A Map of Security Risks Associated with Using COTS

Ulf Lindqvist and Erland Jonsson
Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
{ulfl, erland.jonsson}@ce.chalmers.se

As published in *Computer*, Vol. 31, No. 6, June 1998, pp. 60-66.

Cover Feature

A Map of Security Risks Associated with Using COTS

The widespread use of commercial off-the-shelf products in combination with increased internetworking calls for an analysis of the associated security risks. Combining Internet connectivity and COTS-based systems results in increased threats from both external and internal sources.

Ulf Lindqvist
Erland Jonsson
Chalmers University of Technology

The traditional security design approach has been one of risk avoidance, not only in systems with high-security (military grade) requirements but also in medium-security systems, such as those typically found in financial institutions and corporate research departments. The design approach has been to construct mainly customer-specific solutions using security mechanisms such as physical "air gap" separation, information flow analysis, and strict or formal development and verification methods. However, there are several reasons why these techniques are applied less frequently today:

- Developing entirely customer-specific solutions is usually far more expensive and in all cases more time-consuming than purchasing COTS products.
- Some security mechanisms, particularly cryptography, have proven so difficult to implement correctly that developers should be provided with ready-made building blocks, relieving them of the risk of introducing subtle but serious flaws.
- Most organizations want connectivity and internetworking rather than physical separation. (Physical separation—also called "sneaker net"—requires manual intervention to transport data between a protected system and the outside world.)

More and more members of the security community realize the impracticality and insufficiency of risk avoidance as the sole doctrine. This was understood long ago in the reliability community, where fault tolerance was developed as a complement to fault prevention.[1] It turns out that strict development procedures can only *reduce* the number of flaws in a complex system, not eliminate every single one. Vulnerabilities may also be introduced by changes in the system environment or the way the system operates. Therefore, both developers and system owners must anticipate security problems and have a strategy for dealing with them.[2] This is particularly important with COTS-based systems, because system owners have no control over the development of the components.

SECURITY-RELATED COTS PRODUCTS

Any type of COTS component might have an impact on the overall system security, depending on how it is used in the system. Therefore every type of COTS product could be security-related. On the other hand, not all COTS products are designed without relevant security concerns. Some COTS products are indeed designed, implemented, and evaluated according to medium or even high levels of security functionality and assurance (although these products are in the minority). Examples of COTS products intended to improve security include cryptographic software (and hardware), network firewalls, and antivirus tools.

The open question, however, is what level of security one can attain by composing a system of different products. An ideal design goal would be to make the overall system security independent of how some untrusted components behave, but that is often difficult to accomplish in practice.

COTS operating systems deserve special attention, for several reasons:

- Operating systems are perhaps the most widespread COTS products.
- Only a handful of different basic types of COTS operating systems exist from which to choose, and they show wide variation in their security functionality and assurance.

0018-9162/98/$10.00 © 1998 IEEE

- Many application programs rely on the operating system to enforce security mechanisms, such as user identification, authentication, and access control.

TAXONOMY OF SECURITY RISKS

Every situation in which the use of computers can affect something valuable (for example, human lives or health, privacy, economic assets, or national security) involves risks. Peter G. Neumann informally defines a risk as "a potential problem, with causes and effects," although pointing out that there is no standard definition of the term.[3]

Here we are mainly concerned with security risks, which we define as

- the system, through human misuse, experiences loss of confidentiality, integrity, or availability for any of its resources; or
- the system, through misuse or by accident, experiences the introduction of a security vulnerability. (A security vulnerability is a flaw that could later be exploited to cause a loss of confidentiality, integrity, or availability.)

Our taxonomy is a map of potential problem areas. It can be used to aid the analysis of security risks when using systems that to some extent contain COTS components. It is based on the typical phases in the establishment of the system.

Component design

Some security risks originate from the design of the COTS components and are consequently beyond the control of the customers:

- *Inadvertently flawed component design.* The components may have various types of bugs, some of which may affect security.
- *Intentionally flawed component design.* The components may contain intentional security flaws, such as backdoors, viruses, or Trojan horses (for a more detailed explanation of this problem, see the sidebar "Defining the Confinement Problem.").
- *Excessive component functionality.* A component may have many more features than the customer needs or even knows about, and so the customer might not realize the true security implications of including the component in the system.
- *Open or widely spread component design.* Although most academic security researchers (including ourselves) promote openness and public scrutiny for better security, a risk does exist if details of the component design are widely known outside the customer organization. Even worse, from a security point of view, is the common situation in which the design is known to a large development organization and its partners but not to the customers.
- *Insufficient or incorrect documentation.* The developer might not provide the customer with the documentation needed to correctly and securely integrate the component into the system.

