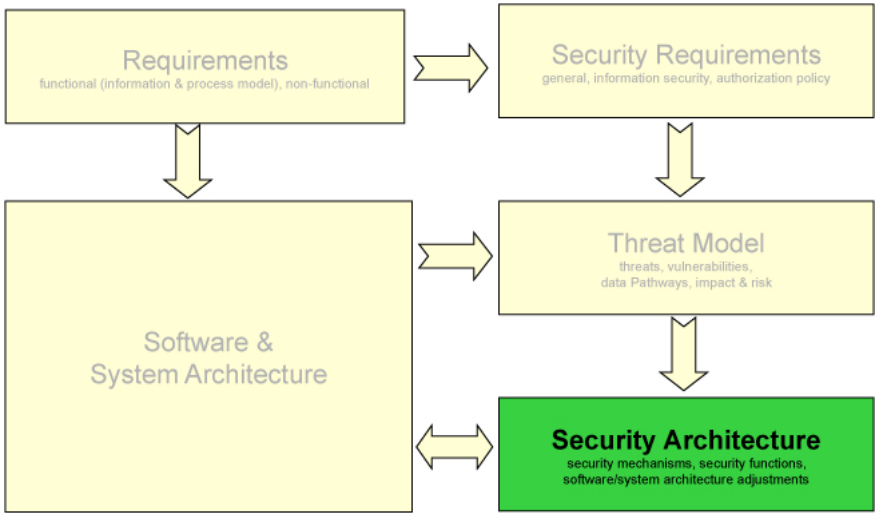
Security Design

Security Engineering

David Basin
ETH Zurich

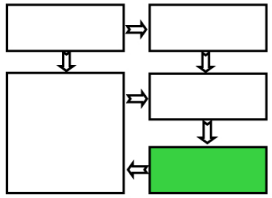
Road map



☞ **Risk management and safeguards**

- Security design principles and best practices
- Automatically generating security implementation
- Security design options
- Security design as an iterative process

Security risk



- Strategies for handling risk

  Avoid it by implementing appropriate safeguards

  Transfer it by allocating it to other systems, people, organization's assets, or to insurance

  Assume it by accepting it and controlling it with given resources

- Risk analysis considered in detail in separate module

  E.g., calculation of impact, risks, etc.

- Here we look at ways of employing **safeguards** and **countermeasures** during design to **avoid** or **mitigate** risk.

Handling risk: strategies for risk reduction

- **Avoid** the risk, by changing requirements for security or other system characteristics (followed by redesign/implementation)
- **Transfer** the risk, by allocating it to other systems, people, organization's assets or by buying insurance
- **Assume** the risk, by accepting it and controlling it with available resources

Corresponding costs must be taken into account

Safeguards and countermeasures

- **Avoidance controls**
  - Safeguards used to proactively minimize risk of exploits
  - Either reduce their likelihood or impact
- Examples
  - Encryption and authentication; PKIs
  - System security architecture (DMZ, firewalls, VPNs)
  - Secure communication
  - Physical security
  - Interruption prevention
  - Procedural measures: information security program and compliance to policies and standards.

Safeguards and Countermeasures II

- **Assurance**
  - Tools and strategies to ensure the effectiveness of existing controls and safeguards
- Examples
  - Application security reviews
  - Testing with respect to recommendations (e.g., NIST 800-42)
  - Penetration testing
  - Periodic perimeter scans
  - Vulnerability assessments
  - Regular updates/patches

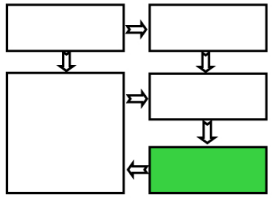
Safeguards and Countermeasures III

- **Detection**
  - Techniques and programs to ensure early detection, interception, and response to security breaches
  - Example: intrusion detection systems
  - Also virus scanners and audits
- **Recovery**
  - Planning and response services to rapidly restore a secure environment and investigate sources of a security breach
  - Examples: business continuity planning, impact analyses, crisis management, disaster recovery planning

Road map

- Risk management and safeguards
- ☞ **Security design principles and best practices**
- Automatically generating security implementation
- Security design options
- Security design as an iterative process

Security design principles



- Defining a security architecture is a **creative process**

  Supported by designer's experience, general principles, and patterns

- Use **proven patterns** and **principles**, e.g.
  - Avoid security by obscurity
  - Least privilege
  - Keep security mechanisms simple
  - Defense in depth
  - Detect intrusions

- If designers lack experience, **ask specialists**

Security design principles (cont.)

- Use **standards** and **best practices**[1]
  - BSI IT Baseline Protection Manual
  - Open Web Application Security Project (OWASP)
  - Rules and recommendations for specific industry or product
- **Example**: OWASP SQL Injection Prevention Cheat Sheet[2]
  - **Primary defense options**

    Option 1. Use Prepared Statements (Parameterized Queries)

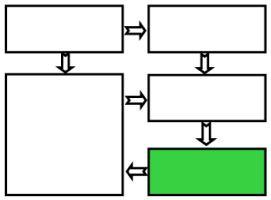
    Option 2. Use Stored Procedures

    Option 3. Escape all user supplied input
  - **Additional defenses**
    - enforce least privilege
    - perform white-list input validation

[1] See www.bsi.bund.de and www.owasp.org

[2] www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

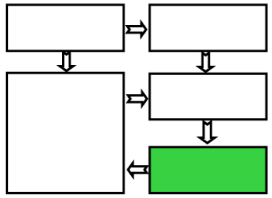
Security design principles (cont.)



- **Consistent security level**
  - Define a baseline and enforce it everywhere!
    Attackers only need a single hole in the defense
  - **Examples**
    - Authenticate and control access on all endpoints
    - Encrypt all network communication

- **Take appropriate measures**
  - Adopt software where needed rather than hacking infrastructure

Security design principles (cont.)



- **Use capabilities provided by the technologies used**
  rather than employing additional components
- **Use mature libraries**, e.g., OpenSSL, Signpost OAuth, ...
- **Use proprietary solutions as a last resort**
  - Standard solutions are usually more secure, efficient, and robust than home-grown ones
  - Maintenance?
- **Generate implementations to avoid programming errors**
  - Access control is good candidate
  - Requires high-quality generators

