

# Model-based Security Engineering (Keynote)

Jan Jürjens  
Software & Systems Engineering  
Dep. of Informatics, TU Munich, Germany\*  
<http://www4.in.tum.de/~juerjens>

## Abstract

The current state of the art in security-critical software is far from satisfactory: New security vulnerabilities are discovered on an almost daily basis. To improve this situation, we develop techniques and tools that perform an automated analysis of software artefacts for security requirements (such as secrecy, integrity, and authenticity). These artefacts include specifications in the Unified Modeling Language (UML), annotated source code, and run-time data such as security permissions. The security analysis techniques make use of model-checkers and automated theorem provers for first-order logic. We give examples for security flaws found in industrial software using our tools.

## 1 IT Security Today

### 1.1 Problems

Attacks against computer networks, which the infrastructures of modern society and modern economies rely on, can cause substantial financial damage.<sup>1</sup> They can even threaten humans lives.<sup>2</sup> Due to the increasing interconnection of systems, such attacks can be waged anonymously and from a safe distance. Thus networked computers need to be secure.

The high-quality development of security-critical systems is difficult. Still, many systems are developed, deployed, and used over years that contain significant security weaknesses. Software engineering is the key to improve security: Over 90% of security incidents reported to the CERT Coordination Center result from defects in software requirements, design, or code.<sup>3</sup> Many flaws and possible sources of misunderstanding have been found in protocol or system

---

\*From Oct. 2006: Open University, UK.

<sup>1</sup>For example, US companies were reported to loose \$7 million per year averagely due to successful attacks on critical business assets, according to the 2004 CSI/FBI Computer Crime and Security Survey, [www.gocsi.com](http://www.gocsi.com).

<sup>2</sup>For example, Dutch hackers managed to break into US DOD computer systems in 1990/91 and managed to access military-critical information about the Persian Gulf War which they allegedly tried to sell to the Irak [Den03].

<sup>3</sup>SEI/CERT, online at [www.cert.org](http://www.cert.org) .

specifications, sometimes years after their publication or use. For example, the observations in [Low96] were made 17 years after the well-known Needham–Schroeder authentication protocol had been published in [NS78]. Many vulnerabilities in deployed security-critical systems have been exploited, sometimes leading to spectacular attacks. For example, as part of a 1997 exercise, an NSA hacker team demonstrated how to break into US Department of Defense computers and the US electric power grid system, among other things simulating a series of rolling power outages and 911 emergency telephone overloads in Washington, DC, and other cities [Sch99]. While there are of course many more recent examples of security breaches, this particular example also shows that there is more to be concerned about than website defacements and creditcard misuse.

Computer breaches do significant damage, as a study by the Computer Security Institute shows: Ninety percent of the respondents detected computer security breaches within the last 12 months. Forty-four percent of them were willing and able to quantify the damage. These 223 firms reported \$455,848,000 in financial losses [Ric03].

## 1.2 Causes

While tracing requirements during software development is difficult enough, enforcing security requirements is intrinsically subtle, because one has to take into account the interaction of the system with motivated adversaries that act independently. Thus security mechanisms, such as security protocols, are notoriously hard to design correctly, even for experts. Also, a system is only as secure as its weakest part or aspect.

Security is compromised most often not by breaking dedicated mechanisms such as encryption or security protocols, but by exploiting weaknesses in the way they are being used [And01]: According to A. Shamir, the Israeli state security apparatus is not hampered in its investigations by the fact that suspects may use encryption technology that may be virtually impossible to break. Instead, other weaknesses in overall computer security can be exploited [Sha99]. As another example, the security of Common Electronic Purse Specifications (CEPS) [CEP01] transactions depends on the assumption that it is not feasible for the attacker to act as a relay between an attacked card and an attacked terminal. However, this is not explicitly stated, and it is furthermore planned to use the CEPS over the Internet, where an attacker could easily act as such a relay. This is investigated in [Jür04]. As a last example, [Wal00] attributes the failures in the security of the mobile phone protocol GSM among other reasons to the failure to acknowledge limitations of the underlying physical security, such as misplaced trust in terminal identity and the possibility to create false base stations.

