TUHH
Hamburg University of Technology

# Security Analysis of Software Design

**Riccardo Scandariato**
Institute of Software Security, TUHH, Germany
ric***do . scanda***to @ tuhh.de

**Master Course “Secure Software Engineering”**
**Summer Semester 2022**

# Learning objectives

- What are architectural weaknesses?
  - CAWE

**Reading material**
1) Mehdi Mirakhorli, Common Architecture Weakness Enumeration (CAWE), http://blog.ieeesoftware.org/2016/04/common-architecture-weakness.html
2) And also https://cwe.mitre.org/data/definitions/1008.html

- How to find architectural weaknesses with model-based security analysis?
  - Manual inspection vs automated checking (UMLsec)

**Reading material**
Jan Jürjens, *Model-Based Security Engineering with UML*,
Chapter 4 of the book "Secure Systems Development with UML"
Link: https://link.springer.com/chapter/10.1007/3-540-26494-9_4

# Goals of model-based security analysis

- Discover security/privacy issues early on, even before developing the code (forward engineering case)

- Certification of a system (also existing one)
  - Evidence that risks are identified
  - Evidence that ‘reasonable’ security mechanisms are in place

# Model Based Security Analysis

Why security analysis at the architecture level? Aren't security tests on the implementation enough?

- A different type of issues (than, e.g., implementation vulnerabilities)
  - Using Weak Authentication (e.g., "API keys")
  - Unprotected Storage of Credentials
  - Permission Re-delegation
  - Download code without integrity checking or origin authN
  - …
- Late detection and fixing of these flaws can be difficult / expensive (e.g., require refactoring)

# Architectural Flaws

- **Flaws of Omission.** Such design flaws result from ignoring a security requirement or potential threats.
- **Flaws of Commission.** Such design flaws refer to the design decisions which were made and could lead to undesirable consequences.
- **Flaws of Realization.** The design decision is correct but the implementation of that suffers from a coding mistake. Code vulnerabilities TERRITORY

- Common Architecture Weakness Enumeration (CAWE)
- *By Mehdi Mirakhorli*
- http://blog.ieeesoftware.org/2016/04/common-architecture-weakness.html

# Share your opinion

- What is wrong in the next slide?

# An example

**Listing 1** An example of an incorrect implementation of the tactic "Authenticate Actors" in a Web application written in PHP resulting in an authentication-bypass.

```
1   $auth = $_COOKIES['authenticated'];
2   if (!$auth) {
3       if (authenticate($_POST['username'], $_POST['password'])) {
4           // save the cookie to be sent out in future responses
5           setcookie('authenticated', '1', time()+60*60*2);
6       } else {
7           showLoginScreen(); // request user to login
8           die('\n'); // kill the process
9       }
10  }
11  performPrivilegedAction();
```

# Share your opinion

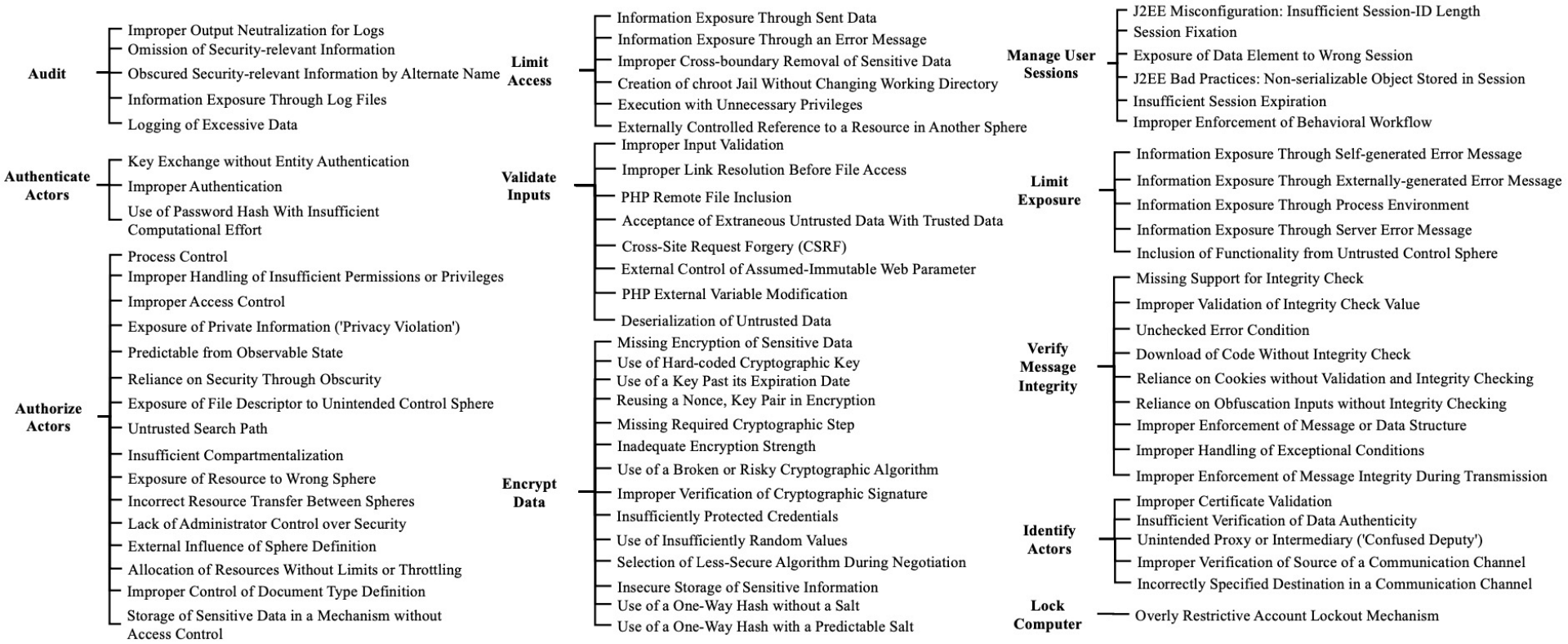
- Trust that the client will not add a bogus “authenticated” cookie
- Is that reasonable?
- Client-side authentication is very easy to break

