



# Security at Architectural Design Level

**Riccardo Scandariato**

Institute of Software Security, TUHH, Germany

ric\*\*\*do . scanda\*\*\*to @ tuhh.de

Put name of event HERE



# Learning objectives

- **What models** are interesting for security? And **what properties** are represented?
- **What can I do** with models?
  - Analysis, testing, generation, ...
- **How to build** a secure software architecture?

## Reading material on Security Tactics

Joanna Santos, et al., An Empirical Study of Tactical Vulnerabilities, JSS, 2019



# Software model

- Provides an abstraction of the system
- Software engineering perspective
  - Parts/components and interfaces
  - Functionality/logic ...

# We focus on architectural design

"The software architecture of a computing system is the **structure** of the system, which comprise software **components**, the **externally visible properties** of those components, and the **relationships** among them"

Len Bass

"The fundamental concepts and **properties** of a system in its **environment** embodied in its **elements**, **relationships**, and in the **principles** of its **design** and **evolution**."

ISO/IEC/IEEE 42010

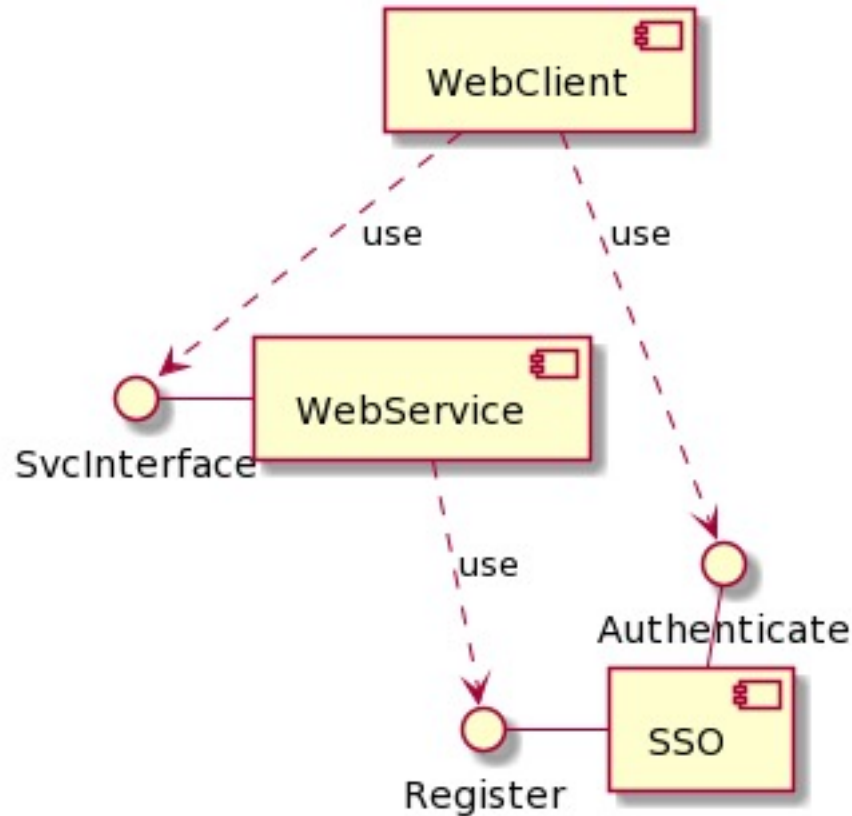
<http://www.iso-architecture.org/ieee-1471/defining-architecture.html>

The set of **design decisions** that determine the quality properties of a system

Jan Bosch

# Security – Commonly used models

## UML Component Diagram





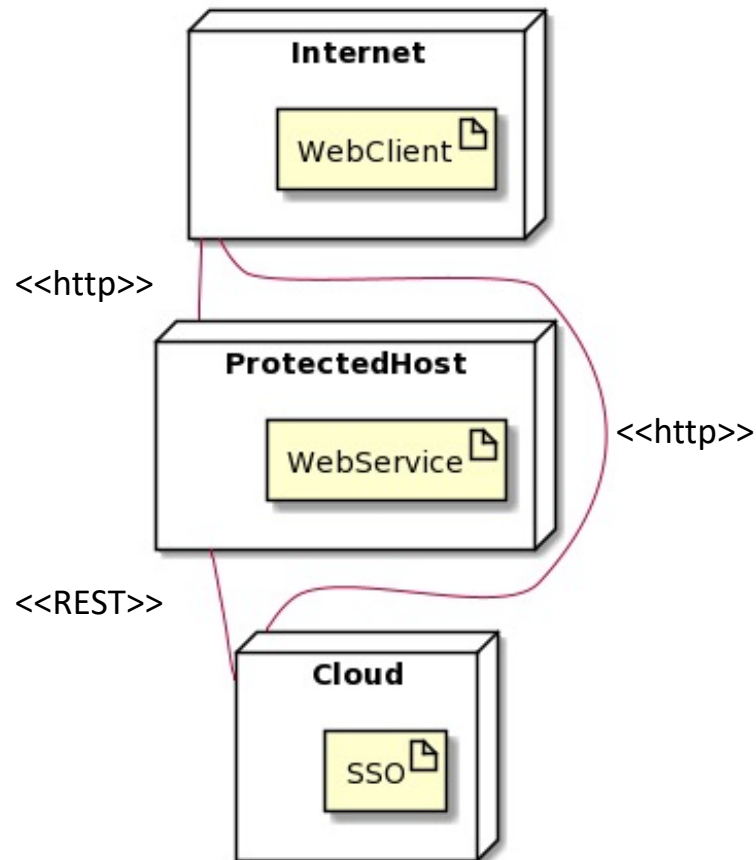
# Security – Commonly used models

## UML Sequence Diagram



# Security – Commonly used models

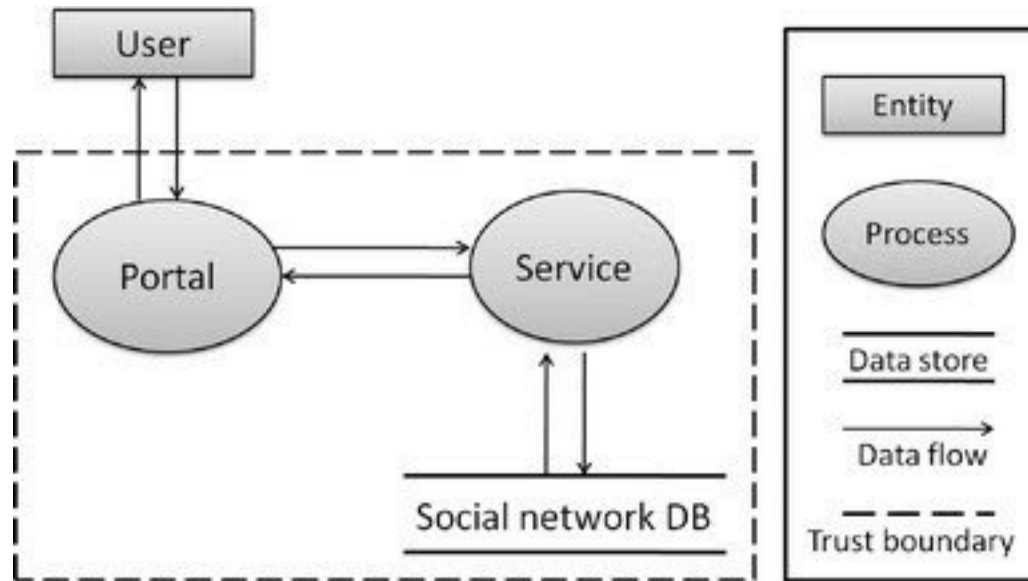
## UML Deployment Diagram



# Security – Commonly used models

## Data Flow Diagram

We'll see this later  
(cf. Threat analysis)