Some security risks originate from the design of the COTS components and are consequently beyond the control of the customers.

Component procurement

There are also security risks associated with purchasing and delivering components:

- *Insufficient component validation.* A component purchase might not fully conform with the customer's *real* security requirements, which are not necessarily the same as the customer's *specified* requirements.
- *Delivery through insecure channel.* For example, in downloading a software component via the Internet, the product might be manipulated along the way by a third party (an intermediary attack) or the customer might be tricked into downloading a manipulated product from a site controlled by the attacker, instead of the real product from the vendor's site.

Component integration

Integrating components, which is a step in the design of the composed system, has the following risks:

- *Mismatch between product security levels.* A common problem when integrating different products is that the security level must be set to the lowest common denominator to make the products work together. For example, in the Microsoft Windows NT File System, user access to local system files and folders can be restricted to read-only permission to prevent accidental or intentional modification. However, for Microsoft Office 97 to work properly, the user must be given write permission for a number of system folders and files.[4]
- *Insufficient understanding of integration requirements.* The integrators might not fully understand all of the preconditions for secure integration of the products, for example, that some components must be physically protected.

Internet connection of system

When the system is connected to the Internet, a number of additional risks must be considered:

- *Increased external exposure.* By connecting the system to the Internet, exposure expands to a large number of potential external attackers who otherwise would not have any data communication path to the system.

Defining the Confinement Problem

The *confinement problem* is a classic computer security problem. Basically, it is a question of how to limit the actions of an executing program that normally has all the privileges of its invoking user and, therefore, can do anything the invoker can do.

One example of such a program is a Trojan horse. A Trojan-horse program appears to be benign and to behave as expected by the invoker but, in addition to or instead of the expected actions, does something malicious.

A classic and most elegant example of a Trojan horse was presented by Ken Thompson.[1] Thompson describes a portion of code hidden in the C compiler on a Unix system. When Thompson's modified compiler compiles the login program, it inserts a backdoor into the login binary that, for example, grants access to any account upon entering a "master key" password. To ensure that the Trojan horse does not disappear when the C compiler is replaced by a new version, Thompson's compiler detects when it is used to compile a new C compiler and inserts the Trojan code into the object code of the new compiler.

Unfortunately, Trojan horses present not only a theoretical problem for researchers but a serious threat to ordinary users of computer systems. In the world of computing today, factors such as increased software complexity, dynamic code linking, user abstraction from underlying functions, Internet connectivity, and frequent downloading of executable code from various sources all facilitate the insertion of Trojan horses and make them difficult to discover.

A general problem

However, the confinement problem does not apply only to Trojan horses. Untrusted software that is not malicious might still have side effects that are unexpected by the invoker and may cause security problems. This should be of particular concern to developers of systems that include COTS components.

Traditionally, computer security has focused on confidentiality, that is, making sure that only authorized subjects (people, processes) have read access to certain information. Therefore, an early definition of the confinement problem concerns information leakage: "…the problem of confining a program during its execution so that it cannot transmit information to any other program except its caller."[2] Great effort has been expended on the analysis and elimination of covert channels (unauthorized communication paths through which a process could transmit confidential information).

The basic problem can be stated more generally: "What is the appropriate way to confine an untrusted program so that it can do everything it needs to do to meet the user's expectations, but nothing else?" The imprecise definition does not give much hope for a final solution, and the real difficulty lies in correctly specifying the permitted behavior of the program.

Suggested solutions

The general problem could be addressed, however, if it were possible always to follow the principle of least privilege, which states that every subject should operate using the least set of privileges necessary to complete the job.[3] Several access control models designed with the purpose of enforcing that principle have been proposed.[4] With military applications in mind, designers have developed operating systems implementing *mandatory access control*, which bases confinement on data classification labels and personnel clearance.