- CAWE > Authenticate Actors - *(1010)* > Use of Client-Side Authentication - *(603)*

# CAWE

## Common Architectural Weakness Enumeration

Accepted as a “view” in CWE




Pls, look at this ➡ https://cwe.mitre.org/data/definitions/1008.html

# Model-based security analysis

- Inspection guidelines
- Algorithmically
  (e.g., model checking, pattern matching, etc.)
- Threat and risk analysis → Later in this course ;)

# Types of model-based security analysis

- Inspection guidelines
  (performed manually, possibly tool assisted)

**Design Flaw 1: Missing authentication**

**Description** This refers to the absence of an authentication mechanism in the system. Apart from external entities, like users or other systems the system may interact with, authentication may be necessary within the system between processes/components/datastores that are located in different trust boundaries.

**Detection**

- Consider the external entities (users/subsystems) that interact with the system and which assets of the system they can access.
- Determine the processes that interact with high-value assets in the system.
- For each interaction examine:
  - If it is an entity: Does the entity go through an authentication point in order to access the asset?
  - If it is a process: Is the identity of a process accessing datastores or processes in a different part of the system (trust boundaries – requires different privilege levels) verified?

K. Tuma et al., **Automating the Early Detection of Security Design Flaws**, MODELS 2020

- Benchmark is a 'trendier' term
  (see CIS – Center for Internet Security)
  - Also more focus on tool support for the rules

# Manual vs Automated Model Analysis

Accessing the software design/software architecture for detection of flaws

| Analysis | Advantages | Disadvantages |
|---|---|---|
| **Manual** | • Interpretation of improper representation of models<br>• Fewer false positives | • Time consuming<br>• Requires expertise<br>• No completeness guarantee |
| **Automated** | • Faster<br>• Easily re-executed if model changes | • Specific model with precise notation is required<br>• Effort to add additional info to models |

# UMLSec

# Unified Modeling Language (UML)

**UML:** Industry standard object oriented modeling technique
*Relatively* precisely defined
Widely adopted and accepted

**UML Diagrams**

Rich set of diagrams, covering a spectrum of abstractions (more/less detailed descriptions)

Visual representation of the architecture and detailed design of complex software systems

| UML Diagrams | |
|---|---|
| Structural Diagrams | Behavioral Diagrams |
| Class Diagram | Use Case Diagram |
| Component Diagram | Activity Diagram |
| Deployment Diagram | State Machine Diagram |
| Object Diagram | Sequence Diagram |
| Package Diagram | Communication Diagram |
| Profile Diagram | Interaction Overview Diagram |
| Composite Structure Diagram | Timing Diagram |

# UMLsec – Phylosophy

- Annotate design diagrams with various recurring security requirements (secrecy, integrity, authenticity…) and security assumptions
- Annotations as
  - Stereotypes
  - Tags
- Goal
  - Documentation / keep track of info
  - Formal semantics → tool-supported analysis

# UMLSec Extension Mechanisms

UML profile collects the relevant definitions of stereotypes, tagged values, and constraints

**Stereotypes** Define new sub-types of modelling elements, hence extending the UML metamodel. Stereotype definition can include zero or more tags

***Example: <<guarded>> can only be used on Objects***

**Tagged values:** Name-value pair that add properties to model elements. Can be used in the context of a stereotypes that defines them

***Example: {guard = obj} identifies the guard object***

Foo Boolean values, {tag} means {tag = true}

**Constraints** Define the formal semantics of a model element (e.g., written in first-order logic). That is, the desired security property.

***Example: "guarded objects only accessible via guard object"***

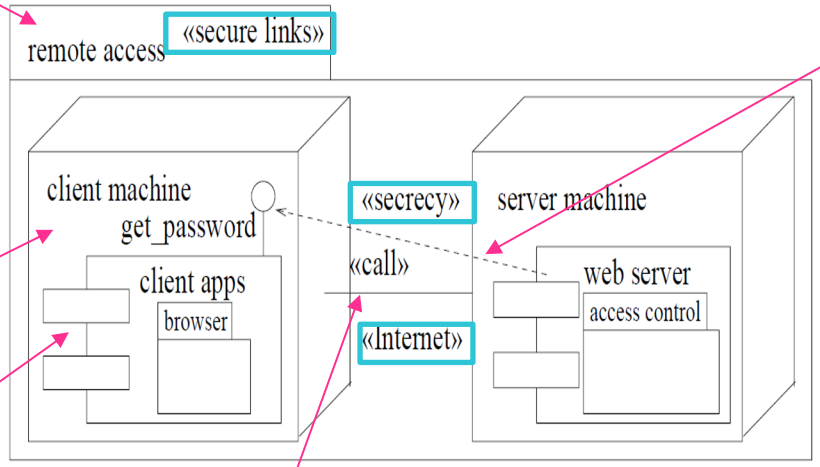
# Example

| Stereotype | Base Class | Tags | Constraints | Description |
|---|---|---|---|---|
| Internet | link | | | Internet connection |
| encrypted | link | | | encrypted connection |
| LAN | link,node | | | LAN connection |
| wire | link | | | wire |
| secure links | subsystem | | dependency security matched by links | enforces secure communication links |
| secrecy | dependency | | | assumes secrecy |
| integrity | dependency | | | assumes integrity |
| high | dependency | | | high sensitivity |

