university of koblenz
Computer Science

6. Security

Engineering Web and Data-intensive Systems

Dr. Volker Riediger - Winter Term 2022/23

ist *Institute for Software Technology*

Security

- Web Application Security
- Access Control

This lesson is informational - content will not be part of examinations related to winter term 2022/23.

Image: colourbox.de

Disclaimer: This lecture only scratches the surface of application security and privacy by giving some examples.

To address those very important topics more in-depth, please consider specialized courses like

Advances in Secure Software Engineering

Advanced Topics in Web-based and Data-intensive Software and its Security

Data Protection in Software Development

Security in computers networks and mobile systems

6.1 Web Application Security

Related Web Resources

- OWASP - Open Web Application Security Project
- CVE - Common Vulnerabilities and Exposures
- NIST CSRC - Computer Security Resource Center
- NVD - National Vulnerability Database
- CWE - Common Weakness Enumeration
- CPE - Common Platform Enumeration
- CVSS - Common Vulnerability Scoring System
- German Authorities
  - BSI
  - CERT Bund
- …and many, many more…

No Privacy without Security…

- **Security**
  - Secure communication
  - Secure data storage
  - Authentication
  - Authorization
  - Access control
  - Monitoring
  - Auditing/Logging
  - …

- **Privacy**
  - Data protection
  - Data categorization
  - Data minimization
  - Data governance
  - Need-to-know principle
  - Access policies
  - Laws, regulations
  - …

