3.3: UMLsec

v. 26.06.2013
3.3 UMLsec

Literatur:
Unibibliothek (e-Book): http://www.ub.tu-dortmund.de/katalog/titel/1361890
Papier-Version: http://www.ub.tu-dortmund.de/katalog/titel/1091324
• Kapitel 4.1
Einordnung
3.3 UMLsec

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Model-Driven Architecture
  - Sicherheitsanforderungen
  - UMLsec
  - UML-Analysis
  - Design Principles
  - Examples
    - TLS Variant
    - CEPS Purchase
Introduction of UMLsec

- **UML extension UMLsec**
  - allows to express security-related information within diagrams in UML system specification.
  - in form of a UML profile using the standard UML extension mechanisms.

- **Stereotypes and tags:** used to formulate security requirements and assumptions on the system environment.

- **Constraints**
  - give criteria that determine whether the requirements are met by the system design, by referring to the execution semantics.
  - can be checked automatically using tool support\(^1\).

---

\(^1\) Jan Jürjens, Secure Systems Development with UML, Springer 2004. Chap 6
Outline

- List requirements on a UML extension for secure systems development.
- Discuss how far our extension meets these requirements.
- Explain details of the extension by means of examples.
- Demonstrate the usefulness of the extension
  - enforcing established rules of secure systems design
  - indicate with an example how one could use UMLsec to apply security patterns.

- Formulate necessary properties of an UML extension for secure systems development.
  - Like the OMG Requests for Proposals (RFPs): distinguish mandatory and optional requirements.

Main **mandatory requirements:**

- Provide basic **security requirements** such as secrecy, integrity, authenticity.
- Allow considering different **threat scenarios** depending on adversary strengths.
- Allow including important **security concepts** (e.g. *tamper-resistant hardware*).
- Allow incorporating **security mechanisms** (e.g. access control).
- Provide **security primitives** (e.g. (a)symmetric encryption).
- Allow considering underlying **physical security**.
- Allow addressing **security management** (e.g. secure workflow).

The **optional requirement:**

- Include **technology-specific** security concepts (Java, smart cards, CORBA, …)
Note:

- **Goal:** not to aim for completeness by including all kinds of security properties as primitives.
- **Focus** on those that have a comparatively intuitive and universally applicable formalization, such as secrecy, integrity, and message authentication.
- **Other properties,** such as entity authenticity, have meanings that depend more on the context of their specific use.
  - Can be added by more sophisticated users on-the-fly.
Add security-relevant information to UML model elements.

Define labels for UML model elements:

- called stereotypes.

Different stereotypes available:

- Security assumptions on the physical level of the system, such as stereotype `<<Internet>>`.
- Security requirements on the logical structure of the system or on specific data values, such as stereotypes `<<secrecy>>`, `<<critical>>`.
- Security policies that system parts are supposed to obey, such as stereotypes `<<fair exchange>>`, `<<secure links>>`, `<<data security>>`, `<<no down – flow>>`. 
UMLsec: General Ideas (2)

- Activity diagram:
  - secure control flow, coordination
- Class diagram:
  - exchange of data preserves security levels
- Sequence diagram:
  - security-critical interaction
- Statechart diagram:
  - security preserved within object
- Deployment diagram:
  - physical security requirements
- Package:
  - holistic view on security
Give profile following the structure in [UML03]:

- **Applicable Subset:** Profile concerns all of UML.

- **Stereotypes, Tagged Values, and Constraints:**
  - List of stereotypes from UMLsec, their tags and constraints and corresponding tags (all DataTags).
  - The stereotypes do not have parents.
  - Concepts apply both to type and instance level.
  - For simplicity focus on the instance level
  - By "subsystem" we mean, more precisely, "subsystem instance".