**Subsystem (think of package)**
Integrate the information between the different kinds of diagrams and between different parts of the system specification

Deployment diagram with

Security annotations




Dependency (between components)

Node

Component

Link (between nodes)

# UMLSec: usage scenarios
## (from more abstract to more concrete)

## *Usecase diagrams*

*Seen before*

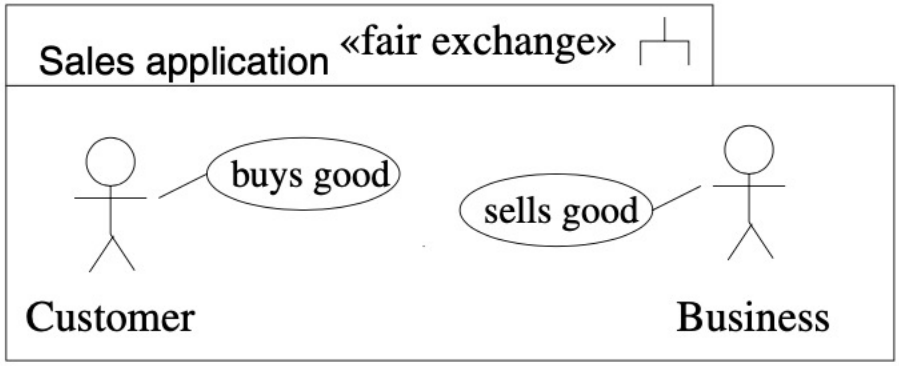



**Fig. 2.1.** Use case diagram for business application

**In UMLsec: Capture security requirements**

## *Activity diagrams*

*Specify the control flow between several components within the system, usually at a higher degree of abstraction than statecharts and sequence diagrams. They can be used to put objects or components in the context of overall system behaviour or to explain use cases in more detail.*

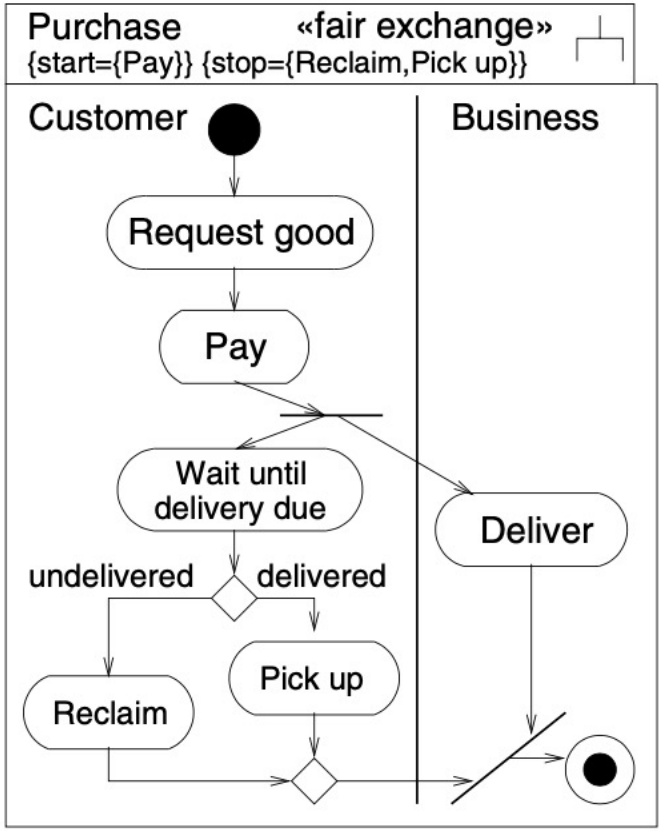



**Fig. 2.2.** Purchase activity diagram

**In UMLsec: Define secure business processes**

***Deployment diagrams***

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

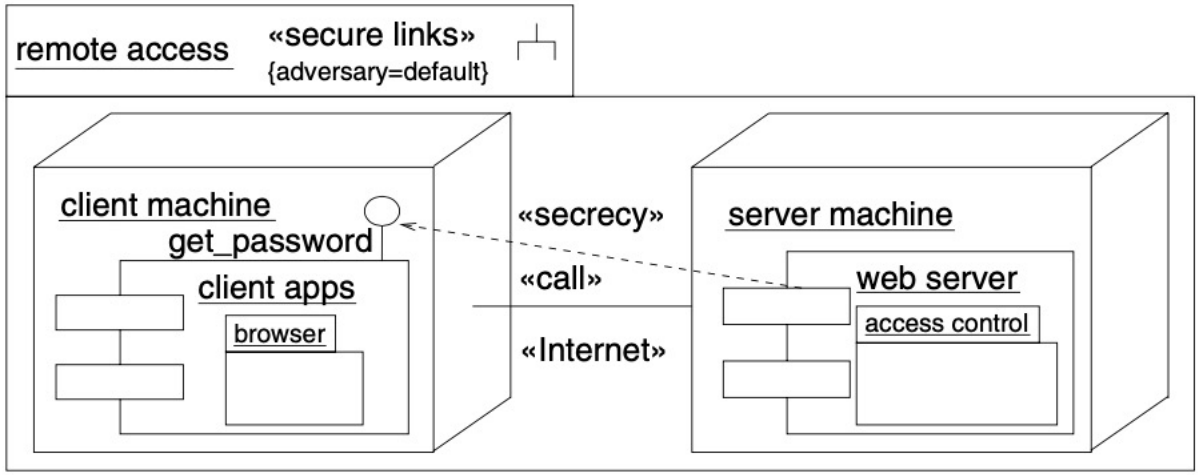



**Fig. 2.3.** Example *secure links* usage

**In UMLsec: Check physical security**

## *Sequence diagrams*

*Describe interaction between objects arranged in time sequence and also sequence of the messages exchanged.*