Top 10 Security Risks in Web Applications

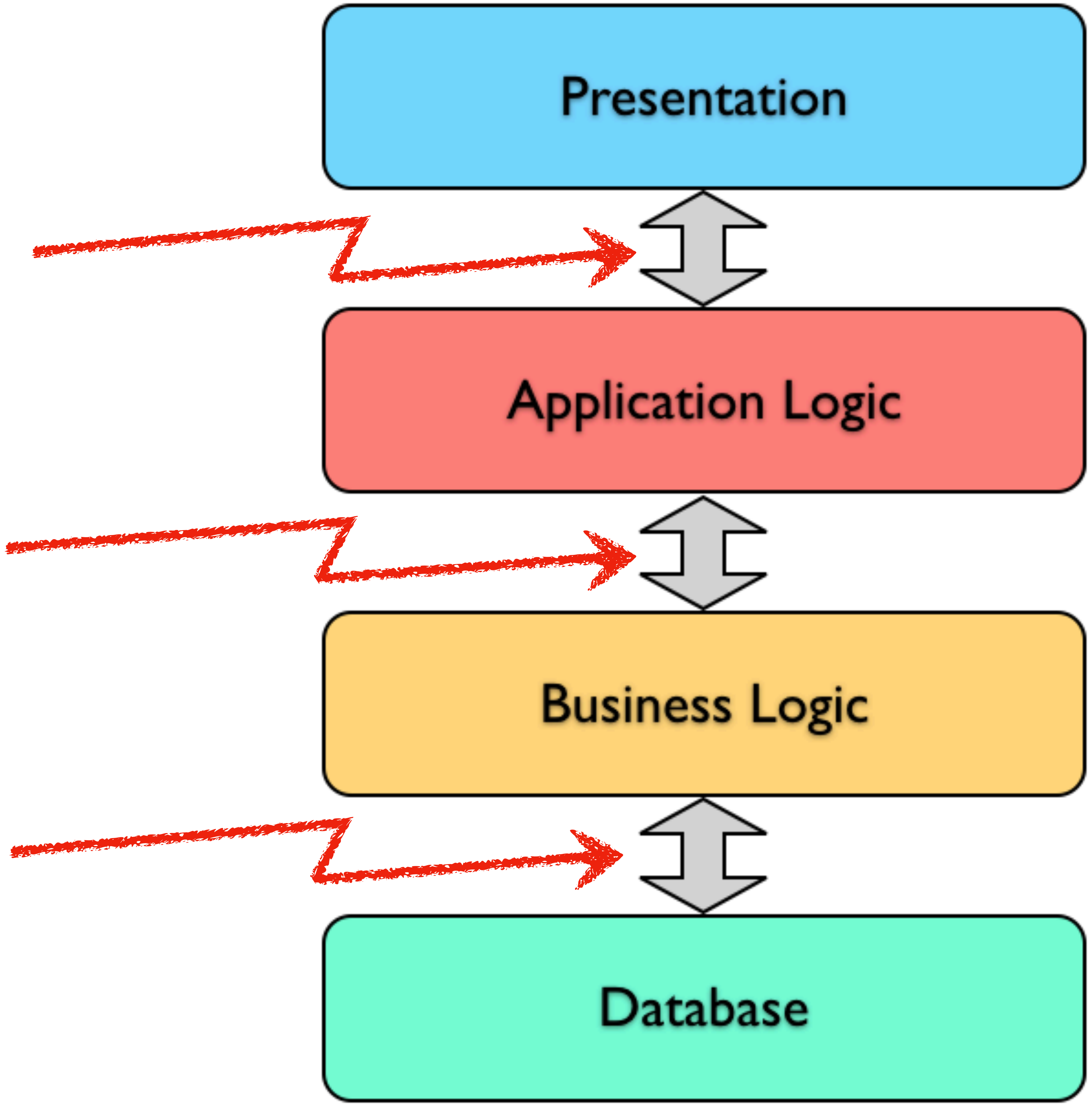
according to OWASP