- **UMLsec requires no prerequisite profiles.**

---

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Tags</th>
<th>Constraints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fair exchange</td>
<td>subsystem</td>
<td>start, stop, adversary</td>
<td>after start eventually reach stop</td>
<td>enforce fair exchange</td>
</tr>
<tr>
<td>provable</td>
<td>subsystem</td>
<td>action, cert, adversary</td>
<td>action is non-deniable</td>
<td>non-repudiation requirement</td>
</tr>
<tr>
<td>rbac</td>
<td>subsystem</td>
<td>protected, role, right</td>
<td>only permitted activities executed</td>
<td>enforces role-based access control</td>
</tr>
<tr>
<td>Internet</td>
<td>link</td>
<td></td>
<td></td>
<td>Internet connection</td>
</tr>
<tr>
<td>encrypted</td>
<td>link</td>
<td></td>
<td></td>
<td>encrypted connection</td>
</tr>
<tr>
<td>LAN</td>
<td>link</td>
<td>node</td>
<td></td>
<td>LAN connection</td>
</tr>
<tr>
<td>wire</td>
<td>link</td>
<td></td>
<td></td>
<td>wire</td>
</tr>
<tr>
<td>smart card</td>
<td>node</td>
<td></td>
<td></td>
<td>smart card node</td>
</tr>
<tr>
<td>POS device</td>
<td>node</td>
<td></td>
<td></td>
<td>POS device</td>
</tr>
<tr>
<td>issuer node</td>
<td>node</td>
<td></td>
<td></td>
<td>issuer node</td>
</tr>
<tr>
<td>secrecy</td>
<td>dependency</td>
<td></td>
<td></td>
<td>assumes secrecy</td>
</tr>
<tr>
<td>integrity</td>
<td>dependency</td>
<td></td>
<td></td>
<td>assumes integrity</td>
</tr>
<tr>
<td>high</td>
<td>dependency</td>
<td>secrecy, integrity,</td>
<td>dependency security matched by links «call», «send»</td>
<td>enforces secure communication links</td>
</tr>
<tr>
<td>critical</td>
<td>object,</td>
<td>authenticity, high,</td>
<td>respect data security</td>
<td>structural interaction data security</td>
</tr>
<tr>
<td></td>
<td>subsystem</td>
<td>fresh adversary</td>
<td></td>
<td>basic data security requirements</td>
</tr>
<tr>
<td>secure links</td>
<td>subsystem</td>
<td></td>
<td></td>
<td>information flow condition</td>
</tr>
<tr>
<td>secure dependency</td>
<td>subsystem</td>
<td></td>
<td></td>
<td>information flow condition</td>
</tr>
<tr>
<td>data security</td>
<td>subsystem</td>
<td></td>
<td></td>
<td>information flow condition</td>
</tr>
<tr>
<td>no down-flow</td>
<td>subsystem</td>
<td></td>
<td>prevents down-flow</td>
<td>information flow condition</td>
</tr>
<tr>
<td>no up-flow</td>
<td>subsystem</td>
<td></td>
<td>prevents up-flow</td>
<td>information flow condition</td>
</tr>
<tr>
<td>guarded access</td>
<td>subsystem</td>
<td></td>
<td>guarded objects accessed through guards</td>
<td>information flow condition</td>
</tr>
<tr>
<td>guarded</td>
<td>object</td>
<td>guard</td>
<td></td>
<td>guarded object</td>
</tr>
<tr>
<td>Tag</td>
<td>Stereotype</td>
<td>Type</td>
<td>Multip.</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>---------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>start</td>
<td>fair exchange</td>
<td>state</td>
<td>*</td>
<td>start states</td>
</tr>
<tr>
<td>stop</td>
<td>fair exchange</td>
<td>state</td>
<td>*</td>
<td>stop states</td>
</tr>
<tr>
<td>adversary</td>
<td>fair exchange</td>
<td>adversary model</td>
<td>1</td>
<td>adversary type</td>
</tr>
<tr>
<td>action</td>
<td>provable</td>
<td>state</td>
<td>*</td>
<td>provable action</td>
</tr>
<tr>
<td>cert</td>
<td>provable</td>
<td>expression</td>
<td>*</td>
<td>certificate</td>
</tr>
<tr>
<td>adversary</td>
<td>provable</td>
<td>adversary model</td>
<td>*</td>
<td>adversary type</td>
</tr>
<tr>
<td>protected</td>
<td>rbac</td>
<td>state</td>
<td>*</td>
<td>protected resources</td>
</tr>
<tr>
<td>role</td>
<td>rbac</td>
<td>(actor, role)</td>
<td>*</td>
<td>assign role to actor</td>
</tr>
<tr>
<td>right</td>
<td>rbac</td>
<td>(role, right)</td>
<td>*</td>
<td>assign right to role</td>
</tr>
<tr>
<td>secrecy</td>
<td>critical</td>
<td>data</td>
<td>*</td>
<td>secrecy of data</td>
</tr>
<tr>
<td>integrity</td>
<td>critical</td>
<td>(variable,</td>
<td></td>
<td>integrity of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>expression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>authenticity</td>
<td>critical</td>
<td>(data, origin)</td>
<td>*</td>
<td>authenticity of data</td>
</tr>
<tr>
<td>high</td>
<td>critical</td>
<td>message</td>
<td>*</td>
<td>high-level message</td>
</tr>
<tr>
<td>fresh</td>
<td>critical</td>
<td>data</td>
<td>*</td>
<td>fresh data</td>
</tr>
<tr>
<td>adversary</td>
<td>secure links</td>
<td>adversary model</td>
<td>1</td>
<td>adversary type</td>
</tr>
<tr>
<td>adversary</td>
<td>data security</td>
<td>adversary model</td>
<td>1</td>
<td>adversary type</td>
</tr>
<tr>
<td>integrity</td>
<td>data security</td>
<td>(variable,</td>
<td></td>
<td>integrity of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>expression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>authenticity</td>
<td>data security</td>
<td>(data, origin)</td>
<td>*</td>
<td>authenticity of data</td>
</tr>
<tr>
<td>guard</td>
<td>guarded</td>
<td>object name</td>
<td>1</td>
<td>guard object</td>
</tr>
</tbody>
</table>
Stereotypes and tags in more detail.

