Orchestrating Security and System Engineering for Evolving Systems

Fabio Massacci[1] , Fabrice Bouquet[2], Elizabeta Fourneret[2], Jan Jurjens[3], Mass S. Lund[4], Sébastien Madelénat[5] , JanTobias Muehlberg[6], Federica Paci[1], Stéphane Paul[5], Frank Piessens[6], Bjornar Solhaug[4], Sven Wenzel[3]

[1]Univ. of Trento, IT, [2]Lab. d'Inform. de Franche-Comté, FR, [3]TU. Dortmund, DE,[4]SINTEF ICT, NO, [5]Thales Research & Tech., FR, , [7]Katholieke Univ. Leuven, BE,
{name.surname}@{disi.unitn.it, lifc.univ-fcomte.fr, cs.tu-dortmund.de, sintef.no, thalesgroup.com, cs.kuleuven.be}

Abstract. How to design a security engineering process that can cope with the dynamic evolution of Future Internet scenarios and the rigidity of existing system engineering processes? The SecureChange approach is to orchestrate (as opposed to integrate) security and system engineering concerns by two types of relations between engineering processes: (i) vertical relations between successive security-related processes; and (ii) horizontal relations between mainstream system engineering processes and concurrent security-related processes. This approach can be extended to cover the complete system/software lifecycle, from early security requirement elicitation to runtime configuration and monitoring, via high-level architecting, detailed design, development, integration and design-time testing. In this paper we illustrate the high-level scientific principles of the approach.

Keywords: We would like to encourage you to list your keywords in this section.

1 Introduction

It is taken for granted that Future Internet scenarios, being them on content, services or things will be characterized by a quick pace of evolution: it should be possible to quickly design new services, or swiftly quickly integrate new devices providing new and interesting contents to the end users. This quick pace of evolution should be supported at all times by maintaining security and trust properties.
This assumption is somehow at odds with strong opposing forces that are currently shaping the engineering process in industry. In order to cope with complexity and quality control the system, software, and service engineering process in industry has been (is, and will likely be) subject to strong push towards rigidity, especially when strong security requirements are at stake. The need to show compliance with standards e.g. ISO 15288 [18] and ISO 12207 [17], respectively for system and

software engineering makes often the engineering process rigid. On one hand stakeholders demand flexibility to accommodate changes; on the other hand they demand compliance to standards in process and product. Process rigidity is further increased when security aspects standards are further taken into account. The use of ISO 2700x, EBIOS, CRAMM, BSIMM or SDLC might be mandated by customers or regulations and the design process must also be compliant with those standards.
For complex systems the engineering process is often supported by artifacts (UML models of the system to be, DOORS format for requirements, UML risk profiles in CORAS etc), and companies tend to adapt and customize these artifacts to fit their needs and application domains (e.g. by using Eclipse GMF), in order to decompose and compartmentalize the work. Some parts of the processes might also be outsourced so that a shared artifact may no longer exist.
In this scenario, integrating security and trust concerns which address simultaneously the calls for fast changes and hard compliance is difficult. While it is widely recognized that security considerations must be considered from the start, most research proposals have focused on new fully integrated security-system engineering processes (eg [34,13,14]). This is the "default solution" in many European Projects: yet another integrated process for security (service, content, things) engineering.
Yet, all integrated processes have significant difficulties in adoption. The main reason behind these difficulties is that security-related activities (e.g. assessment, engineering and assurance [5,30]) must comply with the constraints and pace of the legacy mainstream engineering processes, methods and tools (e.g. [18,17]). The rigidity factors that we have mentioned above makes each step of engineering process highly customized and de facto unchangeable, as the switching cost would be too high. So there is no chance to adopt an entirely new security engineering process that can cope with the dynamicity and evolution challenges of the Future Internet.
So what happens when a security requirement or a threat model changes? Changes must percolate through each step, with its own specific security artifacts and security processes and many errors might be introduced in this endeavor.
Is there another way?

1.1 The Contribution of this paper

The gist of the SecureChange project is to propose a security engineering process that can be deployed in practice without requiring a practically impossible integration of all its parts. Our idea is to deal with evolution and to accommodate proprietary steps by using ***orchestration*** instead of integration [10]. In order to orchestrate the various steps in the process we consider two types of relations: (i) vertical relations between successive security-related processes; and (ii) horizontal relations between mainstream system engineering processes and concurrent security-related processes.