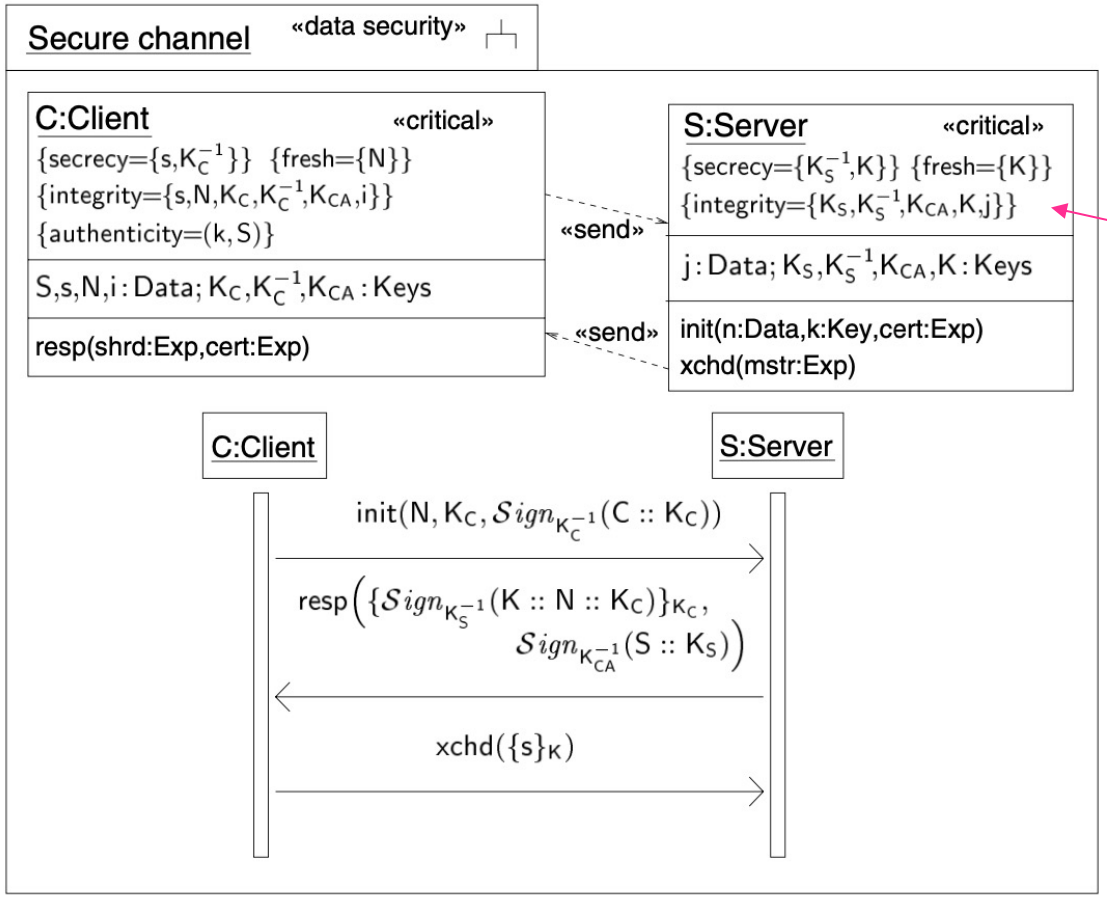



**Fig. 2.4.** Key exchange protocol

## *Class diagrams*

*Define the static class structure of the system: classes with attributes, operations, and signals and relationships between classes. On the instance level, the corresponding diagrams are called* ***object*** *diagrams.*

**In UMLsec: Define security critical interactions**

***Statechart diagrams***

*Give the dynamic behaviour of an individual object or component: events may cause a change in state or an execution of actions.*

