Developing Secure Web-based Applications with UML: Methods and Tools

Jan Jürjens

Software & Systems Engineering
TU Munich, Germany

juerjens@in.tum.de
http://www.jurjens.de/jan

Personal introduction + history

Me: Leading the Competence Center for IT-Security at Software & Systems Engineering, TU Munich

- Extensive collaboration with industry (BMW, HypoVereinsbank, T-Systems, Deutsche Bank, Siemens, Infineon, ...)
- PhD in Computer Science from Oxford Univ., Masters in Mathematics from Bremen Univ.
- Numerous publications incl. 1 book on the subject

This tutorial: part of series of 30 tutorials at international conferences. Continuously improved (please fill in feedback forms).

A Need for Security

Society and economies rely on computer networks for communication, finance, energy distribution, transportation...

Attacks threaten economical and physical integrity of people and organizations.

Interconnected systems can be attacked anonymously and from a safe distance.

Networked computers need to be secure.

Problems

Many flaws found in designs of security-critical systems, sometimes years after publication or use.

Spectacular Example (1997):

NSA hacker team breaks into U.S. Department of Defense computers and the U.S.electric power grid system. Simulates power outages and 911 emergency telephone overloads in Washington, D.C..

Causes I

- Designing secure systems correctly is difficult.
  Even experts may fail:
  - Needham-Schroeder protocol (1978)
  - attacks found 1981 (Denning, Sacco), 1995 (Lowe)
- Designers often lack background in security.
- Security as an afterthought.

Causes II

„Blind“ use of mechanisms:

- Security often compromised by circumventing (rather than breaking) them.
- Assumptions on system context, physical environment.

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (Lampson, Needham).

Difficulties

Exploit information spreads quickly.

No feedback on delivered security from customers.

Previous approaches

„Penetrate-and-patch“:

- insecure
- disruptive

Traditional formal methods: expensive.

- training people
- constructing formal specifications.

Goal: Security by design

Consider security

- from early on
- within development context
- taking an expansive view
- in a seamless way.

Secure design by model analysis.

Secure implementation by test generation.

Holistic view on Security

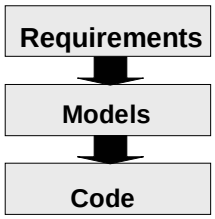
„An expansive view of the problem is most appropriate to help ensure that no gaps appear in the strategy“ (Saltzer, Schroeder 1975).

But „no complete method applicable to the construction of large general-purpose systems exists yet“ - since 1975.

Model-based Development

Goal: easen transition from human ideas to executed systems.

Increase quality with bounded time-to-market and cost.



Using UML

UML: unprecedented opportunity for high-quality critical systems development feasible in industrial context:

- De-facto standard in industrial modeling: large number of developers trained in UML.
- Relatively precisely defined (given the user community).
- Many tools in development (also for analysis, testing, simulation, transformation).

Challenges

- Adapt UML to critical system application domains.
- Correct use of UML in the application domains.
- Conflict between flexibility and unambiguity in the meaning of a notation.
- Improving tool-support for critical systems development with UML.

UMLsec: Goals

Extensions 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

The UMLsec profile

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

Use associated constraints to evaluate specifications and indicate possible weaknesses.

Ensures that UML specification provides desired level of security requirements.

Link to code via test-sequence generation.

This tutorial

Background knowledge on using UML for critical systems development.

- UML basics, including extension mechanisms.
- Extensions of UML (UMLsec, UML-RT, ...)
- UML as a formal design technique.
- Tools.
- Case studies.

Concentrate on security-critical systems.

Explain how to generalize approach to other criticality requirements.

Before we start …

We have more material than we can usefully cover within the given time frame.

Let's make selection based on your background/interests:

- UML background (no, beginner, advanced)
- working background (industrial, academic)
- application domain interests

Roadmap

Prologue

UML

UMLsec: The profile

Security analysis

Using Java security, CORBAsec

Case studies

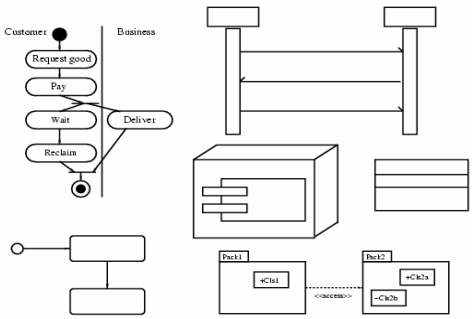
UML 2.0, Testing, Tools

UML

Unified Modeling Language (UML):

