Encapsulating Rules of Prudent Security Engineering (Position Paper)

Jan Jürjens*

Computing Laboratory, University of Oxford, GB

Vers. 6.VI.01 of a paper at IWSecP01. Current version and other material: www.jurjens.de/jan .

Abstract. In practice, security of computer systems is compromised most often not by breaking dedicated mechanisms (such as security protocols), but by exploiting vulnerabilities in the way they are employed. Towards a solution of this problem we aim to encapsulate rules of prudent security engineering in such a way that a system specification formulated in (a formal core of) the Unified Modeling Language (UML, the industry-standard in object-oriented modelling) can be evaluated wrt. these rules, violations be indicated and suggestions for modifications be derived.

1 Introduction

In the context of computer security, "an expansive view of the problem is most appropriate to help ensure that no gaps appear in the strategy" [SS75].

However, it has remained true over the last 25 years that "no complete method applicable to the construction of large general-purpose systems exists yet" [SS75] that would ensure security, inspite of very active research and many useful results addressing particular subgoals [Sch99].

Thus problems often arise from design limitations and implementation errors rather than defects in security mechanisms [And94, And01]. Therefore research in avoiding design and implementation errors is one of the main challenges in computer security research [Sch99].

For instance, in the case of GSM security [Wal00], some examples for security weaknesses arising in this way are

- the failure to acknowledge limitations of the underlying physical security (misplaced trust in terminal identity; false base stations),
- an inadequate degree of flexibility to upgrade security functions over time and

* jan@comlab.ox.ac.uk – http://www.jurjens.de/jan – Supported by the Studienstiftung des deutschen Volkes and the Computing Laboratory.

– lack in the user interface wrt. communicating security-critical information (no indication to the user that encryption is on).

We aim to draw attention to such design limitations during the design phase, before a system is actually implemented.

We use a formal core of the Unified Modeling Language (UML [RJB99], the industry-standard in object-oriented modelling) to encapsulate knowledge on prudent security engineering and thereby make it available to developers of security-critical systems [Jür01d, Jür01b, Jür01c, Jür01a]. More precisely, we use a fragment of UML together with a formal semantics (which helps us formulate our concepts). So far, there exists no universally agreed-upon formal semantics for all of UML, but even if there will never be one, one can use our approach (by incorporating the security checks into a tool and explaining them informally, just as programming languages are usually used without a formal semantics, however unsatisfactory this may be).

Currently a large part of effort both in verifying and in implementing specifications is wasted since these are often formulated imprecisely and unintelligibly [Pau98]. Being able to express security-relevant information in a widely used design notation helps alleviate this problem (especially if this allows designers to reuse concepts and results formulated in this notation). Additionally, this may reduce misunderstandings that may arise between different parties involved in designing and evaluating security-critical systems (cf. e.g. [Gol00]).

Since we are using a simplified formal fragment of UML, we may reason formally, showing e.g. that a given system is as secure as certain components of it. Furthermore one may go beyond formal verification and make use of techniques more feasible in practice, such as specification-based testing (e.g. following ideas in [JW01b]).

In this position paper, we explain our approach by showing how it relates to the principles of security engineering set out in [SS75], using examples from earlier work (for which the details have to be omitted and can be found in the respective references given). We also mention briefly how one might apply our approach to investigate security protocols in the system context.

2 Design Principles for Secure Systems

We demonstrate by examples how our approach relates to the rules stated in [SS75].

Economy of mechanism Our approach addresses this "meta-property" by providing developers (possibly without background knowledge in security) with guidance on the employment of security mechanisms who might otherwise be tempted to employ more complicated mechanisms since these may seem more secure.

Fail-safe defaults One may verify that a system is fail-safe by showing that certain security-relevant invariants are ensured throughout the execution of the system, i. e. in particular if the execution is interrupted at some point (possibly due to malicious intent of one of the parties involved). An example is secure log-keeping for audit control.