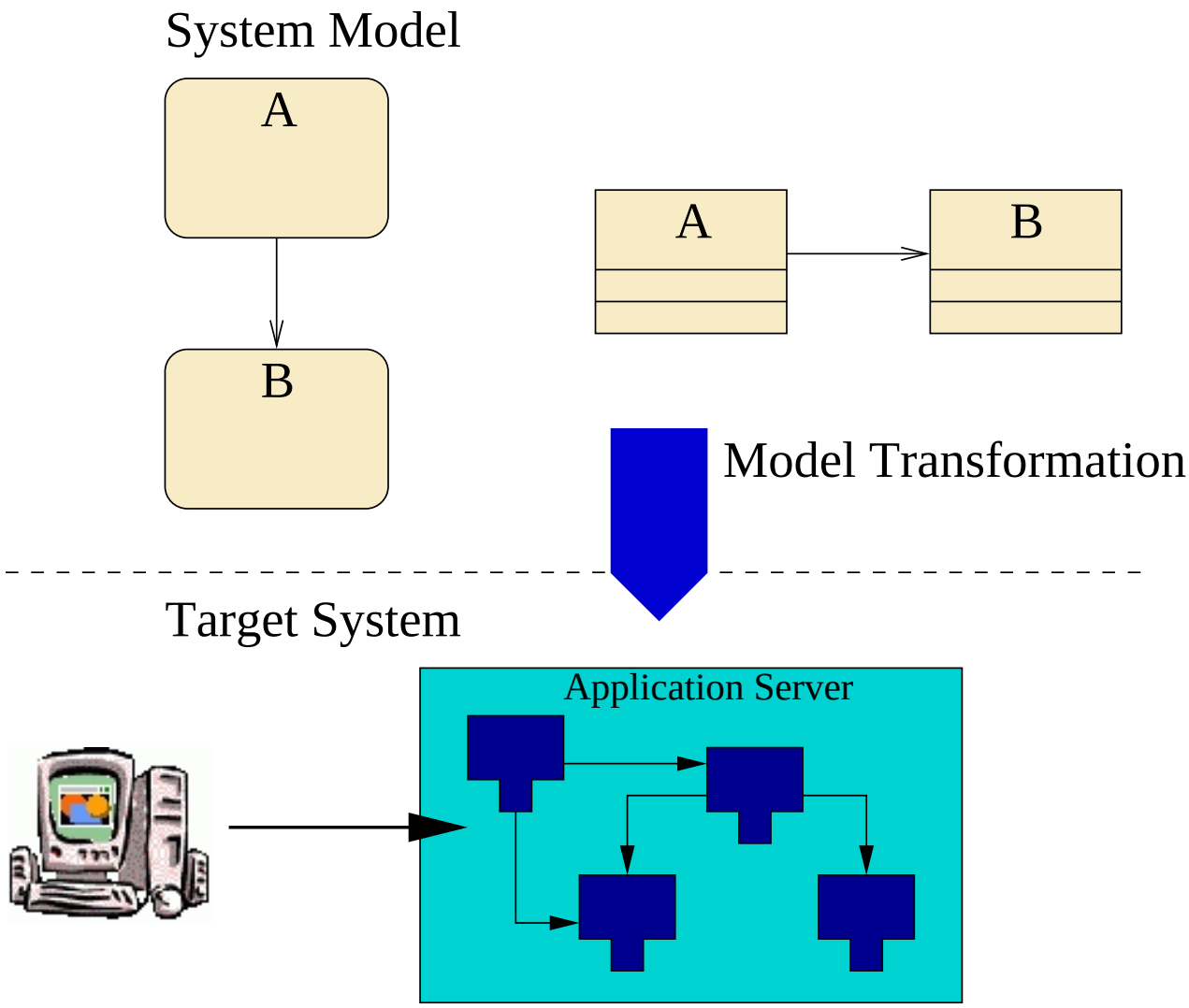
Road map

- Risk management and safeguards
- Security design principles and best practices

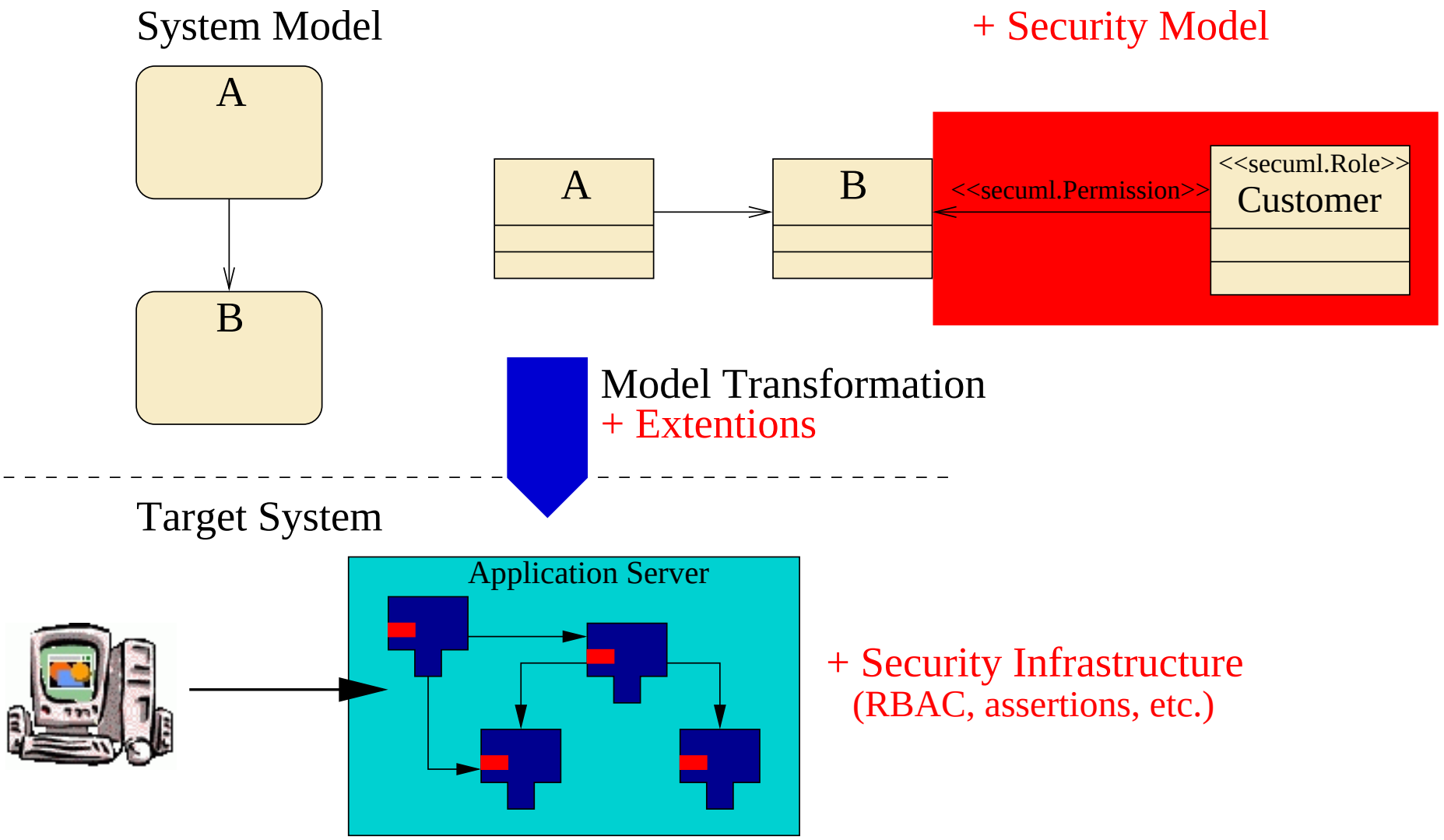
☞ **Automatically generating security implementation**

- Security design options
- Security design as an iterative process

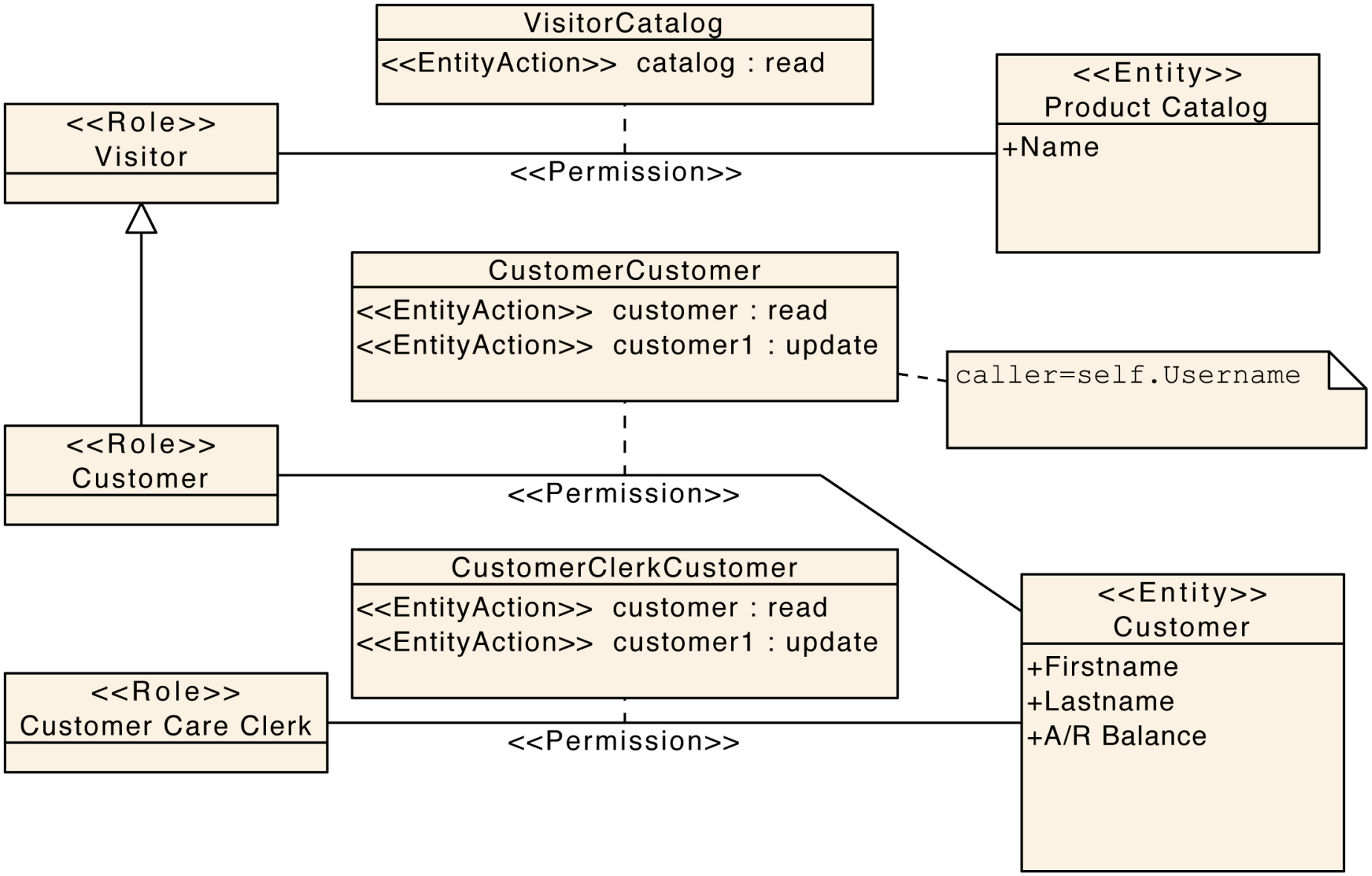
Starting point: model-driven architecture



