

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.



TUM Jan Jürjens, TU Munich: Model-based Security Engineering

Challenge: Security

Security is **holistic** property:

- Attackers often **circumvent** (not: **break**) mechanisms.
- **Transform** (in)secure components to **secure systems** ?

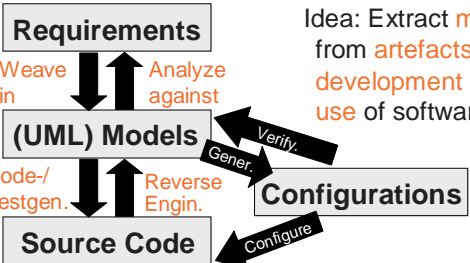


„Those who think that their problem can be solved by simply applying cryptography don't understand cryptography and don't understand their problem“ (B. Lampson / R. Needham).

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 3

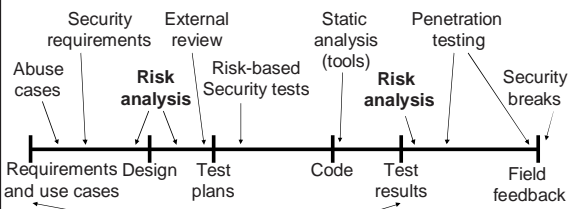
Model-based Security Engineering

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



→ **Tool-supported, theoretically sound, efficient automated security design & analysis.**

Secure System Lifecycle

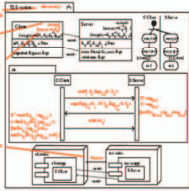


Model-based Security Engineering [McGraw 2003]

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,FOSAD05,ICSE05]

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 6

Example: Crypto-based Distributed System

Attacker may ...

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

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

(cf. [Dolev, Yao 1982])

TUM Jan Jürjens, TU Munich: Model-t

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]

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 8

First-order Logic: Basic Rules

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

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

For **evolving** adversary knowledge define

$$\forall E_1, E_2. (knows(E_1) \wedge knows(E_2) \Rightarrow knows(\{E_1\}_{E_2}) \wedge knows(Dec_{E_2}(E_1)) \wedge \dots)$$

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 9

Example: TLS Variant

Presented at IEEE Infocom 1999.

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

TUM

Example: Translation to Logic

$Ext_{K_{CA}}(C_s) = S \wedge Ext_{K_C}(Dec_{K_C^{-1}}(C_s)) = N$

$knows(N) \wedge knows(K_C) \wedge knows(Sign_{K_C^{-1}}(C::K_C)) \wedge \forall init_1, init_2, init_3. [knows(init_1) \wedge knows(init_2) \wedge knows(init_3) \wedge snd(Ext_{init_2}(init_3)) = init_2 \Rightarrow knows(\{Sign_{K_S^{-1}}(\dots)\}_{\dots}) \wedge [knows(Sign_{\dots})] \wedge \forall resp_1, resp_2. [\dots \Rightarrow \dots]]$

TUM Jan Jürjens, TU Munich: Model-based Security Engineering

Analysis

Check whether can **derive** $knows(s)$ e.g. using e-Setho.

Surprise: **Yes!**
 → Protocol does **not** preserve secrecy of s .

Why? Use Prolog-based **attack generator**.

```

input_formula(tls_abstract_protocol_axiom, (
  ! [ArgS_11, ArgS_12, ArgS_13, ArgC_11, ArgC_12] : (
    ! [DataC_KK, DataC_k, DataC_n] : (
      % Client -> Attacker (1. message)
      (
        knows(n)
        & knows(k_c)
        & knows(sign(conc(c, k_c)))
        & % Server -> Attacker (2. message)
        (
          (
            knows(ArgS_11)
            & knows(ArgS_12)
            & knows(ArgS)
            & (? [X])
          )
        )
      )
    )
  )
)
=> (
  know(symenc(secret, DataC_k))
)

```

TUM Jan Jürjens, TU Munich: Model-based Security Engineering

Man-in-the-Middle Attack

$$C \xrightarrow{N_i::K_C::\text{Sign}_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::\text{Sign}_{K_A^{-1}}(C::K_A)} S$$

$$A \xrightarrow{\{\text{Sign}_{K_S^{-1}}(K_j::N_i)\}_{K_A}::\text{Sign}_{K_{CA}^{-1}}(S::K_S)} S$$

$$C \xleftarrow{\{\text{Sign}_{K_S^{-1}}(K_j::N_i)\}_{K_C}::\text{Sign}_{K_{CA}^{-1}}(S::K_S)} A$$

$$C \xrightarrow{\{s\}_{K_j}} A \xrightarrow{\{s\}_{K_j}} S$$

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 13

The Fix

$$K' := \text{snd}(\text{Ext}_{K_{CA}}(\text{arg}_{C,1,2}))$$

$$k := \text{fst}(\text{Ext}_{K'}(\text{Dec}_{K_C^{-1}}(\text{arg}_{C,1,1})))$$

$$\text{fst}(\text{Ext}_{K_{CA}}(\text{arg}_{C,1,2})) = S \wedge$$

$$\text{snd}(\text{Ext}_{K'}(\text{Dec}_{K_C^{-1}}(\text{arg}_{C,1,1}))) = N_i \wedge$$

$$\text{fbd}(\text{Ext}_{K_S}(\text{Dec}_{K_C^{-1}}(\text{arg}_{C,1,1}))) = K_C$$

e-Setheo: **Proof** that $\text{knows}(s)$ not derivable.
 Note **completeness** of FOL (but also undecidability).

