UP and Security: Overview of UMLsec

Jan Jürjens

Competence Center for IT Security
Software & Systems Engineering
TU München

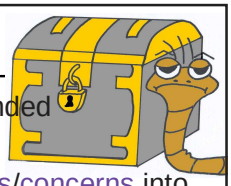
juerjens@in.tum.de
http://www.umlsec.org

Software Engineering & Security

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

Traditional formal methods: expensive.
- training people
- constructing formal specifications.

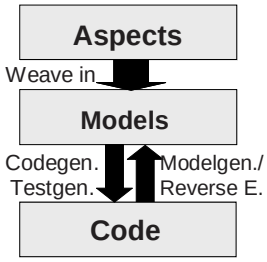
Security by Design



Increase security with bounded investment in time, costs:
- Weave in security aspects/concerns into artefacts arising in industrial development and use of security-critical systems (UML models, source code, configuration data).
- Tool-supported, theoretically sound, efficient automated security synthesis.

Model-based Security with Aspects

- Weave in security aspects into UMLsec models.
- Generate code (or tests) from models.
- Generate models from evolving or legacy code.



UMLsec: Goals

Extension for secure systems development.
- evaluate UML specifications for weaknesses in design
- encapsulate established rules of prudent secure engineering as checklist
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- make certification cost-effective

UMLsec: How

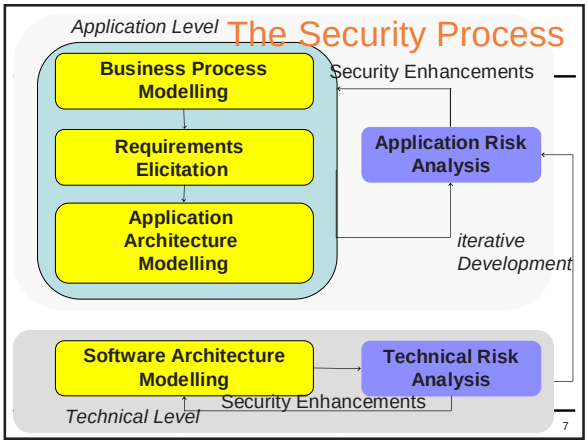
Recurring security requirements, adversary scenarios, concepts offered as stereotypes with tags on component-level.

Use associated constraints to verify specifications using automated theorem provers and indicate possible weaknesses.

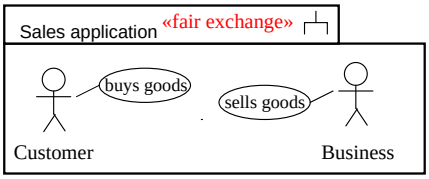
Ensures that UML specification provides desired level of security requirements.

Link to code via round-trip engineering etc.

The Security Process



Requirements with Use Case Diagrams



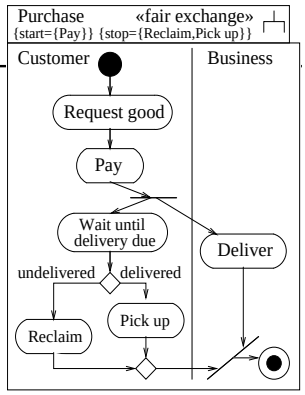
Capture security requirements in use case diagrams.

Constraint: need to appear in corresponding activity diagram.

Fair Exchange

Customer buys good from a business.

How can enforce fair exchange:
After payment, customer is eventually either delivered good or able to reclaim payment (or vc.vs.).



≪fair exchange≫

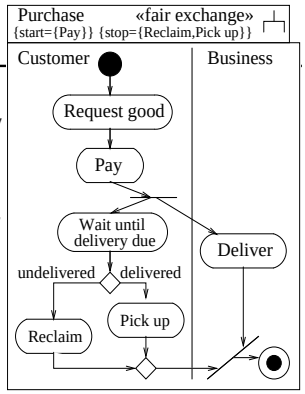
Ensures generic fair exchange condition.

Constraint: after a {start} state in activity diagram is reached, eventually reach {stop} state.

(Cannot be ensured for systems that an attacker can stop completely.)

Example

≪fair exchange≫ fulfilled if adversary cannot stop system: After payment, customer is eventually either delivered good or able to reclaim payment.



≪Internet≫, ≪encrypted≫, …

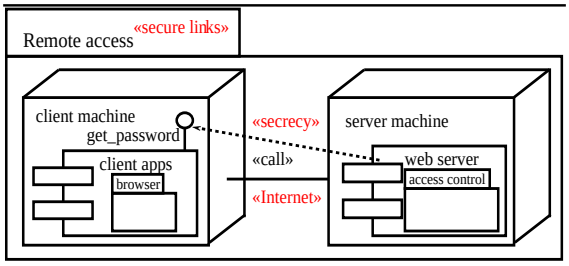
Kinds of communication links resp. system nodes.

For adversary type A, stereotype s, have set $Threats_A(s) \in \{delete, read, insert, access\}$ of actions that adversaries are capable of.

Default attacker:

Stereotype	$Threats_{default}$ ()
Internet	{delete, read, insert}
encrypted	{delete}
LAN	∅
smart card	∅

Secure Architecture



Architecture secure against default adversary ?