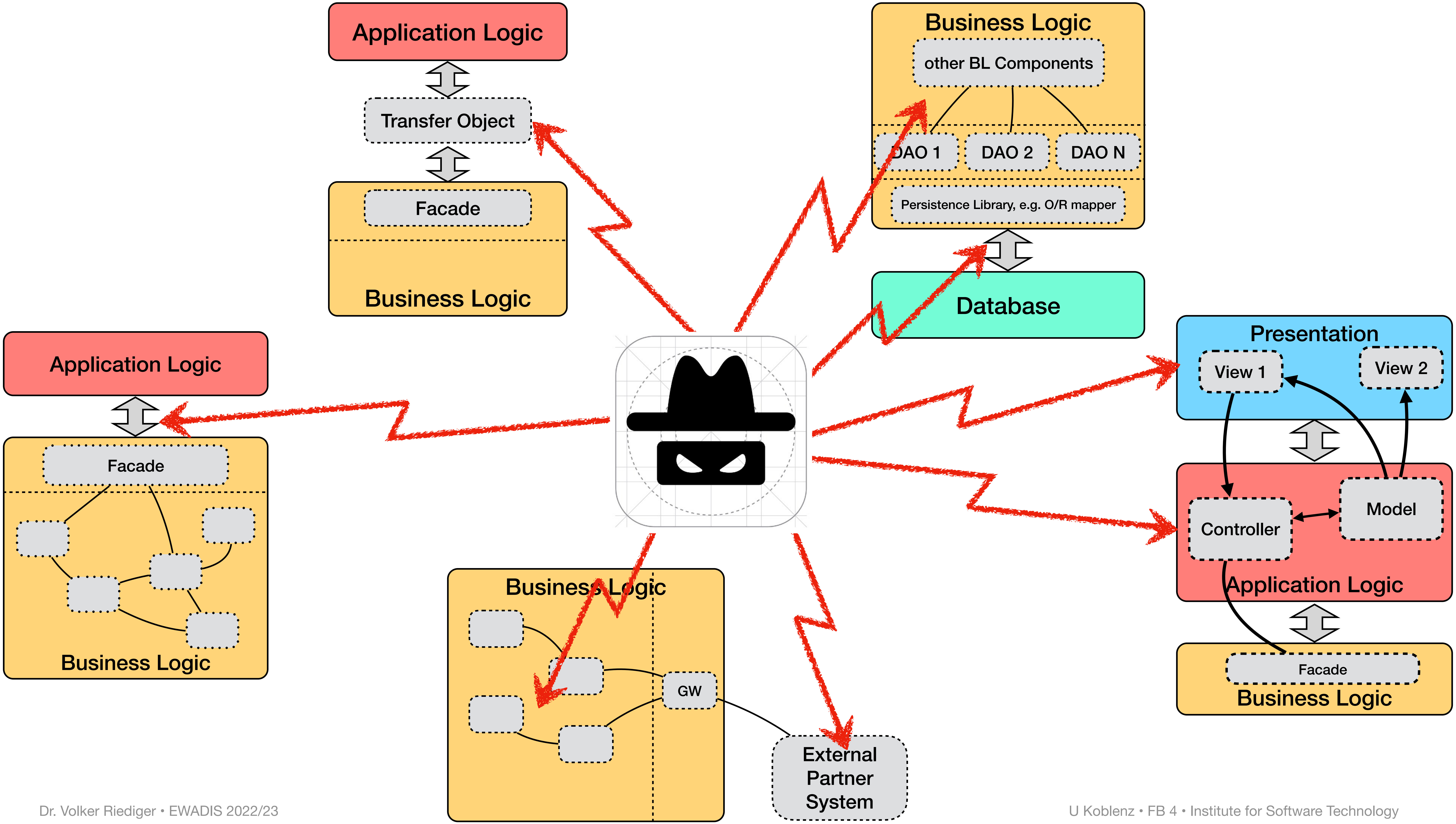
1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

