Tools for Model-based Security Engineering: Models vs. Code[*]

Jan Jürjens and Yijun Yu
Computing Department, The Open University, GB
http://www.computing.open.ac.uk/people/{j.jurjens,y.yu}

ABSTRACT

We present tools to support model-based security engineering on both the model and the code level. In the approach supported by these tools, one firstly specifies the security-critical part of the system (e.g. a crypto protocol) using the UML security extension UMLsec. The models are automatically verified for security properties using automated theorem provers. These are implemented within a framework that supports implementing verification routines, based on XMI output of the diagrams from UML CASE tools. Advanced users can use this open-source framework to implement verification routines for the constraints of self-defined security requirements.

In a second step, one verifies that security-critical parts of the model are correctly implemented in the code (which might be a legacy implementation), and applies security hardening transformations where is that not the case. This is supported by tools that (1) establish traceability through refactoring scripts and (2) modularize security hardening advices through aspect-oriented programming. The proposed method has been applied to an open-source implementation of a cryptographic protocol implementation (JESSIE) in Java to build up traceability mappings and security aspects. In that application, we found a security weakness which could be fixed using our approach. The resulting refactoring scripts and security aspects have found reusability in the Java Secure Socket Extension (JSSE) library.

Categories and Subject Descriptors: D.2.2 Software Engineering: Design Tools and Techniques -Computer Aided Software Engineering (CASE), D.2.4 Software Engineering: Software/Program Verification

General Terms: Security.

Keywords: Security, Model-based Software Engineering, UML, Verification Framework, Code Analysis, Refactoring, Security Hardening.

Model-based Security Engineering

Understanding the security goals provided by software making use of cryptography is one of the major challenges with security-critical systems. Any support to aid secure systems development is thus dearly needed. Towards this goal, the security extension UMLsec for the Unified Modeling Language (UML) [3] allows us to include security requirements as stereotypes with logical constraints. In this paper we present automated tool-support for the analysis of UMLsec models against security requirements by checking the constraints associated with the UMLsec stereotypes. Besides presenting a general, extensible framework for implementing verification routines for the constraints associated with security-critical UML stereotypes, we focus on a plug-in that utilizes an automated theorem-prover (ATP) for first-order logic (FOL) to verify security properties of UMLsec models which make use of cryptography (such as cryptographic protocols), which was explained in [5]. To do so, the analysis routine extracts information from behavioral UML diagrams that may contain additional specific cryptography-related information. If the analysis reveals that there is an attack, an attack generation script written in Prolog generates the attack trace.

To link the model-based security analysis to the code level, we have shown how to insert cryptographic assertions into implementations of cryptographic protocols to ensure that the analyzed cryptographical protocol design is implemented securely [6]. To our experience, it is however non-trivial to insert the right assertions at the right place in the program. As the implementation or the used libraries evolve, the instrumentation may not anymore guarantee the correct link to the protocol design. Moreover, it is not clear whether and how such assertions can be reliably transferred to a different implementation of the protocol. As such assertions tend to crosscut in the code, a good candidate to instrument the program is aspect-oriented programming (AOP). However, aspectJ cannot intercept arbitrary control flow at the statement level (and does not intend to do so either, for good reasons). Our tools therefore maintain traceability between the design and the implementation of a cryptographic protocol through techniques and tools for refactoring scripting and aspect-oriented programming that are supported respectively by the JDT[1] and AJDT[2] in the Eclipse Integrated Development Environment (IDE). Other Java IDE's (e.g. Netbeans) are applicable as long as both refactoring scripting and aspectJ are supported.

Note that our goal is not to provide an automated full formal verification of Java code but to increase understanding of the security properties enforced by cryptoprotocol imple-

[*] This work was partially supported by the Royal Society within the project Modelbased Formal Security Analysis of Crypto Protocol Implementations.

[1] www.eclipse.org/jdt

[2] www.eclipse.org/ajdt

ASE'07, November 4–9, 2007, Atlanta, Georgia, USA.
ACM 978-1-59593-882-4/07/0011.

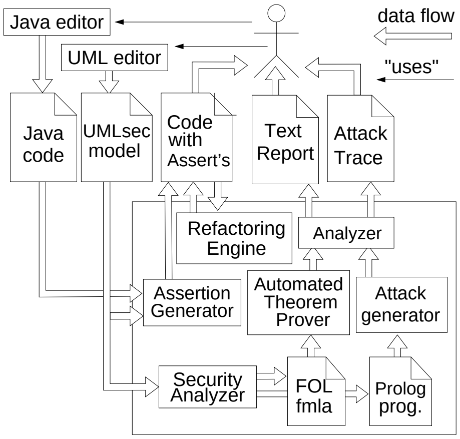

Figure 1: Tool-flow of the MBSE suite

mentations (which may be legacy implementations) in a way as automated as possible. Because of the abstractions, the approach may produce false alarms (which however have not surfaced yet in practical examples). Note that our focus here is on high-level security properties such as secrecy and authenticity, and not on detecting low-level security flaws such as buffer-overflow attacks.