The orchestrated process is based on the *separation of concern principle*. An important advantage of separation of concern is that in-depth expertise in the respective domains is not a prerequisite. The orchestrated process allows the separate domains to leverage on each other without the need of full integration. However consistency of concerns must still ensured. For example security risk managers, requirement managers, and system designers share a minimal set of concepts which is the interface between their own processes: each process is conducted separately and

only when a change affects a concept of the interface, the change is propagated to the other domain following the ideas behind conceptual mappings [4] and relations [15].
In the next section we present the overall Security Engineering process and then discuss the specification (Sec.3) and design (Sec.5) steps of the lifecycle with their interplay with risk assessment (Sec.4). Then we focus on validation (Sec.6) and verification (Sec.7) steps and conclude the paper (Sec.8).

2 The Security and System Engineering Orchestrated Process

A system lifecycle typically has eight phases as illustrated in Figure 1: (i) architecting, (ii) specification, (iii) design, (iv) realisation or acquisition, (v) integration and verification, (vi) validation and deployment, (vii) operation and maintenance, and (viii) disposal. During the evolution process, a system may occupy several of these phases at the same time: earlier specs might be going already through security testing while new requirements might still be at the architectural phase. Security risk management activities can be conducted regardless of the system lifecycle phase although the pursued goals may differ.

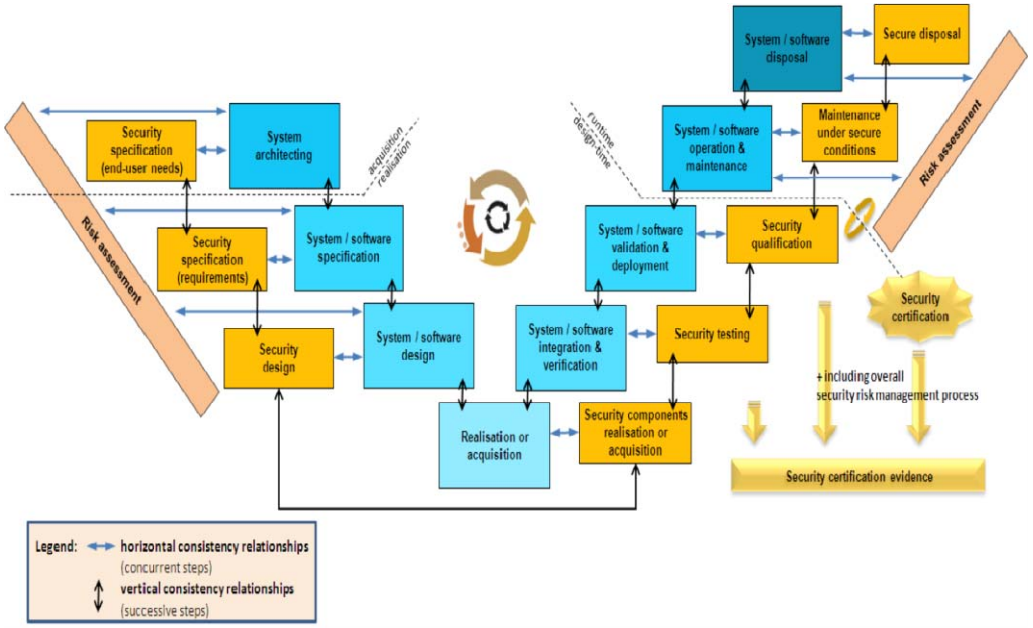

Fig. 1. One Mainstream and Security System/Software Engineering Processes

The first phase is classically performed by the customer, using architectural frameworks (AF) such the NATO AF. It produces end-user requirements as documented in a call-for-tender. During this phase, the main goal of the security activities is to elicit security needs, possibly consolidated by a threat assessment.

The following phases, except operation, are mainly performed by the system provider. During the specification phase, the main goal of the security activities is to define the system requirements, and thus gain early assurance that the proposed architectural solution is sound with respect to security concerns. This step encompasses a high-level risk assessment [24] backing-up the specification of security requirements.

During this phase it is important to be able to quickly update models and bring them in synch [29,1]. The end customer might be involved in the loop and must be able to do some form of what-if scenarios [31]. For example it must be possible to identify possible evolutions and discuss possible tactical solutions in the choice of the components (e.g. the residual risk that a particular component might become useless depending on the outcome of a standardization body).