Thus it is not enough to ensure correct functioning of security mechanisms used. They cannot be “blindly” inserted into a security-critical system, but the overall system development must take security aspects into account in a coherent way [SS75]. More specifically, one can say that “those who think that their problem can be solved by simply applying cryptography don’t understand cryptography and don’t understand their problem” (R. Needham). In fact, according to [Sch99], 85% of Computer Emergency Response Team (CERT) security advisories could not have been prevented just by making use of cryptography. Thus,

given the current state of software security, just using encryption to protect communication still leaves most weaknesses unresolved, and has been compared to using an armored car to deliver credit card information “from someone living in a cardboard box to someone living on a park bench” [VM02]. Building trustworthy components does not suffice, since the interconnections and interactions of components play a significant role in trustworthiness [Sch99].

Lastly, while functional requirements are generally analyzed carefully in systems development, security considerations often arise after the fact. Adding security as an afterthought, however, often leads to problems [Gas88, And01].

It has remained true over the last 30 years since the seminal paper [SS75] that no coherent and complete methodology to ensure security in the construction of large general-purpose systems exists yet, in spite of very active research and many useful results addressing particular subgoals [Sch99], as well as a large body of security engineering knowledge accumulated [And01]. Such a methodology would allow the computer security engineer to construct a system in a way similar to how a civil engineer would build a bridge. In contrast, today ad hoc development leads to many deployed systems that do not satisfy important security requirements. Thus a sound methodology supporting secure systems development is needed.

### 1.3 State of the Art in Practice

In practice, the traditional strategy for security assurance has been “penetrate and patch”: It has been accepted that deployed systems contain vulnerabilities. Whenever a penetration of the system is noticed and the exploited weakness can be identified, the vulnerability is removed. Sometimes this is supported by employing friendly teams trained in penetrating computer systems, the so-called “tiger teams” [Wei95, McG98].

For many systems, this approach is not ideal: Each penetration using a new vulnerability may already have caused significant damage, before the vulnerability can be removed. For systems that offer strong incentives for attack, such as financial applications, the prospect of being able to exploit a discovered weakness only once may already be enough motivation to search for such a weakness. System administrators are often hesitant to apply patches, *especially* in critical systems, since applying the patch may disrupt the service [And01]. Having to create and distribute patches costs money and leads to loss of customer confidence. Patches may contain security threats themselves, such as the FunLove virus in a Microsoft hotfix distributed in April 2001 [Mic01].

It would thus be preferable to consider security aspects more seriously in earlier phases of the system life-cycle, before a system is deployed, or even implemented, because late correction of requirements errors costs up to 200 times as much as early correction [Boe81].

The difficulty of designing security mechanisms correctly has motivated quite successful research using mathematical concepts and tools to ensure correct design of small security-critical components such as security protocols, including [BAN89, KMM94, Low96, Pau98]. The goal is to establish crucial requirements at the specification level through formalization and proof, which may be mechanically assisted or even automated. Note that it is not possible to actually prove a system secure in an absolute sense: Proofs can only be performed with respect to models which are necessarily abstractions from reality. Attackers can

always try to go beyond the limitations of a given model to still attempt an attack. Nevertheless, a model-based security analysis is useful, because certain attacks can be prevented and the required effort for successful attacks increased. Also, often problems with a specification are detected just by trying to make it sufficiently precise for formal analysis.

Unfortunately, due to a perceived high cost in personnel training and use, formal methods have not yet been employed very widely in industrial development [Hoa96, Hei99, KK04]. To increase industry acceptance in the context of security-critical systems, it would be beneficial to integrate security requirements analysis with a standard development method, which should be easy to learn and to use [CW96]. Also, security concerns must inform every phase of software development, from requirements engineering to design, implementation, testing, and deployment [DS00].