Extension: model-driven security



Example: authorization policy



Example: generated code

Declarative access control (XML):

```
<method-permission>
  <role-name>Customer</role-name>
  <method>
    <ejb-name>CustomerImpl</ejb-name>
    <method-intf>Local</method-intf>
    <method-name>read</method-name>
    <method-params/>
  </method>
</method-permission>
```

Java-Assertions:

```
public CustomerData read() {
  ...
  if ctx.isCallerInRole(Customer") {
   // OCL: caller = self.owner.name
   accessAllowed = ctx.getCallerPrincipal().getName().equals(getOwner());
  }
  if (!accessAllowed) throw new AccessControlException("Access denied");
  _cancel();
}
```

Medium-scale case study
secure e-shop for pets

- Model
  - System model: **30 entities**
  - Access control: **6 role** and **60 permissions (15 constraints)**
- Generated code
  - XML: **5000 lines** (of total 13000 lines)
  - Java: **2000 lines** (of total 20000 lines)
- Development time: **3 weeks**
  - Only **2 days** for modeling and implementation of access control

$\Rightarrow$ **Study shows feasibility and benefit of approach**

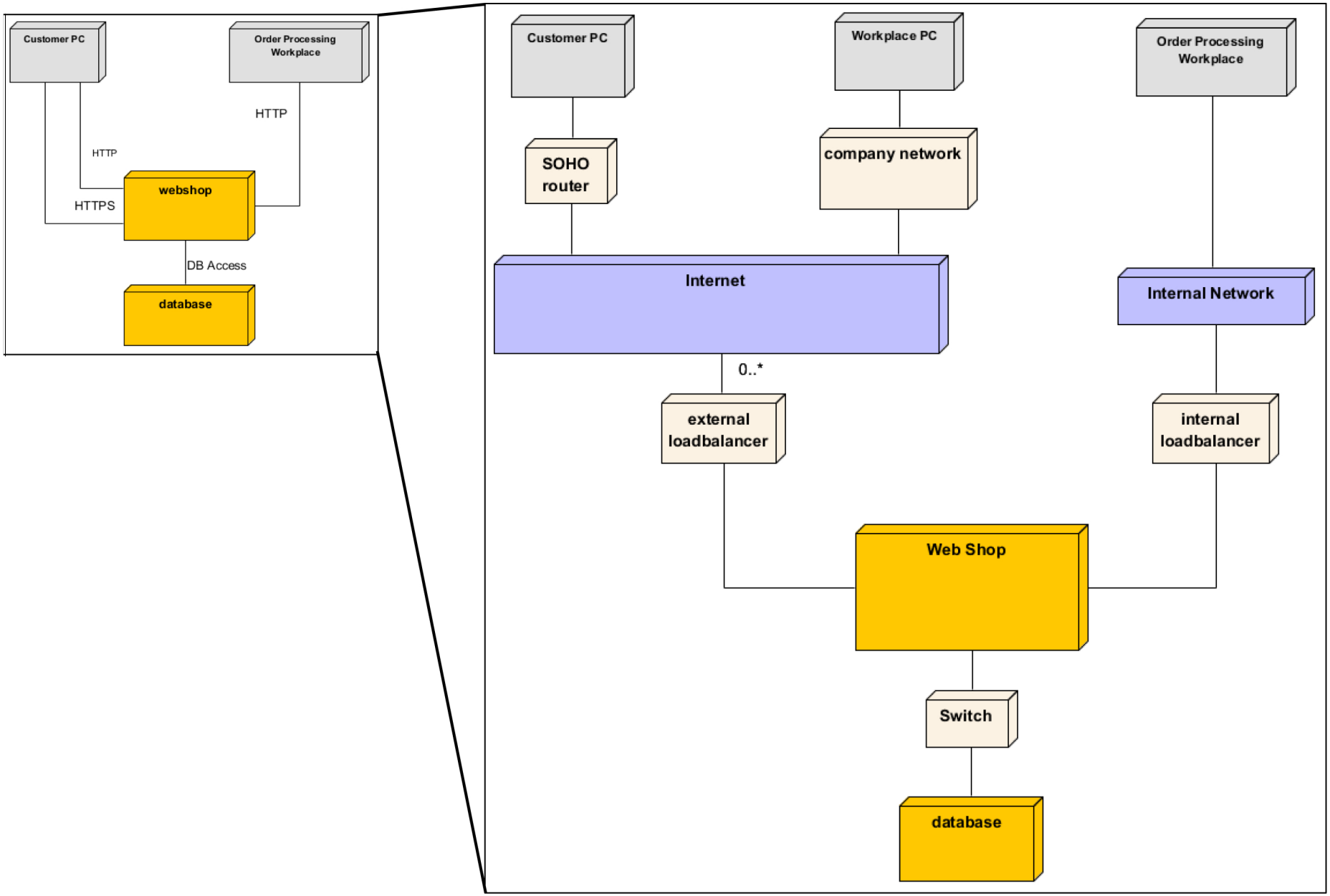
Large-scale industry projects
Passenger information system, NYC subway

- Environment: UML design model + proprietary code generators
- Approach
  - Added permissions for service access to UML model
  - Implemented generator for access control
  - Generated all access control artifacts directly from the model
- Lessons learned
  - Substantially simplifies access control design and implementation (just a drop-down box in the model)
  - Increased overall acceptance and support
  - AC implementation managed independently by security group