A more recent approach is *domain-and-type enforcement* (DTE), in which an attribute called a domain is associated with each subject and another attribute called a type is associated with each object. A central matrix specifies whether a particular mode of access to objects of a type is granted or denied to subjects in a domain.[5]

Another type of confinement mechanism is the Java security model,[6] which is an example of language-based confinement of downloaded mobile code. Cryptographic methods for ensuring authenticity and integrity of programs are often suggested, but their main drawback is that they only solve the problem of confirming the author's identity and that the program has not been altered by someone else; they are of little help when the user does not trust the author.

References

1. K. Thompson, "Reflections on Trusting Trust," *Comm. ACM*, Aug. 1984, pp. 761-763.
2. B.W. Lampson, "A Note on the Confinement Problem," *Comm. ACM*, Oct. 1973, pp. 613-615.
3. J.H. Saltzer and M.D. Schroeder, "The Protection of Information in Computer Systems," *Proc. IEEE*, Sept. 1975, pp. 1,278-1,308.
4. R.S. Sandhu, "Lattice-Based Access Control Models," *Computer*, Nov. 1993, pp. 9-19.
5. W.E. Boebert and R.Y. Kain, "A Further Note on the Confinement Problem," *Proc. 30th Int'l Carnahan Conf. Security Technology*, IEEE Press, Piscataway, N.J., 1996, pp. 198-203.
6. G. McGraw and E.W. Felten, *Java Security: Hostile Applets, Holes & Antidotes*, John Wiley & Sons, New York, 1996.

- *Intrusion information and tools easily available*. An insider who decides to attack the system can get a great deal of applicable information from the Internet.
- *Executable content*. Many World Wide Web pages have executable content (for example, Java applets) that automatically downloads and executes on a user's computer when viewing the page in a Web browser. Credulous users might well run programs that attack their system.
- *Outward channel for stolen information*. The Internet connection constitutes a channel that can covertly and conveniently export information stolen from the system, for example, by internal attackers or by programs planted by external attackers.

System use

Some risks are related to how the users operate the system:

- *Unintended use.* The system can be used in an unintended way, for example, to store and process data that are more sensitive than the system was designed to handle or to attack other systems.
- *Insufficient understanding of function.* Users might not be able to judge their adherence to the security policy if they do not fully understand a function. For example, they might not know whether or not a particular program transmits passwords in the clear over the network.

System maintenance

Finally, there are risks involved in the maintenance of the system:

- *Insecure updating.* In the same way as the initial software delivery is risky if performed via an insecure channel, software updates can be modified in transit or system owners can be fooled into installing fraudulent updates.
- *Unexpected side effects.* Any changes made to components in the system can have unexpected side effects and might introduce new security vulnerabilities.
- *Maintenance backdoors.* The history of computer insecurity contains many cases in which developers left open backdoors for convenient testing and maintenance of their products. However, such backdoors can be misused by anyone who knows or finds out about them.

ANALYZING RISKS TO PRIVACY IN A DATABASE SYSTEM

We were invited to investigate the security of a privacy-oriented database system under development. The system, which was based mainly on COTS products, was designed to strongly protect the privacy of the individuals recorded in the database. Our study revealed a large number of security problems that we reported to the developers and later further analyzed in terms of underlying causes and possible remedies.[5]

The system was intended for personal registers in government or municipal services, offices in health care and in social care, and other public services. The major design goal was to make it virtually impossible to link a sensitive record in the database to an individual without proper authorization. To accomplish this separation of data, the designers used a combination of cryptographic devices and record pseudonymity. Their idea was to split identifying data and descriptive data between two separate databases, using an individual identification number, similar to a Social Security number, as the link between them. Figure 1 illustrates how data could be split between the two databases.

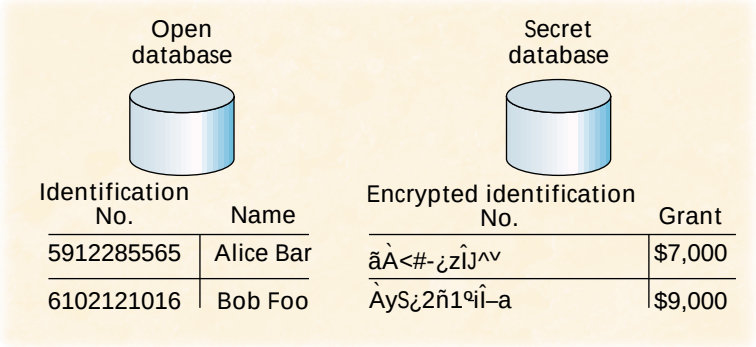

Figure 1. The two databases with simple examples of records.