Some other challenges for using sound engineering methods for secure systems development exist. Currently a large part of effort both in analyzing and implementing specifications is wasted since these are often formulated imprecisely and unintelligibly, if they exist at all. If increased precision by use of a particular notation brings an additional advantage, such as automated tool support for security analysis, this may however be sufficient incentive for providing it. Since software developers often hesitate to learn a particular formal method to do this, because of limited resources in time and training, one needs to instead use the artifacts that are at any rate constructed in industrial software development. Examples include specification models in the Unified Modeling Language (UML), source code, and configuration data. Also, the boundaries of the specified components with the rest of the system need to be carefully examined, for example with respect to implicit assumptions on the system context [Gol00]. Lastly, a more technical issue is that formalized security properties are not always preserved by refinement, which is the so-called *refinement problem* [RSG<sup>+</sup>01]. Since an implementation is necessarily a refinement of its specification, an implementation of a secure specification may, in such a situation, not be secure, which is clearly undesirable. Also, it hinders the use of stepwise development, where one starts with an abstract specification and refines it in several steps to a concrete specification which is implemented, allowing mistakes to be detected early in the development cycle, and thus leading to considerable savings: Without preservation of security by refinement, developing secure systems in a stepwise manner requires one to redo the security analysis after each refinement step. Hence, we need formalizations of security requirements that are indeed preserved under refinement.

Influential software engineering models (CMM, Iso9000, V-Modell XT) consider security concerns to a very limited degree or do not consider them at all. Conversely, key security engineering documents (e.g. Common Criteria) do not address software engineering issues. Cooperation of two communities (security and software engineering) has until recently been limited [DS00]. As a result, security concerns are often considered only after deployment, when attacks or security bugs are found and it is very expensive to fix the problems. Instead, to reduce the number of security defects, security concerns must be regarded as key factors throughout the whole software development process, rather than being considered only after deployment (e.g., by applying patches) whenever incidents happened already.

## 2 Towards a Solution

### 2.1 Model-based Security Engineering

In MBSE [Jür02, Jür04, Jür05a, Jür05b, Jür06, Jür07], recurring security requirements (such as secrecy, integrity, authenticity and others) and security assumptions on the system environment, can be specified either within a UML specification, or within the source code (Java or C) as annotations. The associated tools [UML04] (Fig. 1) generate logical formulas formalizing the execution semantics and the annotated security requirements. Automated theorem provers and model checkers automatically establish whether the security requirements hold. If not, a Prolog-based tool automatically generates an attack sequence violating the security requirement, which can be examined to determine and remove the weakness. This way we encapsulate knowledge on prudent security engineering as annotations in models or code and make it available to developers who may not be security experts. Since the analysis that is performed is too sophisticated to be done manually, it is also valuable to security experts.

One can use MBSE within model-based development (Fig. 2). Here one first constructs a model of the system. Then, the implementation is derived from the model: either automatically using code generation, or manually, in which case one can generate test sequences from the model to establish conformance of the code regarding the model. The goal is to increase the quality of the software while keeping the implementation cost and the time-to-market bounded. For security-critical systems, this approach allows one to consider security requirements from early on in the development process, within the development context, and in a seamless way through the development cycle: One can first check that the system fulfills the relevant security requirements on the design level by analyzing the model and secondly that the code is in fact secure by generating test sequences from the model. However, one can also use our analysis techniques and tools within a traditional software engineering context, or where one has to incorporate legacy systems that were not developed in a model-based way. Here, one starts out with the source code. Our tools extract models from the source code, which can then again be analyzed against the security requirements. Using MBSE, one can incorporate the configuration data (such

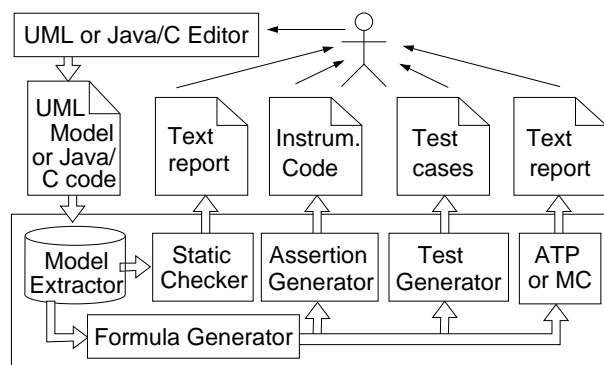


Figure 1: Model-based Security Tool Suite

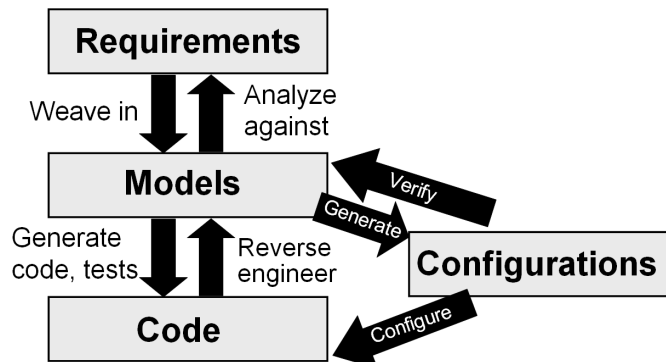


Figure 2: Model-based Security Engineering

as user permissions) in the analysis, which is very important for security but often neglected.