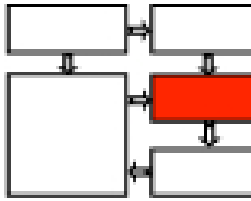
Road map

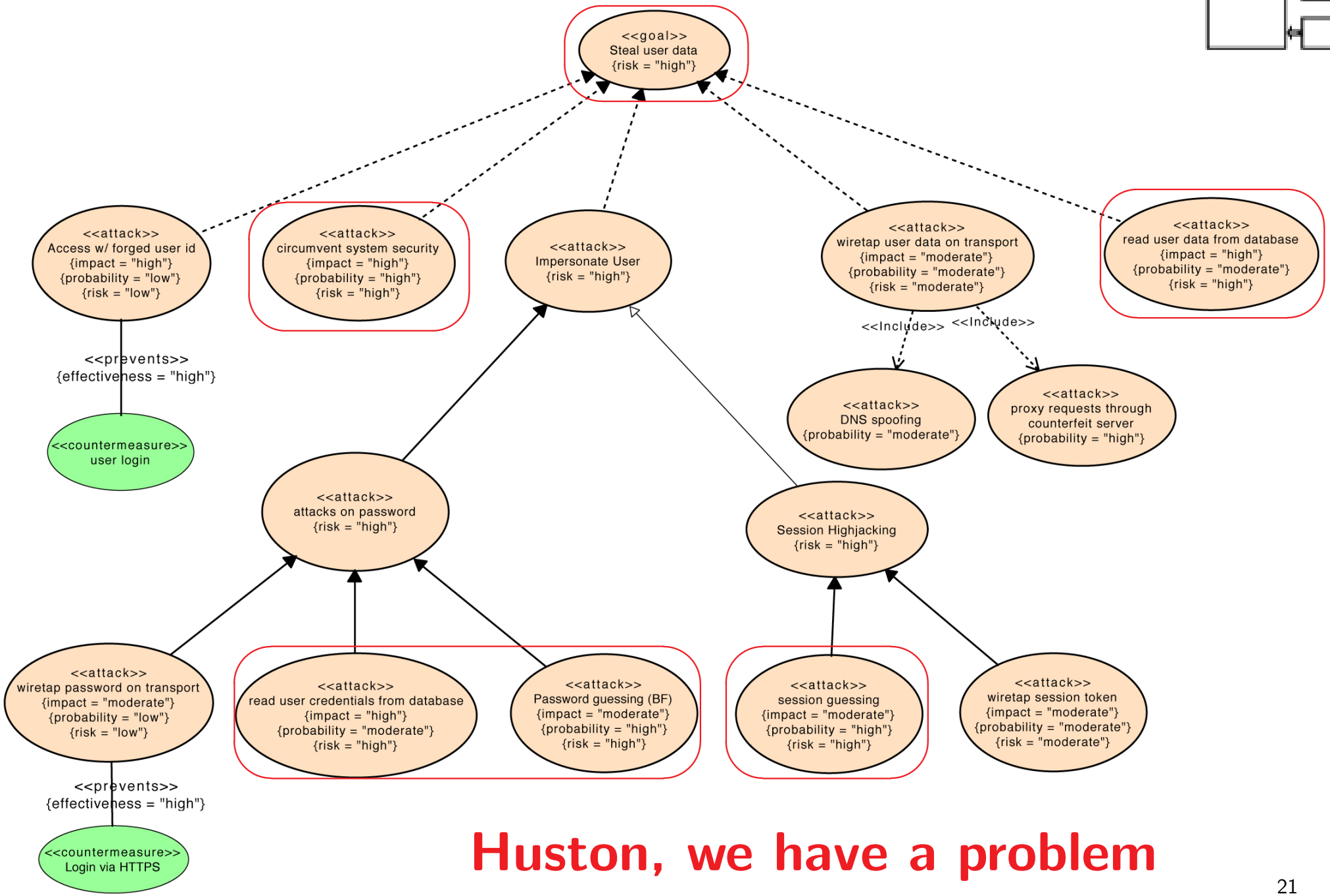
- Risk management and safeguards
- Security design principles and best practices
- Automatically generating security implementation
- ☞ **Security design options**
- Security design as an iterative process

Recall system architecture



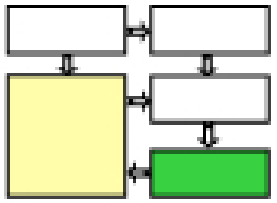
Threat Model (Version 1)





Huston, we have a problem

Security design options

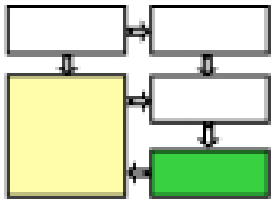


- Threats can be countered in many ways.
- Countermeasures can be categorized as follows:
  1. **Integrate or configure existing security mechanisms**
     Utilize system security mechanisms
  2. **Implement security functions in application logic**
  3. **Refactor the software architecture**
     Reduce "attack surface"
- Of course, do not lose sight of other requirements!

  Performance, scalability, user experience, cost, ...

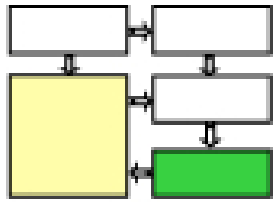
Let us look at these categories in more detail

1) Security mechanism integration



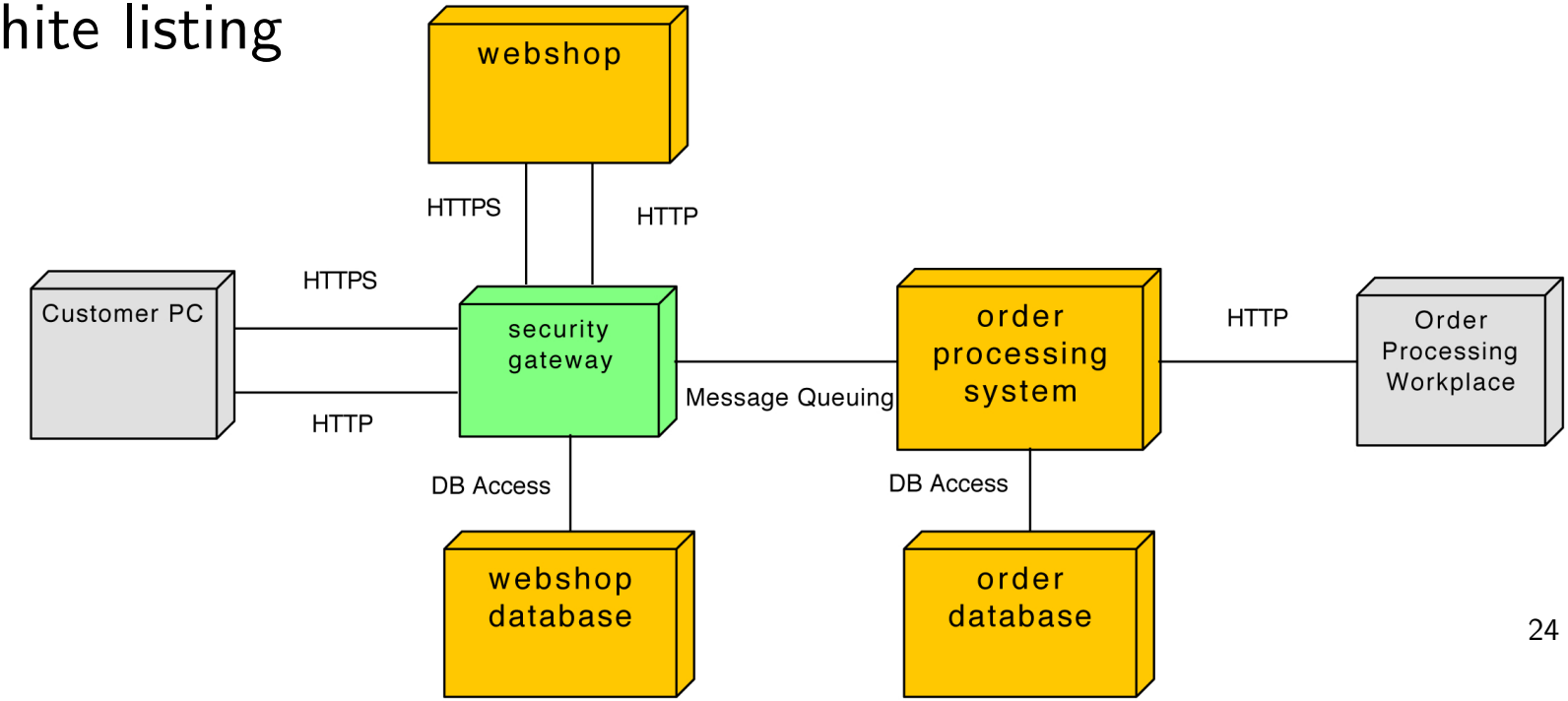
- **Integrate, configure, or employ existing mechanisms**
  - Network: firewalls, VPNs, network-authentication and access control (e.g. based on IP addresses)
  - OS: hardening, file system access control
  - Application/webserver: transport encryption, authentication, access control (e.g., based on URL)
- **Pros**
  - Baseline protection (standard recommended measures, e.g., BSI))
  - No application support required, so good for black-box integration
  - Usually well-understood by IT departments and administrators
- **Cons**
  - Application layer attacks cannot be stopped at the network level
  - Nontrivial, e.g., deep packet inspection is complex ($\leadsto$ vulnerabilities) and causes bottlenecks