In contrast, during the design phase, the system degrees of freedom slowly freeze. As time goes by, any major change in design has increasingly significant costs, may require going back to the customer and could lead to unacceptable delays. Changes must be managed differently. The main goal is to make sure that the security properties are preserved across evolution. We accept the change because we know that security properties won't change. Technically this is an obvious observation. We could just re-verify the design after the change and see if the properties still hold. The challenge is to just specify the "delta" and use patterns or stereotype to capture only the "delta" of the change and specify the conditions on the delta that preserve security properties [22]. Proven security design patterns may be used. Security risk assessment is performed in parallel, re-defining security objectives until residual risks are acceptable. Some early validation techniques [23] may be applied in order to gain early assurance that the system design is sound.

The main goal of the security activities during the realisation or acquisition phase is to implement or acquire the countermeasures. In some cases, when the proposed security controls are elementary or available off-the-shelf, this activity may be carried out as part of the mainstream engineering process. When SOA technology is the targeted platform, security-as-a-service might be the right solution [16].

During the integration & verification phase, the main goal of the security-related activities is to integrate and test the countermeasures. As for realisation or acquisition, the integration of the security countermeasures may be carried out as part of the mainstream engineering process; however testing represents a security-specific task, aiming at proving that the information system protects data and maintains functionality as intended.

During the validation/quality check phase, the main goal of the security-related activities is the security qualification of the system, which will potentially lead to certification. The qualification of a product gives evidence of the robustness of the security services of the product. It is based on: (i) the verification of the conformity of the product with the security characteristics specified in the target, on the basis of an evaluation realized by a laboratory approved by a certification authority, e.g. ANSSI in France; (ii) the approval, by the certification authority, of the relevance of the security target with respect to the planned use and the requested level of qualification. This qualification allows: a) to separate the purely technical assessment of the system from a wider assessment of its ability to protect sensitive information in given conditions; b) to recognize that the same system can allow for the protection of

information of different levels, and thus can obtain various levels of approval, according to the conditions of use.
During the operation & maintenance phase, the main goal of security risk management is to monitor the effectiveness of the countermeasures to determine the extent to which the controls are implemented correctly, operating as intended, and producing the desired outcome with respect to meeting the security requirements for the system or enterprise. In case security is found to be flawed, the previous activities may be performed anew to ensure an acceptable level of risk.

3 Secure Specification and Design

This section focuses on the early steps of the system development lifecycle, i.e. security requirement specification, security design and security risk assessment, which are carried out concurrently to system specification and design. We explain here the intuition behind the orchestration aspects (see [10] for details).
The security analysis during the requirement phase is based on an iterative security methodology for evolving requirements (SeCMER for short, see [6] for details). It supports: (i) requirements elicitation, (ii) automatic detection of requirement changes that cause the violation of security properties and (iii) argumentation analysis to check that security properties are preserved and to identify new security properties that should be taken into account.
These steps are present in almost all “traditional” integrated security engineering methodologies (see e.g. [13,14,8]). The distinguishing feature here is that we do not envisage the existence of a single integrated model. Every iteration of the SeCMER process starts with the requirements elicitation phase, which processes a change request as a set of incremental changes on a requirement model. Different aspects of security requirements can be modelled with different state-of-the-art requirement languages such as SI* [27] or Problem Frames (PFs) [31] or use traditional text-based description such as DOORS [7]. An underlying conceptual models maps the notions SI* to the other models such as PFs or other target security domain specific languages [26]. In this way an analyst can use its favourite (or mandatory) model to capture some requirements and use another model for validation purposes. For example, by using the propositions in the requirements model, argumentation analysis [33] determines whether the design has exploitable vulnerabilities that might expose valuable assets to malicious attacks.
For this approach to work we need an automatic orchestration process which uses event-condition-action rules called evolution rules [2,1]. Whenever a change to the requirement model matches some evolution rule(s), the change is automatically detected, and the transformation engine applies specified actions on the requirement model and checks whether the existing security goals are still satisfied after the change. If this is not the case, the change is passed onto the argumentation process, in order to consider whether the security goal can be restored. When even that is not possible, the security goal will be passed back to the elicitation process where the goal will have to be renegotiated and/or reformulated. The same technology can then be used across all vertical and horizontal relations.

4 The Mutual Evolution of Risk and Requirement Analysis

The risk assessment process must be as well adapted as well: a static risk methodology cannot cope with system evolution and the very dynamic process for managing security requirements that we have just discussed. In this section we exemplify the process by using the CORAS approach [24] which is a model-driven approach to risk assessment that is closely based on the ISO 31000 [19] standard. An alternative using Thales's risk assessment modelling language is described in [10].

