Model-based Security Engineering
Jan Jürjens
Competence Center for IT Security Software & Systems Engineering Informatics, Technical University Munich from 1 Oct: Open University, UK juerjens@in.tum.de http://www4.in.tum.de/~juerjens

Personal Introduction
Leading the Competence Center for IT-Security at Software & Systems Engineering, TU Munich
• Extensive collaboration with industry (BMW, HypoVereinsbank, T-Systems, Munich Re, O2, Deutsche Bank, Siemens, Infineon, Allianz, …)
• PhD in Computer Science from Oxford Univ., Masters in Mathematics from Bremen Univ.
• Numerous publications incl. 2 books with Springer on the subject
• From 1 Oct: Senior Lecturer at Open University, UK.

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
Requirements
Weave in
(UML) Models
Code/Test
Analyze against
Reverse
Design
Configurations
Source Code

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

Secure System Lifecycle

Estimate Security

Model-based Security Engineering
Design: Encapsulate prudent security engineering rules.
Analysis: Formally based, automated, efficient tools.
Note: emphasis on high-level requirements.

UMLsec

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, FOSSAD05, ICSE05]
Example: Crypto-based Distributed System

Adversary knowledge: $k, y, x, z$

Example: TLS Variant

Presented at IEEE Infocom 1999.
Goal: send secret protected by session key using fewer server resources.

Security Analysis in First-order Logic

Approximate adversary knowledge set from above:
Predicate $\text{knows}(E)$ 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 $\text{knows}(s)$ from model-generated formulas using automatic theorem prover.

First-order Logic: Basic Rules

Define $\text{knows}(E)$ for any $E$ initially known to adversary.
Define cryptosystem, E.g.: $\text{Dec}_{k^{-1}}((E)_k) = E$
For evolving adversary knowledge define
\[
\forall E_1, E_2 \left( \text{knows}(E_1) \land \text{knows}(E_2) \Rightarrow \right.
\]
$\text{knows}((E_1)_{E_2}) \land
\text{knows}(\text{Dec}_{E_2}(E_1)) \land
\ldots
\]

Example: TLS Variant

Translation to Logic

Example: TLS Variant

Analysis

Check whether can derive $\text{knows}(s)$ e.g. using e-Setheo.
Surprise: Yes!
Protocol does not preserve secrecy of $s$.
Why? Use Prolog-based attack generator.

Jan Jürgens, TU München: Model-based Security Engineering
Man-in-the-Middle Attack

\[
\begin{align*}
N_i &= \mathcal{K}_C \oplus \text{Sign}_{\mathcal{K}_C}(C \oplus \mathcal{K}_C) \\
N_j &= \mathcal{K}_A \oplus \text{Sign}_{\mathcal{K}_A}(C \oplus \mathcal{K}_A)
\end{align*}
\]

\[
\begin{align*}
\{\text{Sign}_{\mathcal{K}_C}(K_j : N_i)\} &\mathcal{K}_A \oplus \text{Sign}_{\mathcal{K}_A}(S : \mathcal{K}_S) \\
\{\text{Sign}_{\mathcal{K}_A}(K_j : N_i)\} &\mathcal{K}_C \oplus \text{Sign}_{\mathcal{K}_C}(S : \mathcal{K}_S)
\end{align*}
\]

The Fix

\[
k^0 = \text{skd}(\text{Dec}_{\mathcal{K}_S}(\text{msg}))
\]

\[
\text{skd}(\text{Dec}_{\mathcal{K}_S}(\text{msg})) = \text{Dec}_{\mathcal{K}_S}(\text{msg})
\]

\[
\text{e-Setho: Proof that } \text{knows(s)} \text{ not derivable.}
\]

Note completeness of FOL (but also undecidability).

Refinement & Composability

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. [FME01]

Similar: Established composability for certain security requirements under suitable assumptions. [Concur01]

Layered Security Protocols

System layer on top uses security services below.

- client authenticity
- confidentiality, integrity, server authenticity

\[
\begin{align*}
\text{Security properties additive?} \quad &? \\
\text{Theorem: Yes, under suitable conditions.} \\
\end{align*}
\]

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: verify code against interface spec.

Surprise: Essentially no existing work (eg for crypto prots)!

Experiences

Can generate behavioral models from code (e.g. CFGs). Problem: too concrete
\Rightarrow understanding + automated verification hard (even with annotations).

Constructing abstract specifications from practical software is manually intensive.