## 2.2 Security Design Analysis using UMLsec [Jür04]

The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate the security requirements and assumptions. *Constraints* give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics of the used fragment of UML. The security-relevant information added using stereotypes includes security assumptions on the physical level of the system, security requirements related to the secure handling and communication of data, and security policies that system parts are supposed to obey. The UMLsec tool-support in Fig. 2) can be used to check the constraints associated with UMLsec stereotypes mechanically, based on XMI output of the diagrams from the UML drawing tool in use [UML04, Jür05b]. There is also a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. Thus advanced users of the UMLsec approach can use this framework to implement verification routines for the constraints of self-defined stereotypes. The semantics for the fragment of UML used for UMLsec is defined in [Jür04] using so-called *UML Machines*, which is a kind of state machine with input/output interfaces similar to Broy's Focus model, whose behavior can be specified in a notation similar to that of Abstract State Machines (ASMs), and which is equipped with UML-type communication mechanisms. On this basis, important security requirements such as secrecy, integrity, authenticity, and secure information flow are defined. To support stepwise development, we show secrecy, integrity, authenticity, and secure information flow to be *preserved* under refinement and the composition of system components. We have also developed an approach that supports the secure development of layered security services (such as layered security protocols). UMLsec can be used to specify and implement security patterns, and is supported by dedicated secure systems development processes, in particular an Aspect-Oriented Modeling approach which separates complex security mechanisms (which implement the security

aspect model) from the core functionality of the system (the primary model) in order to allow a security verification of the particularly security-critical parts, and also of the composed model.

### 2.3 Code Security Assurance [Jür05a, Jür06]

Even if specifications exist for the implemented system, and even if these are formally analyzed, there is usually no guarantee that the implementation actually conforms to the specification. To deal with this problem, we use the following approach: After specifying the system in UMLsec and verifying the model against the given security goals as explained above, we make sure that the implementation correctly implements the specification with techniques explained below. In particular, this approach is applicable to legacy systems. In ongoing work, we are automating this approach to free one of the need to manually construct the UMLsec model.

**Run-time Security Monitoring using Assertions** A simple and effective alternative is to insert security checks generated from the UMLsec specification that remain in the code while in use, for example using the `assertion` statement that is part of the Java language. These assertions then throw security exceptions when violated at run-time. In a similar way, this can also be done for C code.

**Model-based Test Generation** For performance-intensive applications, it may be preferable not to leave the assertions active in the code. This can be done by making sure by extensive testing that the assertions are always satisfied. We can generate the test sequences automatically from the UMLsec specifications. More generally, this way we can ensure that the code actually conforms to the UMLsec specification. Since complete test coverage is often infeasible, our approach automatically selects those test cases that are particularly sensitive to the specified security requirements.

**Automated Code Verification against Interface Specifications** For highly non-deterministic systems such as those using cryptography, testing can only provide assurance up to a certain degree. For higher levels of trustworthiness, it may therefore be desirable to establish that the code does enforce the annotations by a formal verification of the source code against the UMLsec interface specifications. We have developed an approach that does this automatically and efficiently by proving locally that the security checks in the specification are actually enforced in the source code.

**Automated Code Security Analysis** We developed an approach to use automated theorem provers for first-order logic to directly formally verify crypto-based Java implementations based on control flow graphs that are automatically generated (and without first manually constructing an interface specification). It supports an abstract and modular security analysis by using assertions in the source code. Thus large software systems can be divided into small parts for which a formal security analysis can be performed more easily and the results

composed. Currently, this approach works especially well with nicely structured code (such as created using the MBSE development process).

**Secure Software-Hardware Interfaces** We have tailored the code security analysis approach to software close to the hardware level. More concretely, we considered the industrial Cryptographic Token Interface Standard PKCS 11 which defines how software on untrustworthy hardware can make use of tamper-proof hardware such as smart-cards to perform cryptographic operations on sensitive data. We developed an approach for automated security analysis with first-order logic theorem provers of crypto protocol implementations making use of this standard.