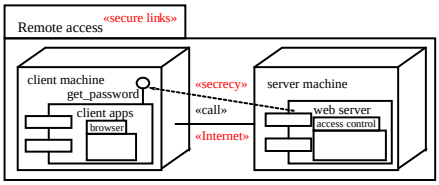
≪secure links≫

Ensures that physical layer meets security requirements on communication.

Constraint: for each dependency *d* with stereotype *s* ∈ {≪secrecy≫, ≪integrity≫} between components on nodes *n*≠*m*, have a communication link *l* between *n* and *m* with stereotype *t* such that

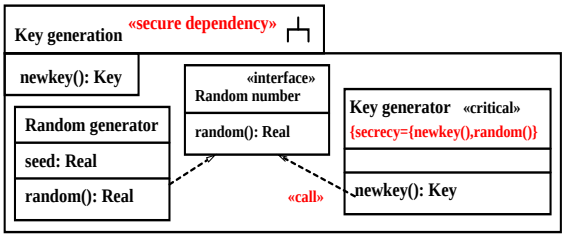
- if *s* = ≪secrecy≫: have read ∉ $Threats_A(t)$.
- if *s* = ≪integrity≫: have insert ∉ $Threats_A(t)$.

Example ≪secure links≫



Given default adversary type, constraint for stereotype ≪secure links≫ violated: According to the $Threats_{default}$(Internet) scenario, ≪Internet≫ link does not provide secrecy against default adversary.

Secure Data Structure



Data structure secure ?

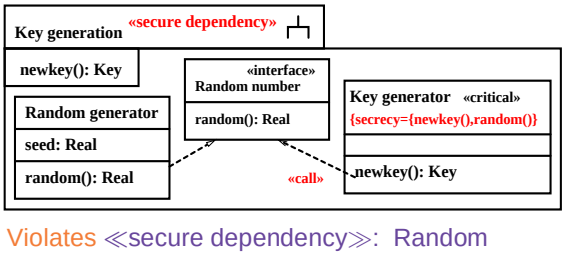
≪secure dependency≫

Ensure that ≪call≫ and ≪send≫ dependencies between components respect security requirements on communicated data given by tags {secrecy}, {integrity}.

Constraint: for ≪call≫ or ≪send≫ dependency from *C* to *D* (and similarly for {integrity}):

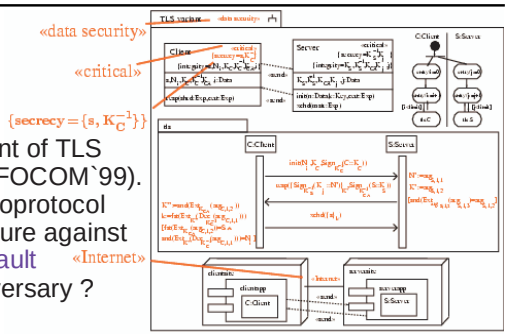
- Msg in *D* is {secrecy} in *C* if and only if also in *D*.
- If msg in *D* is {secrecy} in *C*, dependency stereotyped ≪secrecy≫.

Example ≪secure dependency≫



Violates ≪secure dependency≫: Random generator and ≪call≫ dependency do not give security level for random() to key generator.

Secure Use of Cryptography



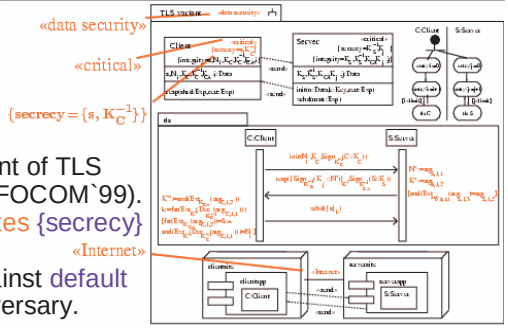
Variant of TLS (INFOCOM`99).
Cryptoprotocol secure against default adversary ?

≪data security≫

Security requirements of data marked ≪critical≫ enforced against threat scenario from deployment diagram.

Constraints: Data marked {secrecy}, {integrity}, {authenticity}, {fresh} fulfills respective formalized security requirements.

Example ≪data security≫



Variant of TLS (INFOCOM`99).
Violates {secrecy} of s against default adversary.

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.

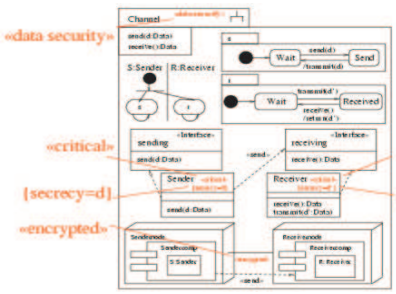
Security management: Use activity diagrams.

Technology specific: Java, CORBA security.

Model-based Security Aspects

- Define abstract security aspect.
- Define concretization (e.g. protocol).
- If possible, give conditions under which it is secure to weave in aspect using concretization, e.g. by simulation argument.