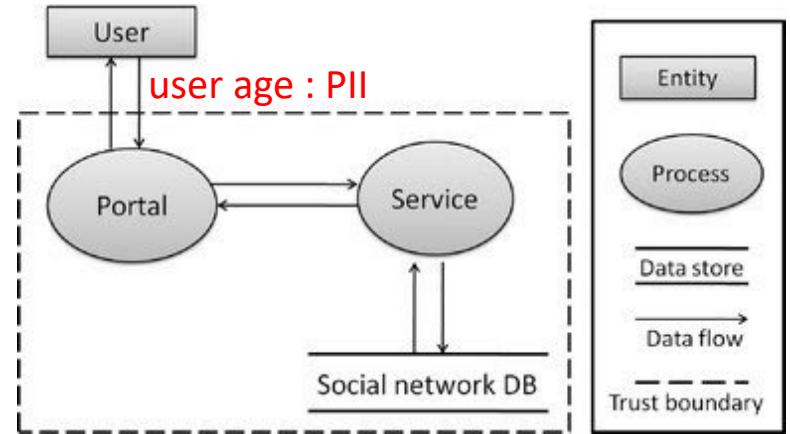
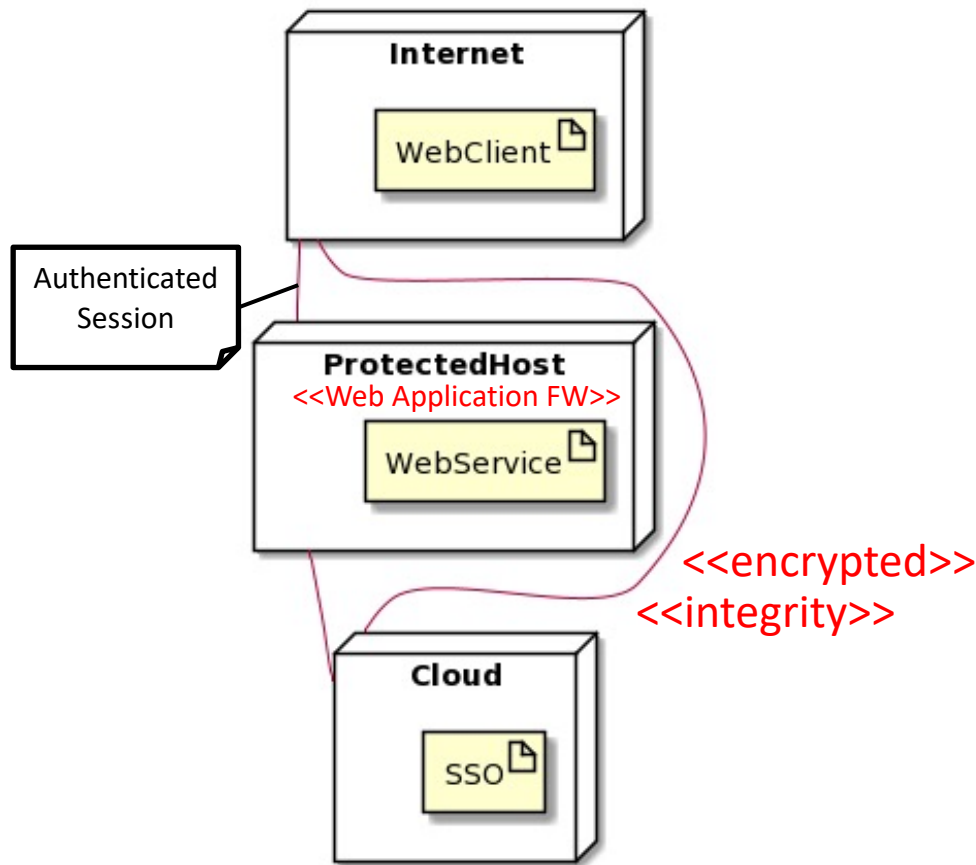


# Security model

- Provides an abstraction of the system
- Software engineering perspective
  - Parts/components and interfaces
  - Functionality/logic ...
- **Security engineering perspective**
  - Data/assets, their sensitivities
  - Information flows
  - Security mechanisms
  - Security assumptions/expectations ...

# Security engineering perspective

## Examples



PII : personally identifiable information

*The problem of semantics* 😊

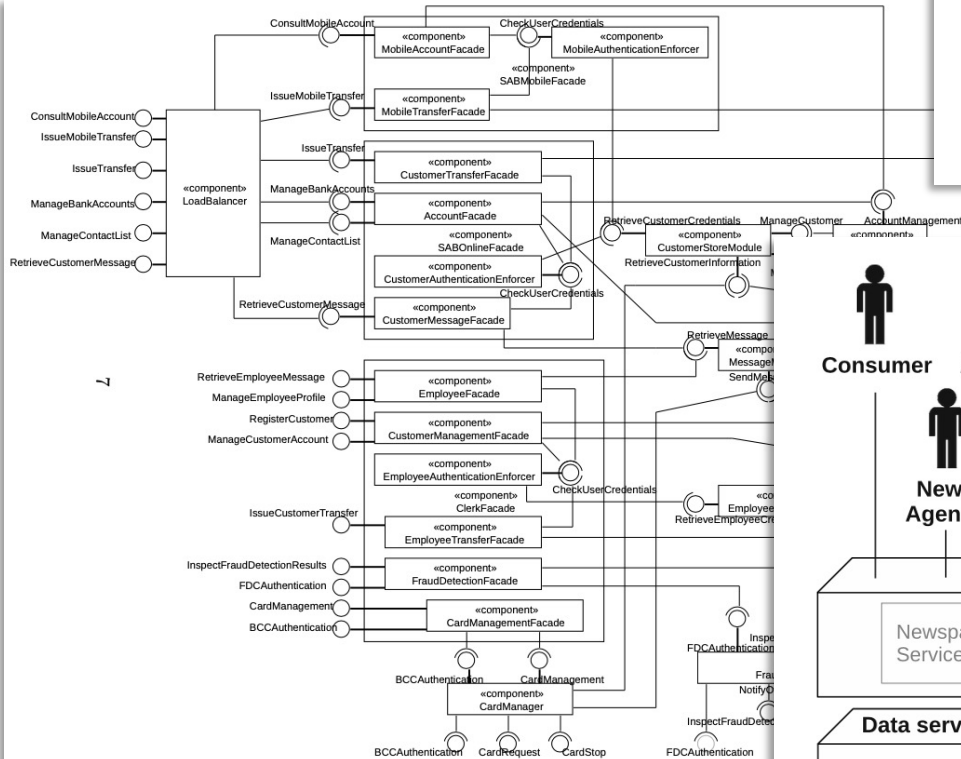
# Model, where from?

