Model-based Security Engineering for Evolving Systems

Jan Jürjens

Open University (GB)
Microsoft Research (Cambridge)
Robinson College (Univ. Cambridge)

http://www.jurjens.de/jan
Security: Some Problems

„Blind“ use of security mechanisms:
• Security usually compromised by circumventing (rather than breaking) them.
• Assumptions on system context, physical environment.
• Attacker may use unintended/unnoticed functionality

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (R. Needham).
Architectural Layers
Critical System Lifecycle

- Critical requirements
- Abuse cases
- Risk analysis
- Requirements and use cases
- External review
- Risk-based tests
- Design
- Static analysis (tools)
- Test plans
- Code
- Risk analysis
- Risk-based tests
- Test results
- Static analysis (tools)
- System Monitoring
- System breaks
- Field feedback

Model-based Security Engineering

[McGraw 2003]
Model-based Security Engineering

Requirements

(UML) Models

Source Code

Configurations

Runtime System

Weave in
Analyze against
Code-/Testgen.
Rev.Eng. Verify
Configure
Verify
Monitor
Execute
Configure

Evolution
Model-based Security with UMLsec

Extension of the Unified Modeling Language (UML) for secure systems development.

- evaluate UML models for security
- encapsulate established rules of prudent secure engineering
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- can use in certification
Insert recurring security requirements, adversary scenarios, security mechanisms as predefined markers. Use associated logical constraints to verify specifications using model checkers and ATPs based on formal semantics. Ensures that UML specification enforces the relevant security requirements wrt Dolev-Yao type adversaries.  

[FASE01, UML02, FOSAD05, ICSE05]
What Does UMLsec Cover?

Security requirements: <<secrecy>>, ...

Threat scenarios: Use $\text{Threats}_{\text{adv}}(\text{ster})$.

Security concepts: For example <<smart card>>.

Security mechanisms: E.g. <<guarded access>>.

Security primitives: Encryption built in.

Physical security: Given in deployment diagrams.

Security management: Use activity diagrams.

Technology specific: Java, CORBA security.
Refinement, Composability, Aspects, Services

Need to refine models down to code. Common formalizations of security properties not preserved by refinement. Bad: re-verify after each step (incl code).

**Theorem:** Our notion of model refinement preserves security requirements.  

Similar: Established composability for certain security requirements under suitable assumptions.  

Also: Demonstrated how to apply security using aspect-oriented weaving / service orientation.
Layered Security Architectures

System layer on top uses security services below.

- confidentiality, integrity, server authenticity
- client authenticity

\[ \text{confidentiality, integrity, server authenticity} + \text{client authenticity} = \text{?} \]

Security properties \textbf{additive} ?

Theorem: \textbf{Yes}, under suitable conditions.

[Safecomp03]
Jan Jürjens: Models for Secure Change

Model Verification

\[ \forall arg_1, \ldots, arg_n . \ (knows(arg_1) \land \ldots \land knows(arg_n) \land cond(arg_1, \ldots, arg_n) \Rightarrow knows(exp(arg_1, \ldots, arg_n))) \]

... ()

\begin{align*}
& \text{knows(ArgC}_3) \\
& \& \text{equal(fst(ArgC}_3, \text{type_serverkeyexchange}) \}
& \& \text{equal(snd(ext(snd(snd(ArgC}_3), k\_ca)), skey)) \}
& \& \text{equal(snd(ext(snd(ArgC}_2), k\_ca)), fst(snd(ArgC}_3))}
\end{align*}

\[ \Rightarrow (\text{(knows(ArgC}_4_1) \& \text{equal(ArgC}_4_1, \text{type_serverhello_done})) \]

\[ \Rightarrow (\text{(true \& equal(ClientKeyExchange, enc(premasterkey, skey))}) \]

\[ \$ \text{------------------- Conjecture -- } \]

input_formula(attack, conjecture, (\text{knows(mastersecret)}) )\].

[FASE05, ICSE05, ICSE06]

analyzing results ...

model found/total failure
time limit information: 19 total / 18 strategy (leaving wrapper).
task myUML_PID1491 on atbroy1 has status SUCCESS (model found by strategy 300) consuming 1 seconds deleting temporary files.
e-SETHEO done. exiting
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 where feasible!
Model vs. Implementation

- "meaning"
- "meaning"

compare meaning!

Sent and received data

Backtrace assignments

Elements of connections

Defined during model creation

Implement -ation

Sent and received data

Find

Equal?

Has

Implement -ation

Elements of connections

Jessie – using RSA & Server authentication

C: Client

S: Server

[Abstract model]

Jan Jürjens: Models for Secure Change
Interface spec of SSL

- Identify program points:
  - value \((r)\)
  - receive \((p)\)
  - guard \((g)\)
  - send \((q)\)

II) Check guards enforced
Checking Guards

Guard $g$ enforced by code?

- Generate runtime check for $g$ at $q$ from diagram: simple + effective, but performance penalty.
- Testing against checks (symbolic crypto for inequalities).
- Automated formal local verification: conditionals between $p$ and $q$ logically imply $g$ (using ATP for FOL).

[ICFEM02] [ASE06]
msg = Handshake.read(din, certType);

session.trustManager.checkServerTrusted (peerCerts, suite.getAuthType());

msg = new Handshake(Handshake.Type.CLIENT_KEY_EXCHANGE, ckex);

msg.write (dout, version);

only possible way without throwing exception

[[equal(fst(extr_S(c_s)),S)]]
System Evolution

Systems increasingly live longer than planned

Example: Year-2000-bug

Problem: Quality assurance for complex evolving systems

=> EU Project “Secure Change”
Maintaining Traceability under Evolution

Code evolution can potentially multiply testing effort.
Need to re-verify code parts that changed, re-do integration testing etc.
Particular problem when testing for sophisticated properties (such as security) since requires particular effort.