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 14

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]

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 15

Layered Security Protocols

System layer on **top** uses **security** services **below**.

client authenticity

confidentiality, integrity, server authenticity

=

confidentiality, ... + client authenticity ?

Security properties **additive** ? [Safecom03]
Theorem: **Yes**, under suitable conditions.

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 16

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) !

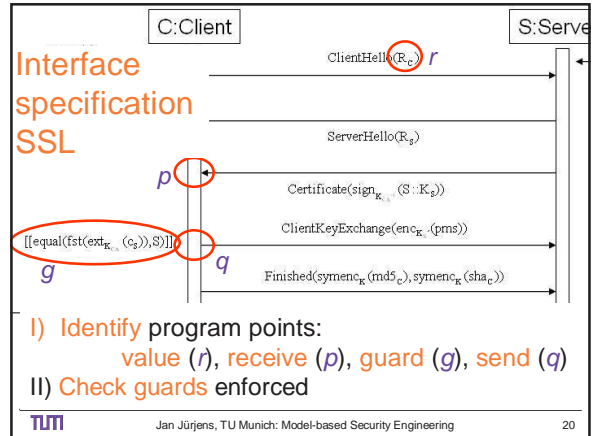
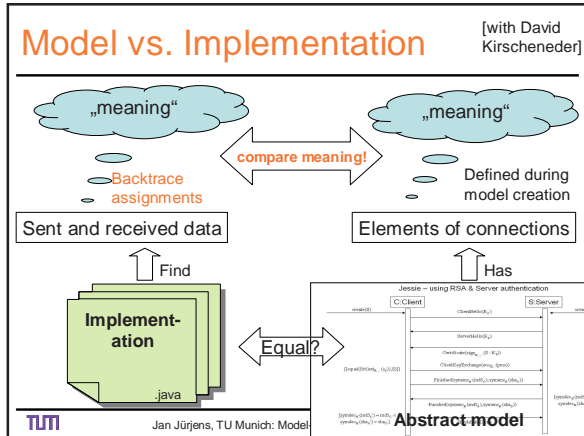
TUM Jan Jürjens, TU Munich: Model-based Security Engineering 17

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

TUM Jan Jürjens, TU Munich: Model-based Security Engineering 18

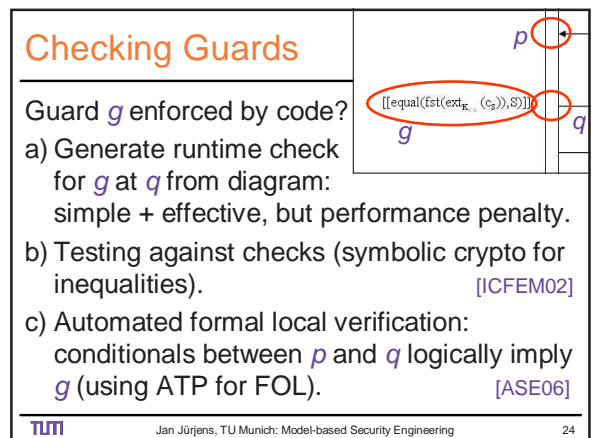
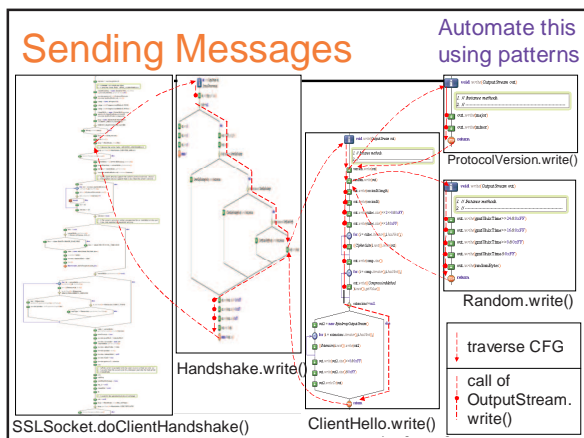
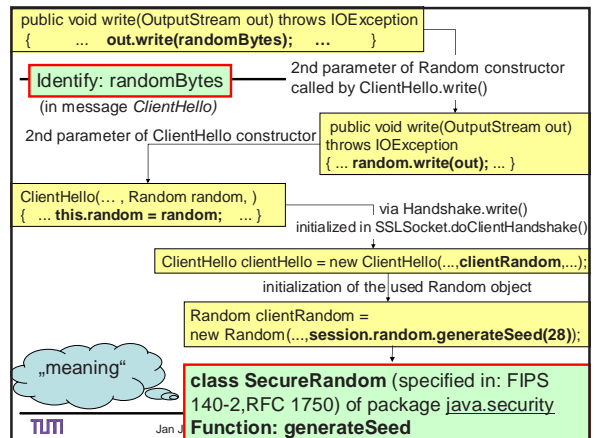