- Top-down, as an up-front blueprint
  - “Security concept” developed in safety-critical domains (automotive, aviation, medical)
- Bottom-up, reconstructed by experts
  - Common case when security analysis starts
  - E.g., most web-based systems
- Bottom-up, automatically extracted from code
  - Research field (ArchSec, Gravity, etc)

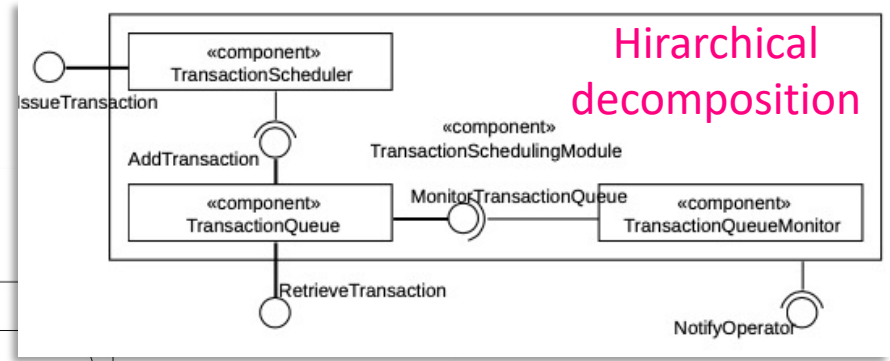


# Architectural documentation

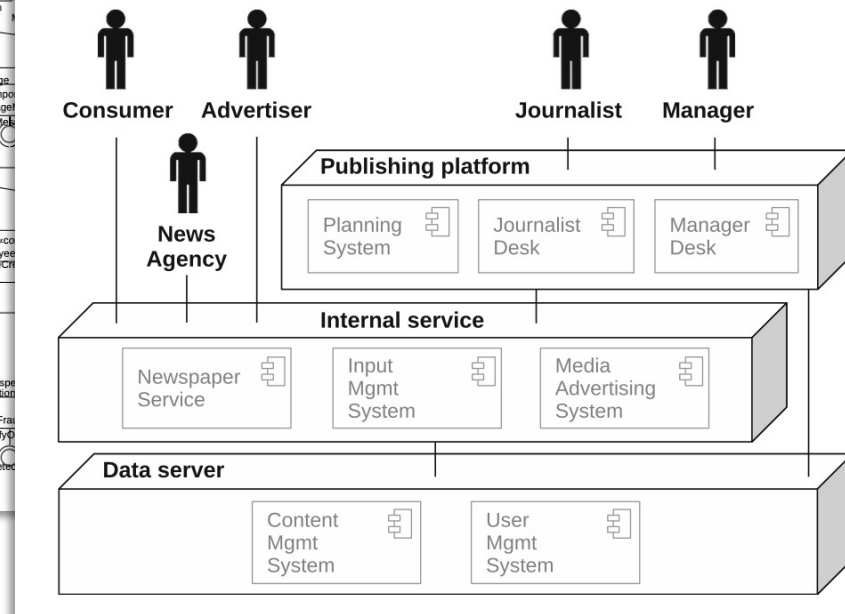
## Academic dream



Component diagram



Hierarchical decomposition

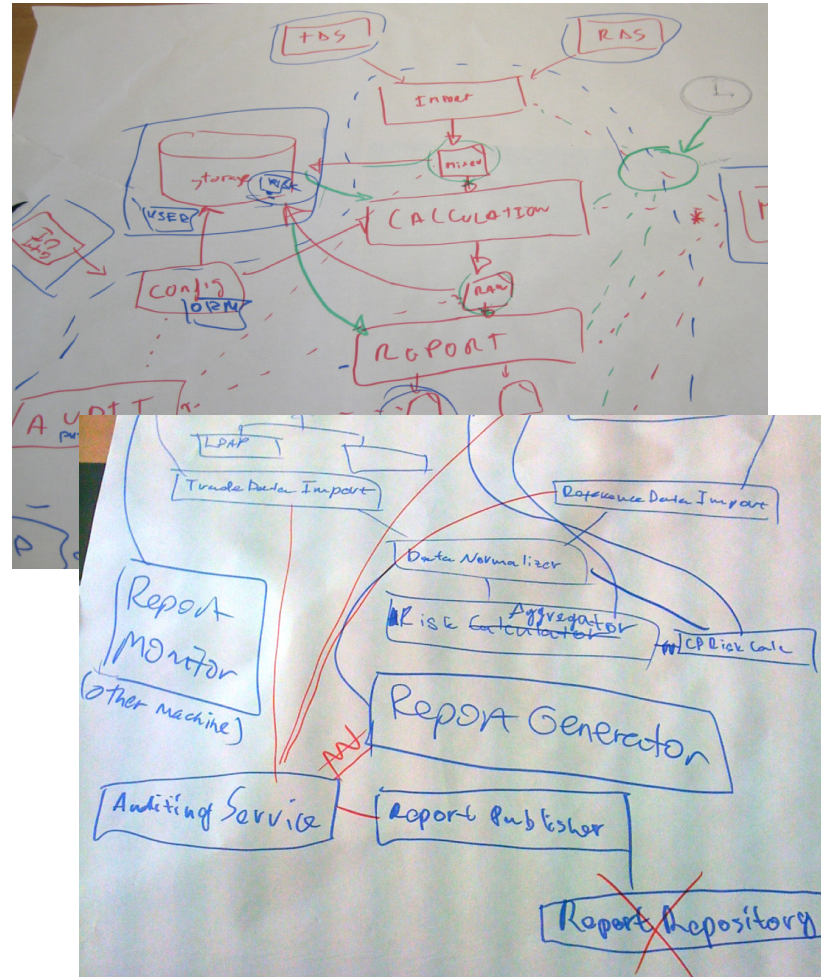


Deployment diagram



# Architectural documentation

## Real life





# Models, what for?

Next week

- Model analysis
- Code generation
- Model-based test generation
- More (e.g., monitoring, metrics...)

# Model-based code generation

- Derive (draft) implementation code and configuration
- E.g., code: Authorization pattern → **Security aspect woven in code**
- E.g., config: access control policy
  - derive roles from context/component diagrams
  - **derive permissions from use cases, workflows, etc.**

# Model-based test generation

- Functional security testing
  - E.g., test rules in a firewall, given the components that are present in the model
  - E.g., test access control rules, provided the roles in the model
- Security vulnerability testing (penetration)
  - Generate attack sequences using the system topology
  - Model-based fuzzing (e.g., alter order of messages in a protocol), from a sequence diagram