- Constraints use security-aware interpretation of UML diagrams.
- \(<\langle \text{fair exchange}\rangle, \langle\text{provable}\rangle, \langle\text{secure links}\rangle, \langle\text{data security}\rangle\>\):
  - parameterized over adversary type w.r.t. which the security requirements should hold.
- \(\{\text{adversary}\}\): values of the form \((T;C)\).
  - \(T\): Adversary type, such as \(T = \text{default}\) for the adversary defined later, which may also be self-defined.
    - If omitted \(T = \text{default}\).
  - \(C\): Logical condition on the previous knowledge \(K_p^A\) of the adversary\(^1\).
    - If omitted \(C\) ensures that data included in \(\{\text{secrecy}\}\) tag of \(\langle\text{critical}\rangle\) does not appear as subexpressions in \(K_p^A\).
- \(a^*\) represents an arbitrary multiplicity of a tag.

---

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.4
Well-formedness Rules

- Constraints associated with stereotypes:
  - give a range from structural syntactic conditions,
    - such as \texttt{<<secure links>>},
  - to relatively deep semantic conditions,
    - such as \texttt{<<no down-flow>>}.
  - advantage:
    - first find violations against simpler structural conditions, then
      analyse the behavioral part of the specification
    - automated mechanical verification is also available\(^1\)

- Seems to be more efficient than trying to establish the overall security
  all at once.

- Industrial setting: allows a scaling of the necessary costs.

Examples for usage of stereotypes

- Examples are just for illustration.
  - No formal proofes for stated properties.
  - Only essential fragments of subsystems of stereotype in question.

- Substantial case-studies for performing security analyses with UMLsec will be discussed in later section.
fair exchange
(for use case diagrams)

- Transactions should be performed in a way that prevents both parties from cheating.
- Applicable to subsystems containing a use case diagram.
  - Can be refined by another subsystem only if that is also stereotyped `<<fair exchange>>`.
- Only informal meaning, as opposed to the stereotypes below.
  - “refinement" is meant here in an informal sense.
- Shows how security requirements (as stereotypes) in other kinds of diagrams below can also conveniently be included in use case diagrams.
Use case diagram describing the following situation:

- a customer buys a good from a business.
- trade should be performed in a way that prevents both parties from cheating.
  - Add requirement by adding «fair exchange» to the subsystem containing the use case diagram.

Capture security requirements in use case diagrams.
- Constraint: need to appear in corresponding activity diagram.
<<fair exchange>> applied to subsystems containing an activity diagram

- associated tags \{start\}, \{stop\}, \{adversary\}.
- \{start\}, \{stop\} take pairs \((\text{good}; \text{state})\) as values,
  - \text{good} is the name of a good to be sold, can be omitted if only one good is to be sold
  - \text{state} is the name of a state.
- \{adversary\} adversary type relative to which the security requirement should hold.
- for every \text{good} to be sold, whenever a \{start\} \text{state} in the activity diagram is reached, eventually a \{stop\} \text{state} will be reached, when the system is executed in presence of an adversary of the type \text{A} specified in \{adversary\}.
Example
<<fair exchange>>

Use case in more detail by giving the activity diagram.

- Customer buys goods from a business.
- Adversary type irrelevant
  - no communication structure specified
- How can fair exchange be enforced?
- Requirement <<fair exchange>> formulated by referring to the activities in the diagram.

```
Purchase <<fair exchange>>
{start={Pay}} {stop={Reclaim,Pick up}}
```

### Diagram
```
Customer
    ↓
Request good
    ↓
Pay
    ↓
Wait until delivery due
    └── undelivered
Reclaim
    └── delivered
Pick up
    ↓
Deliver
```

Business
Ensures generic fair exchange condition.

Constraint:

- Actions listed in `{start}`, `{stop}` should be linked
  - if one of the former is executed then eventually one of the latter will be.
- Formalized wrt. formal semantics of the used fragment of UML.
- Can be checked automatically.
Formalization \textbf{<<fair exchange>>}

Formalized for a given subsystem \textit{S}:

- \textit{S} fulfills the constraint of \textbf{<<fair exchange>>} with respect to adversary type \textit{A} if for every good to be sold following condition holds:
  - For every execution \textit{e} of \( [[S]]_A \) there exists number \( n \in \mathbb{N} \) such that for every sequence \( I_1, ..., I_n \) of input multi-sets there exists an execution \( e' \) which is an extension of \( e \) and then processes the inputs in \( I_1, ..., I_n \), such that there are at least as many \{\textit{stop}\} states in \( e' \) as there are \{\textit{start}\} states in \( e \), with respect to the relevant good.
Revisit example

<<fair exchange>>

<<fair exchange>> fulfilled?