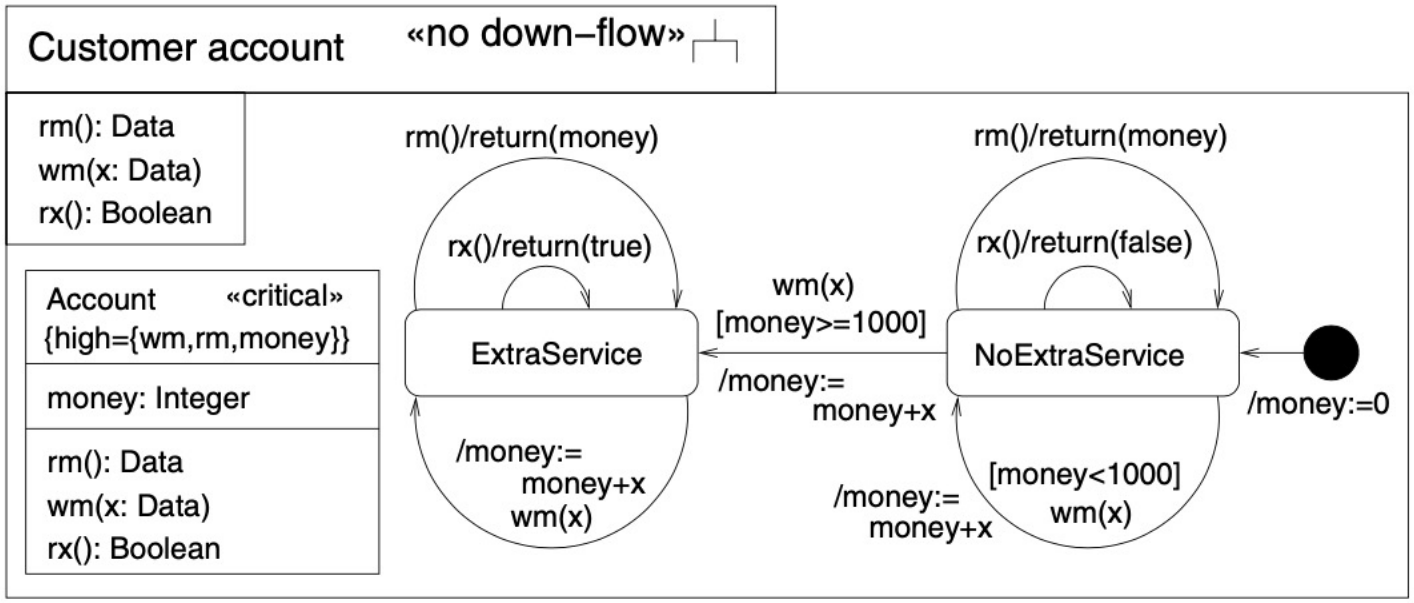



**Fig. 2.5.** Customer account data object

**In UMLsec: Information flow analysis**

Study at home

# UMLsec profile at a glance

| Stereotype | Base Class | Tags | Constraints | Description |
|---|---|---|---|---|
| fair exchange | subsystem | start, stop, adversary | after start eventually reach stop | enforce fair exchange |
| provable | subsystem | action, cert, adversary | action is non-deniable | non-repudiation requirement |
| rbac | subsystem | protected, role, right | only permitted activities executed | enforces role-based access control |
| Internet | link | | | Internet connection |
| encrypted | link | | | encrypted connection |
| LAN | link, node | | | LAN connection |
| wire | link | | | wire |
| smart card | node | | | smart card node |
| POS device | node | | | POS device |
| issuer node | node | | | issuer node |
| secrecy | dependency | | | assumes secrecy |
| integrity | dependency | | | assumes integrity |
| high | dependency | | | high sensitivity |
| critical | object, subsystem | secrecy, integrity, authenticity, high, fresh | | critical object |
| secure links | subsystem | adversary | dependency security matched by links | enforces secure communication links |
| secure dependency | subsystem | | «call», «send» respect data security | structural interaction data security |
| data security | subsystem | adversary, integ., auth. | provides secrecy, integrity, authenticity, freshness | basic data security requirements |
| no down-flow | subsystem | | prevents down-flow | information flow condition |
| no up-flow | subsystem | | prevents up-flow | information flow condition |
| guarded access | subsystem | | guarded objects accessed through guards | access control using guard objects |
| guarded | object | guard | | guarded object |

# Stereotypes 1/3

**Internet, encrypted LAN: Denote communication links**- Stereotypes on links in deployment diagrams denote the corresponding requirements on communication links nodes. Each link or node carries at most one of these stereotypes.

**Secure Dependency**

This stereotype, used to label subsystems containing object diagrams or static structure diagrams, ensures <<call>> or <<send>> dependencies respect the security requirements on the data that may be communicated along them, as given by the tags secrecy, integrity and high of the stereotype <<critical>>

**Secrecy, integrity, high**

Stereotypes denote dependencies in static structure or component diagrams that provide security requirement for the data that is sent as arguments or return values of operations or signals.

# Stereotypes 2/3

**Secrecy**

<<call>> or <<send>> dependencies in object or component diagrams stereotyped <<secrecy>> provide security requirement for the data that is sent as arguments or return values of operations or signals

Both are used in the constraint of the stereotype <<secure links>>

**Critical**

This stereotype labels objects or subsystem instances containing data that is critical in some way, which is specified in more detail using the tags secrecy, integrity, fresh and high.

**No down flow**

This stereotype of subsystems enforces secure information flow by making use of the associated tag high. According to the <<no-down flow>> constraint, the stereotyped subsystem prevents down-flow wrt messages and their return messages specified as high

# Stereotypes 3/3

**Fair exchange**

Tags *start* and *stop* whenever a start state in the activity diagram is reached, then eventually corresponding stop state will be reached.

**Provable**

Tags *action* and *cert* whenever a start state in the activity diagram is reached, then eventually corresponding stop state will be reached.

**Guarded Access**

Each object in the subsystem that is <<guarded>> can only be accessed through the objects specified by the tag guard attached
to <<guarded>> object.

**Guarded**

Labels objects (in particular in the scope of the stereotype <<guarded access>> above) that are supposed to be guarded. It has a tagged value guard which defines the name of the corresponding guard object.

Only for your info

# Summary of UMLsec tags

| Tag | Stereotype | Type | Multip. | Description |
|---|---|---|---|---|
| start | fair exchange | state | * | start states |
| stop | fair exchange | state | * | stop states |
| adversary | fair exchange | adversary model | 1 | adversary type |
| action | provable | state | * | provable action |
| cert | provable | expression | * | certificate |
| adversary | provable | adversary model | * | adversary type |
| protected | rbac | state | * | protected resources |
| role | rbac | (actor, role) | * | assign role to actor |
| right | rbac | (role, right) | * | assign right to role |
| secrecy | critical | data | * | secrecy of data |
| integrity | critical | (variable, expression) | * | integrity of data |
| authenticity | critical | (data, origin) | * | authenticity of data |
| high | critical | message | * | high-level message |
| fresh | critical | data | * | fresh data |
| adversary | secure links | adversary model | 1 | adversary type |
| adversary | data security | adversary model | 1 | adversary type |
| integrity | data security | (variable, expression) | * | integrity of data |
| authenticity | data security | (data, origin) | * | authenticity of data |
| guard | guarded | object name | 1 | guard object |