As an example for modelling audit control with our approach we give a part of the specification of the unlinked load transaction of the smart-card based Common Electronic Purse Specifications (CEPS) [CEP01] given in [Jür01b]. The simplified behaviour of the card is given in Figure 1 (since it just serves as an illustration for our approach we omit the explanation which can be found in [Jür01b]). We use a UML statechart diagram, which

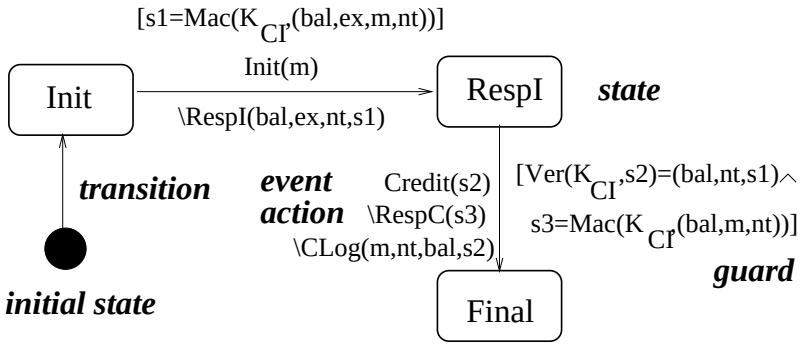

Fig. 1. Statechart for card

is a notation for state machines where, intuitively, a label \Msg(args) on a transition means to output the message Msg with arguments args, Msg(args) means to trigger the transition on input of Msg whose arguments are assigned to the variable args and [condition] means to trigger the transition only if condition is fulfilled.

In the context of this load transaction, one aspect of audit security is that the cardholder should only be lead to believe (e.g. when checking the card with a portable card reader after the transaction) that a certain

amount has been correctly loaded if she is later able to *prove* this using the card – otherwise the load acquirer could first credit the card with the correct amount, but later in the settlement process claim that the cardholder tries to fake the transaction. Thus we have to check the following audit security condition on the attributes of the audit log object CardLog (again this is just for illustration; details can be found in [Jür01b]).

Correct amount: $s2$ and $s1$ verify correctly (say $\mathsf{Ver}(K_{CI}, \mathsf{CardLog}.s2) = (bal', nt', s1')$ and $\mathsf{Verify}(K_{CI}, s1') = (bal'', ex'', m'', nt'')$), and additionally we have $\mathsf{CardLog}.m = m''$ (i. e. the correct amount is logged).

This way one can e.g. uncover unstated assumptions on the trust relations between the protocol participants on which its security relies (e.g. that audit security relies on the fact that the load acquirer trusts the card issuer; details cf. [Jür01b]).

Complete mediation This principle can be enforced e.g. in Java by using guarded objects [Gon99]. Their use however is not entirely straightforward [Gon98]. We demonstrate how one can ensure proper use of guards to enforce this principle can be enforced with an example from [Jür01c].

Suppose that a certain micropayment signature key may only be used by applets originating at and signed by the site Finance (e.g. to purchase stock rate information on behalf of the user), but this access should only be granted five times a week.

The guard object can be specified as in Figure 2 (where ThisWeek counts the number of accesses in a given week and weeklimit is true if the limit has not been reached yet). One can then prove (informally or using a formal logic such as [ABLP93]) that certain access control requirements are enforced by the guards.

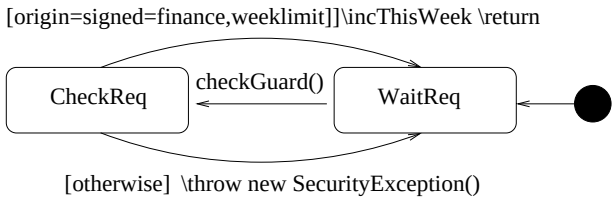

Fig. 2. Statechart MicGd

In this situation, a specification satisfies complete mediation if the access to every object is guarded by a guard object. More feasibly, one can specify a set of sensitive objects and say that a specification satisfies mediation wrt. these objects if they are guarded. One may then give a general policy that defines which access restrictions the guard objects should enforce.

Open design Our approach aims to contribute to the development of a system whose security does not rely on the secrecy of its design.