- The *open database* would contain publicly available data such as name and address, with plaintext individual identification numbers as record identification fields.
- The *secret database* would contain sensitive data, with encrypted individual numbers as record identifiers. That is, the records in the secret database are pseudonymous rather than anonymous. The designers did not require encryption of any other fields in the secret database unless that information could be used to link confidential information to specific individuals.

The design goal was to make the database system resistant to many different kinds of potential attackers, ranging from dishonest or disgruntled current or former employees to other organizations and even foreign governments. Furthermore, even with physical access to the client hosts, to a copy of the databases from the server host, or to both, the attackers should not be able to violate the security policy of the system. The developers summarized the comprehensive security policy of the system in two statements:

- *Confidentiality.* Only authorized users should be able to link records in the secret database to a single individual.
- *Integrity.* Only authorized users should be able to modify records in a meaningful way.

Most of the system components were COTS products from IBM: The computers were PS/2 PCs running OS/2 2.11, the Transaction Security System[6] (TSS) was used for encryption and authentication, the database-management system (DBMS) was DB2/2, and the OS/2 LAN Server was used for network communication between the clients and the server.

The major design goal was to make it virtually impossible to link a sensitive record in the database to an individual without proper authorization.

Several of the problems exposed in our study could be traced to the fact that all of the COTS components (except the TSS) were developed with lower security requirements than the composed database system. The developers had failed to make the security of the system independent of the (in)security of those components.

Risks revealed

A selected subset of the problems we found shows typical COTS-related security risks.

- *Trojan horse in client.* Users had to present a smart card and a secret PIN to start the application and the cryptographic operations that were needed to identify records in the secret database. Owing to the lack of security protection mechanisms in the operating system, nothing could prevent an attacker with physical access to a client host from installing a Trojan horse that could, for example, record all transactions the user performs.
- *Information leaking to swap file.* The virtual memory swap file of OS/2 might contain sensitive information. The application has no control over what information is transferred to this file. It would be possible for an attacker to collect information by searching this file after the authorized user has finished working.
- *DBMS log files.* The log files of the DBMS contained, among other things, information about the origin of records, which could be used to correlate records in the two databases.
- *DBMS ordering of records.* The designers had no control over the ordering of records in the database. In the system analyzed, records were always added in the same order in both the open and secret databases. With access to copies (disk or backups) of the two databases, it would be an easy task to identify all the secret records.

The last item is perhaps the most evident example of how a "simple" but serious problem can be overlooked when developers rely on general-purpose products to solve problems for which those products were not designed.

All of these problems can be categorized as insufficient understanding of integration requirements. Or, if we choose to consider the components as unsuitable for this type of application, as insufficient component validation.

RISKS EXPERIENCED IN INTRUSION EXPERIMENTS

There is currently no established method to quantitatively measure the security of a system in comparison with other systems. However, there are guidelines for building systems with a certain level of security functionality and assurance, and developers can submit their system to a third-party evaluator who will try to determine whether the system was built according to the guidelines. In the reliability field, guidelines exist for building high-reliability systems, as well as methods to test the system and measure its reliability in an artificial operational environment, typically through various fault-injection methods.[7]

With that analogy in mind, and with the objective of finding operational security measurements, we conducted intrusion experiments in which students were encouraged to attack a certain system for a limited period of time, under careful supervision and with the requirement that all their activities be reported and documented.

Thus far, we have performed one pilot experiment and three full-scale experiments on a Unix system (SunOS 4.*x*) and one full-scale experiment on a Novell NetWare system. We have analyzed the first Unix experiment and presented a model of the intrusion process,[8] as well as a taxonomy of intrusion techniques and results. Analysis of the other experiments is in progress.

We chose to use ordinary students as attackers instead of experienced crackers and to provide them with standard user accounts. In this way, we would model the insider threat; that is, when legitimate users of a system for some reason decide to extend or misuse their privileges. We also ensured that each test environment represented a standard installation of a common COTS-based computing system.

