Can We Win the Security Game?

I am pleased and honored to have been asked to serve on *IEEE Security & Privacy*'s editorial board. This publication offers a refreshing bridge over the gap in the literature between pure academic papers and the world of the practitioner.

CHARLES C. PALMER
IBM T.J. Watson Research Center

Like many of today's security professionals, I did not begin my career working in security. I started with operating systems and networking in the oil industry and then at IBM Research. In 1996, I had finished an early version of Domain Name Systems Security Extensions (DNSSEC) and had been applying genetic algorithms to network design when I was offered the chance to found IBM's ethical hackers team. The goal was to help customers understand their security stance and security shortcomings by trying to break into their systems.

I could have continued my research, but this new team sounded like a lot of fun. It turned out to be fascinating work. We helped many clients while IBM's security consultants learned from us. After a few years, the consultants took over and the team moved on to tool building and operating system security issues. Since then, I have been leading other security- and privacy-related departments at the T.J. Watson Research Center, whose focus has ranged from cryptography research to secure hardware to Web services security. But enough about me—let's look at where we are with security and privacy.

Increasing vulnerabilities

Currently, computer security and personal information privacy are popular topics. Unfortunately, that's not because everyone is applauding today's systems as being well secured or, in some cases, even securable. Enterprises and regular folks rely so heavily on their computers that incidents like the recent plague of Blaster and Sobig worms have caused widespread alarm. (According to Paul Wood of Messagelabs [www.msnbc.com/news/955498.asp], in the four days starting 18 August 2003, one in 17 emails sent around the world had been affected by Sobig.F, making it the fastest infection ever.)

The constant threat of viruses and worms, combined with the still-strong concern about 9/11-like disasters and business continuity, has pushed security to the top of everyone's list. *InfoWorld*'s 2003 Security Survey (www.infoworld.com/pdf/special_report/SecurRep2003.pdf) found broad agreement regarding the top threats to enterprise network security: Trojans, viruses, worms, and other malicious code; hackers; employee error; and employee, ex-employee, and partner sabotage. If you visit CERT's Web site and look at past years' statistics (www.cert.org/stats/cert_stats.html), you'll find that roughly 10 new vulnerabilities are reported each day. And while we all complain about how disrupting it is to constantly install patches or update virus signatures on our personal systems, what about the enterprise that has thousands of machines with different problems, patch levels, and problem-avoidance procedures? Finally, how should we feel if that enterprise is operating a critical national infrastructure component, such as energy management, telecommunications, or transportation?

How did we get into this mess?

If it hurts when you do that, then don't do it!

Far too often, this old punch line applies to products' security and privacy aspects. Whether because of a complicated interface, endless configuration options, or incompatible point solutions written by indifferent programmers, many products are too hard to use, introduce unacceptable performance costs, or just don't work. Even products that specifically focus on helping with security and privacy challenges are sometimes hopelessly complex or obscure. As a result, users and system administrators often just turn off security controls or ignore their alerts.

We're in this mess primarily because of three bad habits:

- *Fast development.* In the computer boom time of the 90s, the focus was on getting the software out fast, gain market position, and then (maybe) working with the first several customers to resolve any bugs or problems. Missing a market opportunity was far more deadly for a company than releasing software before it was ready. There was also a severe shortage of experienced

programmers, which meant that many under-experienced people were writing thousands of lines of software as fast as they could.

- *Add-on security*. Too often, security and privacy considerations are underspecified, weak, or simply omitted from a product's design. Adding strong, effective security and privacy capabilities to an existing design or implementation is just not possible. For example, imagine adding air bags to an automobile after it rolled off the assembly line; would you feel safe driving that car?
- *Feature creep*. Having escaped the constraints of small memory sizes and slow processor speeds, software developers seem to have little motivation to resist the rampant feature creep in today's systems. As systems have become increasingly complex, so have their programming models. Today's software development tools haven't stepped up to the task of helping programmers efficiently design and implement these complicated systems, and they certainly don't help them develop secure software.

So, with all these problems, what do we do now?

Security engineering

Writing software is unlike any other kind of engineering humans have ever done. When a skilled potter makes a water pitcher, the most important indicators of its quality are evident to the pitcher's buyer: attributes such as a good handle, a well-formed spout point for pouring, and being free of leaky cracks, to name a few. As engineering matured, so did the tools engineers used, which furthered the complexity of the things they built.

Over time, consumers of these goods became less able to assess their quality and were forced to trust the engineers. For example, when the Romans built their aqueducts, the typical citizen, who fully understood the pitcher, did not understand the details of collecting and making water flow to the fountain, at which they filled their pitchers. Clearly, if an aqueduct leaked, citizens might notice it. But other, more subtle, design bugs would surely have gone unnoticed. Over time, those same engineers developed standard practices for building aqueducts and other systems, so that others might learn from their experiences and avoid disasters.

Fast forward to today, when software engineers build such complex things that the vast majority of humans (including other engineers!) have no idea how they work. These creations defy even the very tools of the traditional engineering trade: reliable tools for standard benchmarks, rigorous mathematical analyses of an implementation's accuracy, complete test coverage of all its potential uses, and so on.

Software engineering techniques attempt to address these concerns. Such techniques typically require engineers to spend time during the design phase carefully specifying how the target system should work using highly expressive modeling tools such as the Unified Modeling Language (UML). We then bring out tools that convert design diagrams into code, formal modeling techniques that verify a design, and test-coverage tools that dutifully run for days, all to ensure that the code does what it is supposed to do.