The CORAS method consists of a risk assessment method, a language for risk modelling, as well as a tool to support the activities of the method. For changing and evolving systems the challenge is that previous security risk assessment results may become invalid and obsolete after the occurrence of a change. SecureChange addresses this challenge by generalising the CORAS approach [24] to the setting of evolving systems [25]. The generalised approach incorporates techniques for tracing changes from the target system to the risk model. For the secure specification and design, this is depicted in Figure 1 by the horizontal relations from risk assessment to the system specification and design, respectively. A change that is proposed for the latter can be traced to affected parts of the risk model that in turn are reassessed and updated. The risk assessment will document how the security risks evolve, and moreover identify options for risk treatment that can be taken into account during specification and design so as to keep the security risks acceptable under change. Moreover, risks that change should explicitly be analysed and documented as such.

A key principle is that only the parts of the risk picture that are affected by system changes should be analysed anew. The same principles of orchestration and separation of concerns are applied in this setting. The key concepts in the requirements models (loosely speaking the requirements APIs) are mapped into the concepts of the risk assessment model (the risk's APIs). Then the evolution rules orchestrate the process by keeping the models in synch. For example, a change request during the specification phase may lead to changes in the requirement model, such as the identification of new assets. As part of the interface between the requirement and risk model, this model artefact can be passed to the security risk assessment. By separation of concern, the risk assessment proceeds with this change separately until eventually the interface is invoked again and treatments are passed back and included in the requirement model as security goals.

5 Managing Evolution During the Design Step

As we have already mentioned the analysis of changes during the design process must change focus and try to identify as much as possible the changes that preserve the security properties established during the earlier phases.

In the SecureChange project we have used the UMLsec profile [23] to perform the security design and its early validation. The UMLsec profile is based on the mainstream software engineering modelling language UML and defines stereotypes which are used together with tags to formulate the security requirements and assumptions of a system at design time. These corresponds to the links in the

horizontal relations in Figure 1 between the mainstream system engineering process and the security engineering process.
The profile has been extended to the UMLseCh profile for specifying one or more evolutions on a model [22]. The stereotypes «add», «del», and «substitute» can be used to precisely define which model elements are to be added, deleted, and substituted in a model, and together with constraints in first-order predicate logic that allow to coordinate and define more than one evolution path. 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.
Based on the UMLseCh stereotypes used in the diagrams we can compute all possible "delta"s of a model and check whether an evolution of the design will preserve the security properties of the system. Besides the ability to analyse the security properties of all possible evolutions, we have an efficiency gain, since the analysis of the delta performs better than the re-evaluation of the complete design models.
Once the security properties have been confirmed to hold we can use the model to propagate the security-preserving changes to the later phases, thus moving the process along the vertical relations in Figure 1. During the realization phase one can use the code generation tools generally available for UML. In the testing phase we can use model-based testing tools such as the one presented later in this paper. A further possibility, once we have the horizontal traceability link between security properties and system properties is to generate security monitors from the UMLsec specification to monitor the secure execution of the system.

6 Security Testing of Evolving Systems

With the overall goal to provide means of security verification and validation for evolving systems, Secure Change has a strong focus on verification techniques and test generation that can be applied to implementation artefacts during the development and deployment phases of the software life-cycle.
Following the vertical relations in Figure 1 on the design models one can automatically generate the test models [11] where the traceability to the security requirements (the horizontal relations) is preserved.
We can classify tests [12] with respect to requirements evolution: only the first time ever all tests are *new*; as the system evolves and requirements and design models are orchestrated together we can progressively divide tests in eight different groups: *new*, *un-impacted* (by evolution), *re-executed* (i.e. impacted by evolution but the test values are not changed), *outdated* (i.e. deleted by evolution), *failed* (i.e. animation of test failed, e.g. due to a modification of the system behaviour), *updated* (i.e. same as previous test, but oracle values changed), *adapted* (i.e. new version of test to cover a previous requirement) and *removed* (by the user).
These eight status of the test are used to structure the test repository into four sets of test suites: *evolution* (contains new, adapted and updated tests exercising the novelties of the system), *regression* (contains un-impacted and re-executed tests exercising the unmodified parts of the system), *stagnation* (contains outdated and failed tests, which are invalid w.r.t. the current version of the system), *deletion* (contains removed tests).