Assumption: Have textual specification. Then:
- construct interface spec from textual spec
- analyze interface spec for security
- verify that software satisfies interface spec
Model vs. Implementation

### Elements of connections
- Sent and received data
- Implementation
  - Compare meaning!
- Backtrace assignments
- Defined during model creation

### Interface specification SSL
1. Identify program points: value \((r)\), receive \((p)\), guard \((g)\), send \((q)\)
2. Check guards enforced

### Sending Messages
- Automate this using patterns
  - Handshake.write()
  - ClientHello.write()
  - SSL.Socket.doClientHandshake()

### Checking Guards
- Guard \(g\) enforced by code?
  a) Generate runtime check for \(g\) at \(q\) from diagram: simple + effective, but performance penalty.
  b) Testing against checks (symbolic crypto for inequalities).
  c) Automated formal local verification: conditionals between \(p\) and \(q\) logically imply \(g\) (using ATP for FOL).

---

### Parameter der kryptographischen ClientHello Nachricht

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Effektiv übertragene Daten der ClientHello Nachricht der Jessie implementierung</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>type, getVersion()</td>
</tr>
<tr>
<td>Diver</td>
<td>major, minor</td>
</tr>
<tr>
<td>(p)</td>
<td>(q) &lt; boltTime &gt;&gt; (q) &lt; boltTime &gt;&gt; 0</td>
</tr>
<tr>
<td>(q)</td>
<td>(q) &lt; boltTime &gt;&gt; 0</td>
</tr>
<tr>
<td>r, c</td>
<td>randomBytes</td>
</tr>
<tr>
<td>SId</td>
<td>sessionId, length</td>
</tr>
<tr>
<td>Lcp</td>
<td>sessionId, (timeout::size() &lt;&lt; 1) &gt;&gt; 3 &amp; 0x8F</td>
</tr>
<tr>
<td>LKcomp</td>
<td>compSize, N</td>
</tr>
<tr>
<td>comp. 1</td>
<td>comp. N</td>
</tr>
</tbody>
</table>

Currently do this manually using code assertions

### Implementation (Jessie): Identify Values
- 2nd parameter of Random constructor called by ClientHello.write()
- 2nd parameter of ClientHello constructor
- ClientHello.random = new Random(\(\ldots\), session.random.generateSeed(28));
- Class SecureRandom (specified in: FIPS 140-2, RFC 1750) of package java.security
- Function: generateSeed

### Checking Guards
- Guard \(g\) enforced by code?
  a) Generate runtime check for \(g\) at \(q\) from diagram: simple + effective, but performance penalty.
  b) Testing against checks (symbolic crypto for inequalities).
  c) Automated formal local verification: conditionals between \(p\) and \(q\) logically imply \(g\) (using ATP for FOL).
Verification of Guards in Code

send represents send command

g: FOL formula with symbols msg, representing n-th argument of message received before program fragment p is executed

\[ p \models g \quad \text{checked in any execution of } p \]
initially satisfying \( d \) before any send write \( p \models g \) for [true] \( p \models g \).

\[ \text{if } c \text{ then } p \text{ else } q \models (c \land d \Rightarrow g, \text{no send in } q) \]

Some Rules (Simplified)

\[
\begin{align*}
[d] & \text{if } c \text{ then } p \text{ else } q = g (c \land d \Rightarrow g, \text{no send in } q) \\
[d] & \text{if } c \text{ then } p \text{ else } q = g (c \land d \Rightarrow g, \text{no send in } p) \\
[d] & \text{if } c \text{ then } p \text{ else } q = g (d \Rightarrow c) \\
[d] & \text{if } c \text{ then } p \text{ else } q = g (d \Rightarrow c) \\
[d] & \text{if } c \text{ then } p \text{ else } q = g (d \Rightarrow c) \\
[d] & \text{if } c \text{ then } p \text{ else } q = g (d \Rightarrow c) \\
[d] & \text{if } c \text{ then } p \text{ else } q = g (d \Rightarrow c) \\
\end{align*}
\]

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.

Biometric Authentication System

In development by company in joint project.
Uses bio-reference template on smart-card.
Analyze given UML spec.
Discovered three major weaknesses in subsequently improved versions (misuse counter circumvented by dropping / replaying messages, smart-card insufficiently authenticated by mixing sessions).
Common Electronic Purse Specifications

Global elec. purse standard (Visa, 90% market). Smart card contains account balance, performs crypto operations securing each transaction. Formal analysis of load and purchase protocols: three significant weaknesses: purchase redirection, fraud bank vs. load device owner.

Bank Application

Security analysis of web-based banking application, to be put to commercial use (clients fill out and sign digital order forms). Layered security protocol (first layer: SSL protocol, second layer: client authentication protocol). Security requirements:
- confidentiality
- authenticity

Related Approaches

Ruth Breu et al, David Basin et al: Role-based Access Control using UML
Cf also the workshop series: Critical Systems Development Using Modeling Languages (CSDUML) e.g. at Models 06 (Oct in Genova).

Questions?

More information (papers, slides, tool etc.):
http://www.umlsec.org
from 1 Oct: Open University, UK