Separation of privilege As an example for an instance of this principle, one may easily modify the guard in Figure 2 to require signatures from two different principals on the applet requesting access to the guarded object.

In this context, a specification satisfies separation of privilege wrt. a certain privilege p if there are two or more principals whose signature is required to be granted p, at every point of the execution.

More generally, one can formulate such requirements on a more abstract level using UML activity diagrams and verify behavioural specifications (such as the ones given above) wrt. these requirements.

Least privilege Given functionality requirements on a system, a system specification satisfies the principle of least privilege if it satisfies these requirements and if every proper diminishing of privileges of the entities in the system leads to a system that does not satisfy the requirements. This can be formalised within our specification framework and the condition can be checked.

An example application for this rule are the access control rules enforced by firewalls, e.g. considered in [JW01b].

Least common mechanism Since we follow an object-oriented approach, this principle is automatically enforced in so far as data is encapsulated in objects and the sharing of data between different parts of a system is thus well-defined and can be kept at the minimum of what is necessary. Note that on the programming language level there may be further subtleties (e.g. one should not design public fields or variables that can be accessed directly [Gon99]). It is intended to address these in our specification framework in future work.

Psychological acceptability Wrt. the development process, this principle is addressed by our approach in so far as it aims for ease of use in the

development of security-critical systems, and thus for the psychological acceptability of security issues on the side of the developers.

To consider psychological acceptability of security mechanisms on the side of the *users* of the system one may make use of work using UML for performance engineering (e.g. [PK99]) to evaluate acceptability of the performance impact of security mechanisms. One may also modify this approach to measure the user interaction required by such mechanisms (e.g. for authentication).

3 Protocol contexts

It has been suggested (e.g. in [Aba00]) to investigate the way security mechanisms (such as protocols) are employed in the system context, which in practice offers more vulnerabilities than the mechanisms themselves [And01].

As an example, the security of CEPS transactions depends on the fact that in the immediately envisaged scenario (use of the card for purchases in shops) it is not feasible for the attacker to act as a relay between an attacked card (in a modified terminal) and an attacked terminal. However, this is not explicitely stated, and it is furthermore planned to use CEPS over the Internet [CEP01, Bus.Req.], where an attacker could easily act as such a relay (this is investigated in [JW01a]).

Sometimes such assumptions are actually made explicit, if rather informally (such as "R_1 should never leave the LSAM in the clear." [CEP01, Tech. Sp. p.187]).

Being able to formulate precisely the assumptions on the protocol context, to reason about protocol security wrt. them, and to be able to communicate them to developers and clients would thus be useful.

In our UML-based approach, security protocols can be specified using message sequence charts. An example from [JW01a] is given in Figure 3. Assumptions on the underlying physical layer (such as physical security of communication links) can be expressed in implementation diagrams (cf. [Jür01d]), and the behaviour of the system context surrounding the protocol can be stated using statecharts and reasoned about as indicated above.

4 Conclusion

In this position paper, we used the computer security design principles put forward in [SS75] to illustrate our approach towards encapsulating rules

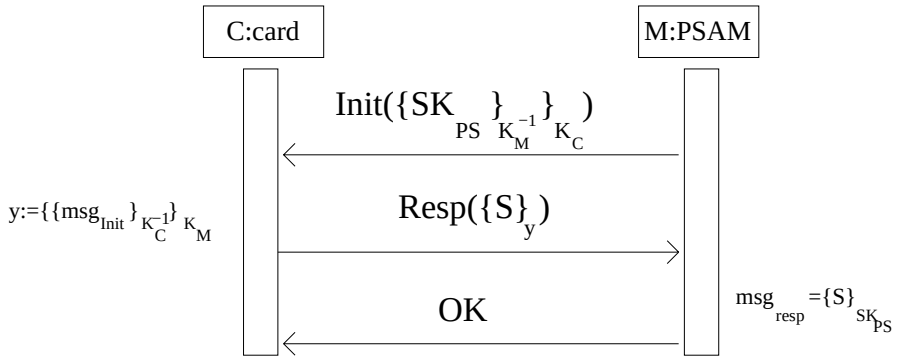

Fig. 3. MSC for CEPS purchase transaction

of prudent security engineering using a formal core of the object-oriented design notation UML (the current industry standard in object-oriented modelling). Using this approach one may evaluate system specifications wrt. these rules, and obtain suggestions for modifications in case of violations. We also mentioned briefly how to use our approach to consider the security of protocols in the system context.