# Other uses

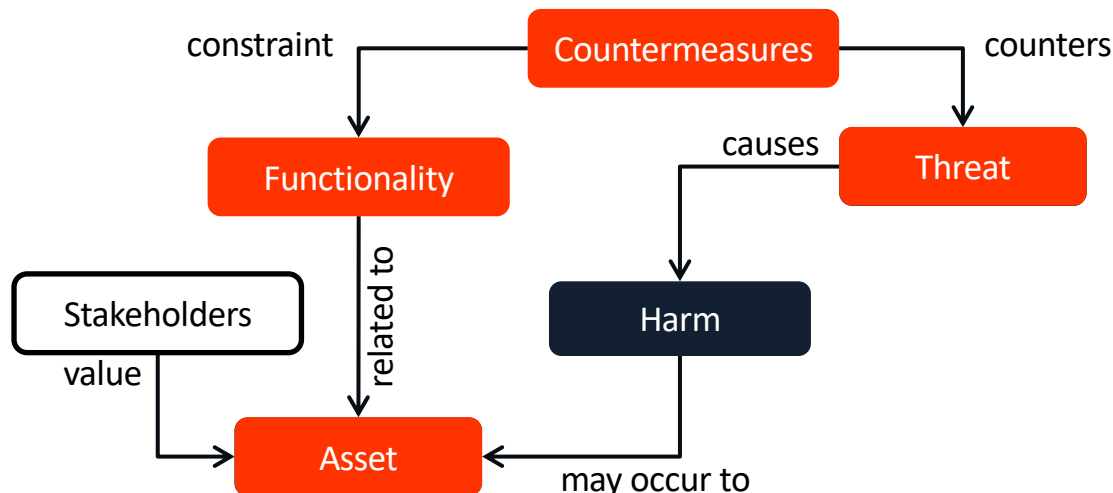
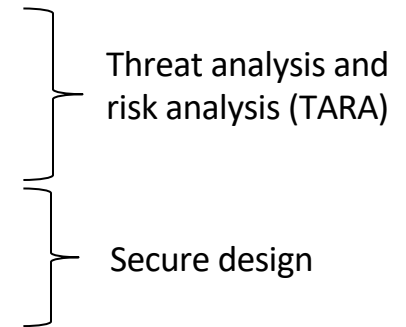
- Model-based runtime **monitoring**
  - Monitoring the security assumptions made in the model
  - E.g., communication is encrypted, communication is only allowed between A and B, ...
- Model-to-model **transformations**
  - Hardening the model by adding security countermeasures
  - Making the functionality more GDPR compliant
- Compute security **metrics**
  - Mostly for for certification, but also prediction, etc



# BUILDING SECURITY-AWARE ARCHITECTURAL DESIGN

# Secure architectural design

- Identify the *assets* of interest (by interacting with the stakeholders)
- Understand the relationship *asset*  $\leftrightarrow$  *functionality*
- Identify *threats* and their importance (impact, likelihood)
- Implement constraints (i.e., *countermeasures*) to deal with threats
- Hence achieving the security goals



# In the toolbox

- Previous slide has a “magic” step 😊  
*“Implement constraints (i.e., **countermeasures**) to deal with threats”*
- What security knowledge do we use?
  - Security principles
  - Security tactics
  - Security patterns / security solutions



# SECURITY DESIGN PRINCIPLES



# Security Meta Principles

## A. Simplicity

- Fewer components and cases to fail
- Fewer possible inconsistencies
- Easy to understand

## B. Restriction

- Minimize access and inhibit communication

## C. Minimal assumptions

- Avoid trust



# Security Design Principles

1. Least Privilege
2. Fail-Safe Defaults
3. Economy of Mechanism
4. Complete Mediation
5. Open Design
6. Separation of Privilege
7. Least Common Mechanism
8. Psychological Acceptability

(VERY old)

J. Saltzer , D. Schroeder, The Protection of Information in Computer Systems, *Proceedings of the IEEE* 63(9), 1975



# 1. Least Privilege

- A subject/process should be given only those privileges necessary to complete its task
  - Function, not identity, controls
  - Rights added as needed, discarded after use
- **Architecture:** component only has privileges to interact with other appropriate components
- Common violation:
  - Browsing the Internet while logged as ***administrator*** or ***root***





## 2. Fail-Safe Defaults

- Default action is to **deny access**
- When an action fails, system must be restored to a state as secure as the state it was in when it started the action
- Example
  - Card looked up in vendor database to check for stolen cards
  - If no connectivity, no authentication, but transaction is logged -> NO!



## 3. Economy of Mechanism

- Keep it as **simple** as possible (KISS)
  - Use the simplest solution that works.
  - Fewer cases and components to fail.
  - Minimal retained state (harder for program to get ‘confused’)
- **Reuse** known secure solutions
  - i.e., **don’t write your own cryptography.**



## 4. Open Design

- Security should not depend on secrecy of design or implementation
  - No “Security through obscurity”
  - Refers to security policy and mechanism (not secrets like passwords and crypto keys)
- E.g., do not rely on obfuscation



## 5. Complete Mediation

- **Check every access**
- Usually checked once, on first access:
  - UNIX: File ACL checked on `open()`, but not on subsequent accesses to file
- If permissions change after initial access, unauthorized access may be permitted
- Also important for auditing!



## 6. Separation of Privilege

Require **multiple conditions** to grant access

- Separation of duty
- Compartmentalization
- Defense in depth (or multiple layers of security)



# Separation of Duty

- Functions are divided so that one entity does not have control over all parts of a transaction.
- Example:
  - Different persons must initiate a purchase and authorize a purchase.
  - Two different people may be required to arm and fire a nuclear missile.



# Compartmentalization

- Problem: A security violation in one process should not affect others.
- Solution: **isolate** components in deployment
  - Physically
  - Through virtual machines
- Also: **Self-limit** consumption of resources
- Also: **Divide** system into parts which are limited to the specific privileges they require in order to perform a specific task (**privilege separation**)



# Defense in Depth

- Diverse defensive strategies
  - Different types of defenses (protection, detection, reaction)
  - Different implementations of defenses (variety)
  - If **one layer pierced**, next layer may stop
  - Avoid “crunchy on the outside, chewy on the inside” security
- Contradicts “Economy of Mechanism”
  - **Think hard about more than 2 layers**





## 7. Least Common Mechanism

- Mechanisms used to access different resources should not be shared
  - Error or compromises of the mechanism while accessing one resource allow compromise of all other resources
  - Use separate machines, separate networks
  - All data in a blackboard mediated by a blackboard component?
- Contradicts “Economy of Mechanism”?



## 8. Psychological Acceptability

- *Security mechanisms should not add to the difficulty of accessing a resource*
- **Human factors are critical here**
  - Hide complexity introduced by security mechanisms
  - Make system secure in **default** configuration
- **Security vs Usability**