## 2.4 Analyzing Security Configurations

We have also performed research on linking the UMLsec approach with the automated analysis of security-critical configuration data. For example, our tools automatically checks SAP R/3 user permissions for security policy rules formulated as UML specifications [Jür04]. Because of its modular architecture and its standardized interfaces, the tool can be adapted to check security constraints in other kinds of application software, such as firewalls or other access control configurations.

## 2.5 Industrial Applications

of MBSE include:

**Biometric Authentication** For a project with an industrial partner, MBSE was chosen to support the development of a biometric authentication system at the specification level, where three significant security flaws were found [Jür05b]. We also applied it to the source-code level for a prototypical implementation constructed from the specification [Jür05a].

**Common Electronic Purse Specifications** MBSE was applied to a security analysis of the Common Electronic Purse Specifications (CEPS), a candidate for a globally interoperable electronic purse standard supported by organizations representing 90 % of the world's electronic purse cards (including Visa International). We found three significant security weaknesses in the purchase and load transaction protocols [Jür04], proposed improvements to the specifications and showed that these are secure [Jür04]. We also performed a security analysis of a prototypical Java Card implementation of CEPS.

**Web-based Banking Application** In a project with a German bank, MBSE was applied to a web-based banking application to be used by customers to fill out and sign digital order forms [Jür04]. The personal data in the forms must be kept confidential, and orders securely authenticated. The system uses a proprietary client authentication protocol layered over an SSL connection supposed to provide confidentiality and server authentication. Using the MBSE approach, the system architecture and the protocol were specified and verified with regard to the relevant security requirements.

In other applications [Jür04], MBSE was used ...

- to uncover a flaw in a variant of the Internet protocol TLS proposed at IEEE Infocom 1999, and suggest and verify a correction of the protocol.
- to perform a security verification of the Java implementation Jessie of SSL.
- to correctly employ advanced Java 2 or CORBA security concepts in a way that allows an automated security analysis of the resulting systems.
- for an analysis of the security policies of a German mobile phone operator.
- for a security analysis of the specifications for the German Electronic Health Card in development by the German Ministry of Health.
- for the security analysis of an electronic purse system developed for the Oktoberfest in Munich.
- for a security analysis of an electronic signature pad based contract signing architecture under consideration by a German insurance company.
- in a project with a German car manufacturer for the security analysis of an intranet-based web information system.
- with a German chip manufacturer and a German reinsurance company for security risk assessment, also regarding Return on Security Investment.
- in applications specifically targeted to service-based, health telematics, and automotive systems.

### 3 Outlook

Given the current unsatisfactory state of computer security in practice, MBSE seems a promising approach, since it enables developers who are not experts in security to make use of security engineering knowledge encapsulated in a widely used design notation. Since there are many highly subtle security requirements which can hardly be verified with the “naked eye”, even security experts may profit from this approach. Thus one can avoid mistakes that are difficult to find by testing alone, such as breaches of subtle security requirements, as well as the disadvantages of the “penetrate-and-patch” approach. Since preventing security flaws early in the system life-cycle can significantly reduce costs, this gives a potential for developing securer systems in a cost-efficient way. MBSE has been successfully applied in industrial projects involving German government agencies and major banks, insurance companies, smart card and car manufacturers, and other companies. The approach has been generalized to other application domains such as real-time and dependability. Experiences show that the approach is adequate for use in practice, after relatively little training. On the basis of the book [Jür04] and the associated tutorial material and tools [UML04], usage of UMLsec can in fact be rather easily taught to industrial developers.

**Acknowledgements** The research summarized in this keynote has benefitted from the help of too many people to be able to include here; they are listed in [Jür04].

