Software Engineering for Secure Systems

Danilo Bruschi*, Bart De Win**, Mattia Monga*

*Università degli Studi di Milano, Milan, Italy
**Katholieke Universiteit Leuven, Leuven, Belgium
bruschi@dico.unimi.it,bart.dewin@cs.kuleuven.ac.be, monga@ðico.unimi.it

A workshop summary

The SESS'05 held in St. Louis, MO on May 15-16 was intended to be a venue to discuss techniques for building and validating secure applications. Workshop attendees (about 40 people) came both from the software engineering and the security community, raising a fruitful discussion and exchange of ideas and problem perspectives.

Speaker presentations were organized in three sessions.

The first session addressed *security at the architectural level* of applications. Demir et~al. [1] proposed an aspect oriented architectural description language to handle security features in distributed systems. Rits et~al. [2] described their approach for inspecting (in an aspect oriented fashion) access control policies in multi-layer applications. Banerjee et~al.[3] related trustworthiness of complex applications to their architectural parts: components, connectors, and configurations. Ren et~al.[4] suggested that connectors can be used to enforce many security policies at design time. De~Win et~al.[5] discussed design by contract when applied to security concerns. Verhanneman et~al.[6] focused on requirement traceability in the context of access control for distributed systems.

The second session tackled the problem of *evaluating trust, risk, and security properties from high level descriptions of a system*. Lee et~al.[7] presented a structured and systematic approach to certification and accreditation in the context of Department of Defense's projects. Dwaikat and Parisi-Presicce[8] proposed a quantitative risk assessment based on architectural information. Mead and Stehney[9] described a methodology for elicitation and risk assessment of security requirements. Sohr et~al.[10] showed the advantages of using OCL for expressing authorization policies at design level.

In the third session a number of *software engineering techniques that may improve the security of applications* were presented. Breech and Pollock [11] described a framework for testing language-based protection mechanisms. Weber et~al.[12] classified software flaws according to the nature of the defect in the source code. Masri and Podgurski[13] detailed the use of dynamic flow analysis for checking security policies. Jochen et~al.[14] discussed the challenge of trusting polymorphic components. Gegick and Williams[15] proposed a pattern language for describing vulnerabilities.

Lesson learned

A significant part of the discussions focussed on the differences (if any) between secure software engineering and plain secure engineering. Quite obviously secure software engineering implies secure engineering and software engineers should learn to use the language and the conceptual tools of security and safety specialists. However the software part of complex systems is often the hardest to secure, due to its intrinsic complexity. Software engineers should exploit their understanding of software to manage this complexity taking into account security and safety concerns. Security requirements should be elicited correctly, traced, implemented and validated, as usual. However, they inherently cross-cut different abstraction levels and they are exceptionally difficult to validate because they are often expressed as negative properties. As a result, new approaches to testing security mechanisms are needed. Moreover, security issues encompass components, connectors, and configurations, thus security contracts among services should account for them all. Enforcing security contracts and policies requires (or at least is greatly simplified by) run-time monitoring, that should be part of most of secure applications. In any case, using a software system implies a risk, but it has also a value; however, both of them are extremely difficult to quantify and compare, since they are domain- and stakeholder-sensitive. Finally, raising the return on investment of the security effort is crucial for many environments to adopt and install secure software engineering methods.

References

[1] O. E. Demir, P. Devanbu, N. Medvidovic, and E. Wohlstadter, "DISCOA: architectural adaptions for security and QoS,

[2] M. Rits, B. D. Boe, and A. Schaad, "XacT: a bridge between resource management and access control in multilayered applications"

[3] S. Banerjee, C. A. Mattmann, N. Medvidovic, and L. Golubchik, "Leveraging architectural models to inject trust into software systems"

[4] J. Ren, R. Taylor, P. Dourish, and D. Redmiles, "Towards an architectural treatment of software security: A connector-centric approach"

[5] B. D. Win, F. Piessens, J. Smans, and W. Joosen, "Towards a unifying view on security contracts"

[6] T. Verhanneman, F. Piessens, B. D. Win, and W. Joosen, "Requirements traceability to support evolution of access control"

[7] S.-W. Lee, R. Gandhi, and G.-J. Ahn, "Establishing trustworthiness in services of the critical infrastructure through certication and accreditation"

[8] Z. Dwaikat and F. Parisi-Presicce, "Risky trust: Risk-based analysis of software systems"

[9] N. R. Mead and T. Stehney, "Security quality requirements engineering (SQUARE) methodology"

[10] K. Sohr, L. Migge, and G.-J. Ahn, "Articulating and enforcing authorisation policies with UML and OCL"

[11] B. Breech and L. Pollock, "A framework for testing security mechanisms for program-based attacks"

[12] S. Weber, P. Karger, and A. Paradkar, "A software flaw taxonomy: Aiming tools at security"

[13] W. Masri and A. Podgurski, ¡ÈUsing dynamic information flow analysis to detect attacks against applications"

[14] M. Jochen, A. A. Anteneh, L. Pollock, and L. Marvel, "Enabling control over adaptive program transformation for dynamically evolving mobile software validation"

[15] M. Gegick and L. Williams, "Matching attack patterns to security vulnerabilities in software-intensive system designs"