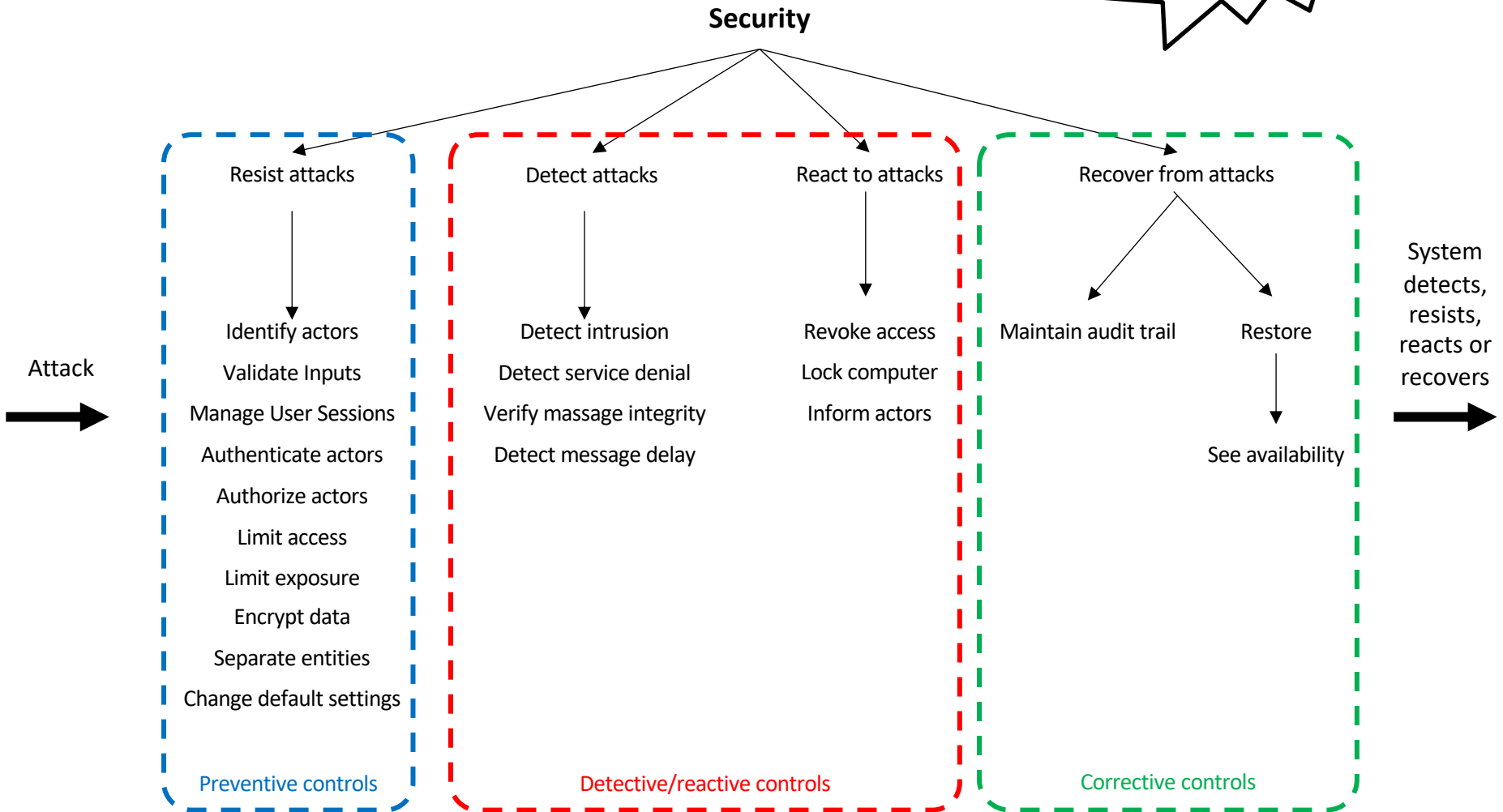




# SECURITY TACTICS



# Tactics for secure design




# Tactics for secure design


Category	Tactic	Description
<b>Resist Attacks</b>	Identify Actors	<i>Identifies the external agents that provide inputs into the systems</i>
	Validate Inputs	<i>Sanitizes, neutralizes and validates any externally provided inputs to minimize malformed data from entering the system and preventing code injection in the input data</i>
	Manage User Sessions	<i>Retains the information or status about each user and his/her access rights for the duration of multiple requests</i>
	Authenticate Actors	<i>Verifies the authenticity of actors (i.e. to check if the actor is indeed who it claims to be).</i>
	Authorize Actors	<i>Enforces that agents have the required permissions before performing certain operations, such as modifying data</i>
	Limit Access	<i>Limits the amount of resources that are accessed by actors, such as memory, network connections, CPU, etc.</i>
	Limit Exposure	<i>Minimizes the attack surface through designing the system with the least needed amount of entry points</i>
	Encrypt Data	<i>Maintains data confidentiality through use of encryption libraries</i>
	Separate Entities	<i>Places processes, resources or data entities in separate boundaries to minimize the impacts attacks</i>
	Change Default Settings	<i>Forces users to configure the system before use by changing the default (and potentially less secure) configuration.</i>
<b>React to Attacks</b>	Revoke Access	<i>In case of attacks, the system denies access to resources to everyone until the malicious behavior ends</i>
	Lock Computer	<i>Lockout mechanism that takes effect in case of multiple failed attempts to access a given resource</i>
	Inform Actors	<i>In case of malicious activities, the users/administrators or other entities that are in charge of the system are notified.</i>
<b>Detect Attacks</b>	Detect Intrusion	<i>Monitors network traffic for detecting abnormal traffic patterns caused by intrusion attempts</i>
	Detect Service Denial	<i>Monitors incoming traffic for detecting Denial Of Services (DoS) attacks.</i>
	Verify Message Integrity	<i>Ensures integrity of data, such as messages, resource files, deployment files, and configuration files</i>
	Detect Message Delay	<i>Detects malicious behavior through observing the time spent on delivering messages. In case messages are taking unexpected times to be received, the system may detect a potential data leakage.</i>
<b>Recover from Attacks</b>	Audit	<i>Logs user activities in order to identify attackers and modifications to the system</i>

# Risk-aware design


- **Preventive**: avoid incidents before they occur
  - E.g., access control to avoid disclosure
- **Detective/Reactive**: respond to incidents while they occur
  - E.g., detect anomalous activity and lock down the network
- **Corrective**: handle incidents after they have occurred (cf. **resilience**)
  - E.g., restore correct state from backup



E.g., First line of defense



E.g., Second line of defense

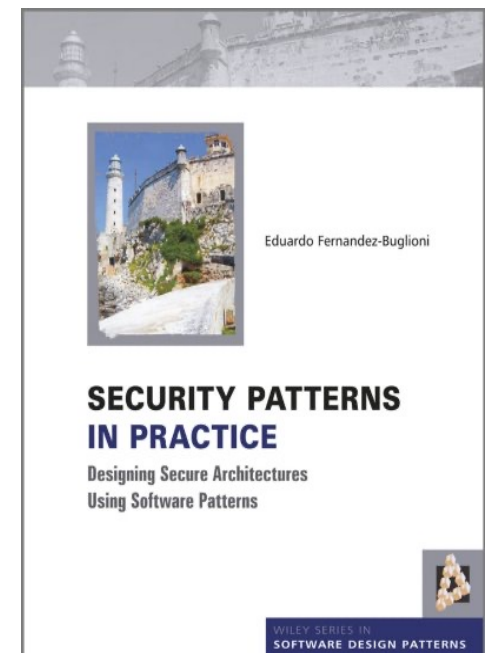
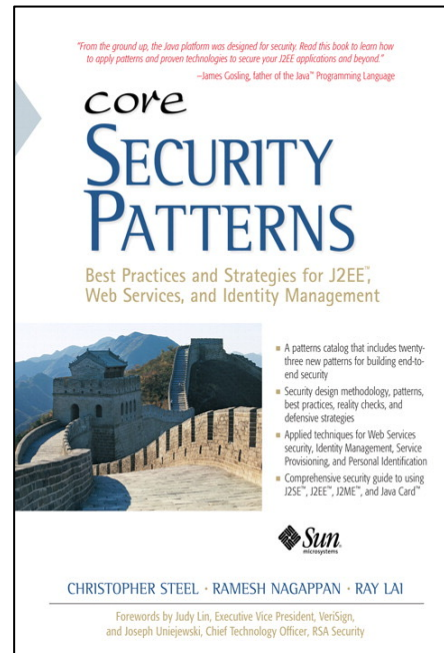
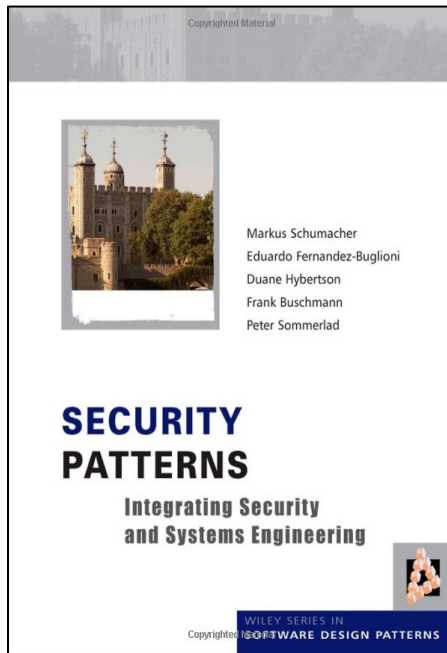