# Key security requirements

| Stereotype | Base Class | Tags | Constraints | Description |
|---|---|---|---|---|
| fair exchange | subsystem | start, stop, adversary | after start eventually reach stop | enforce fair exchange |
| provable | subsystem | | ble | non-repudiation requirement |
| rbac | subsystem | | executed | enforces role-based access control |
| Internet | link | | | Internet connection |
| encrypted | link | | | encrypted connection |
| LAN | link, node | | | LAN connection |
| wire | link | | | wire |
| smart card | node | | | smart card node |
| POS device | node | | | POS device |
| issuer node | node | | | issuer node |
| secrecy | dependency | | | assumes secrecy |
| integrity | dependency | | | assumes integrity |
| high | dependency | | | high sensitivity |
| critical | object, subsystem | secrecy, integrity, authenticity, high, fresh | | critical object |
| secure links | subsystem | adversary | dependency security matched by links | enforces secure communication links |
| secure dependency | subsystem | | « call », « send » respect data security | structural interaction data security |
| data security | subsystem | adversary, integ., auth. | provides secrecy, integrity, authenticity, freshness | basic data security requirements |
| no down-flow | subsystem | | | information flow condition |
| no up-flow | subsystem | | | information flow condition |
| guarded access | subsystem | | guarded objects accessed through guards | access control using guard objects |
| guarded | object | | | guarded object |

Non-repudiation of actions

Coarse-grain access control

Confidentiality & integrity of communications

Information flow properties

Architectural access control

Study at home

# UMLsec: Supported Security Requirements

**Fair Exchange**- This requirement postulates that the trade is performed in a way that prevents both parties from cheating

**Non-Repudiation**- An action cannot subsequently be denied successfully. That is, the action is provable, usually wrt. some trusted third party

**Secure Logging-** The auditing data is at each point during the transaction of the system consistent with the actual state of the transaction (to avoid the possibility of fraud by interrupting the transaction)

**Message Authenticity or Data origin Authenticity-** Allows to identify the original source of data in the past

**Entity Authenticity-** Allows to identify active participation of a participant in a particular protocol at that time

**Guarded Access-** Access control ensures that only legitimate parties have access to a security-relevant part of the system. Access control can be enforced by guards.

# UMLsec: Supported Security Requirements

**Freshness-** A message is fresh if it is created under the current execution round of the system under consideration and cannot replay an older message by the attacker

**Secure Information Flow-** This requirement is to ensure there is no indirect leakage of sensitive information from a trusted to an untrusted part. Trusted parts of a system are often marked as high, untrusted parts as low

**Secrecy and Integrity-** These are main data security requirements. A subsystem *S* preserves the secrecy (a.k.a.confidentiality) of an expression *E* from adversary *A* if *E* never appears in the knowledge set *K* of *A* during execution of *S.* Integrity means that some information can be modified only by legitimate parties.

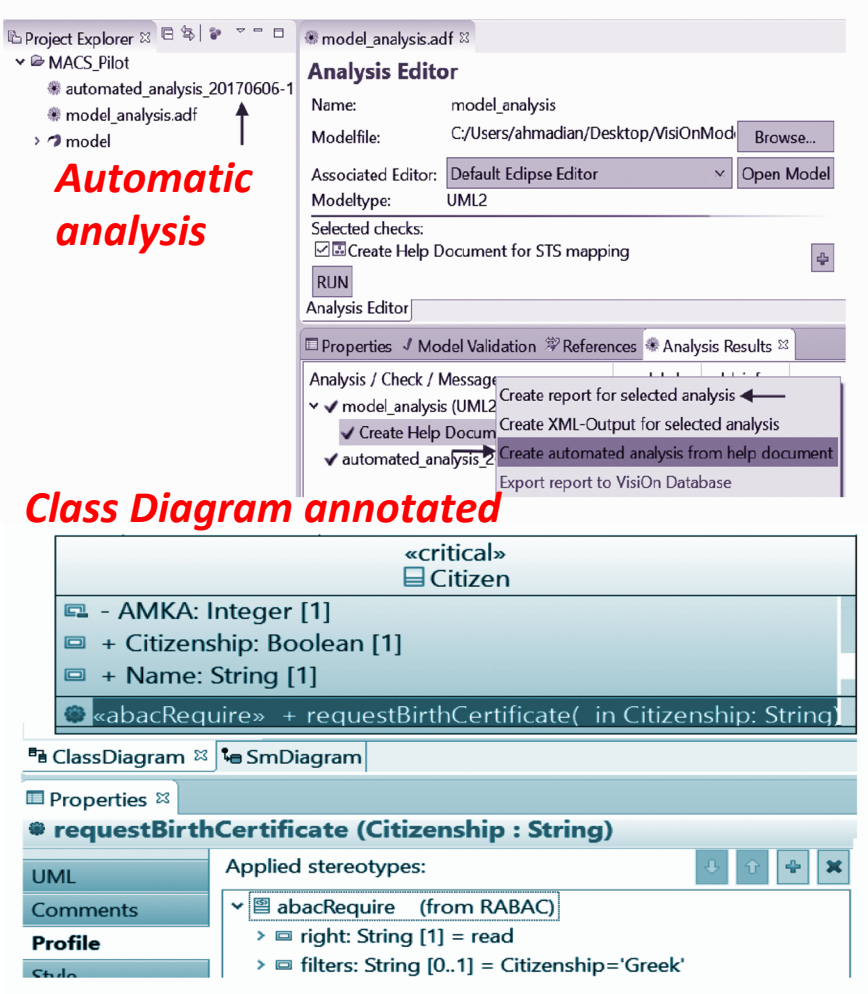
**Secure Communication Link**- Sensitive communication between different parts of a system needs to be protected. The relevant requirement of a secure communication link is here assumed to provide secrecy and integrity for the data in transit.