→ Want to automatically trace code evolution to model level so can automatically reuse earlier tests.

[Bauer, Jurjens, Yu 09]
Basic observation: Most system changes can be reduced to two kinds:

• Adding / removing parts of the system.
• Basic refactoring operations to hold system parts together despite changes.

When adding / removing code parts we need to assume that the corresponding models are also added / removed.

For evolution by refactoring can achieve automated model-code traceability e.g. using Eclipse Refactoring Language Toolkit (LTK) / XML based refactoring scripts.

Maintain model-code synchrony using continuous integration scripts (with CruiseControl / Apache Ant).
Evolution as Code Refactoring

Diagram:

DESIGN

- Virtualizing refactorings
- Concretizing refactorings

IMPLEMENTATION

- Programs
- Tests

- Programs
- Tests
Refactoring: Tool Supports
Applied our approach to a series of implementations of the Java Secure Sockets Extension library:

- Jessie 1.0.0
- Jessie 1.0.1
- JSSE 1.6

Demonstrated that our model-code co-evolution approach is robust even across major software changes.
## Refactoring JSSE

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Program entities</th>
<th>Identif.</th>
<th>Refactoring op.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $C$</td>
<td><code>clientHello</code></td>
<td><code>C</code></td>
<td><code>rename.type</code></td>
</tr>
<tr>
<td>2. $S$</td>
<td><code>serverHello</code></td>
<td><code>S</code></td>
<td><code>rename.type</code></td>
</tr>
<tr>
<td>3. $P_{ver}$</td>
<td><code>session.protocol</code></td>
<td><code>P_{ver}</code></td>
<td><code>extract.temp</code></td>
</tr>
<tr>
<td></td>
<td><code>version</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. $R_C$</td>
<td><code>clientRandom</code></td>
<td><code>R_C</code></td>
<td><code>rename.local.variable</code></td>
</tr>
<tr>
<td>$R_S$</td>
<td><code>serverRandom</code></td>
<td><code>R_S</code></td>
<td><code>rename.local.variable</code></td>
</tr>
<tr>
<td>5. $S_{id}$</td>
<td><code>sessionId</code></td>
<td><code>S_{id}</code></td>
<td><code>rename.field</code></td>
</tr>
<tr>
<td></td>
<td><code>sessionId</code></td>
<td></td>
<td><code>rename.local.variable</code></td>
</tr>
<tr>
<td>6. Ciph$[]$</td>
<td><code>session.enabledSuites</code></td>
<td><code>Ciph</code></td>
<td><code>extract.temp</code></td>
</tr>
<tr>
<td>7. Comp$[]$</td>
<td><code>comp</code></td>
<td><code>Comp</code></td>
<td><code>extract.temp</code></td>
</tr>
<tr>
<td>8. Veri</td>
<td>Lines 1518–1557</td>
<td><code>Veri</code></td>
<td><code>extract.method</code></td>
</tr>
</tbody>
</table>
# Refactoring: Performance

<table>
<thead>
<tr>
<th>Messages in sequence</th>
<th>op.</th>
<th>diff</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1: $C \rightarrow S : (P_{\text{ver}}, R_C, S_{id}, \text{Ciph}[], \text{Comp}[])$</td>
<td>7</td>
<td>31</td>
<td>13.891</td>
</tr>
<tr>
<td>S2. $S \rightarrow C : (P_{\text{ver}}, R_S, S_{id}, \text{Ciph}[], \text{Comp}[])$</td>
<td>5</td>
<td>20</td>
<td>9.437</td>
</tr>
<tr>
<td>S3. $S \rightarrow C : \text{Certificate}[X509Cert_s]$</td>
<td>2</td>
<td>2</td>
<td>1.474</td>
</tr>
<tr>
<td>S4. $C : \text{Veri}(X509Cert_s)$</td>
<td>2</td>
<td>2</td>
<td>3.854</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total of 7 messages and 3 checks</td>
<td>27</td>
<td>86</td>
<td>40.303</td>
</tr>
</tbody>
</table>
Applications of MBSE

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.

ns: Models for Secure Change
Conclusions

Model-based Security Engineering using UMLsec:
• aims to covers all kinds of security aspects.
• formally based approach
• automated tool support
• industrially used methods
• integrated approach (models, source-code, configuration data)
• ongoing Secure Change project: support system evolution
Overview

Security Engineering

Models
- Analysis Framework
- Model Checking
- Autom. Theorem Proving
- Refinement

Components
- UMLsec
  - Formal Semantics
  - Runtime Checks
  - Model-based Testing
  - Autom. Theorem Proving

Code
- Runtime Checks
- Model-based Testing
- Autom. Theorem Proving

IT Security

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

Foundations
- Fault-tolerance
- Security
- Management

Dependable Systems Development
- Engineering
  - Risk Assessment
  - Permissions
  - Business Processes
  - Security Investment
  - Firewall Configurations

Real-time

Reliability

Security
Ongoing Work

• Security Verification of Crypto Protocol Implementations in C: Use VCC to verify C code. (with Andy Gordon, MSR Cambridge; RS Industrial Fellowship & 2 PhD projects)
• Modelling for Compliance (EPSRC CASE PhD project with British Telecom)
• Security Engineering for Lifelong Evolvable Systems (EU FP7 Integrated Project)
• RS Joint International Project with TU Munich on Formal Security Analysis of Cryptoprotocol Implementations
• RS Joint International Project with NII Tokyo on Relating Security Requirements and Design
• HIRING NOW: Postdocs / PhD students (at TU Dortmund / Fraunhofer ISST, Germany)!
Questions?

More information (papers, slides, tool etc.):
http://jurjens.de/jan
J.Jurjens@open.ac.uk