Purchase «fair exchange»
{start={Pay}} {stop={Reclaim,Pick up}}

Customer

Request good

Pay

Wait until delivery due

undelivered delivered

Reclaim

Pick up

Business

Deliver
Revisit example
<<fair exchange>>

<<fair exchange>> fulfilled:

- After payment:
  - customer is able to either pick up the delivery or reclaim the payment.

Can't be ensured for systems which an attacker can stop completely.
A subsystem $S$ may be labeled $<<\text{provable}>>$.

Tags: $\{\text{action}\}$,$\{\text{cert}\}$, and $\{\text{adversary}\}$.

- $\{\text{cert}\}$ contains an expression
  - proof that the action at the state in $\{\text{action}\}$ was performed.
- $\{\text{adversary}\}$ specifies an adversary type relative to which the security requirement should hold.

$S$ may output expression $E \in \text{Exp}$ in $\{\text{cert}\}$ only after the state in $\{\text{action}\}$ is reached, when executed in presence of an adversary of the type $A$ specified in $\{\text{adversary}\}$.

- Here certificate in $\{\text{cert}\}$ is unique for each subsystem instance.
More formally: \( S \) fulfills the constraint if the following holds for adversary type \( A \):

\[
\text{for (execution } e \text{ of } [[S]]_A) \{ \\
    \text{if (expression in } \{\text{cert}\} \text{ is given as output at a state } S \text{ in } e) } \\
    \text{then } \{ \text{state in } \{\text{action}\} \text{ appears as current state before } S \text{ in } e \}. \\
\}
\]

To avoid illegitimate repayment claims, in \(<\text{fair exchange}>\) example:

- Employ \(<\text{provable}>\) with regard to state Pay.
- Ensure that Reclaim payment action checks whether Customer can provide proof of payment.
role-based access control
<<rbac>>

- Applicable to subsystems containing activity diagram
- Enforces role-based access control in the business process specified in the activity diagram.

Tags: {protected}, {role}, and {right}.

- {protected} contains states in the activity diagram, to which the access should be controlled.
- {role} list of pairs (actor; role)
  - actor actor in activity diagram, role is a role.
- {right} has a list of pairs (role; right)
  - role is a role
  - right represents the right to access a protected resource.

Requires that actors in the activity diagram only perform activities for which they have the appropriate rights.
For a subsystem $S$, this is formalized as follows:

- For every actor $A$ in $S$ and every activity $a$ in swimlane of $A$ in the activity diagram in $S$, there exists a role $R$ such that $(A;R)$ is a value of $\{\text{role}\}$ and $(R; a)$ is a value of $\{\text{right}\}$.
Example: Role-based access control ($\langle\langle\text{rbac}\rangle\rangle$)

- Simplified part of a business process
  - credit is being set up for a customer of a bank.
- Bank employees have the right to set up credits.
- For large credits > e.g.10.000, supervisors have to authorize the credit before money is transferred.
- Protected resource: authorize credit activity
  - Supervisor, in her role of credit approver, has appropriate permission
- Diagram is correctly labeled $\langle\langle\text{rbac}\rangle\rangle$
  - the associated constraint is respected.
Example: Role-based access control (<<rbac>>) 

- Example: Instance of the security principle of separation of privilege.
- Ensure that employee is not assigned two roles with associated privileges that are supposed to be separated.
- How to link access control to the level of the technical security architecture is demonstrated using the stereotype <<guarded access>>.
Communication Architecture (1)

- Internet, encrypted, LAN, wire, smart card, POS device, issuer node
  - On links (resp. nodes) in deployment diagrams: denote the respective kinds of communication links (resp. system nodes).
- Require that each link or node carries at most one of these stereotypes.
- For each adversary type $A$, we have a function $\text{Threats}_A(s)$ from
  
  $$s \in \{<<\text{wire>>}; <<\text{encrypted>>}; <<\text{LAN>>}; <<\text{smart card>>}; <<\text{POS device>>}; <<\text{issuer node>>}; <<\text{Internet>>}\}$$

  to a set of strings

  $$\text{Threats}_A(s) \subseteq \{\text{delete}; \text{read}; \text{insert}; \text{access}\}$$

  - node stereotype $s$: $\text{Threats}_A(s) \subseteq \{\text{access}\}$
  - link stereotype $s$: $\text{Threats}_A(s) \subseteq \{\text{delete}; \text{read}; \text{insert}\}$. 
Communication Architecture (2)

Threats_A(s) specifies which kinds of actions an adversary of type \( A \) can apply to nodes or links stereotyped \( s \).

Given UML subsystem \( S \), function Threats_A(s) gives rise to

- \( \text{threats}^A_A(x) \)
  - takes a node or link \( x \) and a type of adversary \( A \)
  - returns set of strings \( \text{threats}^A_A(x) \subseteq \{\text{delete}; \text{read}; \text{insert}; \text{access}\}^2 \).

