Sound Methods and Effective Tools for Model-based Security Engineering with UML

Jan Jürjens

Software & Systems Engineering
TU Munich, Germany

juerjens@in.tum.de
http://www.umlsec.org

Software Engineering & Security

„Penetrate-and-patch“ (aka „banana strategy):

- insecure
- disruptive

Traditional formal methods: limited adoptation in industry.

- training people
- constructing formal specifications.

Model-based Security

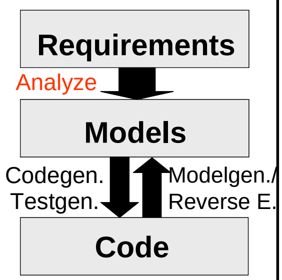
Increase security with bounded investment in time, costs:

- 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.

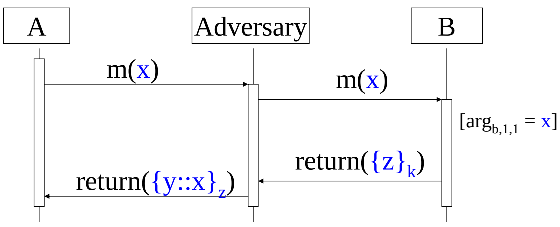
Model-based Security Engineering

- 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



Adversary knowledge:

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

- $\forall e, k. Dec_{k^{-1}}(\{e\}_k) = e$

Security Analysis in First-order Logic

Approximate set of possible data values flowing through system from above.

Predicate *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 *knows(s)* using automated theorem prover.

First-order Logic: Basic Rules

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

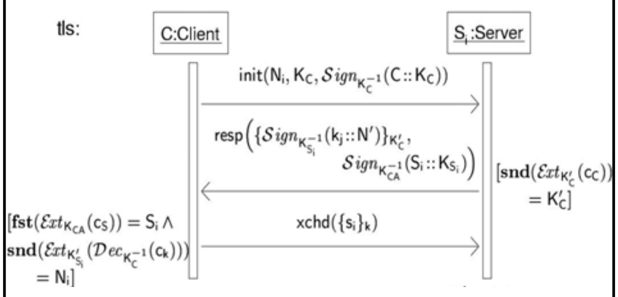
Define cryptosystem. E.g.: $Dec_{K^{-1}}(\{E\}_k) = E$

For evolving knowledge define

$\forall$ *E₁,E₂.(knows(E₁)* $\wedge$ *knows(E₂)* $\Rightarrow$
knows(E₁::E₂) $\wedge$ *knows({E₁}$_{E2}$)* $\wedge$
knows(Dec$_{E2}$(E₁)) $\wedge$ *knows(Sign$_{E2}$ (E₁))* $\wedge$
knows(Ext$_{E2}$ (E₁)))

$\forall$ *E.(knows(E)* $\Rightarrow$
knows(head(E)) $\wedge$ *knows(tail(E)))*

Given Sequence Diagram …



… Translate to 1st Order Logic

Sequence diagram connection
TR1=(in(msg_in),cond(msg_in),out(msg_out))
followed by *TR2* gives predicate *PRED(TR1)=*
8 msg_in. [knows(msg_in)Æ cond(msg_in)
) knows(msg_out)
Æ PRED(TR2)]

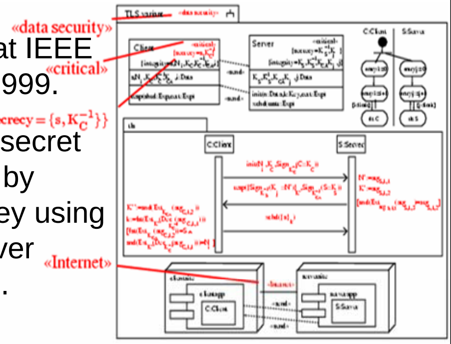
Abstraction (e.g. from senders, receivers): find all attacks, may have false positives.

Check whether can derive threat conjecture (e.g. *knows(s)* for a secret *s*) from axioms.

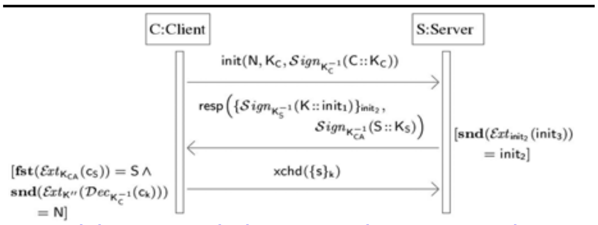
Example: Proposed Variant of TLS (SSL)

Presented at IEEE Infocom 1999.

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

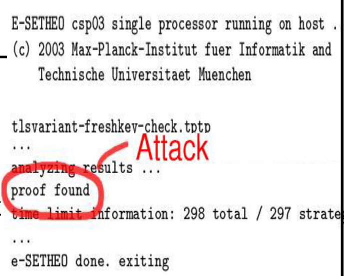


Example: Translation to Logic



knows(N)Æ knows(K$_C$)Æ knows(Sign$_{KC-1}$(C::K$_C$))
Æ 8 init₁,init₂,init₃.[knows(init₁)Æ knows(init₂)Æ
knows(init₃) Æ snd(Ext$_{init2}$(init₃)) = init₂
) knows({Sign$_{KS-1}$(…)}$_{…}$)Æ […] Æ […)…]…]

Surprise …

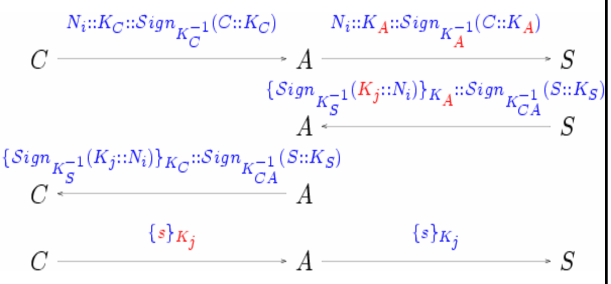


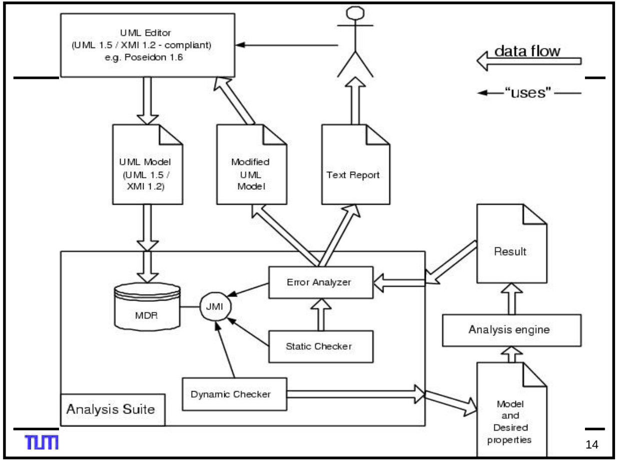
Can derive *knows(s)*.
That is: Protocol does not preserve secrecy of *s* against adversaries.

➔ Completely insecure wrt stated goals.

But why ? Use prolog-based attack generator.

Man-in-the-Middle Attack





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 recombing sessions).

Related Work

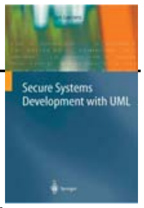
UML and security:

- 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]