- visual modelling for OO systems
- different views on a system
- high degree of abstraction possible
- de-facto industry standard (OMG)
- standard extension mechanisms

A glimpse at UML



Used fragment of UML

Use case diagram: discuss requirements of the system

Class diagram: data structure of the system

Statechart diagram: dynamic component behaviour

Activity diagram: flow of control between components

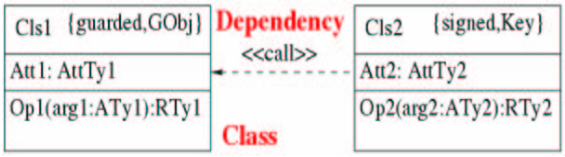
Sequence diagram: interaction by message exchange

Deployment diagram: physical environment

Package/Subsystem: collect diagrams for system part

Current: UML 1.5 (released Mar 2003)

UML run-through: Class diagrams



Class structure of system.

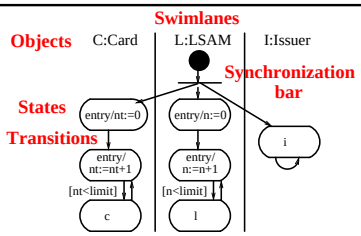
Classes with attributes and operations/signals; relationships between classes.

UML run-through: Statecharts

Dynamic behaviour of individual component.

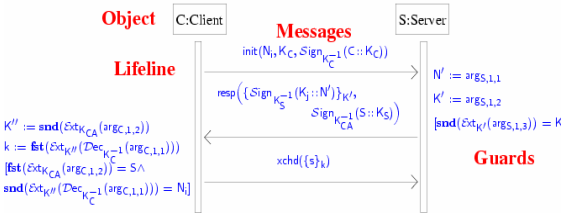
Input events cause state change and output actions.

UML run–through: Activity diagrams



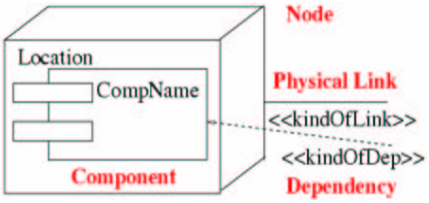
Specify the control flow between components within the system, at higher degree of abstraction than statecharts and sequence diagrams.

UML run-through: Sequence Diagrams



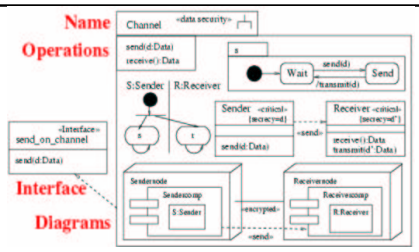
Describe interaction between objects or components via message exchange.

UML run-through: Deployment diagrams



Describe the physical layer on which the system is to be implemented.

UML run-through: Package



May be used to organize model elements into groups.

UML extension mechanisms

Stereotype: specialize model element using ≪label≫.

Tagged value: attach {tag=value} pair to stereotyped element.

Constraint: refine semantics of stereotyped element.

Profile: gather above information.

Roadmap

Prologue

UML

UMLsec: The profile

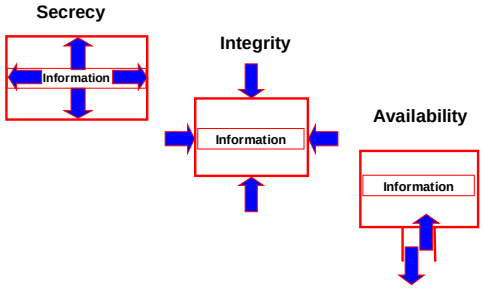
Security analysis

Using Java security, CORBAsec

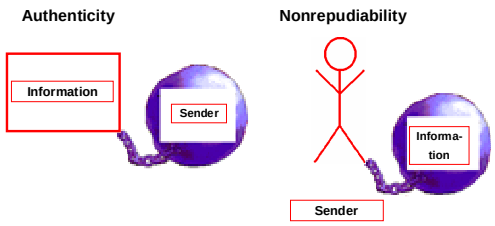
Case studies

UML 2.0, Testing, Tools

Basic Security Requirements I



Basic Security Requirements II



UMLsec profile (excerpt)

Stereotype	Base class	Tags	Constraints	Description
Internet	link			Internet connection
secure links	subsystem		dependency security matched by links	enforces secure communication links
secrecy	dependency			assumes secrecy
secure dependency	subsystem		call, send respect data security	structural interaction data security
no down-flow	subsystem	high	prevents down-flow	information flow
data security	subsystem		provides secrecy, integrity	basic datasec requirements
fair exchange	package	start, stop	after start eventually reach stop	enforce fair exchange
guarded access	Subsystem		guarded objects acc. through guards.	access control using guard objects

≪Internet≫, ≪encrypted≫, …

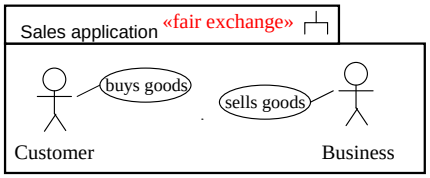
Kinds of communication links resp. system nodes.