1) Mechanism integration example



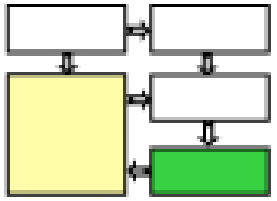
Example: Security Gateway (BSI baseline protection)

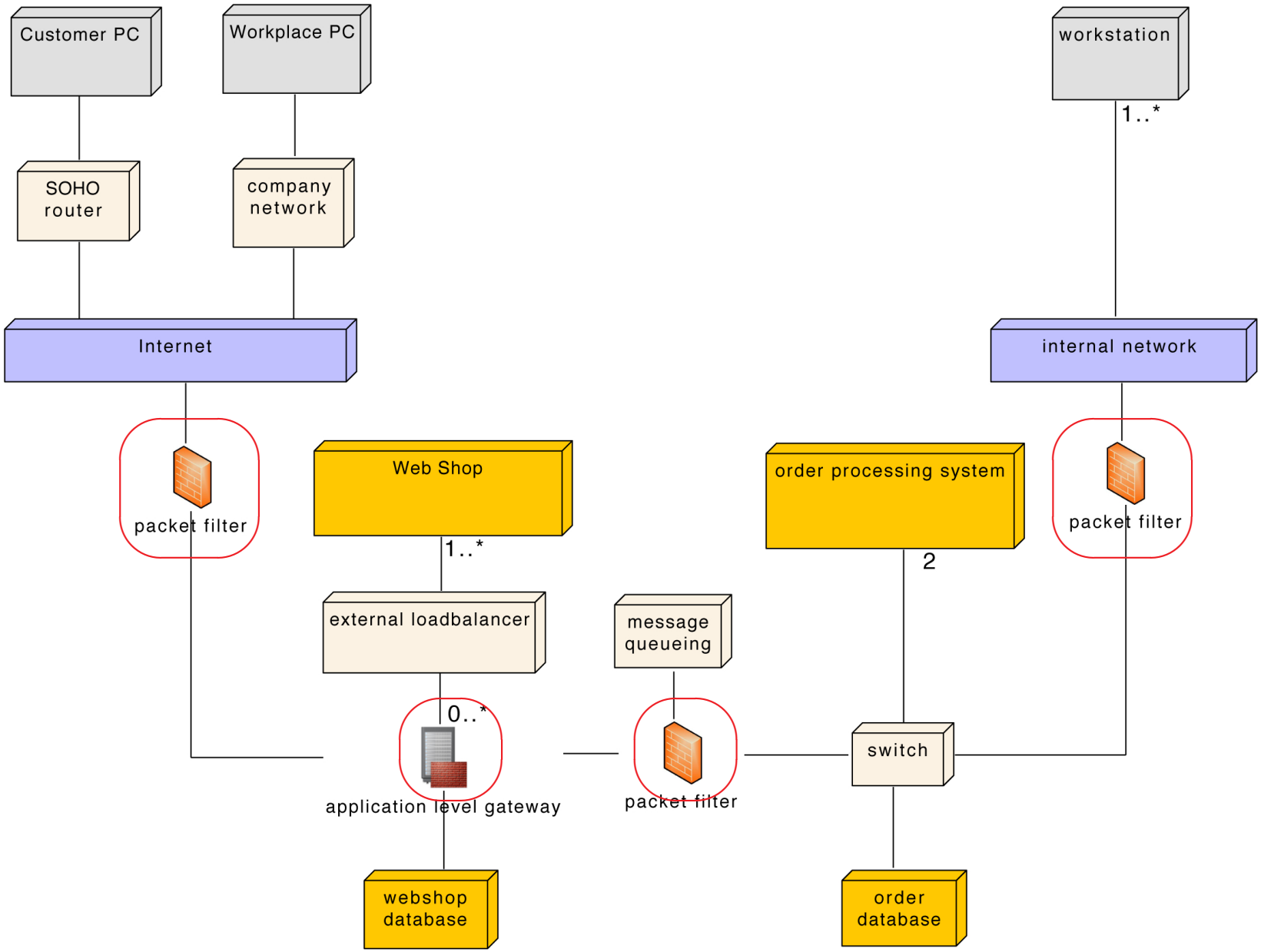
- All traffic from/to Internet and between webshop and database is routed through, and inspected by, the gateway
- Protection services:
  - Network segregation, traffic flow control (intrusion containment)
  - Intrusion detection (guessing attacks)
  - URL white listing



1) Mechanism integration example

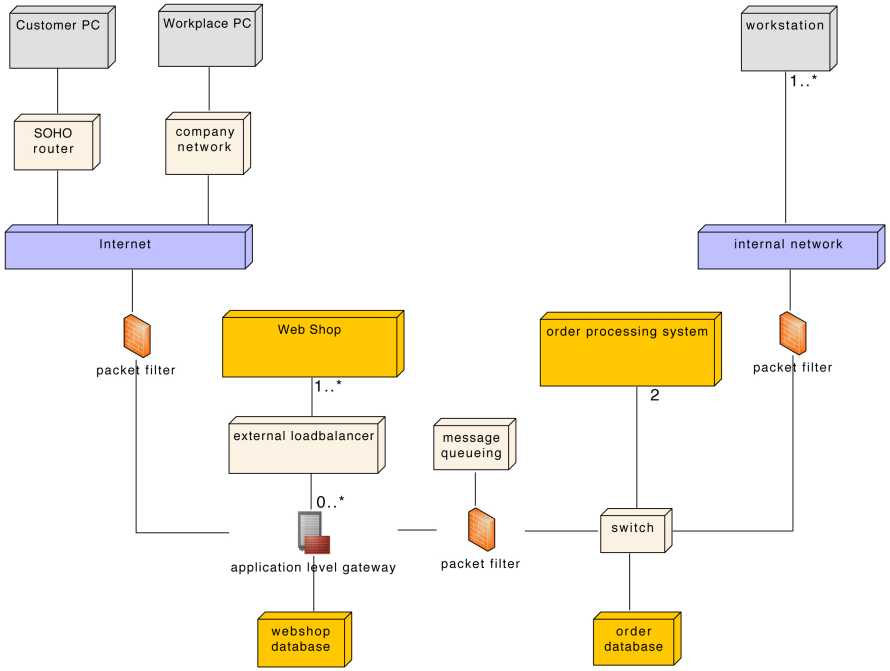
Security gateway: details of network architecture





1) Integration example

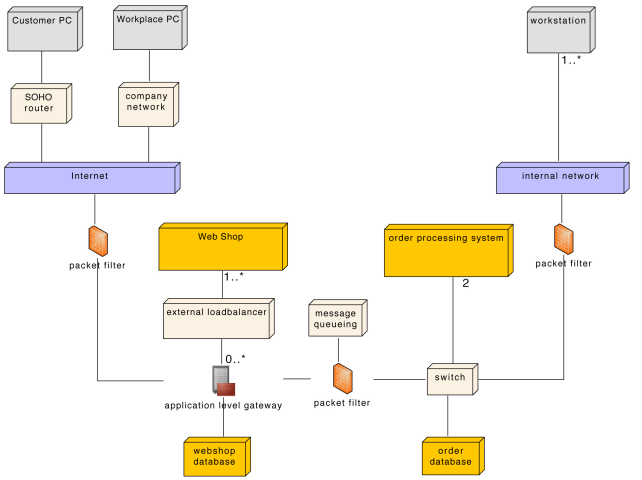
Effects on threat model



- **Attack:** Circumvent system security
  - Countermeasure effectiveness: **high**
    - Only required services exposed, intrusion impact is minimized
  - Impact: **high**
  - Probability: **low** (from **high**)
  - Risk: **moderate** (from **high**)
  - Drawback: another critical component for performance, scalability, and availability

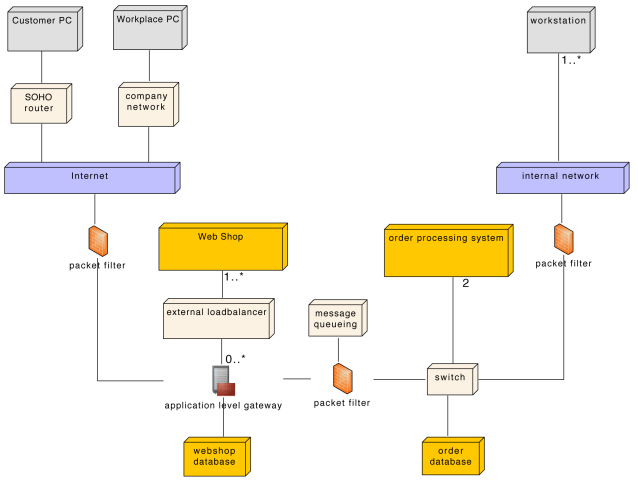
Integration example

Effect on threat model (cont.)