The results of the stated experiments should be interesting to all readers who use comparable systems. Unfortunately, those results are not comforting:

- Almost all attackers performed successful intrusions.
- Several of the intrusions were indeed severe, giving the attacker administrator privileges.
- The Internet provides a vast amount of information on how to successfully attack common systems.
- Known vulnerabilities are often technically difficult to exploit. Still, many of our attackers broke into the system through such holes (often without really understanding why it was possible) by using so-called *exploit scripts* published on the Internet.

The last item describes a serious threat, of which today's system owners must be aware. A relatively small community of technically skilled crackers prepares programs that automatically exploit some vulnerability in a common type of system and makes these programs available on the Internet. Consequently, the group of potential attackers who can perform technically advanced intrusions now includes all who can find, download, and execute these programs—clearly an immensely large number of people.

The exploit scripts (which can be shell scripts, source code, or precompiled binaries) do not always work as distributed, probably to prevent people without any programming skills from using them. However, the errors are sometimes easy to fix, and many exploit scripts are ready to use. Typically, the exploit scripts take advantage of flaws in privileged programs (such as `setuid` programs in Unix) or processes (typically network server processes) by, for example, acting in one of the following ways:

- The exploit script calls the victim program with input data that were unexpected by the author of the victim program. The input data must in some cases be carefully crafted by the author of the exploit script, whereas in other cases the author simply provides an excessive amount of random data.
- The exploit script makes unexpected changes in the execution environment, for example, by moving or renaming files accessed by a privileged process.
- The exploit script retrieves the secret upon which security is based (typically a password or a cryptographic key) through, for example, shrewd guessing or an exhaustive search.

Our experiments resulted in a wide variety of intrusions, each of which would normally be possible owing to a combination of several risks rather than a single one. (Incidentally, this seems to be a general fact.) For example, one well-known intrusion uses the `setuid` mechanism of the Sendmail program. This may be related to the inadvertently flawed component design and excessive component functionality as well as insufficient component validation in the taxonomy. On the other hand, the fact that this intrusion method was known to the attackers may make it referable to the class intrusion information and tools easily available.

A RISK MANAGEMENT APPROACH

The problem of protecting COTS-based systems connected to the Internet is difficult, because this combination increases outsider as well as insider threats. Simply by connecting to the outside world, a system becomes vastly more exposed to external attackers. Furthermore, as observed in our experiments, the existence and availability of exploit scripts and information about flaws also increase the threat from insiders.

Solutions must be sought in the risk management field, where the cost of protection is traditionally weighed against the potential loss caused by a violation of security. A modern risk management philosophy must also include the following:

- *A well-defined and relevant security policy.* The security policy primarily defines what is and is not allowed in terms of system security, although it should also include dictates regarding enforcement, responsibilities, and reporting. In fact, an intrusion is defined as a violation of the security policy (regardless of whether the violation comes from the inside or outside). Without a security policy, you cannot determine if an event is an intrusion, strictly speaking. Hence, the definition of a relevant security policy is a prerequisite for security risk management.
- *Holistic perspective.* System security must be viewed as a holistic property; it is not sufficient to just look at a small number of stronger parts (compare with the database system example). Thus, system security must be considered in a space as well as a time dimension. By time, we mean the system life cycle: Security risks should be estimated in the development, procurement, integration, operation, and maintenance of a system. By space, we mean the structure of the system and the environment in which it is embedded, an environment that includes humans, buildings, and organizations.
- *Confinement of untrusted components.* Processes that are untrusted should be limited in what they can do, ideally to the extent that they can do nothing else than exactly what they are supposed to do (see the sidebar "Defining the Confinement Problem"). For example, a process that normally does not perform any network communication should not be allowed to connect to the network. Confining untrusted COTS components is highly desirable, but is difficult to do in practice. It might be difficult to correctly determine the minimum set of resources that black-box components require to function in all cases. If you fail to do this, the component may not be able to operate in an unforeseen situation. Or you may specify a confinement that is too lax to provide an appropriate level of security. And if you use COTS products to perform the confinement (for example, using a standard Web browser to confine Java applets), the question becomes how far the guardians themselves can be trusted.
- *Partitioning.* The growing complexity of systems and networks is the source of numerous security problems. By partitioning a system into relatively small parts, separated by "watertight bulkheads" in the form of trusted monitoring components, the gain is twofold: Each part becomes less complex and more manageable in terms of security, and the effects of an intrusion are likely to be limited to a single section. The partitioning must be performed with caution, however, to avoid the creation of undesired "single points of failure," for example, a single vulnerable link for the vital communication between two sections.
- *Contingency anticipation.* All systems have security vulnerabilities, and many sites will experience security violations. An organization that has accepted this and made appropri-

A relatively small community of technically skilled crackers prepares programs that automatically exploit some vulnerability in a common type of system and makes these programs available on the Internet.

ate planning will more likely succeed in limiting loss after having been the victim of an intrusion. This contingency planning, preferably performed with the support of software tools, should include methods for intrusion detection, evidence collection, and recovery.