## References

- [And01] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, New York, 2001.
- [BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society, Series A*, 426(1871):233–271, December 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in *ACM Transactions on Computer Systems* 8, 1:18–36 (February 1990).
- [Boe81] B.W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.
- [CEP01] CEPSCO. Common Electronic Purse Specifications, 2001. Business Requirements Version 7.0, Functional Requirements Version 6.3, Technical Specification Version 2.3, available from <http://www.cepsco.com>.
- [CW96] E. Clarke and J. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.
- [Den03] D.E. Denning. *Information Warfare and Security*. Addison-Wesley, 2003.
- [DS00] P. Devanbu and S. Stubblebine. Software engineering for security: A roadmap. In *22nd International Conference on Software Engineering (ICSE 2000): Future of Software Engineering Track*, pages 227–239. ACM, 2000.
- [Gas88] M. Gasser. *Building a secure computer system*. Van Nostrand Reinhold, New York, 1988. Available at <http://nucia.ist.unomaha.edu/library/gasserbook.pdf>.
- [Gol00] D. Gollmann. On the verification of cryptographic protocols – a tale of two committees. In S. Schneider and P. Ryan, editors, *Workshop on Security Architectures and Information Flow*, volume 32 of *ENTCS*. Elsevier, 2000.
- [Hei99] C. Heitmeyer. Formal methods for developing software specifications: Paths to wider usage. In H. R. Arabnia, editor, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 1999)*, pages 1047–1053, 1999.
- [Hoa96] C. A. R. Hoare. How did software get so reliable without proof? In M.-C. Gaudel and J. Woodcock, editors, *Formal Methods Europe 1996 (FME): Industrial Benefit and Advances in Formal Methods*, volume 1051 of *LNCS*, pages 1–17. Springer, 1996.
- [Jür02] J. Jürjens. UMLsec: Extending UML for secure systems development. In *5th Int. Conf. on the Unified Modeling Language (UML)*, LNCS. Springer, 2002.
- [Jür04] J. Jürjens. *Secure Systems Development with UML*. Springer, 2004.

- [Jür05a] J. Jürjens. Code security analysis of a biometric authentication system using automated theorem provers. In *ACSAC'05*. IEEE, 2005.
- [Jür05b] J. Jürjens. Sound methods and effective tools for model-based security engineering with UML. In *27th Int. Conf. on Softw. Engineering*. IEEE, 2005.
- [Jür06] J. Jürjens. Security analysis of crypto-based Java programs using automated theorem provers. In *21st IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2006.
- [Jür07] J. Jürjens. *IT-Security*. Springer, 2007. In preparation.
- [KK04] R. Kilian-Kehr. Can formal verification become mainstream in software engineering ? In J. Jürjens, editor, *FoMSESS 2004*, 2004. Second Workshop of the Working Group on Formal Methods and Software Engineering for Safety and Security (FoMSESS) of the German Computer Society (GI). Available at <http://www4.in.tum.de/~fomseess>.
- [KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, Spring 1994.
- [Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software Concepts and Tools*, 17(3):93–102, 1996.
- [McG98] G. McGraw. Testing for security during development: Why we should scrap penetrate-and-patch. *IEEE Aerospace and Electronic Systems*, April 1998.
- [Mic01] Microsoft TechNet. Information about virus-infected hotfixes, April 25 2001. Available at <http://www.microsoft.com/technet/security/topics/virus/vihotfix.msp>.
- [NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
- [Pau98] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.
- [Ric03] R. Richardson. 2003 CSI/FBI computer crime and security survey. Technical report, Computer Security Institute, San Francisco, May 2003. Available at <http://www.gocsi.com/forms/fbi/pdf.html>.
- [RSG<sup>+</sup>01] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, Reading, MA, 2001.
- [Sch99] F. Schneider, editor. *Trust in Cyberspace*. National Academy Press, Washington, DC, 1999. Available at <http://www.nap.edu/readingroom/books/trust>.
- [Sha99] A. Shamir. Crypto predictions. In *3rd International Conference on Financial Cryptography (FC 1999)*, 1999.

- [SS75] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.
- [UML04] UMLsec group. Security analysis tool, 2004. <http://www.umlsec.org>.
- [VM02] J. Viega and G. McGraw. *Building Secure Software*. Reading, MA, 2002.
- [Wal00] M. Walker. On the security of 3GPP networks. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT*, volume 1807 of *LNCS*, pages 102–103. Springer, 2000.
- [Wei95] C. Weissman. Penetration testing. In M. Abrams, S. Jajodia, and H. Podell, editors, *Information security: an integrated collection of essays*, chapter 11, pages 269–296. IEEE Computer Society Press, Silver Springs, MD, 1995.