# UMLSec: model analysis

# UML Model Analysis with CARiSMA

- Analyse security requirements based on the information from
  (i) formal semantics
  (ii) adversary behaviour

- UMLsec diagrams are converted to first-order logic formulas (including epistemic constructs)
- Analysis of the diagrams using automated first-order logic theorem provers (e.g., e-SETHEO or SPASS)




More information about CARiSMA:
https://rgse.uni-koblenz.de/carisma/
https://youtu.be/b5zeHig3ARw

# Adversary in UMLsec

| Stereotype | Base Class | Tags | Constraints | Description |
|---|---|---|---|---|
| fair exchange | subsystem | start, stop, adversary | after start eventually reach stop | enforce fair exchange |
| provable | subsystem | action, cert, adversary | action is non-deniable | non-repudiation requirement |
| rbac | subsystem | protected, role, right | only permitted activities executed | enforces role-based access control |
| Internet | link | | | Internet connection |
| encrypted | link | | | encrypted connection |
| LAN | link, node | | | LAN connection |
| wire | link | | | wire |
| smart card | node | | | smart card node |
| POS device | node | | | POS device |
| issuer node | node | | | issuer node |
| secrecy | dependency | | | assumes secrecy |
| integrity | dependency | | | assumes integrity |
| high | dependency | | | high sensitivity |
| critical | object, subsystem | secrecy, integrity, authenticity, high, fresh | | critical object |
| secure links | subsystem | adversary | dependency security matched by links | enforces secure communication links |
| secure dependency | subsystem | | « call », « send » respect data security | structural interaction data security |
| data security | subsystem | adversary, integ., auth. | provides secrecy, integrity, authenticity, freshness | basic data security requirements |
| no down-flow | subsystem | | prevents down-flow | information flow condition |
| no up-flow | subsystem | | prevents up-flow | information flow condition |
| guarded access | subsystem | | guarded objects accessed through guards | access control using guard objects |
| guarded | object | guard | | guarded object |

# Adversary in UMLsec

- Type of adversary can be specified in the UML diagram
- If not specified, capability of default attacker is used

| Stereotype | $\text{Threats}_{default}()$ |
|---|---|
| Internet | {delete, read, insert} |
| encrypted | {delete} |
| LAN | ∅ |
| wire | ∅ |
| smart card | ∅ |
| POS device | ∅ |
| issuer node | ∅ |

**Threats from *default* attacker**

| Stereotype | $\text{Threats}_{insider}()$ |
|---|---|
| Internet | {delete, read, insert} |
| encrypted | {delete, read, insert} |
| LAN | {delete, read, insert} |
| wire | {delete, read, insert} |
| smart card | ∅ |
| POS device | {access} |
| issuer node | {access} |

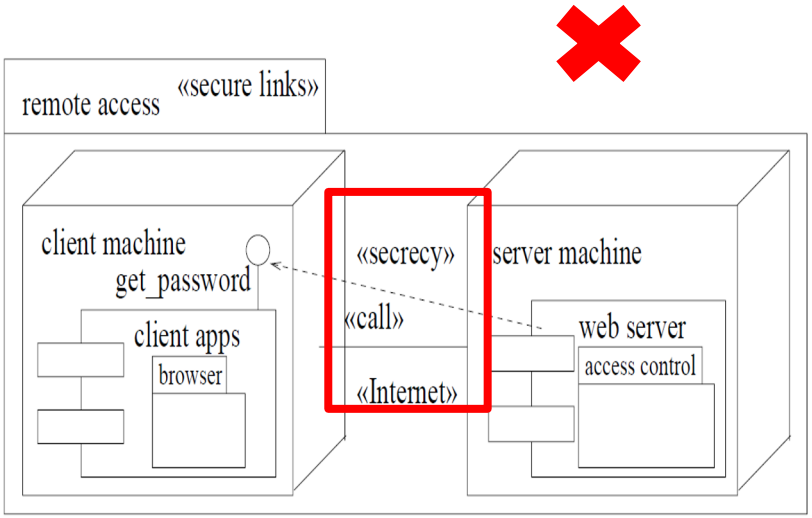
**Threats from *insider* attacker**

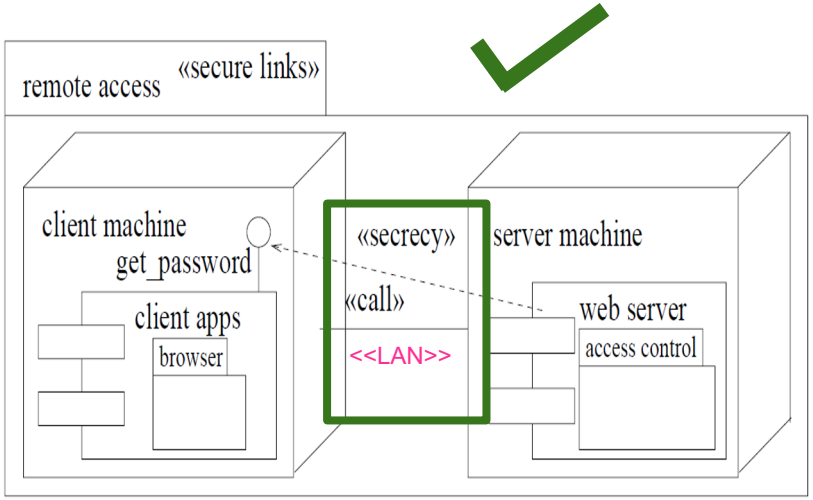
# Security analysis for Secure Links

- The model does not meet the secure requirements against the default adversary:
  - In the model, the call dependency is label with the <<secrecy>> constraint
  - The *link* is labeled as **<<Internet>>**
  - The default attacker has *delete*, ***read*** and *insert* capability

| Stereotype | $\text{Threats}_{default}()$ |
|---|---|
| Internet | {delete, read, insert} |