E.g., Third line of defense



# SECURITY DESIGN PATTERNS

# Security patterns – Fashion items ?







# Single Access Point Example



## Problem

A security model is difficult to **validate** when it has multiple “front doors”, “back doors”, and “side doors” for entering the application

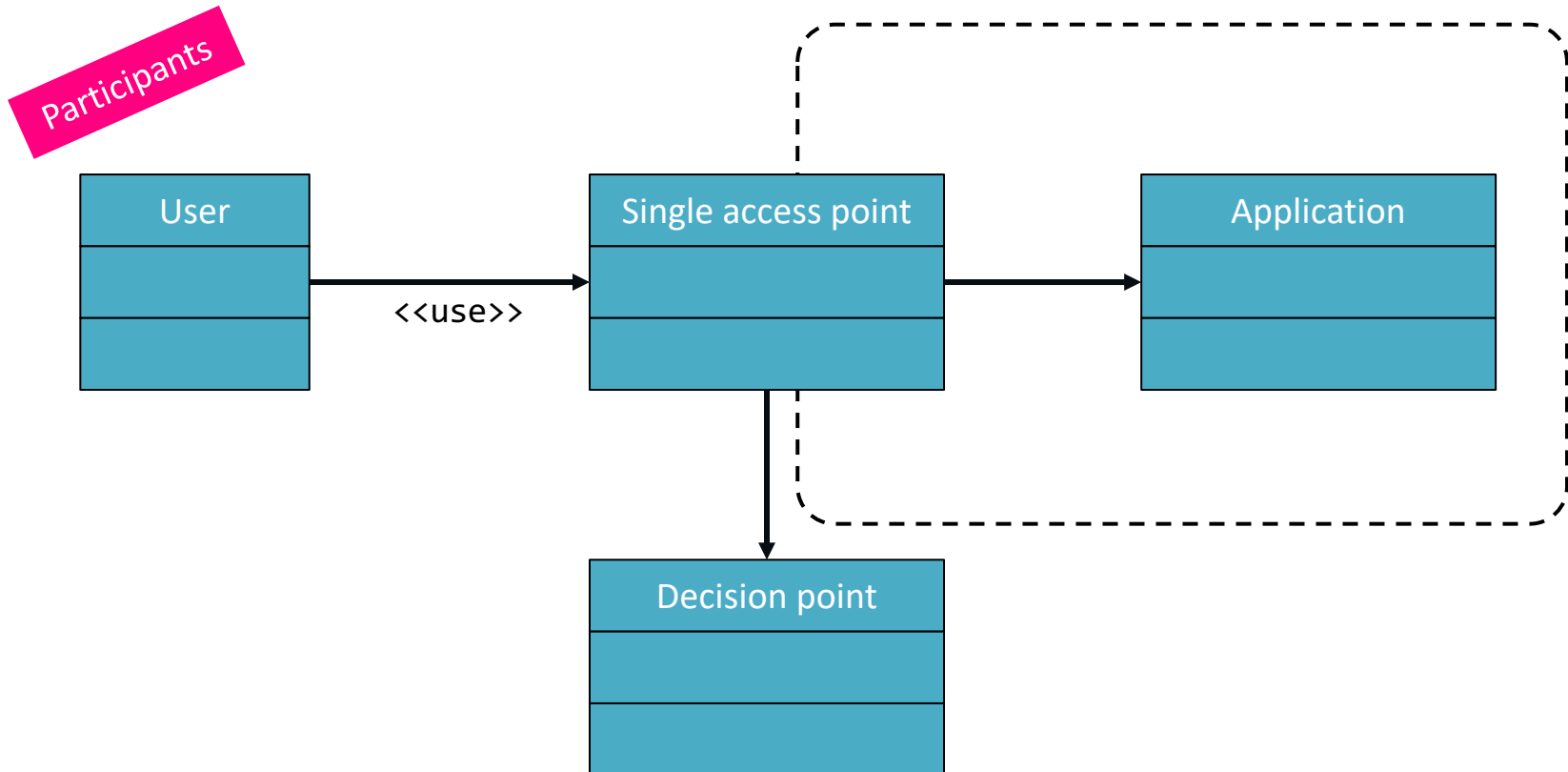
## Solution

Reduce the **attack surface** by setting up only **one way** to get into the system and if necessary, create a mechanism to decide which sub-application to launch.

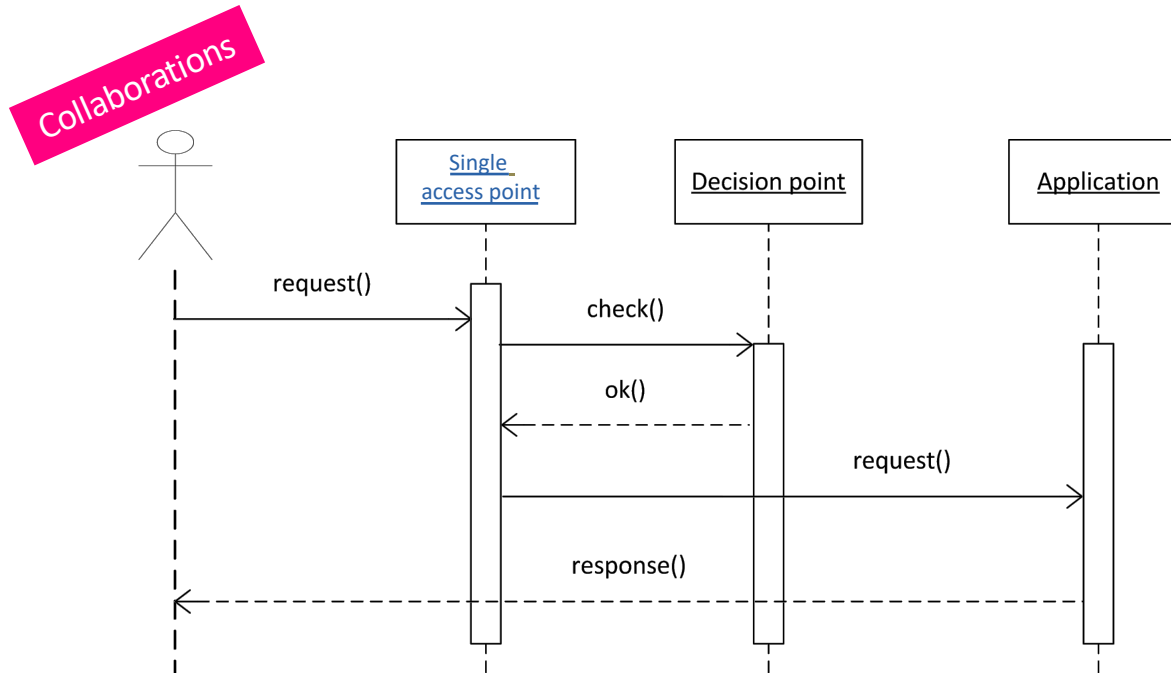
**Known Uses.** UNIX telnet and Windows NT login applications use Single Access Point for logging into the system.

