Dynamic Secure Aspect Modeling with UML: From Models to Code

Jan Jürjens and Siv Houmb

Software & Systems Engineering
Technical Univ. of Munich
http://www4.in.tum.de/~juerjens
juerjens@in.tum.de

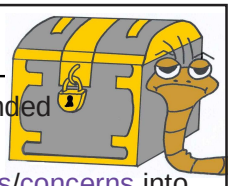
Dep. of Comp. & Inform. Science
Norwegian Univ. of Science and Technology
siv.hilde.houmb@idi.ntnu.no

Software Engineering & Security

„Penetrate-and-patch“:
- insecure
- disruptive

Traditional formal methods: limited adoption in industry.
- training people
- constructing formal specifications.

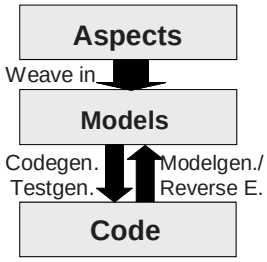
Security by Design



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.



Weaving: Models or Code ?

Weave aspects into model; generate code:

+ admits automated analysis on model-level to validate weaving

Weave aspects into generated code:

+ may be more flexible

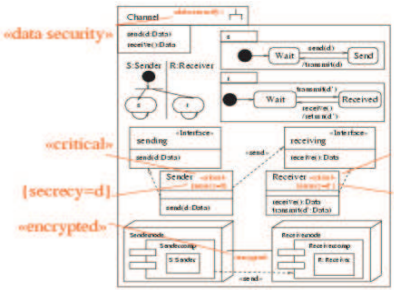
Our approach supports both; we prefer (and present here) the first where possible.

Note: woven models not necessarily meant for human consumption.

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.

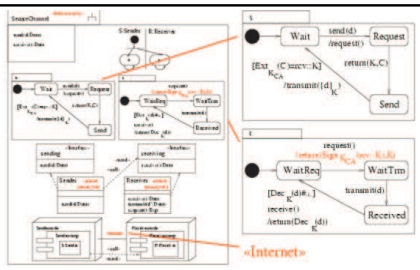
Secure Channel Aspect



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.

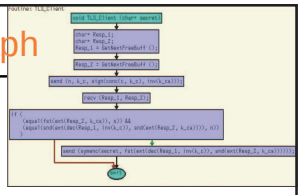
Security Analysis in First-order Logic

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)=*

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

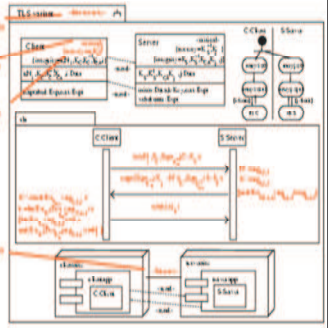
Abstraction (e.g. from senders, receivers): find all attacks, may have false positives.

Analyze with automated prover.

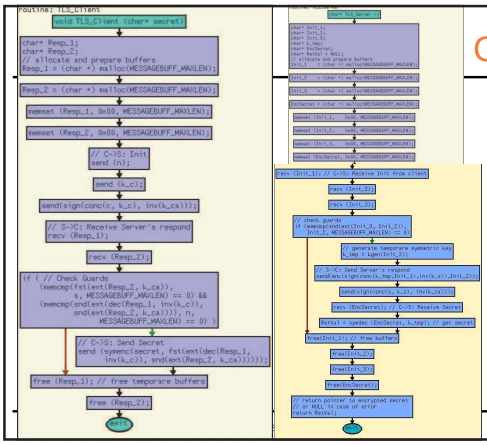
Example: Proposed Variant of TLS (SSL)

Presented at IEEE Infocom 1999.

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



Code



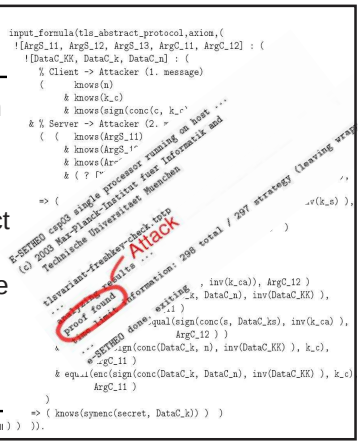
Logic

Check whether can derive *knows(s)*.

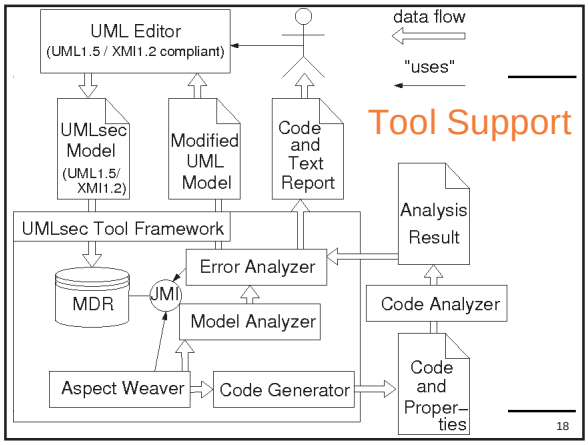
Surprise: Yes !

That means: Aspect concretization does not preserve secrecy of *s*

➔ not secure.



Tool Support



Industrial Application

Biometric Authentication System in development by large German company in joint project.

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

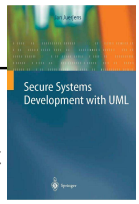
Discovered three significant security weaknesses against subsequently improved versions.

Related Work

- R. France, G. Georg et al.: use security aspect models to describe crosscutting solutions.
- H. Gomaa et al.: Separate application and security concerns in modeling.

So far mostly emphasis on static models. Also so far no integration with code-level verification.

Conclusions, Future Work



Aspect-based Security with UMLsec:

- formally based approach
- sophisticated, automated tool support
- successful use in industrial projects

Further work: Link to formal semantics in development in joint project with B. Selic (IBM Rational) et al.; integrate into NSF initiative by R. France et al..

More information (papers, slides, tool, …):
http://www.umlsec.org