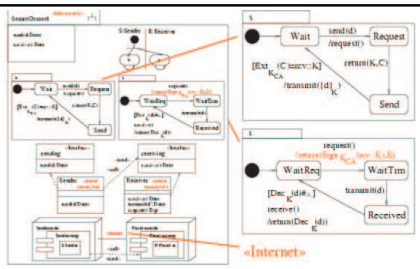
Secure Channel Aspect



Primary model with *directives* for security aspects (cf. *join points* in AspectJ).

To keep *d* secret, must be sent encrypted.

Secure Channel Aspect: Weaving



Exchange certificate and send encrypted data over Internet.

Aspect Validation

Need to prove concretization securely refines abstract aspect. Challenging problem in security.

For secure channel, have generic result.

Often not possible.

➔ Use translation validation on the weaving transformation, before or after code generation.

Translation Validation: Model or Code ?

Model:
+ earlier (less work may have to be redone)
+ more abstract ➔ more efficient
- more abstract ➔ may miss attacks
- code construction not completely automatic
- code generators not formally verified

Code:
+ „the real thing“ (which is executed)

➔ Do both ! (as far as feasible; e.g. where largely automatic). Here: look at code.

Code-level Translation Validation

Logic-based program understanding of crypto protocols in C which is as

- automatic and
- complete

as possible.

Note: can‘t be both perfectly automated and complete: Security in general undecidable.

Abstract and approximate safely.

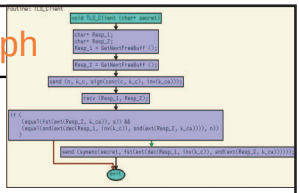
Security Analysis in First-order Logic

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.

E.g. secrecy: For any secret *s*, check whether can derive *knows(s)* using automated theorem prover.

Control Flow Graph



Generate control flow graph (e.g. with aicall (Absint)).

Transform to state machine:

trans(state,inpattern,condition,action,nextstate)

where action can be outpattern or localvar:=value.

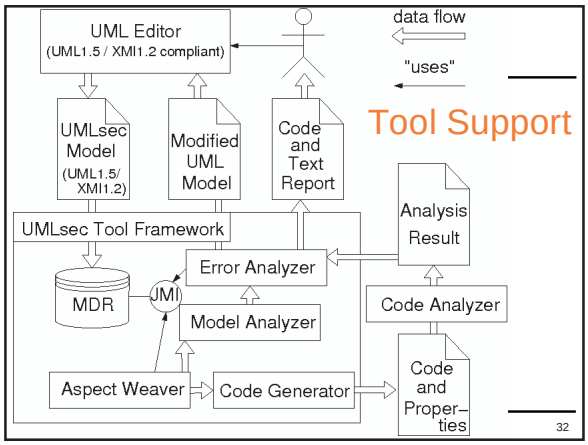
… Translate to First Order Logic

Graph transition
TR1=(in(msg_in),cond(msg_in),out(msg_out))
followed by *TR2* gives predicate *PRED(TR1)=*

$$\forall msg\_in.\ [knows(msg\_in) \wedge cond(msg\_in) \Rightarrow knows(msg\_out) \wedge PRED(TR2)]$$

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

Analyze with automated prover.



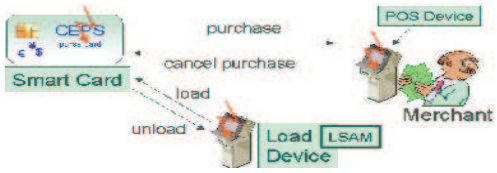
Industrial Applications: Biometry

Biometric Authentication System in development by a company in joint project.

Use our design and validation methods to develop and analyze system.

Discovered three significant security weaknesses against subsequently improved versions.

Common Electronic Purse Specifications



Global electronic purse standard (90% of market).

Smart card contains account balance. Chip performs cryptographic operations securing the transactions.

More fraud protection than credit cards (transaction-bound authorisation).

FAIRPAY

Load protocol

Unlinked, cash-based load transaction (on-line).

Load value onto card using cash at load device.

Load device contains Load Security Application Module (LSAM): secure data processing and storage.

Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.

Banking application

- Security analysis of web-based banking application, to be put to commercial use (clients fill out and sign digital order forms).
- In cooperation with major German bank.
- Layered security protocol
  - first layer: SSL protocol.
  - second layer: client authentication protocol
- Main security requirements:
  - personal data confidential.
  - orders not submitted in name of others.

Some Further Applications

Analysis of SAP access control configurations for German bank

Telematic automobile emergency application of German car company

Electronic signature architecture of German insurance company

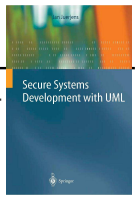
Electronic purse for Oktoberfest

Conclusions

Model-based Security Engineering using UMLsec:

- formally based approach
- automated tool support
- industrially used notation
- integrated approach (specification, source-code, configuration data)

Resources



Jan Jürjens, Secure Systems Development with UML, Springer 2004

ICSE 2005 (atp's), ICSM 2005 (code), Models 2005 (aspects)

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

More information (papers, slides, tool etc.):
http://www.umlsec.org
(user: Participant, password: Iwasthere)