Sound Methods and Effective Tools for Model-based Security Engineering with UML
Jan Jürjens
Software & Systems Engineering
TU Munich, Germany
http://www.umlsec.org

Software Engineering & Security
"Penetrate-and-patch" (aka "banana strategy):
• insecure
• disruptive
Traditional formal methods: limited adoption in industry.
• training people
• constructing formal specifications.

Model-based Security
Increase security with bounded investment in time, costs:
• Extract models from artefacts arising in industrial development and use of security-critical systems (UML models, source code, configuration data).
• Tool-supported, theoretically sound, efficient automated security analysis.

Model-based Security Engineering
• Analyze (UMLsec) models against security requirements.
• Generate code (or tests) from models.
• Generate models from evolving or legacy code.
Goal: model-based = source-code based.

Adversary: Simulation
A
m(x)
return({y::x}z)
Adversary knowledge: k, y, x
B
m(x)
return({z}k)

Security Analysis in First-order Logic
Approximate set of possible data values flowing through system from above.
Predicate \( \text{knows}(E) \) meaning that the adversary may get to know \( E \) during the execution of the protocol.
For any secret \( s \), check whether can derive \( \text{knows}(s) \) using automated theorem prover.
First-order Logic: Basic Rules

Define $knows(E)$ for any $E$ initially known to the adversary.

Define cryptosystem. E.g.: $Dec_{E^{-1}}(\{E\}) = E$

For evolving knowledge define

\[ \forall E_1, E_2. (knows(E_1) \land knows(E_2) \Rightarrow \]
\[ \begin{align*}
& \quad \text{knows}(E_1 \cdot E_2) \land \text{knows}({E_1}E_2) \land \\
& \quad \text{knows}(\text{Dec}_{E_2}(E_1)) \land \text{knows}(\text{Sign}_{E_2}(E_1)) \land \\
& \quad \text{knows}(\text{Ext}_{E_2}(E_1)))
\end{align*} \]

\[ \forall E. (knows(E) \Rightarrow \]
\[ \text{knows}(\text{head}(E)) \land \text{knows}(\text{tail}(E)) \]

Given Sequence Diagram ...

\[
\begin{align*}
\text{TR1} &= \left( \text{in}(msg\_in), \text{cond}(msg\_in), \text{out}(msg\_out) \right) \\
\text{followed by TR2} &\text{ gives predicate PRED(TR1) =} \\
& \begin{align*}
& \forall \text{msg\_in}. \left[ \text{knows}(\text{msg\_in}) \land \\
& \quad \text{cond}(\text{msg\_in}) \Rightarrow \text{knows}(\text{msg\_out}) \land \\
& \quad \text{PRED(TR2)} \right]
\end{align*} \\
\end{align*}
\]

Example: Proposed Variant of TLS (SSL)

Presented at IEEE Infocom 1999.

Goal: send secret protected by session key using fewer server resources.

Example: Translation to Logic

\[
\begin{align*}
\text{knows}(N) \land \text{knows}(K_C) \land \text{knows}(\text{Sign}_{K_C-1}(C::K_C)) \land \\
\begin{align*}
& \forall \text{init}1, \text{init}2, \text{init}3. \left[ \text{knows}(\text{init}1) \land \\
& \quad \text{knows}(\text{init}2) \land \text{knows}(\text{init}3) \Rightarrow \text{snd}(\text{Ext}_{\text{init}2}(\text{init}3)) = \text{init}2 \right] \land \\
& \quad \text{knows}(\text{Sign}_{K_S-1}(\ldots)) \land \ldots \land \text{proof frame}
\end{align*}
\end{align*}
\]

Example: Translation to Logic

Can derive $knows(s)$.
That is: Protocol does not preserve secrecy of $s$ against adversaries.

\[ \Rightarrow \text{Completely insecure wrt stated goals.} \]

But why? Use prolog-based attack generator.
**Man-in-the-Middle Attack**

\[
\begin{align*}
N_1 & \leftarrow \text{K}_C : \text{Sign} \text{K}_C^{-1} (\text{C} : \text{K}_C) \\
N_1 & \leftarrow \text{K}_A : \text{Sign} \text{K}_A^{-1} (\text{C} : \text{K}_A) \\
\text{C} & \rightarrow A \quad S \\
\{\text{Sign} \text{K}_A^{-1} (\text{K}_A : N_1)\} \text{K}_A^{-1} \text{Sign} \text{K}_A^{-1} (\text{S} : \text{K}_A) & \rightarrow A \\
\{\text{Sign} \text{K}_A^{-1} (\text{K}_A : N_1)\} \text{K}_A^{-1} \text{Sign} \text{K}_A^{-1} (\text{S} : \text{K}_A) & \rightarrow A \\
\{\text{v}_j\} & \rightarrow A \quad S
\end{align*}
\]

**Biometric Authentication System**

In development by large German telecommunication company.

In joint project, use presented security analysis tools at given UML specification.

So far, have discovered three major attacks against subsequently improved versions (misuse counter circumvented by dropping / replaying messages, smart-card insufficiently authenticated by recombining sessions).

**Related Work**

- UML and security:
  - C. Montangero et al.: Degas project.
  - D. Basin et al.: Secure UML

- UML verification:
  - Lilius et al.: vUML
  - ...

**Resources**

Jan Jürjens, Secure Systems Development with UML, Springer 2004

Workshop: CSDUML@SAFECOMP05 (Norway, Sept. 05)

Application to C source code: Memocode ‘05

More information (papers, slides, tool etc.):

http://www.umlsec.org

(user: Participant, password: Iwasthere)

Note: International Symposium on Secure Software Engineering (ISSSE 06 - IEEE) [S. Redwine, A. Hall, J. Wing]