The tools are accessible through a web-interface and available as open-source. They have been validated in several industrial projects (see for example [4, 1]), identifying several major security flaws in software during its industrial development.

Model-based Security Verification

The usage of the framework as illustrated in Fig. 1 proceeds as follows. The developer creates a model and stores it in the UML 1.5/XMI 1.2 file format. The file is imported by the verification framework into the internal MDR repository. MDR is an XMI-specific data-binding library which directly provides a representation of an XMI file on the abstraction level of a UML model through Java interfaces (JMI). This allows the developer to operate directly with UML concepts, such as classes, statecharts, and stereotypes. It is part of the Netbeans project. Each plug-in accesses the model through the JMI interfaces generated by the MDR library, they may receive additional textual input, and they may return both a UML model and textual output. There are two kinds of model analysis plug-ins: The static checkers parse the model, verify its static features, and deliver the results to the error analyzer. The dynamic checkers translate the relevant fragments of the UML model into the input language for example of an ATP. The ATP is spawned by the framework as an external process; its results are delivered back to the error analyzer. The error analyzer uses the information received from the static and dynamic checkers to produce a text report for the developer describing the problems found, and a modified UML model, where the errors found are visualized. Besides the automated theorem prover binding presented in this paper there are other analysis plugins including a model-checker binding and plugins for simulation and test-sequence generation. The developer can then use the aspect weaver to weave in security aspects on the model or into the code that can be generated. The resulting code can then again be analzed for security requirements.

The framework is designed to be extensible: advanced users can define stereotypes, tags, and first-order logic constraints which are then automatically translated to the automated theorem prover for verification on a given UML model. Similarly, new adversary models can be defined.

In particular, the automated translation of UMLsec diagrams to first-order logic (FOL) formulas which allows automated analysis of the diagrams using ATPs for FOL is explained in [5]. In case the result is that there may be an attack, in order to fix the flaw in the code, it would be helpful to retrieve the attack trace. Since theorem provers such as e-SETHEO are highly optimized for performance by using abstract derivations, it is not trivial to extract this information. Therefore, we also implemented a tool which transforms the logical formulas explained above to Prolog. While the analysis in Prolog is not useful to establish whether there is an attack in the first place (because it is in order of magnitudes slower that using e-SETHEO and in general there are termination problems with its depth-first search algorithm), Prolog works fine in the case where one already knows that there is an attack, and it only needs to be shown explicitly (because it explicitly assigned values to variables durch its search, which can then be queried).

The user webinterface and the source code of the verification framework is accessible at [2].

Traceable Security Hardening

Software refactoring actions are changes to the internal structure of the software without changing external behavior. For our tools to (1) establish traceability through refactoring scripts and (2) modularize security hardening advices through aspect-oriented programming, we use scripts to automatically repeat refactoring actions found during program analysis such that these actions become semantics-preserving program transformations aiming at improving the program understanding. Specifically, two virtualization mappings are maintainable by our refactoring scripts. The first mapping is from symbols in the designed message sequence chart to program entities. The second mapping is from program entities to the join point entities that are crosscut by aspect advices. Both mappings, when applied reversely, can trace program entities accurately back to the message sequence charts such that one can reflect implementation changes back to the design. The aim is thus to automate the maintenance of traceability mappings created between the cryptographic protocol in design and their open-source implementations.

1. REFERENCES

[1] B. Best, J. Jürjens, and B. Nuseibeh. Model-based security engineering of distributed information systems using UMLsec. In *ICSE 2007*, pages 581–590. ACM, 2007.

[2] J. Jürjens. UMLsec webpage, 2002-07. Accessible at http://www.umlsec.org.

[3] J. Jürjens. *Secure Systems Development with UML*. Springer, 2004.

[4] J. Jürjens. Code security analysis of a biometric authentication system using automated theorem provers. In *21st Annual Computer Security Applications Conference (ACSAC 2005)*. IEEE, 2005.

[5] J. Jürjens. Sound methods and effective tools for model-based security engineering with UML. In *27th International Conference on Software Engineering (ICSE 2005)*. IEEE, 2005.

[6] J. Jürjens. Security analysis of crypto-based Java programs using automated theorem provers. In S. Easterbrook and S. Uchitel, editors, *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006)*. ACM, 2006.