**Source: Wiley Book**

# Single Access Point



# Single Access Point





# Share your opinion

Go to [menti.com](https://menti.com) – PUT CODE HERE



*Which secure design principles are implemented by the Single Access Point?*

*Which secure design tactics are implemented by the Single Access Point?*



# REMINDER: Security Design Principles

1. Least Privilege
2. Fail-Safe Defaults
3. Economy of Mechanism
4. Complete Mediation
5. Open Design
6. Separation of Privilege
7. Least Common Mechanism
8. Psychological Acceptability



# Single Access Point – Principles

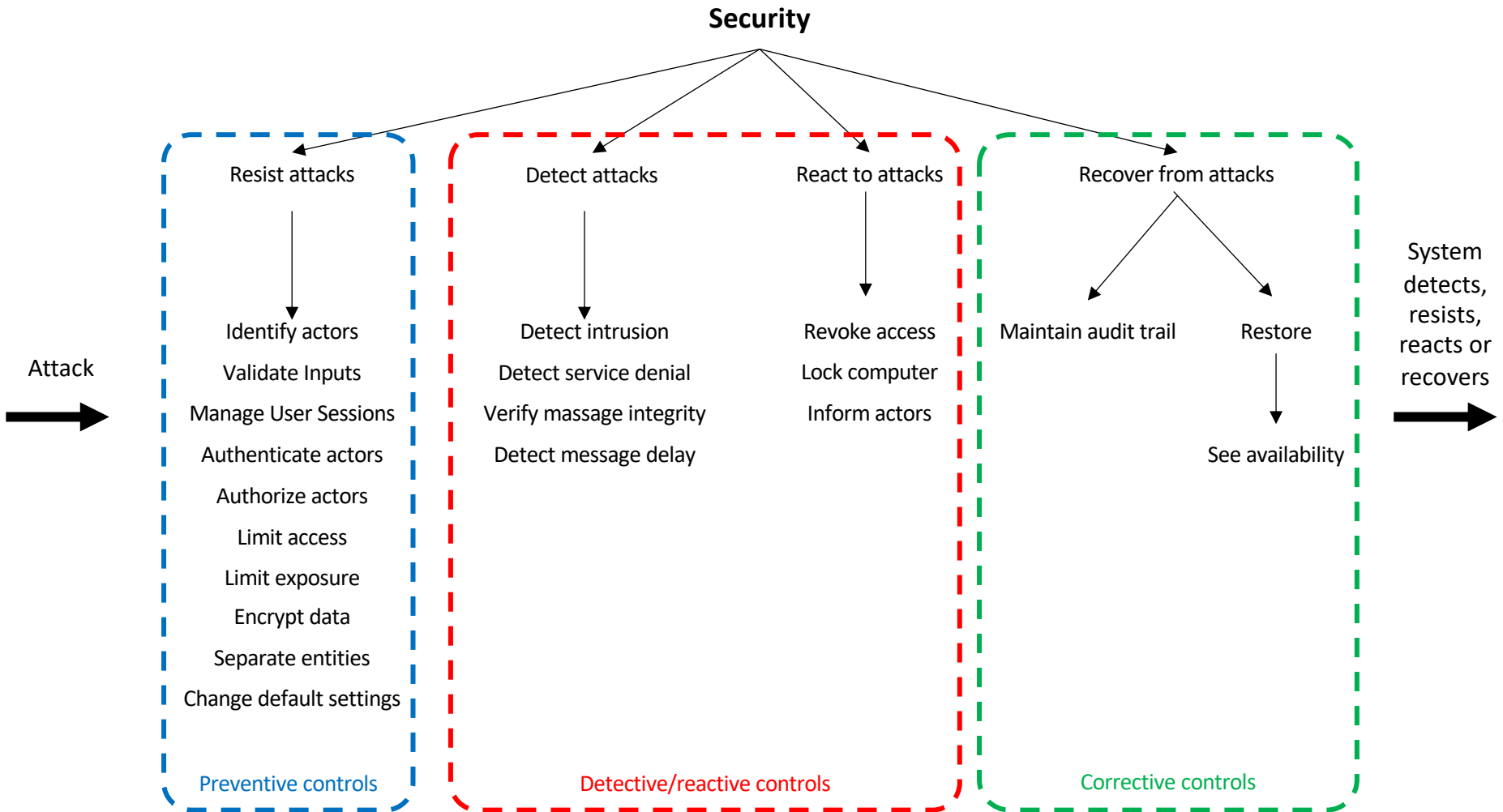
Security Pattern support one or more principles