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	∅

Requirements with use case diagrams



Capture security requirements in use case diagrams.

Constraint: need to appear in corresponding activity diagram.

≪fair exchange≫

Ensures generic fair exchange condition.

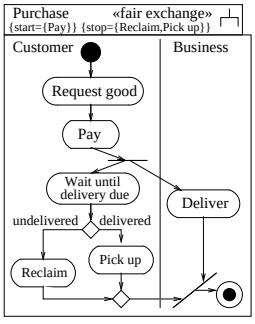
Constraint: after a {start} state in activity diagram is reached, eventually reach {stop} state.

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

Example ≪fair exchange≫

Customer buys a good from a business.

Fair exchange means: after payment, customer is eventually either delivered good or able to reclaim payment.



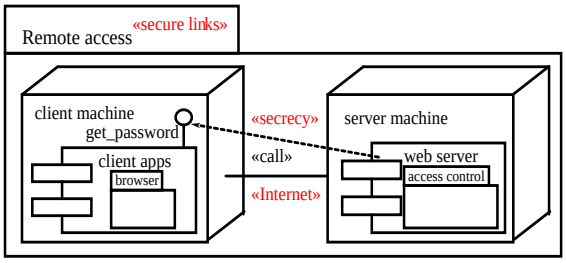
≪secure links≫

Ensures that physical layer meets security requirements on communication.

Constraint: for each dependency *d* with stereotype $s \in \{\ll secrecy \gg, \ll integrity \gg\}$ between components on nodes $n \neq m$, have a communication link *l* between *n* and *m* with stereotype *t* such that

- if $s = \ll secrecy \gg$: have read $\notin Threats_A(t)$.
- if $s = \ll integrity \gg$: have insert $\notin Threats_A(t)$.

Example ≪secure links≫



Given default adversary type, is ≪secure links≫ provided ?

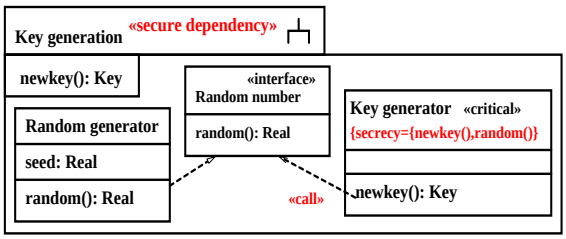
≪secure dependency≫

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≫



≪secure dependency≫ provided ?

≪no down–flow≫

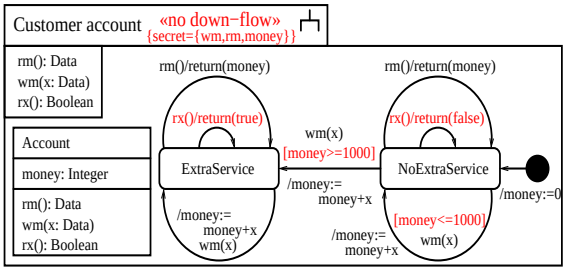
Enforce secure information flow.

Constraint:

Value of any data specified in {secrecy} may influence only the values of data also specified in {secrecy}.

Formalize by referring to formal behavioural semantics.

Example ≪no down-flow≫



≪no down–flow≫ provided ?

≪data security≫

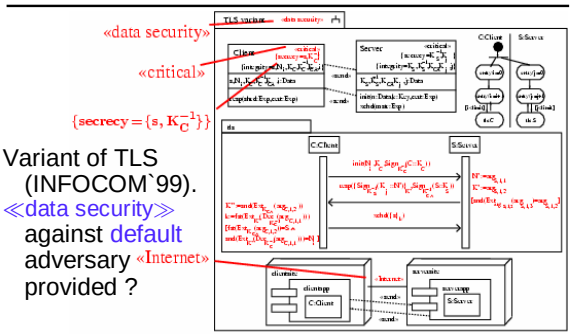
Security requirements of data marked ≪critical≫ enforced against threat scenario from deployment diagram.