Evaluate UML subsystems using their execution semantics\(^1\), by referring to the security framework using UML Machine Systems\(^2\).

Examples for threat sets associated with some common adversary types are the default and insider attacker.

---

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.2
2 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.4
Communication Architecture (3)

Default attacker: outsider adversary with modest capability.

Ability:

- on an Internet link: read, delete, and insert messages.
- on an encrypted Internet link, (such as a virtual private network):
  - delete messages, without knowing their encrypted content, by bringing down a network server.
  - not able to read the plaintext messages or insert messages encrypted with the right key.
- Assume: encryption set up such that the adversary does not get hold of the secret key.
- No direct access to local area network (LAN) and therefore unable to eaves-drop on those connections, nor on wires connecting security-critical devices.
- Smart cards are assumed to be tamperresistant.
  - May not be against more sophisticated attackers.
- Unable to access POS devices or card issuer systems.
Communication Architecture (4)

- For adversary type $A$, *stereotype* $s$, has a set $\text{Threats}_A(s) \subseteq \{\text{delete, read, insert, access}\}$ of actions that adversaries are capable of.

- Default attacker: able to read, delete, insert and access messages on an Internet link.

Default attacker:

<table>
<thead>
<tr>
<th>Stereotypes</th>
<th>Threats$\text{default}(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>${\text{delete, read, insert}}$</td>
</tr>
<tr>
<td>encrypted LAN</td>
<td>${\text{delete}}$</td>
</tr>
<tr>
<td>smart card</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
- Insider attacker, in the context of the electronic purse system\(^1\).
- May access the encrypted Internet link.
  - knowing the corresponding key, and local system components.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Threats(_{\text{insider}}())</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>{delete, read, insert}</td>
</tr>
<tr>
<td>encrypted</td>
<td>{delete, read, insert}</td>
</tr>
<tr>
<td>LAN</td>
<td>{delete, read, insert}</td>
</tr>
<tr>
<td>wire</td>
<td>{delete, read, insert}</td>
</tr>
<tr>
<td>smart card</td>
<td>{access}</td>
</tr>
<tr>
<td>POS device</td>
<td>{access}</td>
</tr>
<tr>
<td>issuer node</td>
<td>{access}</td>
</tr>
</tbody>
</table>

\(^1\) Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 5.3
Dependencies

<<secrecy>>, <<integrity>>, <<high>>

- Used on dependencies in static structure or component diagrams.
- Denote dependencies supposed to provide respective security requirement for the data, sent along them as arguments, return values of operations or signals.
- Used in the constraint for the stereotype <<secure links>>.
Critical Data <<critical>>

- Labels objects or subsystem instances containing data that is critical.
- Tags: \{secrecy\}, \{integrity\}, \{authenticity\}, \{fresh\}, and \{high\}, representing the corresponding security requirements\(^1\).
- \{secrecy\} names of expressions, attributes or message argument variables of current object the secrecy of which is supposed to be protected; name of an operation is allowed to require that its arguments and return values should be kept secret.
- \{integrity\} has as values pairs \((v;E)\)
  - \(v\) variable of object whose integrity should be protected
  - \(E\) set of acceptable expressions that may be assigned to \(v\).

\(^1\) Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.1 and 3.3
Critical Data

- **{authenticity}** contains pairs \((a; o)\) of attributes of the \(<\text{critical}>\) object or subsystem
  - \(a\) stores the data whose authenticity should be provided and
  - \(o\) stores the origin of that data.
- **{fresh}** atomic data (elements of the set \(\text{Data} \cup \text{Keys}\)) that should be freshly generated.
- These constraints are enforced by the constraint of \(<\text{data security}>\) which labels subsystems that contain \(<\text{critical}>\) objects, as explained below.
- **{high}** names of messages that are supposed to be protected w.r.t. secure information flow, as enforced by \(<\text{no down-flow}>\) and \(<\text{no up-flow}>\).
- Synchronous operations: return messages required to be protected.
Secure Communication

- Together with the associated stereotypes <<secrecy>>, <<integrity>>, <<high>>, and <<critical>> one can describe different conditions for ensuring secure data communication with the following stereotypes:
  - <<secure links>>
  - <<secure dependencies>>
  - <<data security>>
Secure Communication

- **<<secure links>>**
  - Ensures that security requirements on the communication dependencies between components are supported by the physical situation, relative to the adversary model under consideration.

- **<<secure dependencies>>**
  - Ensures that the security requirements in different parts of a static structure diagram are consistent.

- **<<data security>>**
  - Ensures that security is enforced on the behavior level.

- One could for example merge the conditions of **<<secure links>>** and **<<secure dependencies>>** to give one stereotype.
Security at Architectural Level: Example

- Business application: part of an e-commerce system
- Supposed to be realized as web application.
- Payment transaction involves transmission of secret data over Internet links.
- "<secure links>" demands that security requirements on communication are met by physical layer.
- Architecture secure against default adversary?
**Stereotype**

<<secure links>>

- Remember $\text{threats}_{A}^{S}(x) \subseteq \{\text{delete; read; insert; access}\}$ with UML subsystem $S$, node or link $x$ and adversary $A$.