Single Access Point

**Economy of mechanisms**

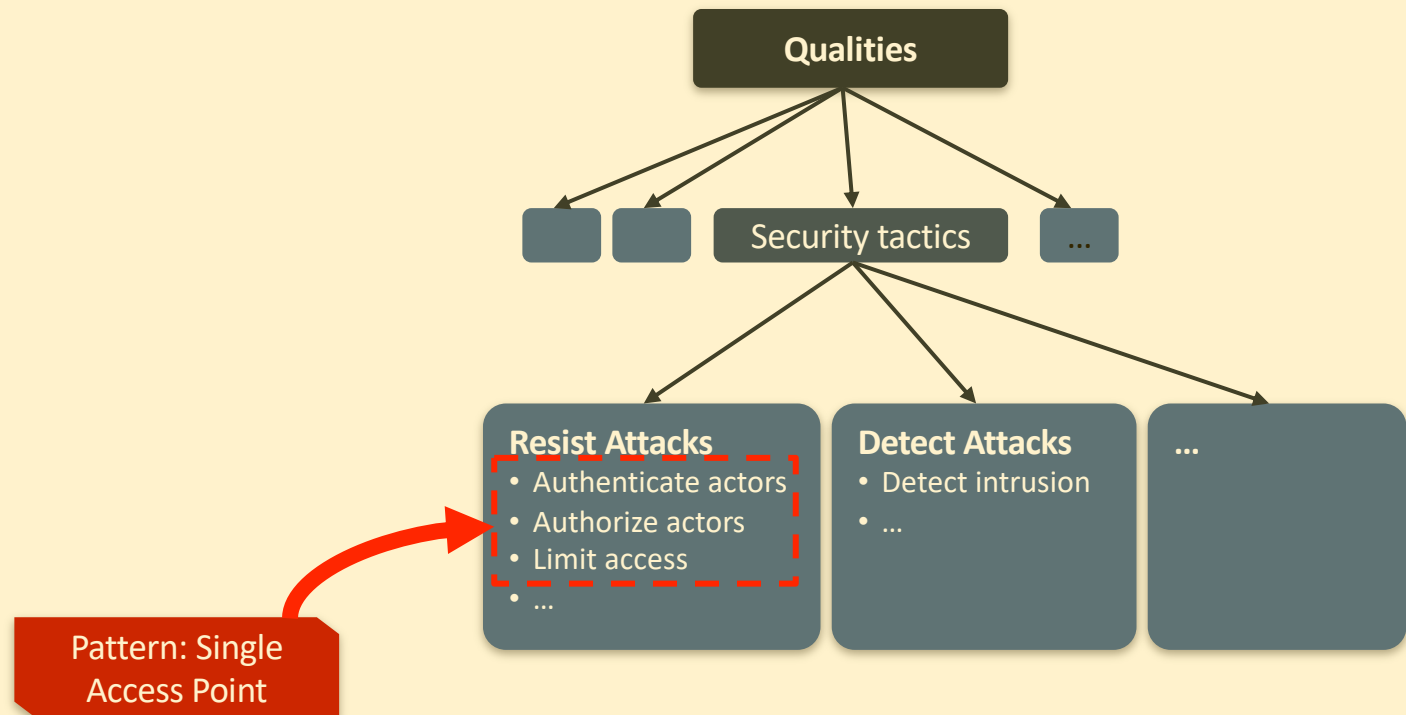
**Complete mediation**

# REMINDER: Tactics for secure design



# Single Access Point – Tactics

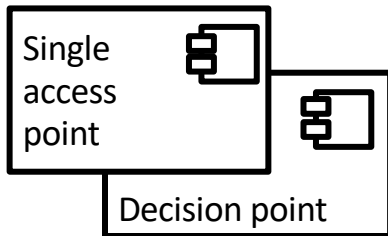
Security Pattern implements one or more security tactics





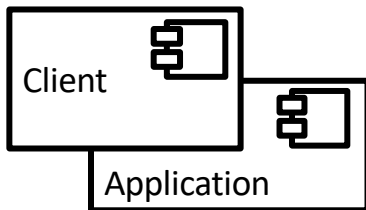
# Some “theoretical” underpinning

## What is a (security) design pattern?



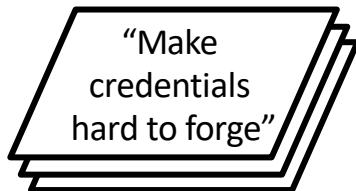
### New components

Added to the design to fulfill **new security functionality**



### Roles

Connect generic solution (pattern) to specific design. **Wire** the newly added components to existing ones



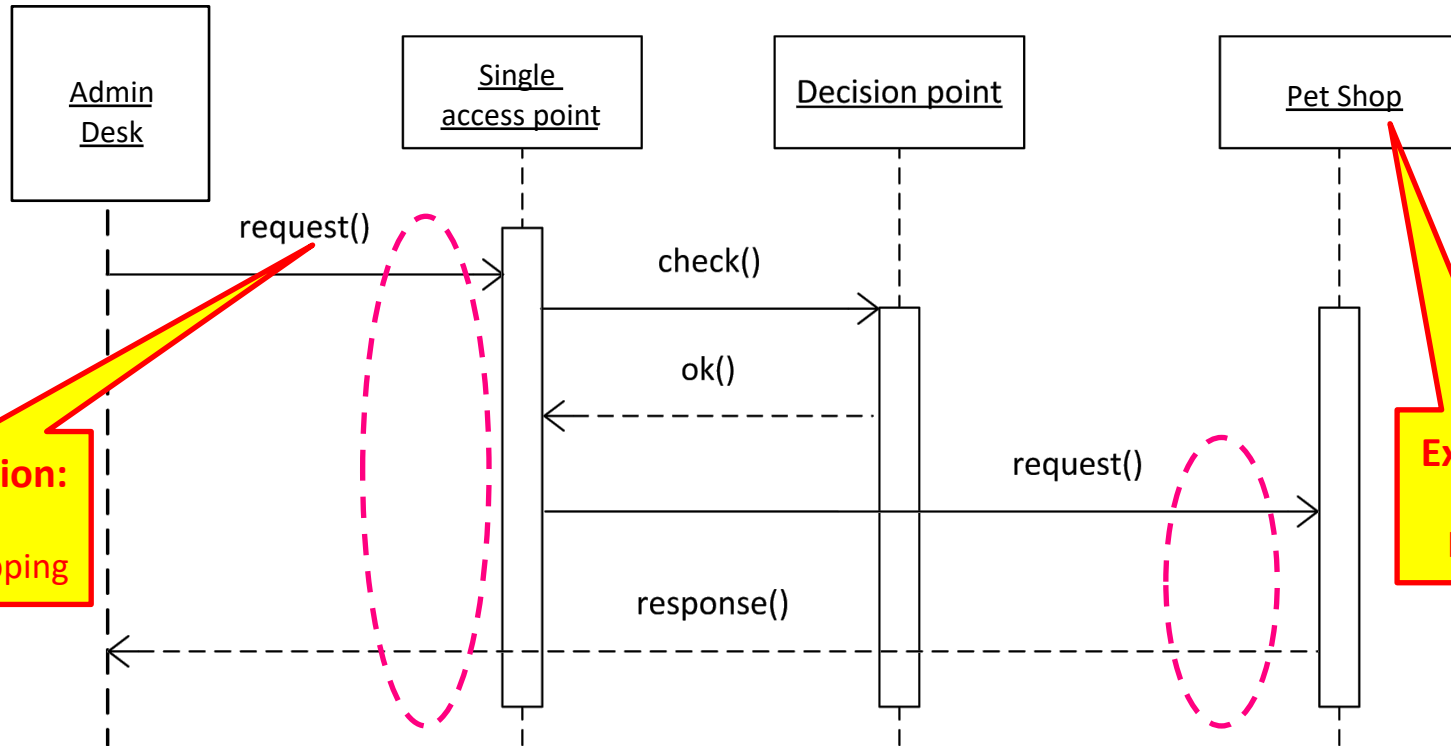
### Expectations

Impose constraints on the rest of the design (**assumptions!**). Need to either **modify** existing components (or “wires”) to satisfy expectations, or **modify design “at large”** (e.g., the account creation functionality, so that a good password is chosen)

# Instantiating a pattern

## Single Access Point

Role (Client → Admin Desk)      New components      Role (Application → Pet Shop)



**Expectation:**  
no  
eavesdropping

**Expectation:**  
no  
back-doors

Wiring

Wiring



# Questions ?

More questions at a later time?



scanda\*\*\*to @ tuhh.de