- **Attack:** URL guessing
  - Countermeasure effectiveness: **moderate**
    - Application state and dynamic URL parameters not considered
  - Impact: **moderate**
  - Probability: **moderate** (from **high**)
  - Risk: **moderate** (from **high**)
- **Attack:** Session guessing
  - Countermeasure effectiveness: **moderate**
    - Detection only and administrative intervention required
    - Detection of untypical behavior depends on experience
  - Impact: **moderate**
  - Probability: **moderate** (from **high**)
  - Risk: **moderate** (from **high**)

1) Integration example (cont.)

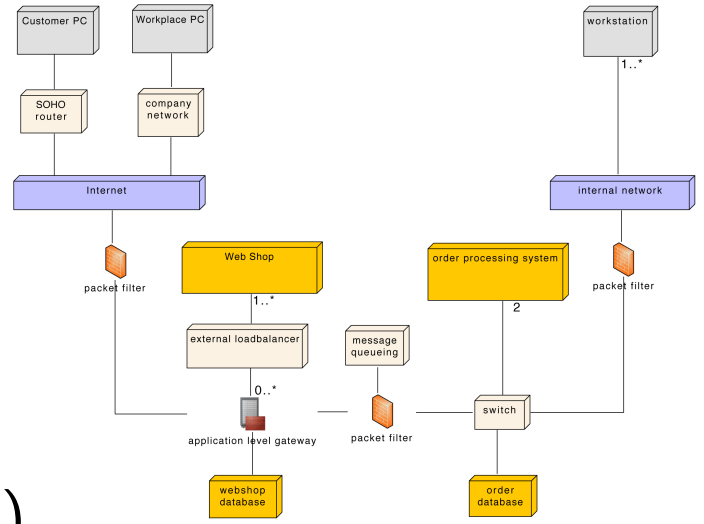


- Secure setup of webserver[3] **(next page)**
  - **Attack:** Circumvent system security
  - Countermeasure effectiveness: **high** (further line of defense)
  - Impact: **high** Probability: **low** (from **high**)
  - Risk: **moderate** (from **high**)
- IP banning after repeated failed login attempts at gateway
  - **Attack:** brute-force password attacks
  - Countermeasure effectiveness: **moderate**
    Repeated logins from one IP address may be proxy access, not an attack. Moreover, attackers may use many IP addresses.
  - Impact: **high** Probability: **moderate** (from **high**)
  - Risk: **high**

[3] Sicherheit von Webanwendungen Manahmenkatalog und Best Practices. See www.bsi.bund.de.

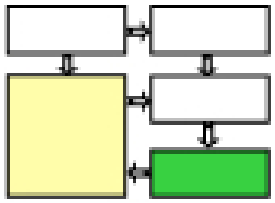
BSI Web Best Practices (sample)

1) Integration example (cont.)



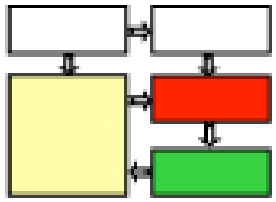
- Utilization of application server and database system functions
  - Transport Security (HTTPS) (wiretapping)
    - Already considered in threat model
  - User authentication (access with forged user id)
    - Already considered in threat model
  - Store hashed user passwords
    - **Attack**: read user credentials from database
    - Countermeasure effectiveness: **high**
      Additional salt hardens against rainbow-table attacks
    - Impact: **high**
    - Probability: **low** (from **high**)
    - Risk: **moderate** (from **high**)

2) Implement security in the application



- Some countermeasures best implemented within application
  - I/O validation & application-specific checks (e.g. input credit card number)
  - Application-level encryption (for communication or of database content)
  - Access control
- Requires that security design is integrated into the development process
- Use of standard security APIs and modules is advisable
- **Pros**
  - Best fit to application (end-to-end security argument)
  - No additional system components (e.g. network appliances), reduces costs for licensing and operation
- **Cons**
  - Expensive (depending on development method)
  - Error prone, test intensive

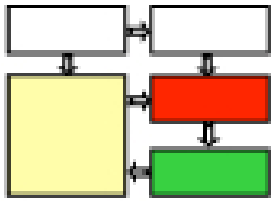
2) Secure application (cont.)



- Session management using unpredictable session tokens
  - **Attack:** session guessing
  - Countermeasure effectiveness: **high**
  - Impact: **moderate**
  - Probability: **low** (from **high**)
  - Risk: **moderate** (from **high**)
- Input validation
  - **Attack:** SQL injection
  - Countermeasure effectiveness: **high**
  - Impact: **high**
  - Probability: **low** (from **high**)
  - Risk: **moderate** (from **high**)
- More on these topics in the implementation module

2) Secure application

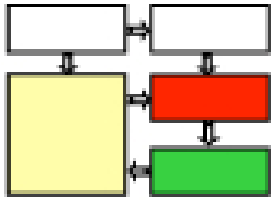
Example: credit card data protection



- Encryption at application level
  - **Attacks**: read user data from database
  - Countermeasure effectiveness: **high**
  - Impact: **high** Probability: **low** (from **high**)
  - Risk: **moderate** (from **high**)
- Masking of credit card number on all user interfaces
  - **Attack**: session guessing
  - Countermeasure effectiveness: **moderate**
    Prevents disclosure of credit card numbers (reduces impact)
  - Impact: **low** (from **moderate**) Probability: **low**
  - Risk: **low** (from **moderate**)

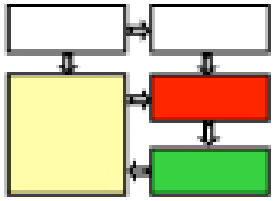
2) Secure application

Example: Credit card data protection (cont.)



- Always ask for checking number (CVC)
  - **Attack**: Reuse of stolen data
  - Countermeasure effectiveness: **moderate**
    Prevents unauthorized credit card transactions (reduces impact)
  - Impact: **low** (from **moderate**)
  - Probability: **low**
  - Risk: **low** (from **moderate**)