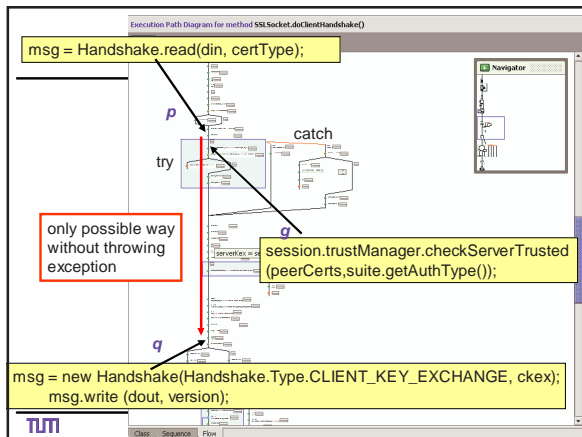
Parameter der kryptographischen ClientHello Nachricht	Effektiv übertragene Daten der ClientHello Nachricht der Jessie Implementierung
C	type.getValue()
Pver	major
	minor
	((gmtUnixTime >>> 16) & 0xFF)
	((gmtUnixTime >>> 8) & 0xFF)
r_c	randomBytes
Sid	sessionId.length
	sessionId
	((suites.size() << 1) >>> 8 & 0xFF)
	((suites.size() << 1) & 0xFF)
LCip	suites_1
	...
	suites_N
	comp.size()
LKomp	comp_1
	...
	comp_N

Implementation (Jessie): Identify Values

Currently do this manually using code assertions

TUM Jan Jürjens, TU Munich: Model-based Security Engineering





Verification of Guards in Code

send: represents send command

g: FOL formula with symbols msg_n representing n^{th} argument of message received before program fragment p is executed

$[d] p \models g$: g checked in any execution of p initially satisfying d before any **send**
write $p \models g$ for $[true] p \models g$.

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

Some Rules (Simplified)

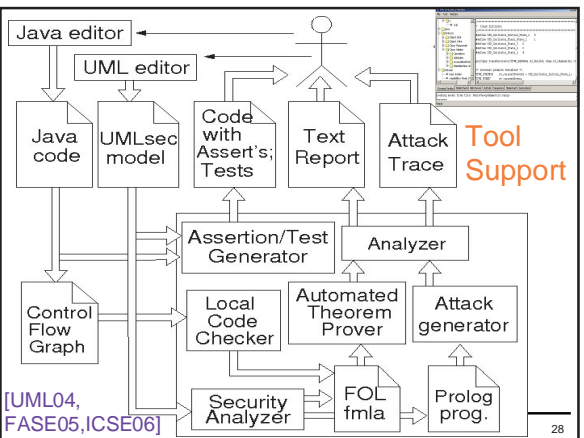
$[d] \text{ if } c \text{ then } p \text{ else } q \models g \iff (c \wedge d \Rightarrow g, \text{ no send in } q)$

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

$[d] \text{ if } c \text{ then } p \text{ else } q \models g \iff (d \Rightarrow c) \iff [d] p \models g$

$[d] \text{ if } c \text{ then } p \text{ else } q \models g \iff (d \Rightarrow \neg c) \iff [d] p \models g \wedge d' \Rightarrow d$

$\frac{[d] p \models g}{x := e; p \models g} d \Rightarrow x = e$



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.



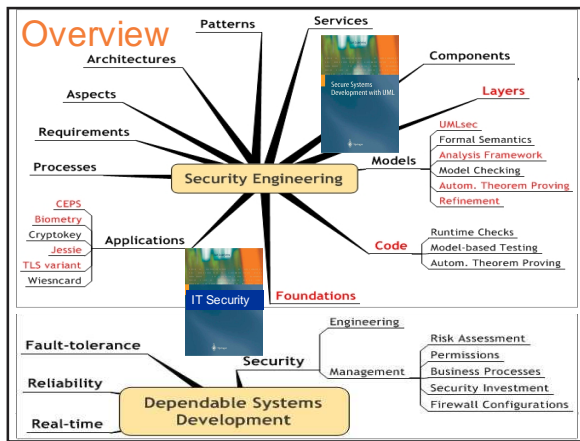
Bank Application

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

Eduardo Fernandez-Medina et al: UML for the Design of Secure Data Bases, Security Architectural Patterns (cf Thursday)

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).