Constraints:

Secrecy of {secrecy} data preserved.

Integrity of {integrity} data preserved.

Example ≪data security≫



Variant of TLS (INFOCOM`99).

≪data security≫ against default adversary «Internet» provided ?

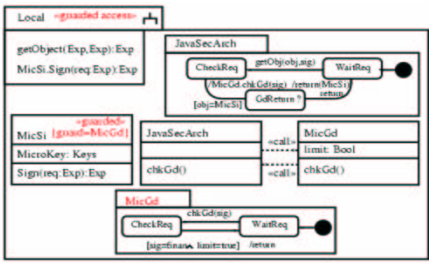
≪guarded access≫

Ensures that in Java, ≪guarded≫ classes only accessed through {guard} classes.

Constraints:

- References of ≪guarded≫ objects remain secret.
- Each ≪guarded≫ class has {guard} class.

Example ≪guarded access≫



Provides ≪guarded access≫:
Access to MicSi protected by MicGd.

Concepts covered by UMLsec

Security requirements: ≪secrecy≫,…

Threat scenarios: Use $\text{Threats}_{adv}(ster)$.

Security concepts: For example ≪smart card≫.

Security mechanisms: E.g. ≪guarded access≫.

Security primitives: Encryption built in.

Physical security: Given in deployment diagrams.

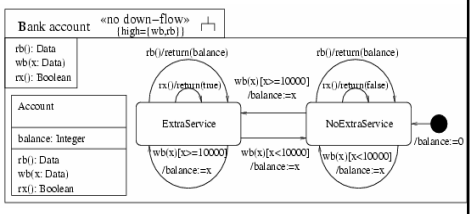
Security management: Use activity diagrams.

Technology specific: Java, CORBA security.

Security Patterns

Security patterns: use UML to encapsulate knowledge of prudent security engineering.

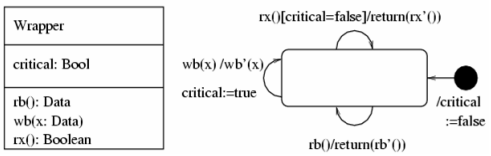
Example:



Does not preserve security of account balance.

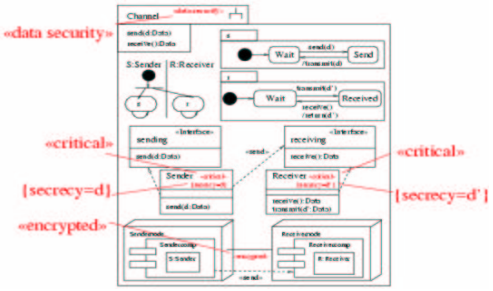
Solution: Wrapper Pattern

Technically, pattern application is transformation of specification.



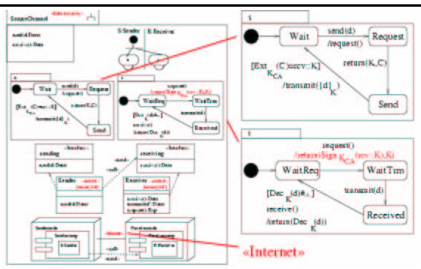
Use wrapper pattern to ensure that no low read after high write.
Can check this is secure (once and for all).

Secure channel pattern: problem



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

Secure channel pattern: (simple) solution



Exchange certificate and send encrypted data over Internet.

Roadmap

Prologue
UML
UMLsec: The profile

Security analysis
Using Java security, CORBAsec
Case studies
UML 2.0, Testing, Tools

Security Analysis

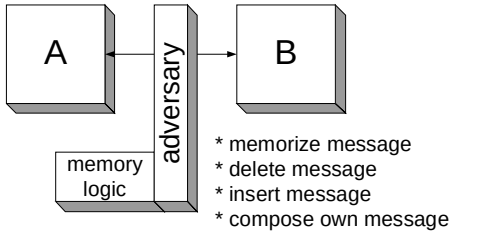
Model classes of adversaries.

May attack different parts of the system according to threat scenarios.

Example: insider attacker may intercept communication links in LAN.

To evaluate security of specification, simulate jointly with adversary model.

Abstract adversary



Security Analysis II

Keys are symbols, crypto-algorithms are abstract operations.

- Can only decrypt with right keys.
- Can only compose with available messages.
- Cannot perform statistical attacks.

