Security: Technical, Social, and Legal Challenges

Bill Arbaugh, University of Maryland at College Park

Information security is increasingly making headlines these days, and usually the news is not good—some new Internet worm, virus, or Trojan is devastating corporate networks. After more than 25 years of research and engineering, why is our information infrastructure more at risk than ever? Unfortunately, there is no simple answer to this question, and there are no simple solutions.

Addressing the problem requires understanding the nature of security and appreciating the need for a more scientific approach. A long-term solution involves significantly improving the design and implementation of software, which demands collaboration between security professionals and software engineers.

A CONTINUOUS PROCESS

What exactly is security? The term itself means little. Many assume that a single security mechanism such as cryptography provides "security," but this is not the case. Effective security is a continuous process involving the design, installation, and operation of a system. In short, security is multidimensional.

The dynamic nature of security makes protecting information system resources difficult. Every aspect of the life cycle of all protected systems is involved. A single design or implementation error can quickly render a presumably secure corporate network insecure. Consider the vulnerabilities associated just with electronic mail and Web browsers: One infected e-mail attachment or piece of malicious downloaded code can spread rapidly, corrupting files, knocking out servers, and crashing computer systems.

Risk management is not well understood within the information security community.

RISK MANAGEMENT

Corporate security architects spend countless hours and money building the cyberspace equivalent of the Maginot line—the fortifications the French built after World War I to prevent another German invasion. Then these architects watch helplessly as hackers, like the German Blitzkrieg, circumvent their costly static defenses and attack them at their weakest point—in this case, consumer applications. Today's security professionals face a conundrum: how to protect everything all of the time as transparently as possible.

As William R. Cheswick and Steven M. Bellovin explained in *Firewalls and Internet Security: Repelling the Wily Hacker* (Addison-Wesley, Reading, Mass., 1994), "any program, no matter how innocuous it seems, can harbor security holes…." Not only can such holes appear in any piece of software (or hardware for that matter), they can be introduced at any point within a product's life cycle. Because security architects have little or no control over most of the life cycle of the products they rely on to protect their infrastructure, they must use risk management daily.

Risk management is a familiar process in the financial and insurance communities, but it is not well understood within the information security community. In fact, risk management as practiced today involves more art than science.

Rules and standards for information security best practices exist, but they are usually ad hoc and based on experience rather than empirical evidence. Insurance underwriters, for example, base life insurance policy decisions on actuarial tables describing the life expectancy of the individuals they insure. However, no such tools are currently available to compute the likelihood that a given information system will suffer a security breach.

A SCIENTIFIC APPROACH

For the security community to make such reliable estimates, it must be able to quantify

- the value of the protected information,
- the threat to that information, and
- the assurance level.

One way to quantify the assurance level is through its dual work factor—

the work required by an adversary to successfully exploit a targeted system. The information security community universally recognizes the work factor as a concept for measuring a cryptographic algorithm's security because all possible attacks against the algorithm are direct and measurable, such as a brute-force strategy to exhaust the key space.

Work factor does not, however, apply to the implementation of cryptographic algorithms—it is quantifiable in theory but not in practice because implementations are susceptible to indirect attacks, usually against external interfaces, which are difficult to quantify.

Paul C. Kocher's calculations of the subtle variations in the time required to perform certain private-key operations demonstrated an entire new series of indirect attacks against presumably protected private keys ("Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," http://www.cryptography.com/timingattack/paper.html). Similarly, Mike Bond and Ross Anderson demonstrated a new class of indirect attacks against a tamper-resistant cryptographic system by manipulating its software application programming interface, causing the system to leak information about its secret keys ("API-Level Attacks on Embedded Systems," *Computer,* Oct. 2001, pp. 67-75).

Both types of attacks result in the recovery of the secret key the cryptographic algorithm uses—the same end as a brute-force strategy—yet they are not included in the initial work factor computation. Arguably, we could quantify timing and API attacks, but what about currently unknown types of attacks? How can we quantify the potential of a trusted insider to inadvertently or maliciously release sensitive information?

SIMPLIFYING DESIGN

We cannot overlook the role of humans in the security equation. One of the 10 security principles that Jerome H. Saltzer and Michael D. Schroeder introduced in their 1975 article, "The Protection of Information in Computer Systems" (*Proceedings of the IEEE,* vol. 63, no. 9, pp. 1278-1308), is psychological acceptance. They correctly recognized that users often either operate security protections incorrectly or, worse, do not use them at all if the mechanisms are difficult to operate or are otherwise an impediment. For example, users commonly fail to secure a product with an insecure default configuration even when the problem is well known.