So how can we be sure that the code will not do other things as well? To meet this challenge, the developer must stop thinking only about what the code should do, and instead think about how the code might be misused, abused, attacked, or crashed. This is the realm of security engineering. Teaching this discipline requires the developer to assume a new mental model during the entire software life cycle, from requirements gathering to design and implementation on through to testing and support. Mark G. Graff and Kenneth R. van Wyk's *Secure Coding* comprehensively covers this topic, and I highly recommend it.[1]

How can software engineers determine that their recently developed software does not pose a security risk? While purists would point out that answering this question for any but the most trivial software is intractable, the security engineer can use these tools and techniques to study and predict the target software's security level. This can lead to a formal security certification for the software when an independent evaluator standardizes and carries out the process. This certified "assurance level" is used somewhat like the US Underwriter's Labs' mark on various appliances: it represents the approval of an objective organization that both the developer and the customer can trust. For the CIO buying secure software, these ratings provide a standard for comparing the assurance level of various products, rather than having to trust marketing messages and brochures.

However, these certifications have their downsides. First, they can be very expensive to obtain, requiring longer, more detailed design and implementation processes and far more documentation than for noncertified products. Furthermore, each certification covers only a single software release—each subsequent release requires another, albeit smaller, evaluation to maintain the certification.

Systems built without specifications cannot fail; they merely present surprises – usually unpleasant.

—Robert Morris Sr.

Both of these facts can result in less frequent software releases.

Some might argue that the evaluators, who are certified by government agencies or by industry groups with their own guidelines and best practices, might be no more skilled than the software developers. This could indeed be the case, but confidence that the software will behave properly increases because the evaluators are objective parties with reputations (and businesses) to protect. They review the software for security and privacy concerns and, if granted certification, must publicly state that they performed the evaluation. When combined with the strong documentation required for such evaluations, the certification process clearly has value. So, while supporters of the formal certification process admit that it is not perfect, there really is no other rational yardstick with which to predict the level of security some software will provide.

Along with changing the way software engineers think about the security their work provides, and how they can convince others that they have succeeded via formal certification, software engineers need some help: the software engineering tools currently used offer little in the way of security and privacy development support. Tomorrow's software engineers need far better tools than we have today to support the security engineering goal. Some of these new tools are beginning to appear now, doing static security analyses of code, deadlock predictions, and even suggesting security and privacy best practices to the developer throughout the software life cycle.

Security must disappear

Although my product development colleagues cringe when they hear me say it, our only hope for securing the enterprise of the future is for security and privacy management to disappear. I'm not advocating that we just give up and go home. Rather, I'm suggesting that the final result of serious efforts to incorporate security and privacy requirements into the software life cycle will result in products that are easy to use and configure, with nonintrusive security and privacy.

While we struggle to manage our current systems' security and privacy, it's clear that the steady march of system complexity will continue, posing even tougher challenges tomorrow. So, in addition to discarding obnoxious user interfaces and overcomplicated configuration screens, we need operational, or runtime, help as well. For this, some autonomic computing concepts (see www.research.ibm.com/autonomic/) can be brought to bear. Concepts of self-optimization, self-protection, and self-healing are goals that security and privacy advocates share. Although the security-specific autonomic computing techniques (see www.research.ibm.com/journal/sj/421/chess.pdf) hold promise, we must tread lightly.

We have already seen the failure of an early attempt at autonomic security when some routers were configured to automatically block traffic from network addresses suspected of initiating attacks. Hackers caused the routers to effectively disconnect themselves from the network by simulating attacks from thousands of harmless addresses.

It is refreshing to see increased industry awareness of products' security and privacy aspects along with increased customer demand for certified products. Despite the statements I've made here, I don't mean to suggest that all products should be certified or mathematically proven to match their specifications. However, as long as we continue to accept shoddy software engineering practices, we will continue to be at an ever-increasing risk of system failures.

The goal is that product developers actively consider their products' security and privacy issues throughout the life cycle, consciously deciding just how far they want to go toward addressing those issues. This security awareness extends to customers as well, who should think about their security and privacy needs. Security and privacy is, of course, just a risk-management game. Our only hope for a secure, privacy-respecting future is for industry, customers, and academia to no longer treat this as a game of chance. □

Reference

1. M.G. Graff and K.R. van Wyk, *Secure Coding: Principles & Practices*, O'Reilly, 2003.

Charles C. Palmer *manages the IBM Thomas J. Watson Research Center's Security, Privacy, and Cryptography Departments. His teams work in the areas of cryptography research, Internet security technologies, Java security, Web services security, privacy, biometrics, secure embedded systems, high-assurance systems, and the global security analysis lab, also known as the ethical hackers lab, which he founded in 1995. He has also been an adjunct professor of computer science at Polytechnic University in New York. Although his primary focus is now security and privacy, Palmer remains active in the genetic algorithms community by participating in program committees. He has a PhD in computer science from Polytechnic University, an MS in computer science from Tulane University, and a BS in computer science from Oklahoma State University. He is a member of the ACM, the IEEE, and the IEEE Computer Society. Contact him at ccpalmer@us.ibm.com; www.research.ibm.com/people/c/cpalmer/.*