Expressions

Exp: term algebra generated by Var ∪ Keys ∪ Data and

- $\_ :: \_$ (concatenation) and empty expression $\mathcal{E}$,
- $\{\_\}\_$ (encryption)
- *Dec ( )* (decryption)
- *Sign ( )* (signing)
- *Ext_( )* (extracting from signature)
- *Hash(_)* (hashing)

by factoring out the equations $Dec_{K^{-1}}(\{E\}_k) = E$ and $Ext_K(Sign_{K^{-1}}(E)) = E$ (for $K \in$ Keys).

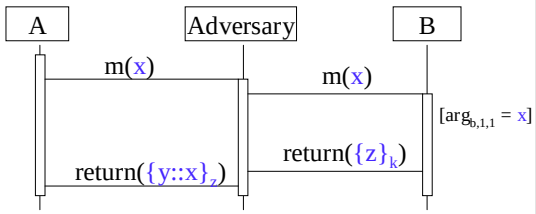
Abstract adversary

Specify set K_A^0 of initial knowledge of an adversary of type *A*. Let K_A^{n+1} be the Exp-subalgebra generated by K_A^n and the expressions received after *n+1*st iteration of the protocol.

Definition (Dolev, Yao 1982).

S keeps secrecy of *M* against attackers of type *A* if there is no *n* with $M \in K_A^n$.

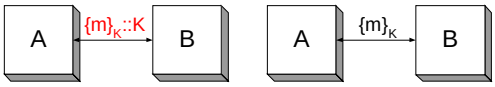
Adversary: Simulation



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

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

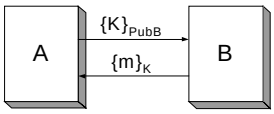
Example: secrecy



Against attacker who can read messages:

- Security of $\{m\}_K::K$ not preserved
- Security of $\{m\}_K$ preserved

Example: secrecy



- Security of *m* is not preserved against an attacker who can delete and insert messages
- Security of *m* is preserved against an attacker who can listen, but not alter the link

Roadmap

Prologue

UML

UMLsec: The profile

Security analysis

Using Java security, CORBAsec

Case studies

UML 2.0, Testing, Tools

Java Security

Originally (JDK 1.0): sandbox.

Too simplistic and restrictive.

JDK 1.2/1.3: more fine-grained security control, signing, sealing, guarding objects, . . .)

BUT: complex, thus use is error-prone.

Java Security policies

Permission entries consist of:

- protection domains (i. e. URL's and keys)
- target resource (e.g. files on local machine)
- corresponding permissions (e.g. read, write, execute)

Signed and Sealed Objects

Need to protect integrity of objects used as authentication tokens or transported across JVMs.

A SignedObject contains an object and its signature.

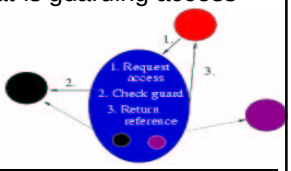
Similarly, need confidentiality.

A SealedObject is an encrypted object.

Guarded Objects

java.security.GuardedObject protects access to other objects.

- access controlled by getObject method
- invokes checkGuard method on the java.security.Guard that is guarding access
- If allowed: return reference. Otherwise: SecurityException



Problem: Complexity

- Granting of permission depends on execution context.
- Access control decisions may rely on multiple threads.
- A thread may involve several protection domains.
- Have method doPrivileged() overriding execution context.
- Guarded objects defer access control to run-time.
- Authentication in presence of adversaries can be subtle.
- Indirect granting of access with capabilities (keys).

→ Difficult to see which objects are granted permission.

⇒use UMLsec

Design Process

(1) Formulate access control requirements for sensitive objects.
(2) Give guard objects with appropriate access control checks.
(3) Check that guard objects protect objects sufficiently.
(4) Check that access control is consistent with functionality.
(5) Check mobile objects are sufficiently protected.

Reasoning

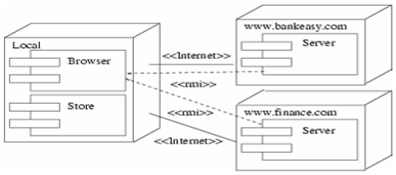
Theorem.

Suppose access to resource according to Guard object specifications granted only to objects signed with *K*.

Suppose all components keep secrecy of *K*.

Then only objects signed with *K* are granted access.

Example: Financial Application



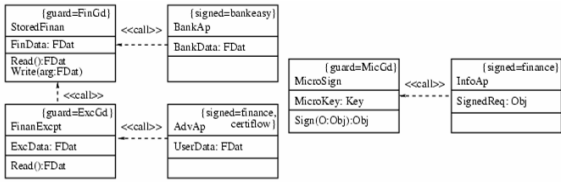
Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain Privileges (step1).