The security and software engineering communities must find ways to develop software correctly in a timely and cost-effective fashion.

Designing complex systems has never been easy, and security systems are no exception. Security can only be effective if it is transparent to both users and administrators. Unfortunately, this is hard to achieve in both commercial and specialized security systems.

Albert Einstein best articulated a generally accepted design principle when he said, "Everything should be as simple as possible, but no simpler." In his 1980 ACM Turing Award lecture, "The Emperor's Old Clothes" (*Comm. ACM*, vol. 24, no. 2, Feb. 1981, pp. 75-83), C.A.R. Hoare applied this philosophy to software design. According to Hoare, "…there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies."

Unfortunately, the vast majority of software designed today falls into the latter category. Radically simplifying modern software design to significantly improve the way *all* software is constructed is critical to achieving effective security.

INTEGRATING SECURITY

An even greater problem is persuading architects of consumer systems to incorporate security into their designs from the beginning rather than retrofitting solutions once vulnerabilities are found. The telephone, cellular, and wireless local area networking industries all had major security problems. Eliminating them required spending considerable time and effort to revise the initial design.

Systems are designed without security as a major concern for many reasons, primarily related to a misunderstanding of economics and performance. Vendors usually do not include robust security in their systems because doing so incurs additional direct costs. Unfortunately, little data is available to indicate whether the lack of a secure design increases the long-term indirect costs. As a result, new ways of measuring security impact and risk are critical.

In his Turing Award lecture, Hoare described four design principles he used to implement Algol 60. The first principle for implementing this algorithmic language was security. "A consequence of this principle," Hoare said, "is that every occurrence of every subscript of every subscripted variable was on every occasion checked at runtime against both the upper and the lower declared bounds of the array."

Yet today, approximately one-half of all security-related vulnerabilities occur due to buffer overflows. We know how to avoid buffer overflows, commercial and open source tools are available to help identify them (although admittedly they are not sufficient yet), and we can use programming languages that provide robust bounds checking at runtime. Why does this problem continue to occur?

I believe, as do many others, that vendors have thus far determined that the cost of developing software correctly far exceeds the potential costs of the downstream problems that not doing so creates. To make any significant improvements in the security of systems, the security and software engi-

neering communities must work closely to find ways to demonstrate that they can develop software correctly in a timely and cost-effective fashion rather than rely on an ineffective penetrate-and-patch strategy (William A. Arbaugh, William L. Fithen, and John McHugh, "Windows of Vulnerability: A Case Study Analysis," *Computer*, Dec. 2000, pp. 52-59).

PRIVACY

Although it is a necessary condition for privacy, security alone is not sufficient. Security provides only the means of controlling the release of sensitive information. It does not prevent presumably trusted parties from releasing that information outside the scope of the agreement that provided the basis for giving it to them—trust is transitive.

Those who wittingly share sensitive data with another assume that the recipient

- will use that information only in accordance with an explicit or implicit privacy agreement, and
- has the means—security mechanisms—in place to protect the information.

Today, unfortunately, both assumptions are seldom true. Several large companies have abruptly revised their privacy policies, using information about their customers in a fashion outside the original privacy agreement, and hackers have electronically stolen identities from company databases and Web sites.

Protecting privacy requires formalizing the nature of the trust relationship between the owner and the holder of information through improved laws and agreements as well as drastically improving security mechanisms and systems.

LEGAL OBSTACLES

As if the technical hurdles facing the security community were not enough, ill-conceived laws severely hamper researchers' ability to verify vendors' security claims. Legislation such as the 1988 Digital Millennium Copyright Act (http://www.loc.gov/copyright/legislation/dmca.pdf) and the Uniform Computer Information Transactions Act (http://www.ucitaonline.com/) makes it a crime to reverse engineer a product containing security mechanisms designed to protect intellectual property, as in the case of the DMCA, and in general as in the case of UCITA.

Overly broad and vague laws have created a cloud of legal uncertainty over an important area of security research and engineering.

These overly broad and vague laws have created a cloud of legal uncertainty over an important area of security research and engineering. When, or if, that cloud will clear is difficult to forecast because both the government and advocates of the legislation are happy with the current state of confusion.

The security and privacy community faces a wide range of difficult technical, social, and legal challenges. In future issues of *Computer,* guest columnists will join me every other month in presenting opinions and overviews of ongoing research related to information security and privacy, highlighting both problems and solutions. ■

***Bill Arbaugh** is an assistant professor in the Department of Computer Science and at the University of Maryland Institute for Advanced Computer Studies at the University of Maryland at College Park. Contact him at waa@cs.umd.edu.*