- *Flaw remediation and active evolution.* A system owner should strive to remove all known vulnerabilities in a system as soon as they are discovered. This is not always easy, because all implications of the remedial actions must be carefully considered. Furthermore, component developers might be unwilling to produce patches. However, our experiments show that removal of all publicly known vulnerabilities would make the attackers' task significantly more difficult. Continuous replacement of components when new security technologies emerge is also important.
- *Support.* The continuous risk management process must be supported by all levels of an organization, from top management down to ordinary users. Organizations using COTS products also need active security support from both developers and third-party vendors.
- *Awareness.* The tradition of covering up security incidents aids only the attackers and must be broken if we will ever have a chance to learn from earlier mistakes. Through education and openness concerning security, people can become more motivated and many risks can be avoided.

The use of COTS systems presents two faces, from a security point of view. On the one hand, security vulnerabilities in those systems will be continuously discovered, owing to the fact that crackers will find it more rewarding to look for flaws and write exploit scripts that can be used to attack many systems. On the other hand, a large customer base should mean that vendors can afford to make an extensive effort to fix security problems. Furthermore, such systems will be closely watched by the security community, and alerts of security problems will be readily announced. However, alerts are of little use if they are not read, understood, spread throughout organizations, and followed by appropriate measures taken by vendors, managers, administrators, and users. Thus, we believe that awareness and openness in security issues are the only means to gain manageable security in COTS-based systems. ❖

References

1. A. Avizienis, "Design of Fault-Tolerant Computers," *Proc. 1967 Fall Joint Computer Conf.*, Thompson Books, Washington, D.C., Vol. 31, 1967, pp. 733-743.
2. J.J. Kahn and M.D. Abrams, "Contingency Planning: What to Do when Bad Things Happen to Good Systems," *Proc. 18th Nat'l Information Systems Security Conf.*, Nat'l Inst. of Standards and Technology/Nat'l Computer Security Center, Baltimore, Md., 1995, pp. 470-479.
3. P.G. Neumann, *Computer-Related Risks*, ACM Press, New York, 1995, pp. 2,348.
4. "OFF97: Security Requirements when Using NTFS Partitions," Article Q169387, Microsoft Corp., Redmond, Wash., 1997.
5. U. Lindqvist, T. Olovsson, and E. Jonsson, "An Analysis of a Secure System Based on Trusted Components," *Proc. 11th Ann. Conf. Computer Assurance*, IEEE Press, Piscataway, N.J., 1996, pp. 213-223.
6. D.G. Abraham et al., "Transaction Security System," *IBM Systems J.*, Vol. 30, No. 2, 1991, pp. 206-229.
7. R.K. Iyer, "Experimental Evaluation," *Proc. 25th Int'l Symp. Fault-Tolerant Computing (Special Issue)*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 115-132.
8. E. Jonsson and T. Olovsson, "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior," *IEEE Trans. Software Eng.*, Apr. 1997, pp. 235-245.

Ulf Lindqvist is a PhD research student in the Department of Computer Engineering at Chalmers University of Technology. His research focuses on computer and network security, especially methods for analysis, categorization, and detection of intrusions and vulnerabilities. Lindqvist received an MS in computer science and engineering and a licentiate of engineering from Chalmers. He is a student member of the IEEE, the IEEE Computer Society, the ACM, and Usenix.

Erland Jonsson is an associate professor of computer security and head of the Department of Computer Engineering at Chalmers University of Technology. His research interests include the quantitative assessment of security, as well as systems with simultaneous security, reliability, and safety requirements. Jonsson received an MS in electrical engineering and a PhD in computer engineering from Chalmers. He is a board member of the Special Interest Group for Security of the Swedish Information Processing Society and a member of the IEEE Computer Society.

Contact the authors at the Department of Computer Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden; {ulfl,erland.jonsson}@ce.chalmers.se.