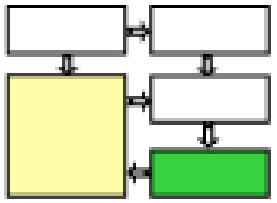
2) Secure application



Example: authorization policy enforcement on all application levels

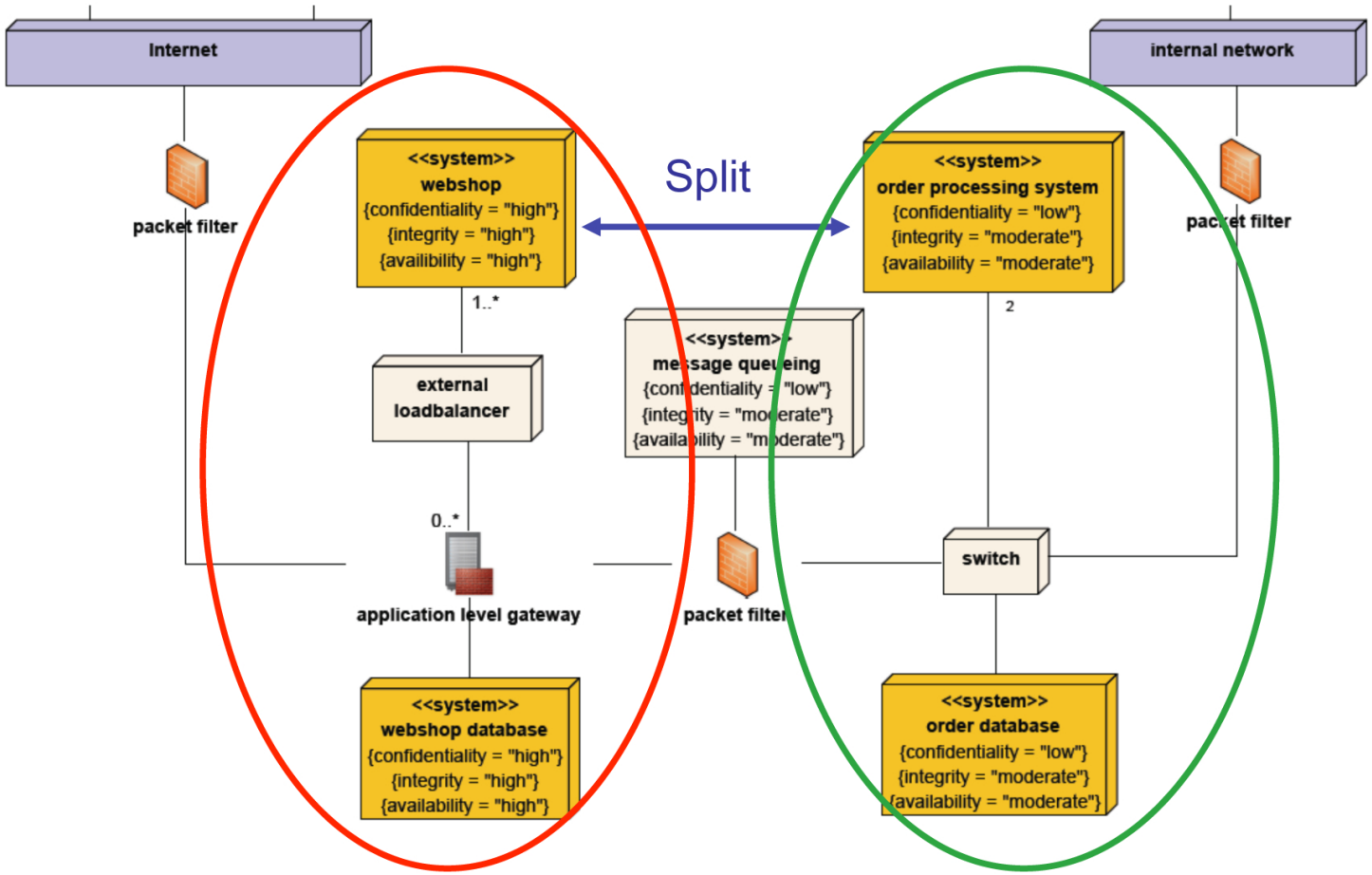
- Components (generating source code automatically!)
  - User interface
  - Business logic
  - Backend services, including database access
- Assessment
  - **Attack**: URL guessing
  - Countermeasure effectiveness: **high**
  - Impact: **moderate**
  - Probability: **low** (from **high**)
  - Risk: **moderate** (from **high**)

3) Refactor software architecture



- **Improve/simplify security design**
  - Reduce attack surface
  - Split systems into highly-critical and less-critical parts
  - Allocate security-sensitive functions to other systems
- **Pros**
  - Simplifies security design (also cheaper and less error-prone)
  - Sometime the only way to achieve security goals
- **Cons**
  - Significant impact on overall project
  - May have negative impact on other functions
  - Causes strongest resistance and conflicts

3) Refactoring example



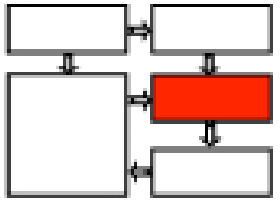
- Split eStore into two applications with dedicated databases
  - **Web Shop**: high availability/security requirements
  - **Order Processing System**: moderate security/availability requirements
- Result
  - Lower operational cost for order processing (simpler architecture)
  - Establishes different security zones

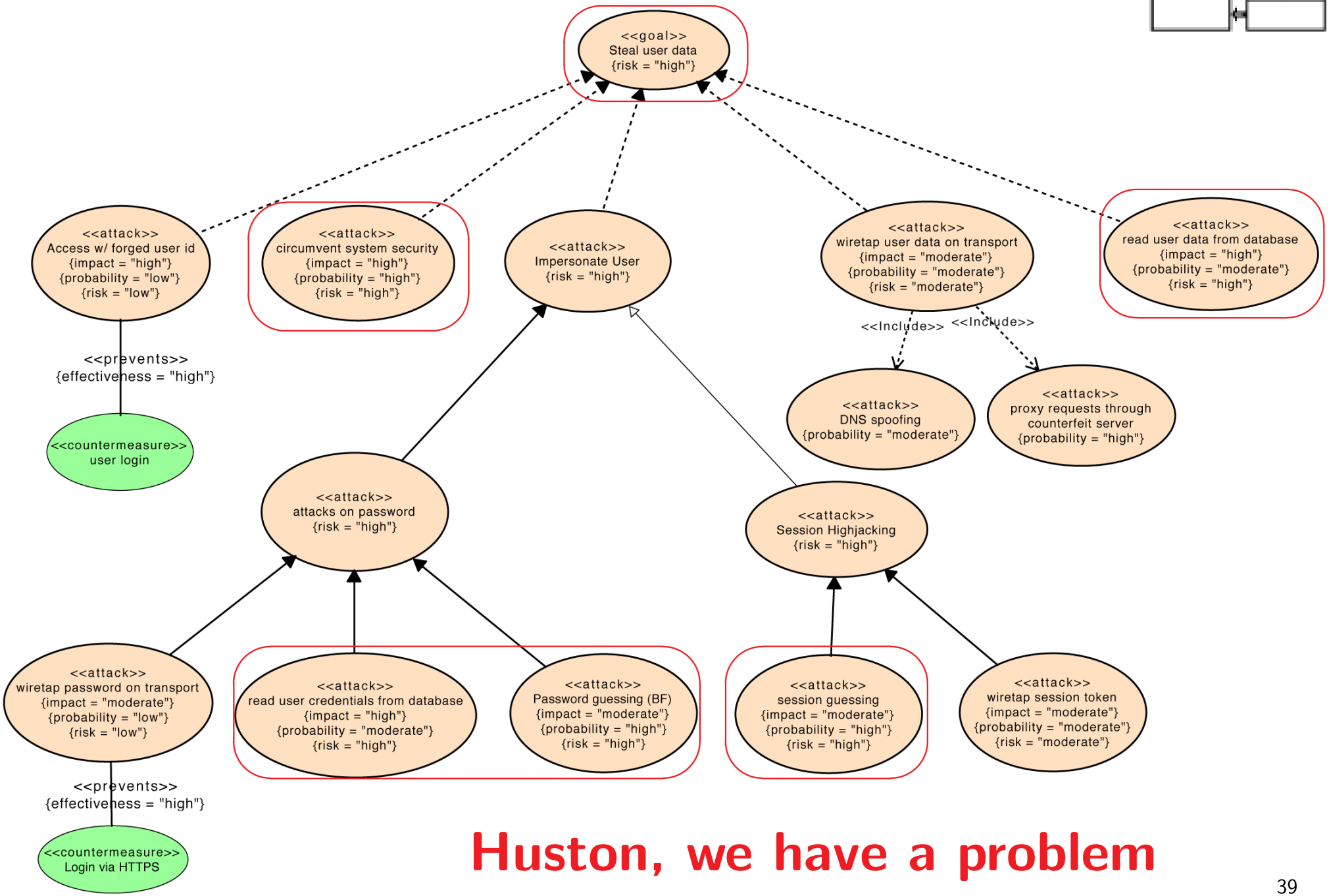
Road map

- Risk management and safeguards
- Security design principles and best practices
- Automatically generating security implementation
- Security design options

☞ **Security design as an iterative process**

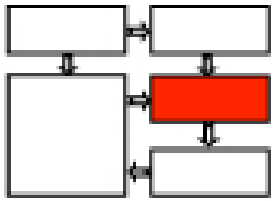
Threat Model (Version 1)