- Applets from and signed by bank read and write financial data between 1 pm and 2 pm.
- Applets from and signed by Finance use micropayment key five times a week.

Financial Application: Class diagram

Sign and seal objects sent over Internet for Integrity and confidentiality.

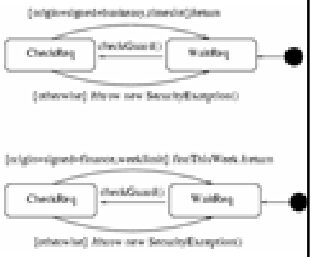
GuardedObjects control access.



Financial Application: Guard objects (step 2)

timeslot true between 1pm and 2pm.

weeklimit true until access granted five times; inc ThisWeek increments counter.



Financial Application: Validation

Guard objects give sufficient protection (step 3).

Proposition. UML specification for guard objects only grants permissions implied by access permission requirements.

Access control consistent with functionality (step 4). Includes:

Proposition. Suppose applet in current execution context originates from and signed by Finance. Use of micropayment key requested (and less than five times before). Then permission granted.

Mobile objects sufficiently protected (step 5), since objects sent over Internet are signed and sealed.

CORBA access control

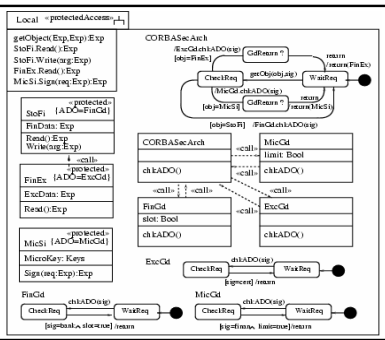
Object invocation access policy controls access of a client to a certain object via a certain method.

Realized by ORB and Security Service.

Use access decision functions to decide whether access permitted. Depends on

- called operation,
- privileges of the principals in whose account the client acts,
- control attributes of the target object.

Example: CORBA access control with UMLsec



Roadmap

Prologue

UML

UMLsec: The profile

Security analysis

Using Java security, CORBAsec

Case studies

UML 2.0, Testing, Tools

Layered Security Protocols

- Protocol on higher layer uses services of protocol on lower layer.
- Big question: security properties additive ?
- Desirable: secure channel abstraction.

client authenticity

confidentiality, integrity, server authenticity

=

confidentiality, … + client authenticity

?

Here: Bank 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.

The Application II

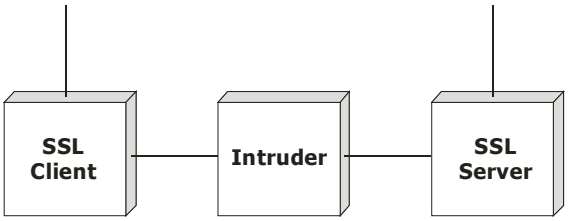
- Two layer architecture.
- When user logs on, an SSL-connection is established (first layer).
  - Provides secrecy, integrity, server authentication but no client authentication (this version).
- Custom-made protocol on top of SSL for client authentication.
- Session key generated by SSL used to encrypt messages on second layer.

SSL Protocol

Provided security services:

- Secure data transmission.
  - Integrity of data.
  - Confidentiality of data.
- Authentication of the server against the client.

Overview UML



Verification of the SSL-Protocol 1

- Authentication:
  It's not possible for the adversary to present himself as the bank's server against the client.

  $\neg(\mathbf{E}(\neg$Server in state *GotPreMasterSecret* $\mathbf{U}$ (Client in state *GotServerFinished*$)))$

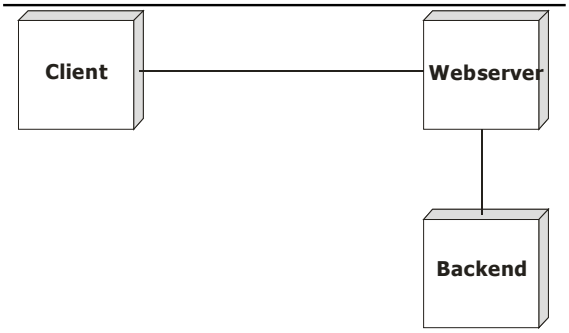
- Also: confidentiality.