With this work, we aim to provide an "expansive view" of computer security and provide a method for development of large security-critical general-purpose systems, as requested in [SS75].

The research on using UML to develop security-critical systems is still in a very early stage; current work addresses the formal foundations of the approach as well as tool-support and application in case-studies (e.g. in the development of an Internet auction system in an MSc project currently in preparation).

Acknowledgements The idea for this line of work arose when doing security consulting for a project during a research visit with M. Abadi at Bell Labs (Lucent Tech.), Palo Alto, whose hospitality is gratefully acknowledged. It has also benefitted from discussions with D. Gollmann, A. Pfitzmann, B. Pfitzmann and others.

References

[Aba00] M. Abadi. Security protocols and their properties. In F. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*. IOS Press, 2000.

[ABLP93] M. Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[And94] R. Anderson. Why cryptosystems fail. *Communications of the ACM*, 37(11):32–40, November 1994.

[And01] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[CEP01] CEPSCO. Common Electronic Purse Specifications, 2001. Business Requirements vers. 7.0, Functional Requirements vers. 6.3, Technical Specification vers. 2.3, available from http://www.cepsco.com.

[Gol00] Dieter Gollmann. On the verification of cryptographic protocols - a tale of two committees. In *Workshop on Security Architectures and Information Flow*, volume 32 of *Electronical Notes in Theoretical Computer Science*, 2000.

[Gon98] Li Gong. Java^{TM} Security Architecture (JDK1.2). http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html, October 2 1998.

[Gon99] Li Gong. *Inside Java 2 Platform Security – Architecture, API Design, and Implementation*. Addison-Wesley, 1999.

[Huß01] H. Hußmann, editor. *Fundamental Approaches to Software Engineering (FASE/ETAPS, International Conference)*, volume 2029 of *LNCS*. Springer, 2001.

[Jür01a] Jan Jürjens. Developing secure systems with UMLsec — from business processes to implementation. In *VIS 2001*. Vieweg-Verlag, 2001. To appear.

[Jür01b] Jan Jürjens. Modelling audit security for smart-card payment schemes with UMLsec. In P. Paradinas, editor, *IFIP/SEC 2001 – 16th International Conference on Information Security*. Kluwer, 2001.

[Jür01c] Jan Jürjens. Secure Java development with UMLsec. 2001. Submitted.

[Jür01d] Jan Jürjens. Towards development of secure systems using UMLsec. In *[Huß01]*, 2001.

[JW01a] Jan Jürjens and Guido Wimmel. Security modelling for electronic commerce: The Common Electronic Purse Specifications. In *First IFIP conference on e-commerce, e-business, and e-government (I3E)*. Kluwer, 2001.

[JW01b] Jan Jürjens and Guido Wimmel. Specification-based testing of firewalls. In *Andrei Ershov 4th International Conference "Perspectives of System Informatics" (PSI'01)*, LNCS. Springer, 2001. To be published.

[Pau98] L. Paulson. Inductive analysis of the Internet protocol TLS (transcript of discussion). In B. Christianson, B. Crispo, W.S. Harbison, and M. Roe, editors, *Security Protocols – 6th International Workshop*, number 1550 in LNCS, page 13 ff., Cambridge, UK, April 1998.

[PK99] R. Pooley and P. King. The unified modeling language and performance engineering. *IEE Proceedings - Software*, 146(1):2–10, 1999.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[Sch99] F. Schneider, editor. *Trust in Cyberspace*. National Academy Press, 1999.

[SS75] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[Wal00] M. Walker. On the security of 3GPP networks. In *Advances in Cryptology – EUROCRYPT*, volume 1807 of *LNCS*. Springer, 2000.