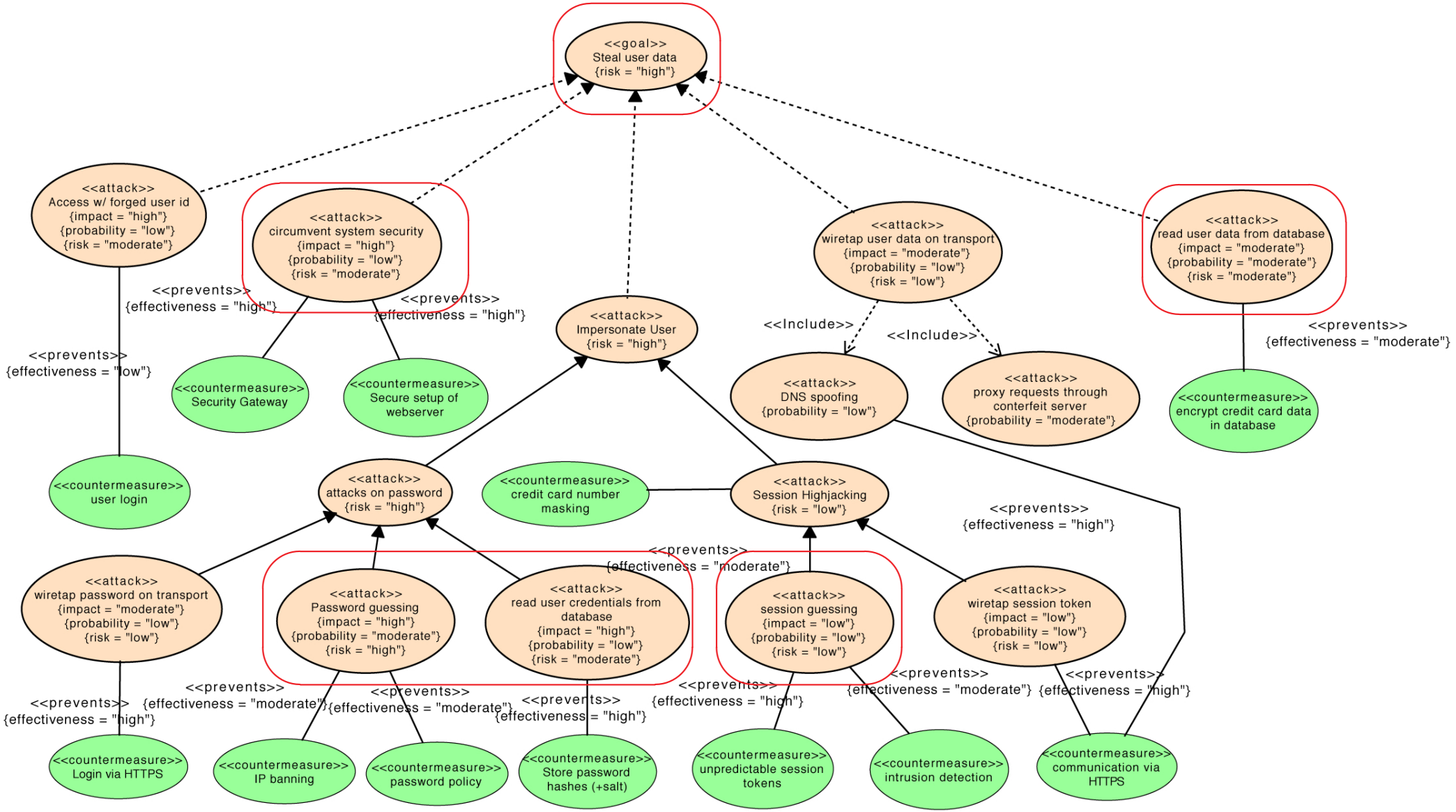




Huston, we have a problem

Threat Model (Version 2)

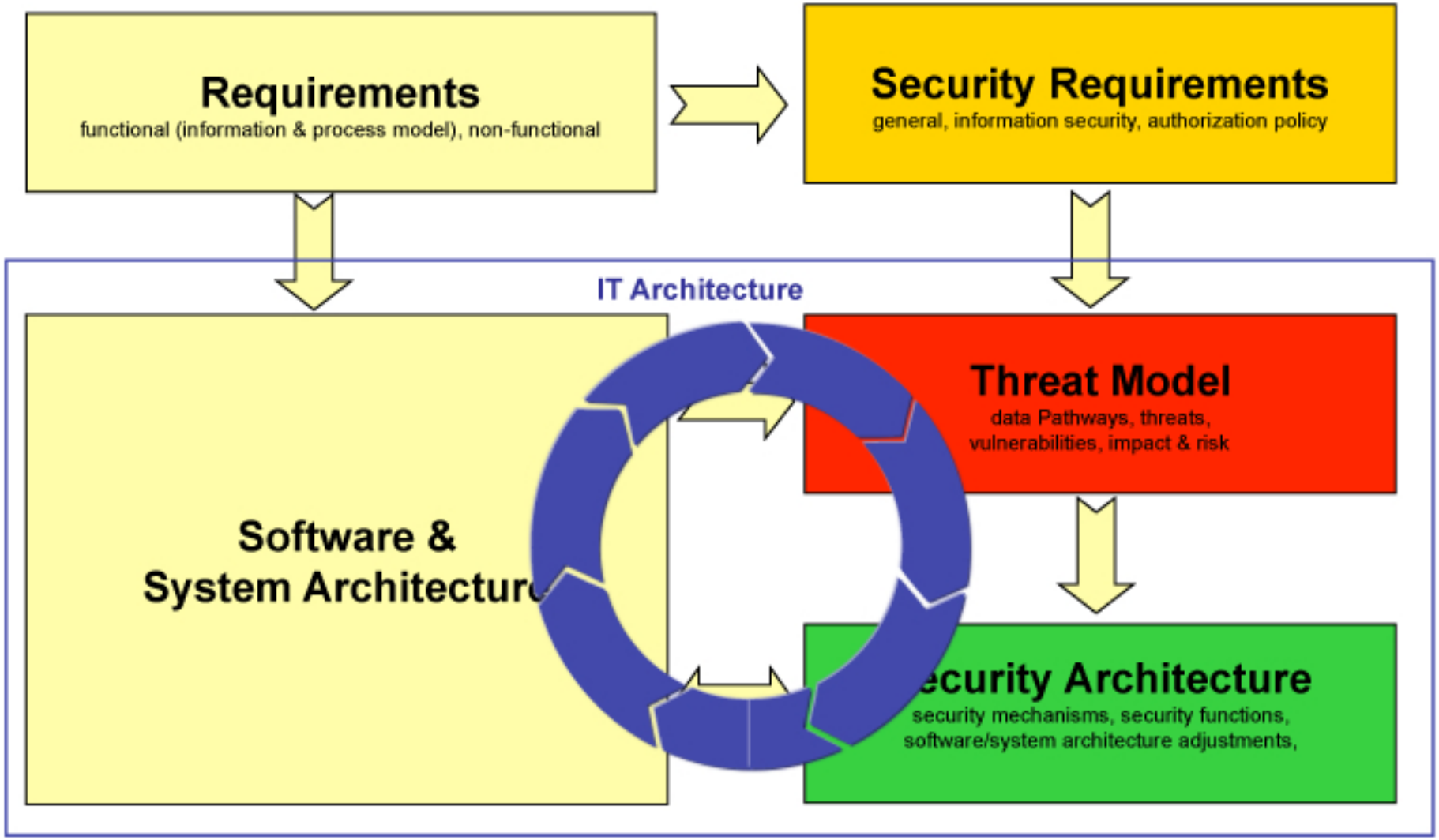




Evaluation and additional considerations

- Critical areas (from earlier analysis)
  - 💣 Input/output validation (XSS, injection, ...)
  - 💣 E-password management and authentication
    - We didn't even consider phishing and password recovery!
  - ✔ Credit card management/payment (but expensive)
    - Credit card data encryption, masking, checking code, and quality assurance
- Another alternative: transfer responsibility
  - ▶ Authentication by 3rd party identity provider, e.g. via OpenId
  - ▶ Payment processing by dedicated payment provider

Security design as an iterative process



Lessons learned

- Risk can be mitigated or transfered
- Mitigation based on different safeguards
- Security design is a creative process guided by principles
- Measures fall into different categories
- Threat model and risk analysis must be synchronized with design
- Code generation can improve quality of security implementations

Literature

- OWASP Resources, www.owasp.org
- BSI Resources, www.bsi.bund.de
- BSI Best Practices for Web Applications (google!)
- D. Basin, J. Doser, T. Lodderstedt, Model Driven Security: from UML Models to Access Control Infrastructures, ACM Transactions on Software Engineering and Methodology, 2006