Model-based Security Engineering for Compliance with Regulatory and Business Requirements

Jan Jürjens

Department of Computing
The Open University, GB

J.Jurjens@open.ac.uk

http://www.umlsec.org
Challenge: Security

Security is holistic property:
- Attackers often circumvent (not: break) mechanisms.
- Transform (in)secure components to secure systems?

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (B. Lampson / R. Needham).
Model-based Security Engineering

Ideas: Extract models from artefacts in development and use of software.

- Tool-supported, theoretically sound, efficient automated security design & analysis.
Secure System Lifecycle

Security requirements
Requirements and use cases
Abuse cases

Risk analysis
Design

External review
Test plans

Risk-based Security tests

Static analysis (tools)
Code

Penetration testing
Test results

Risk analysis

Security breaks

Field feedback

Model-based Security Engineering

Note: emphasis on high-level requirements.

[McGraw 2003]
UMLsec

Insert recurring security requirements, adversary scenarios, security mechanisms as predefined markers. Verify associated logical constraints using model checkers and ATPs (based on formal semantics). Ensures that UML specification enforces relevant security requirements wrt Dolev-Yao type adversaries. [FASE01, UML02, FOSAD05, ICSE05]
**Example: Crypto-based Distributed System**

Adversary knowledge:

\[ k^{-1}, y, x, \{z\}_k, z \]

(return\(\{y::x\}_z\))

\[-\]

Return\(\{z\}_k\)

[arg\(b,1,1 = x\)]

\[-\]

Attacker may ...

- control system parts,
- know data in advance,
- intercept messages,
- delete messages,
- inject messages.

(cf. [Dolev, Yao 1982])
Security Analysis in First-order Logic

Approximate adversary knowledge set from above:

Predicate $knows(E)$ meaning that adversary may get to know $E$ during the execution of the system.

E.g. secrecy requirement:
For any secret $s$, check whether can derive $knows(s)$ from model-generated formulas using automatic theorem prover.  

[ICSE05]
Analysis

Check whether can derive \textit{knows(s)} e.g. using e-Setheo.

Surprise: Yes!

→ Protocol does not preserve secrecy of \textit{s}.

Why? Use Prolog-based attack generator.
Security Analysis: Model or Code?

Model:
+ earlier (less expensive to fix flaws)
+ more abstract \(\Rightarrow\) more efficient
- more abstract \(\Rightarrow\) may miss attacks
- programmers may introduce security flaws
- even code generators, if not formally verified

Code:
+ „the real thing“ (which is executed)

\(\Rightarrow\) Do both!

Surprise: Essentially no existing work (eg for crypto prots)!
Model vs. Implementation

Sent and received data

Implement-ation

Elements of connections

Backtrace assignments

Compare meaning!

Defined during model creation

Equal?

Find

Has

Jan Jürjens, Open University: Moc
Example: Interface spec of SSL

I) Identify program points:
   value (r), receive (p), guard (g), send (q)

II) Check guards enforced
Checking Guards

Guard $g$ enforced by code?

b) Generate runtime check for $g$ at $q$ from diagram: simple + effective, but performance penalty.

c) Testing against checks (symbolic crypto for inequalities). [ICFEM02]

d) Automated formal local verification: conditionals between $p$ and $q$ logically imply $g$ (using ATP for FOL). [ASE06]
msg = Handshake.read(din, certType);

try {
    session.trustManager.checkServerTrusted(peerCerts, suite.getAuthType());
} catch (Exception e) {
    // only possible way without throwing exception
}

msg = new Handshake(Handshake.Type.CLIENT_KEY_EXCHANGE, ckex);
msg.write(dout, version);
Java editor

UML editor

Java code

UMLsec model

Code with Assert's; Tests

Text Report

Attack Trace

Control Flow Graph

Assertion/Test Generator

Analyzer

Local Code Checker

Automated Theorem Prover

Attack generator

Security Analyzer

FOL fmla

Prolog prog.

[UML04, FASE05, ICSE06]
Some Applications

Analyzed designs / implementations / configurations for

- biometry, smart-card or RFID based identification
- authentication (crypto protocols)
- authorization (user permissions, e.g. SAP systems)

Analyzed security policies, e.g. for privacy regulations.

[ACSAC05, ICSE07]
Overview

Security Engineering

- Architectures
- Patterns
- Services
- Components
- Layers
- Models
- Code
- UMLsec
  - Formal Semantics
  - Analysis Framework
  - Model Checking
  - Autom. Theorem Proving
  - Refinement
- Runtime Checks
- Model-based Testing
- Autom. Theorem Proving

Foundations

- Requirements
- Processes

Applications

- CEPS
- Biometry
- Cryptokey
- Jessie
- TLS variant
- Wiesncard

IT Security

- Fault-tolerance
- Reliability

Security

- Engineering

Management

- Risk Assessment
- Permissions
- Business Processes
- Security Investment
- Firewall Configurations

Dependable Systems Development

- Real-time