present users with arcane, unfathomable interfaces. Usability and intuitively easy navigability of security interfaces are key to ensuring that carefully engineered security mechanisms are used in practice and that systems are configured securely. Can trustworthy systems be constructed in a way that makes their use more intuitive? Certain operating system vendors have built their reputations on great user interfaces. We need such innovation for secure systems.

It’s going to happen: some one will use a system in a way...
for which it wasn’t originally intended. Perhaps the system was constructed based on a particular set of environmental assumptions and now it’s being deployed in a completely different context. Use of systems outside of their specification could have a serious impact on their ability to enforce system policies. We know better than to use household bleach to whiten our teeth—do computer systems need similar warning labels regarding their inappropriate use?

Building systems to be both modifiable and extensible is a laudable goal. Providing system-level support for new applications will ensure a system’s continued use. Unfortunately, this can be quite difficult, particularly when faced with the challenge of re-verification. Is there a less onerous way to modernize and improve systems, yet preserve their security properties?

The three articles in our special issue are a combination of retrospective and forward vision. They all reflect the fact that complex systems demand rigorous security engineering. Without it, we will continue to be the unhappy victims of unspecified functionality. Wonderful opportunities are available for both fruitful research on challenging topics and a combination of lessons learned with new technologies to produce systems that do what we expect and nothing more.

References