Authentication protocol

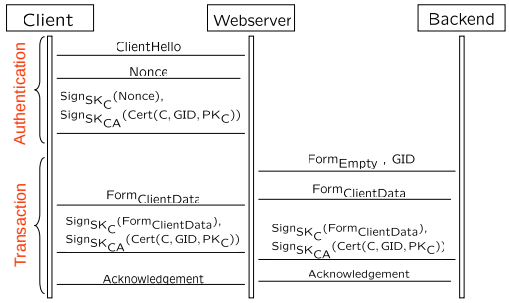
Provided security service:

- Authentication of the client against the bank's server.
- Was not provided by SSL because the underlying software did not support this feature.

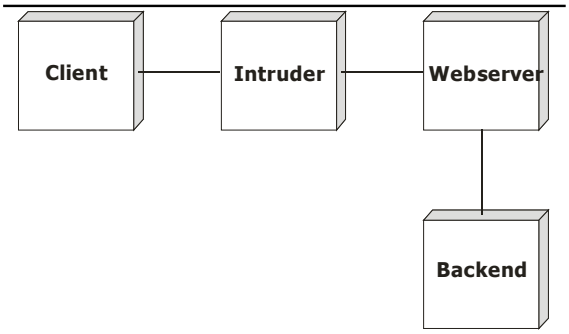
Overview UML



Authentication protocol



Overview UML



Layered Security Protocol

- Adjust adversary model to account for SSL security properties.
- Justify that specialised adversary model wrt. top-level protocol is as powerful as generic adversary wrt. protocol composition.
- Verify top-level protocol wrt. specialised adversary.
- Implies verification of protocol composition.

Verification of the Auth. protocol

- Authentication:
  - It's not possible for the adversary to authenticate under a wrong identity against the web server.

    $\neg(\mathbf{E}(\neg$Client in state *SentNonceCert* $\mathbf{U}$ (Webserver in state *GotSignedNonce* $\wedge$ nonce was signed Client)))

  - Used time: approx. 2 hours 40 minutes.

Verification of the Auth. protocol

- Explanation
  - "There does not exist any path ($\neg\mathbf{E}$)
  - where WebServer is in state *GotSignedNonce*
  - and the Nonce was signed by the client
  - if the Client has not been in state *SentNonceCert* before"
- In other words:
  - If the client didn't send the signed nonce, the Web Server couldn't have received the signed nonce.

Insight

Protocol layering indeed additive wrt. security properties in this particular case.

Generalize to classes of protocols and security requirements.

Further applications

- Variant of the Internet Protocol TLS
- Common Electronic Purse Specifications
- SAP access control configurations
- Biometric authentication system of German telecommunication company
- Automobile emergency application of German car company
- German health card
- Electronic signature application in insurances
- …

Roadmap

Prologue

UML

UMLsec: The profile

Security analysis

Using Java security, CORBAsec

Case studies

UML 2.0, Testing, Tools

Some new concepts in UML 2.0

UML extended with concepts from UML RT (Selic, Rumbaugh 1998).

Focus on software architecture.

New: capsules, ports, connectors.

Capsules, ports, connectors

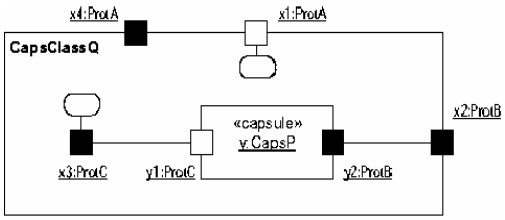
Capsules: architectural objects interacting through signal-based boundary objects (ports).

Port: object implementing interface of capsule. Associated with a protocol defining flow of information.

Connector: abstract signal-based communication channels between ports.

Functionality of capsule realized by associated state machine.

Example



From Selic, Rumbaugh 1998.

Tool-support: Test-generation

Two complementary strategies:

- Conformance testing
- Testing for criticality requirements

Conformance testing

Classical approach in model-based test-generation (much literature).

Can be superfluous when using code-generation [except to check your code-generator, but probably once and for all]

Works independently of criticality requirements.

Conformance testing: Problems

- Complete test-coverage usually infeasible. Need to somehow select test-cases.
- Can only test code against what is contained in the behavioral model. Usually, model is more abstract than code. So may have „blind spots“ in the code.

For both reasons, may miss critical test-cases.

Criticality testing

Shortcoming of classical model-based test-generation (conformance testing) motivates „criticality testing“ (e.g., papers by Jürjens, Wimmel at PSI’01, ASE’01, ICFEM’02).

Goal: model-based test-generation adequate for (security-, safety-) critical systems.

Criticality testing: Strategies

Strategies:

- Ensure test-case selection from behavioral models does not miss critical cases: Select according to information on criticality („internal“ criticality testing).
- Test code against possible environment interaction generated from external parts of the model (e.g. deployment diagram with information on physical environment).