The advantage of our approach is to keep track of test cases through evolution. This allows a better stability and transparency of the test suite which is an important industrial criterion and is a key factor to help the validation team to prioritise test execution. From the test models we can now use an efficient test generation process that takes into account changes in requirements and previous test status.

7 Security Verification of Evolving Systems

In many security critical applications it is also important to achieve stronger guarantees than those achievable with testing. For example embedded Java applet codes for smart card technology supporting identity, banking or health services must be certified with respect to Common Criteria security certification on assurance levels EAL4+ and EAL7. This means formally proving that the embedded software on the smart card device ensures a set of properties related to confidentiality, integrity, availability and authenticity of its assets.

These formal verification steps must be deployed during the realisation and verification phases of the system engineering process illustrated in Figure 1 and therefore must be able to implement automatic rely-guarantee reasoning for programs written in Java and C (as opposed to formal assurance about model specifications used in Section 5). For example, in SecureChange we used the VeriFast prover [21] to verify the absence of memory safety errors, data races, and to prove functional correctness. Applying VeriFast requires the source code of the program under verification to be annotated with method contracts in terms of separation logic [35]. The key strengths of VeriFast are its support for specifying and verifying deep data structure properties, reasoning about concurrent programs and its predictably short verification times, thereby providing immediate feedback to developers and testers.

In order to support evolution a key aspect of verification technologies is to explore possibilities to reuse verification artefacts, such as code annotations, to leverage re-certification of the card when change occurs. In the scenario considered in the project two non-trivial Java Card applets have been annotated and verified with respect to the absence of transaction errors, out-of-bounds operations, invalid casts and null-pointer de-references [29]. Furthermore, the full functional correctness has been proven for one applet. Using VeriFast, we were able to identify a total of thirteen bugs in one of the applets, with a relatively low annotation overhead of about one line of annotations per three lines of code. Ongoing work on VeriFast focuses on further reducing the annotation overhead [29,35] and providing support for highly concurrent usage of data structures [20].

Run-time verification techniques can then be used at load-time to complement the assurance guarantees obtained by development time verification techniques. For example one can use the Security-by-Contract approach to check that the updates of applets on the platforms maintain it security properties [8]

Conclusions

In this paper, we have proposed an overview of model-based security engineering activities performed in symbiosis with mainstream system engineering activities. The baseline approach ensures the consistency of the different models, and allows for change management throughout the complete system lifecycle. The key idea of the project is to automatically orchestrate the changes by providing suitable mapping and traceability link across different artefacts and processes. These features should make industrial adoption easier.
The project's approach has been validated on a large IT system, namely an Air Traffic Management (ATM) in the setting of the introduction of the Arrival Manager (AMAN) for traffic management while the later phases have been validated by considering the evolution of the smart-card systems based on GlobalPlatform. The application to a Service Oriented Architecture has also being validated in the scenario of the security of a multimedia home gateway and this is reported elsewhere [16, 3].

Acknowledgments. This work has been partly funded by the EU FP7 FET IP Secure Change project. We would like to thank other members of the SecureChange consortium and notably G. Bergmann, R. Breu, F. Innerhofer-Oberperfler, A. Tedeschi, and T. T. Thun for many useful suggestions.

References

1. G. Bergmann, et al.: "Change-Driven Model Transformations. Change (in) the Rule to Rule the Change." Software and System Modeling (2011), to appear.
2. G. Bergmann, et al.: "Incremental evaluation of model queries over EMF models." In: Model Driven Engineering Languages and Systems, In Proc. of MODELS'10. 2010.
3. M. Breu, R. Breu, S. Löw: "Living on the MoVE: Towards an Architecture for a Living Models Infrastructure." International Journal on Advances in Software, pp. 290-295, 2010
4. M. Chechik, et. al.: "Relationship-based change propagation: A case study." in Proc, of the ICSE Workshop on Modeling in Software Engineering (MISE '09). pp. 7–12. 2009. IEEE.
5. B. De Win, et. Al: "On the secure software development process: CLASP, SDL and Touchpoints compared", Information and software technology 51(7): 1152-1171, 2009.
6. Deliverable 3.2 "A Methodology for Evolutionary Requirements." Available at www.securechange.eu.
7. DOORS. http://www-01.ibm.com/software/awdtools/doors/.
8. N. Dragoni et al.: "A Load Time Policy Checker for Open Multi-Application Smart Cards". In Proc. of IEEE Policy 2011. IEEE.
9. G. Elahi, E. Yu, and N. Zannone: "A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities," Requirements Engineering 15:41–62, 2010.
10. E. Félix, Deland, F. Massacci, F. Paci: "Managing Changes with Legacy Security Engineering Processes". In Proc. of IEEE Int. Conf. on Intelligence and Security Informatics, 2011.
11. E. Fourneret, et al. "Selective Test Generation Method for Evolving Critical Systems". In Proc. of 1st Int. Workshop on Regression Testing, 2011. IEEE.