- Label subsystems containing static structure diagrams

- Ensures that physical layer meets security requirements on communication.

- Constraint enforces that for each dependency $d$ with stereotype $s \in \{<<\text{secrecy>>}, <<\text{integrity>>}, <<\text{high>>}\}$ between subsystems or objects on different nodes $m \neq n$, have a communication link $l$ between $m$ and $n$ such that:
  - If $s = <<\text{high>>}$ : have $\text{threats}_{A}^{S}(t) = \emptyset$
  - If $s = <<\text{secrecy>>}$ : have read $\notin \text{threats}_{A}^{S}(t)$
  - If $s = <<\text{integrity>>}$ : have insert $\notin \text{threats}_{A}^{S}(t)$
Revisit example

<<secure links>>

Remote access

Constraint for stereotype <<secure links>> fulfilled for default adversaries?
Constraint for stereotype <<secure links>> fulfilled for default adversaries?

- Intuitively: Internet connections do not provide secrecy against default adversaries.
- Technically: Constraint is violated because the dependency carries the stereotype <<secrecy>>, but for <<Internet>> of the corresponding link we have read $\in$ threats$_{\text{default}}$(Internet).
Security annotations consistent across class diagram?
**Stereotype**

**<<secure dependency>>**

- Labels subsystems containing static structure diagrams
- Ensures: **<<call>>** and **<<send>>** dependencies between components respect security requirements on communicated data given by {secrecy}, {integrity} of the stereotype **<<critical>>**.
- More exactly, Constraint enforced is that if there is a **<<call>>** or **<<send>>** dependency from an object or subsystem $C$ to an interface $I$ of an object or subsystem $D$ then the following conditions are fulfilled:
  - For any message name $n$ in $I$, $n \in \{\text{secrecy}\}$ (resp.$\{\text{integrity}\}$, $\{\text{high}\}$) in $C$ if and only if it does so in $D$.
  - If a message name in $I$ appears in $\{\text{secrecy}\}$ (resp. $\{\text{integrity}\}$, $\{\text{high}\}$) in $C$ then the dependency is stereotyped **<<secrecy>>** (resp. **<<integrity>>** resp. **<<high>>**).
Revisit example

<<secure dependency>>

Key generation

newkey(): Key

Random generator
seed: Real
random(): Real

<<interface>>
Random number
random(): Real

Key generator <<critical>>
{secrecy={newkey(),random()}}

<<call>>

:newkey(): Key

<<secure dependency>> fulfilled or not?
Revisit example

<<secure dependency>>

- Violates <<secure dependency>>:
  - Random generator and <<call>> dependency do not give security property \{secrecy\} for random() required by key generator.
Stereotype

<<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.
- Constraint associated with <<data security>> requires that these requirements are met w.r.t. the given adversary model.
- Formalization of this constraint discussed in detail in later section.
Stereotype <<data security>>

- Subsystem $S$ stereotyped <<data security>> respects data security requirements by the stereotypes <<critical>> and the associated tags contained in the subsystem w.r.t. the threat scenario arising from the deployment diagram and adversary type $A$ in {adversary}

More precisely: Constraint given by four conditions, which use the concepts of secrecy, integrity, authenticity, and freshness.

- secrecy: Subsystem preserves secrecy of data designated by {secrecy} against adversaries of type $A$.

- authenticity: For any $(a,o)$ of {authenticity}, $S$ provides the authenticity of the attribute $a$ w.r.t. its origin $o$ against adversaries of type $A$. 
Stereotype <<data security>>

- **integrity**: \{integrity\} of <<critical>> with a value \((v,E)\), the subsystem preserves the integrity of variable \(v\) against adversaries of type \(A\), w.r.t. \(E\) of admissible expressions.
  - If \(E\) is omitted, integrity of \(v\) should be preserved w.r.t. the set of expressions that can be constructed from those in the specification of \(S\).
  - Adversary should not be able to make the variable \(v\) take on a value previously known only to him.

- **freshness**: Within \(S\) stereotyped <<data security>>, any value \(d \in \text{Data} \cup \text{Keys}\) tagged \{fresh\} in the relevant subsystem instance or object \(D\) stereotyped <<critical>> in \(S\) should be fresh in \(D\).
Initial knowledge of the adversary may not contain the data values that, according to the tags of \texttt{<<critical>>}, should be guaranteed \textit{secrecy}, \textit{integrity} or \textit{authenticity}:

- Cannot be achieved if the adversary knows this data initially.
- Further assumptions on the initial adversary knowledge can be specified.
- If admissible expressions or the intended origin of data in \{\textit{integrity}\} and \{\textit{authenticity}\} refer to expressions not locally known at the \texttt{<<critical>>} object where these tags are applied, one can associate these tags with the relevant \texttt{<<data security>>} stereotype.
- Assume that standard adversary not able to break encryption, but can exploit design flaws e.g. in a crypto protocol, for example by attempting so-called „man-in-the-middle“ attacks.
Stereotype <<data security>>

Note:

- Enough for data to be listed with a security requirement in one of the objects or subsystems contained in the subsystem to be required to fulfill the conditions.