Internal Criticality Testing

Need behavioral semantics of used specification language (precise enough to be understood by a tool).

Here: semantics for simplified fragment of UML in „pseudo-code“ (ASMs).

Select test-cases according to criticality annotations in the class diagrams.

Test-cases: critical selections of intended behavior of the system.

External Criticality Testing

Generate test-sequences representing the environment behaviour from the criticality information in the deployment diagrams.

Tool-support: Concepts

Meaning of diagrams stated informally in (OMG 2003).

Ambiguities problem for

- tool support
- establishing behavioral properties (safety, security)

Need precise semantics for used part of UML, especially to ensure security requirements.

Formal semantics for UML: How

Diagrams in context (using subsystems).

Model actions and internal activities explicitly.

Message exchange between objects or components (incl. event dispatching).

For UMLsec/safe: include adversary/failure model arising from threat scenario in deployment diagram.

Use Abstract State Machines (pseudo-code).

Tool-supported analysis

Choose drawing tool for UML specifications

Analyze specifications via XMI (XML Metadata Interchange)

skip compar.

Tool-supported analysis

Commercial modelling tools: so far mainly syntactic checks and code-generation.

Goal: more sophisticated analysis; connection to verification tools.

Several possibilities:

- General purpose language with integrated XML parser (Perl, …)
- Special purpose XML parsing language (XSLT, …)
- Data Binding (Castor; XMI: e.g. MDR)

Data-binding with MDR

MDR: MetaData Repository, Netbeans library (www.netbeans.org)

Extracts data from XMI file into Java Objects, following UML 1.4 meta-model.

Access data via methods.

Advantage: No need to worry about XML.

Framework for CSDUML tools: viki

Implements functionality
- – MDR wrapper
- – File handling
- – Properties management
- – Tool management

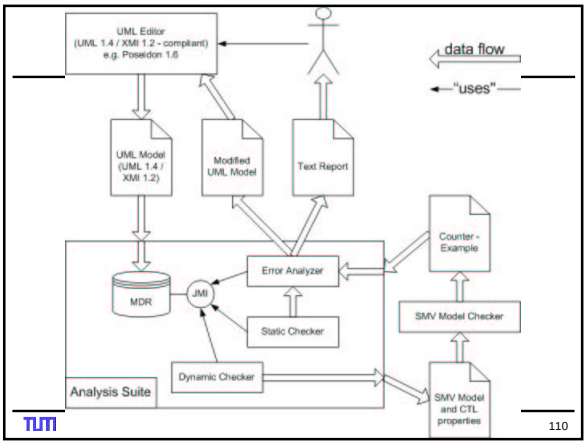
Exposes interfaces
- – IVikiFramework
- – IMdrWrapper
- – IAppSettings

viki Tool

- Works in GUI and/or Text mode
- Implements interfaces
  - – IVikiToolCommandLine
    - Text output only
  - – IVikiToolGui
    - Output to JPanel + menu, buttons, etc
- Exposes set of commands
  - – Automatically imported by the framework

Implementing tools

Exposes a set of commands.
Has its internal state (preserved between command calls).
Every single command is not interactive (read user input only at the beginning).
Framework and analysis tools accessible and available at http://www4.in.tum.de/~umlsec .
Upload UML model (as .xmi file) on website. Analyse model for included criticality requirements. Download report and UML model with highlighted weaknesses.



Connection with analysis tool

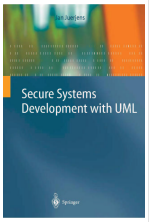
Industrial CASE tool with UML-like notation:
AUTOFOCUS (http://autofocus.informatik.tu-muenchen.de)

- Simulation
- Validation (Consistency, Testing, Model Checking)
- Code Generation (e.g. Java, C, Ada)
- Connection to Matlab

Connect UML tool to underlying analysis engine.

Some resources

Book: Jan Jürjens, Secure Systems Development with UML, Springer-Verlag, 2004
Tutorials: Aug.: WCC (Toulouse). Sept.: SAFECOMP (Potsdam), ASE (Linz), Oct.: UML (Lisbon).
Summer School Lecture: FOSAD (Bertinoro, Italy, Sept.)
Workshop: CSDUML@UML04



More information (papers, slides, tool etc.):
http://www4.in.tum.de/~juerjens/csdumltut
(user Participant, password Iwasthere)

Finally

We are always interested in industrial challenges for our tools, methods, and ideas to solve practical problems.
More info: http://www4.in.tum.de/~secse

Contact me here or via Internet.

Thanks for your attention !