- An attacker can read messages on an Internet link
- Internet connections do not provide secrecy against attacker
- Constraint is violated



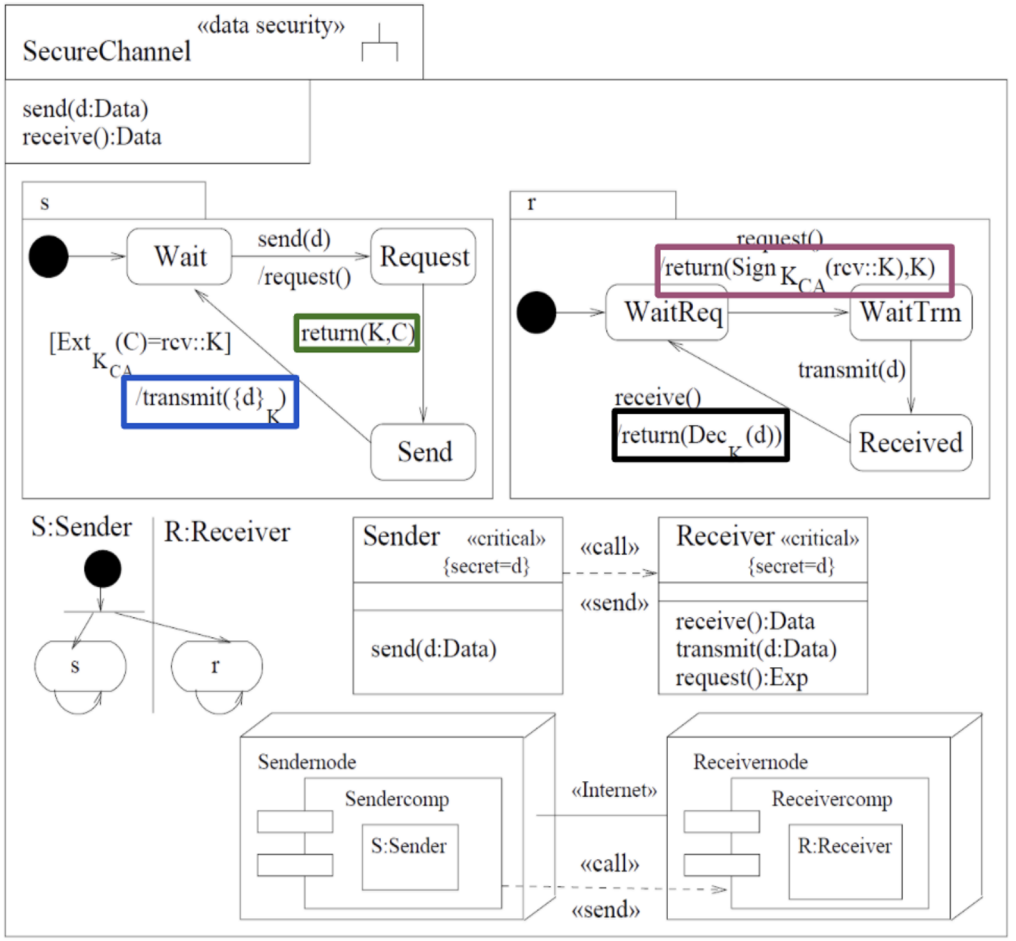



# Security analysis for Data Security

Example is a simple security protocol.

1. The sender requests the public key K together with the certificate to certify authenticity of the key from the receiver
2. Receiver sends certificate and the public key to the sender
3. Sender then sends the data back encrypted using K to the receiver
4. Receiver decrypts the ciphertext from the sender using K

The sender and receiver components can interact with each other because of provided *<<call>> and <<send>>*
An internet connection *<<internet>>* is established between the sender and the receiver




**Example: Security protocol**

Secrecy of d is preserved

# Other approaches

# Analysis based on Formalized Signatures

- Detailed model describing the system

- System descrition model
  - Component-based model
    (components, interface funcions)
  - Deployment model (components in nodes)
- Security specification model
  - Security objectives (e.g., a component is critical)
  - Security controls
    (e.g., component enforces user authentication)
    (e.g., node in trusted zone)

M. Almorsy, et al., *Automated Software Architecture Security Risk Analysis using Formalized Signatures*, **ICSE 2013**

# Attack scenario as OCL signature

```
context System inv Man-in-the-Middle Attack:

self.components->select(C1 |
        C1.DeploymentZone = 'Untrusted' and

        self.components.exists(C2 |
                C2.Channels->exists(Ch |
                        Ch.TargetComponent = C1 and
                        Ch.Encryption = false)

   …
)
```

Model query

Graph navigation

# Security metric as OCL signature

```
context System inv AttackSurface:

self.components
        ->select(C1 |
                 C1.DeploymentZone = 'Untrusted‘)
        ->collect(C2 |
                 C2.Functions)
        ->size()
```

# Analysis based on Formalized Signatures

## How does it work

- OCL signatures are *provided*
  (16 in total, more can be added)

- Tool runs the checks on the models
  (model queries and graph navigation)

- Report of results