- Several nested subsystems may each carry <<data security>>.
  - The conditions are required to hold w.r.t. each of them. Important to note when including one subsystem in another.
Secure Use of Cryptography: Example

Variant of the Internet security protocol TLS proposed in [APS99]

Goal:

- Secure channel over an untrusted communication link between a client and a server.
  - Provide secrecy and server authenticity, as specified by the \{secrecy\} and \{authenticity\}.
- To achieve this, some of local attributes have to satisfy \{integrity\} as well.
  - The adversary should not be able to make these attributes take on a value in his previous knowledge.

Secure Use of Cryptography: Example

Variant of TLS (INFOCOM`99).

Goal is to exchange a secret session key $K$, using public keys $K_C$ and $K_S$, which is then used to encrypt the secret data $s$ before transmission.

Cryptoprotocol secure against default adversary?
Revisit example (Variant of TLS)

<<data security>>

Viologes \{\text{secrecy}\} of s against default adversary.

More details later.
Secure Use of Cryptography: Possible extension

- Properties of secrecy, integrity, and authenticity are taken relative to the considered type of adversary.

- Default adversary is a principal external to the system;

- Adversaries as part of the system under consideration are possible giving adversary access to the relevant system components.
  - by defining $\text{Threats}_{A}(s)$ to contain access for the relevant stereotype $s$.

- E.g.: e-commerce protocol involving customer, merchant, and bank
  - goods being purchased is a secret known only to the customer and merchant, and not the bank.
Secure Use of Cryptography: Possible extension

- Formulated by marking relevant data as “secret“ and by performing a security analysis relative to the adversary model “bank“.
  - Adversary is given access to the bank component by defining Threats() function in a suitable way.

- Note: Adversary does not necessarily have access to the input queue of the system.

- May be sensible, e.g. to apply {secrecy} to a value received by the system from the outside.

- Condition associated with <<data security>> only ensures that stereotyped component keeps the values received by the environment secret.

- Make sure that the environment of the system part under consideration does not make these values available to the adversary either.
Secure Information Flow

- Alternative way of specifying secrecy- and integrity-like requirements.
- Protection against partial flow of information.
- Can be more difficult, especially when handling with encryption.
- Assign to each piece of the system data one of two security levels:
  - high, meaning highly sensitive or highly trusted.
  - low, meaning less sensitive or less trusted.

Given a set of messages $H$ and a sequence $m$ of event multi-sets, we write:

- $m^H$ for the sequence of event multi-sets derived from those in $m$ by deleting all events the message names of which are not in $H$.
- $m_H$ for the sequence of event multi-sets derived from those in $m$ by deleting all events the message names of which are in $H$.
Secure Information Flow: Background

Definition: Given a subsystem $S$ and a set of high messages $H$, we say:

- $A$ prevents down-flow with respect to $H$ if for any two sequences $i; j$ of event multi-sets and any two output sequences $o \in [[S]]_A(i)$ and $p \in [[S]]_A(j)$, $i_H = j_H$ implies $o_H = p_H$ and

- $A$ prevents up-flow with respect to $H$ if for any two sequences $i; j$ of event multi-sets and any two output sequences $o \in [[S]]_A(i)$ and $p \in [[S]]_A(j)$, $i^H = j^H$ implies $o^H = p^H$. 
Secure Information Flow: Background

- Intuitively:
- Prevent down-flow: outputting a non-high (or low) message does not depend on high inputs.
  - rather stringent secrecy requirement for messages marked as high.
- Prevent up-flow: outputting a high value does not depend on low inputs.
  - stringent integrity requirement for messages marked as high.
- This notion is generalization of the original notion of non-interference for deterministic systems\(^1\) to system models that are non-deterministic because of underspecification\(^2\).

---


Secure Information Flow: Example (1)

- Secret attribute `money` containing the amount of money spent by a given customer.
  - Can be read by `rm()`: return value is also secret.
  - Increase `money` with operation `wm(x)`.
- When money exceeds 1000, goes into state `ExtraService`.
- Public operation `rx()` to check whether extra service should be provided.
Secure Information Flow

- Prevent the indirectly leakage out of any partial information about high data via non-high data, as specified by the stereotype <<no down-flow>>.
  - Enforces secure information flow by making use of {high} associated with <<critical>>.
- Intuitively: Value of any data specified in {secrecy} may influence only the values of data also specified in {secrecy}.
- More precisely, formalize by referring to formal behavioural semantics: Constraint for <<no down-flow>> (resp. <<no up-flow>>) is that UML machine \text{Exec}[[S]] for subsystem \text{S} prevents down-flow (resp. up-flow) with respect to messages specified in <<high>> and their return messages.
- E.g. for privacy reasons, it may be important that the observable information on the customer account allows no conclusion about the money spent.
Now we use the stereotype `<<no down-flow >>` to indicate that the object should not leak out any information about secret data, such as the money attribute.