12. E. Fourneret, et. al. “Model-Based Security Verification and Testing for Smart-cards.” In Proc. of ARES’11, 2011. IEEE.
13. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone: “Requirements engineering for trust management: model, methodology, and reasoning,” Internat. Journal of Information Security 5(4):257–274, 2006.
14. C. Haley, R. Laney, J. Moffett, and B. Nuseibeh: “Security requirements engineering: A framework for representation and analysis.” IEEE Trans. Softw. Eng 34:133–153, 2008.
15. J. Hassine, J. Rilling, and J. Hewitt, “Change impact analysis for requirement evolution using use case maps,” in Proc. of the 8th Intl. Workshop on Principles of Software Evolution, pp. 81–90, 2005. IEEE.
16. F. Innerhofer-Oberperfler, M. Hafner, R. Breu: “Living Security – Collaborative Security Management in a Changing World.” In Proc. of IASTED Int. Conf. on Soft. Eng., 2011.
17. ISO 12207, Systems and software engineering — Software life cycle processes, ISO, 2008
18. ISO 15288, Systems and software engineering — System life cycle processes, ISO, 2008
19. ISO 31000, Risk management – Principles and guidelines, ISO, 2009
20. B. Jacobs and F. Piessens: “Expressive modular fine-grained concurrency specification.” In Proc. of POPL’11, pp. 271--282, 2011. ACM.
21. B. Jacobs, J. Smans, and F. Piessens. “A quick tour of the VeriFast program verifier.” In Proc. of APLAS’10, pp. 304--311, LNCS 6461, 2010. Springer.
22. J. Jürjens, L. Marchal, M. Ochoa, and H. Schmidt: “Incremental Security Verification for Evolving UMLsec models.” In Proc. of ECMFA, LNCS. 2011.
23. J. Jürjens. “Secure Systems Development with UML.” Springer, 2005.
24. M. S. Lund, B. Solhaug, K. Stølen: “Model-Driven Risk Analysis – The CORAS Approach.” Springer, 2011
25. M. Soldal Lund, B. Solhaug, K. Stølen: “Risk Analysis of Changing and Evolving Systems using CORAS.” In Foundations of Security Analysis and Design IV, LNCS.
26. F. Massacci, J. Mylopolous, F. Paci, T. T. Tun and Y. Yu. “An Extended Ontology for Security Requirements.” In Proc. of the 1st Internat. Workshop on Information Systems Security Engineering (WISSE’11). London 2011.
27. F. Massacci, J. Mylopoulos, N. Zannone: “Computer-aided support for Secure Tropos.” Automated Software Eng. 14:341-364, 2007.
28. V. Normand, E. Félix, “Toward model-based security engineering: developing a security analysis DSML”, In Proc. of ECMDA-FA, 2009
29. P. Philippaerts, et. al.. The Belgian Electronic Identity Card: A Verification Case Study. In Proc. AVOCS 2011, submitted.
30. System Security Eng. Capability Maturity Model, http://www.sse-cmm.org/index.html
31. M. S. Tran, F. Massacci: “Dealing with Known Unknowns: Towards a Game-Theoretic Foundation for Software Requirement Evolution.” In Proc. of CAiSE 2011. LNCS. Springer.
32. T.T. Tun, et al.: “Early identification of problem interactions: A tool-supported approach.” In Proc. of REFSQ’09, pp. 74-88. LNCS 5512 2009.
33. T.T. Tun, et al.: “Model-based argument analysis for evolving security requirements.” In Proc. of the IEEE SSIRI’10. pp. 88-97, 2010. IEEE.
34. A. van Lamsweerde, “Elaborating security requirements by construction of intentional anti-models,” in Proc. of ICSE’04. pp. 148–157. ACM.
35. F. Vogels, B. Jacobs, F. Piessens, and J. Smans. “Annotation inference for separation logic based verifiers.” In Proc. of FMOODS’11, pp. 319—333, LNCS 6722, 2011. Springer.