No partial leakage of secrets?
Secure Information Flow: Example (3)

- <<no down-flow>> indicates that the object should not leak out any information about secret data, such as the money attribute.

- Violation of the constraint associated with <<no down-flow>>:
  - partial information about the input of the high operation \( \text{wm()} \) leaked out via the return value of the non-high operation \( \text{rx()} \).
How the underlying formalism captures the security flaw using the previous definition:

- sequences $i; j$ of input multi-sets
- sequences $o \in [[A]](i)$ and $p \in [[A]](j)$ of output multi-sets of the UML Machine $A$ giving the behavior of the considered statechart
- with $i_H = j_H$ and $o_H \neq p_H$, where $H$ is the set of high messages.

Consider the sequences

- $i := ([[\text{wm}(0)]) ; \{\text{rx()}\})$
- $j := ([[\text{wm}(1000)]) ; \{\text{rx()}\})$.

Secure Information Flow: Example (5)

Given \( i_H = (\{\} \), \{\text{rx}()\} ) = j_H. \)

Definition of the behavioral semantics of statecharts\(^1\), brings the output multi-sets:

- \( o := (\{\} ), \{\text{return}(\text{false})\} ) \in \text{[[A]]}(i). \)
- \( p := (\{\} ), \{\text{return}(\text{true})\} ) \in \text{[[A]]}(j). \)

\( \Rightarrow \)

- \( o_H = (\{\} ), \{\text{return}(\text{false})\} ) \neq (\{\} ), \{\text{return}(\text{true})\} ) = p_H \)

meaning that the constraint associated with "<no down-flow>" is violated.

Can be detected automatically with the tool support provided for UMLsec\(^2\).

---

\(^1\) Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.2

\(^2\) http://umlsec.de/
Guarded Objects
<<guarded access>>

- Each object in the subsystem stereotyped <<guarded>> can only be accessed through the objects specified by \{\text{guard}\} attached to the <<guarded>> object.

- Formally: assume \( \text{name} \notin K_p^A \) for adversary type \( A \) under consideration and each name \( \text{name} \) of an instance of a <<guarded>> object, meaning that a reference is not publicly available.

- Assume: for each <<guarded>> object there is a statechart specification of an object whose name is given in \{\text{guard}\}.

- To model passing of references.
Example:

- Illustration with a web-based financial application.
- Two institutions offer services over the Internet to local users:
  - Internet bank, Bankeasy
  - financial advisor, Finance.
- Use these services:
  - Local client needs to grant the applets certain privileges.
- Access to local financial data is realized using GuardedObjects.
Guarded Objects

<<guarded access>>

Simplified relevant part of Java Security Architecture

- Receives requests for object references
- Forwards them to the guard objects of the three guarded objects.
- <<guarded>> objects StoFi, FinEx, and MicSi can only be accessed through their associated guard.
  - Subsystem instance fulfills the condition associated with <<guarded access>> w.r.t. default adversaries.
Guarded Objects

Example

Access controls realized by Guard objects FinGd, ExpGd, and MicGd.

- Behavior is specified.

Applets signed by the bank

- Read and write the financial data stored in the local database, but only between 1 pm and 2 pm.
- Enforced by the FinGd guard object.
  - Condition slot is fulfilled if and only if the time is between 1 pm and 2 pm.
**Guarded**

<<guarded>>

- **<<guarded>>** labels objects in scope of **<<guarded access>>** that are supposed to be guarded.\(^1\)

**Tag:**
- `{guard}` name of the corresponding guard object.

- **<<guarded>>** objects
  - **StoFi**, **FinEx**, **MicSi**
  - protected by the `{guard}` objects **FinGd**, **ExpGd**, **MicGd** respectively.

---

\(^1\) Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 5.4
Does UMLsec Meet Requirements?

- Security requirements: Formalizations of basic security requirements provided via stereotypes, such as `<secrecy>`, etc.
- Threat scenarios: using the formal semantics and depending on the modeled underlying physical layer via the sets \( \text{Threats}_{\text{adv}}(\text{ster}) \) of actions available to the adversary of kind \( \text{adv} \).
- Security concepts: For example `<smart card>`.
- Underlying physical security:
  - Addressed by `<secure links>` in deployment diagrams.
- Security primitives:
  - Either built in, such as encryption, or
  - Can be treated, such as security protocols.
- Security managements: Use activity diagrams.
Summary: 3.3 UMLsec

- General Ideas
- Stereotypes
- Communication Architecture
- Critical Data
- Secure Communication
- Secure Information Flow