⇢ Please read the individual descriptions on the OWASP Project Top 10 pages!

Attacks…

- Outsiders, Insiders
- Whitehats, Blackhats
- External attacks via network connections
- Interfaces at layer boundaries especially important w.r.t. security
- …but of course not limited to those interfaces ⇛ also pay attention to internals

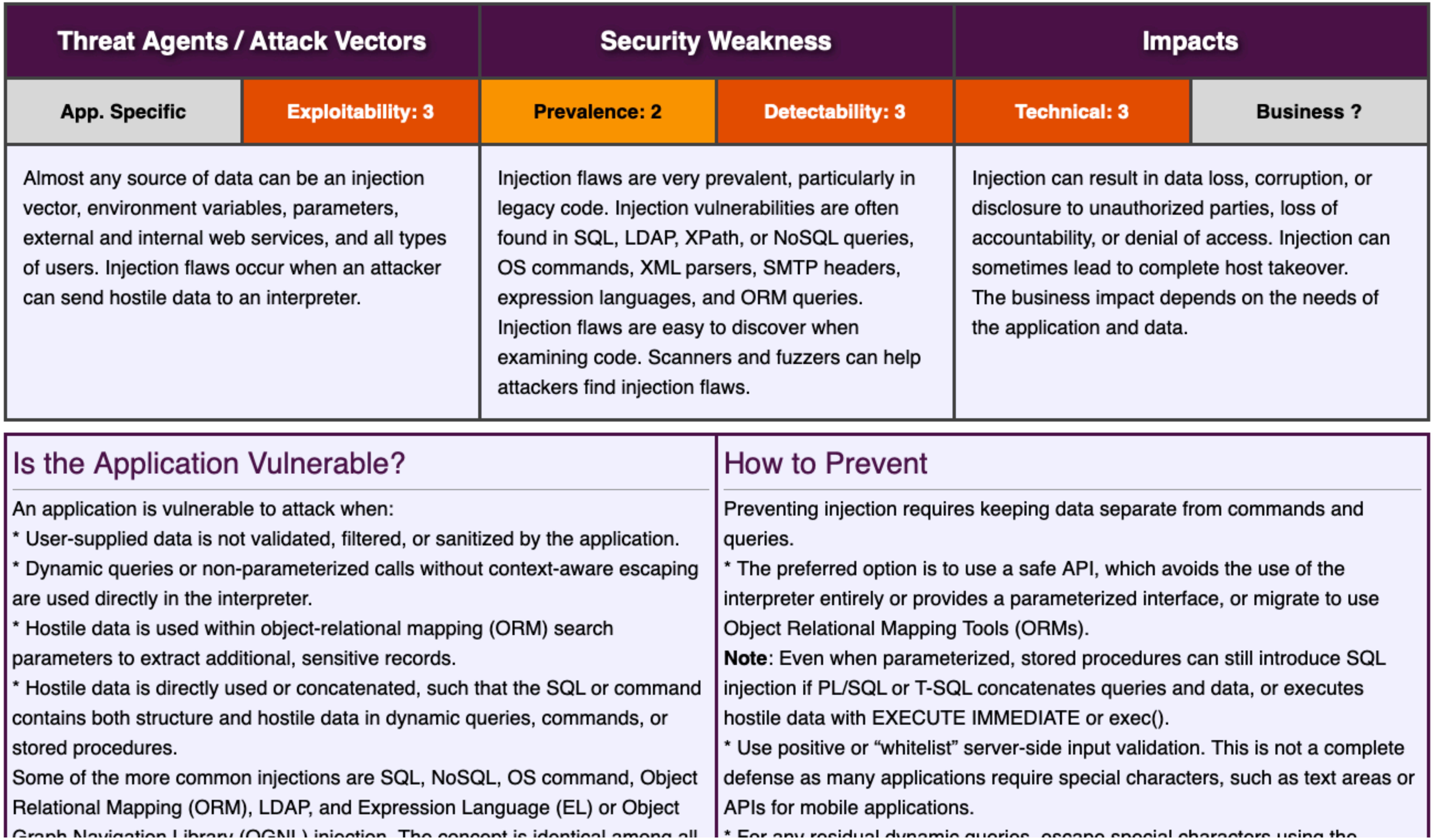




[Spy Icon from colourbox.com]

Injection

[OWASP Page]



Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

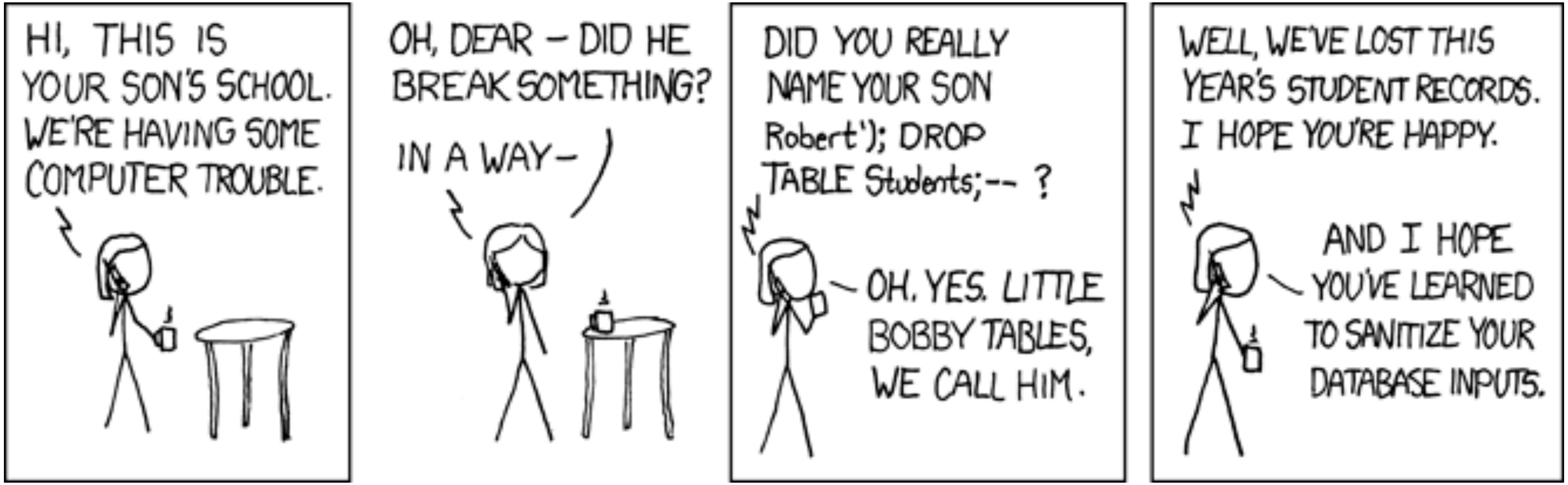
Is the Application Vulnerable?

An application is vulnerable to attack when:
* User-supplied data is not validated, filtered, or sanitized by the application.
* Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
* Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
* Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.
Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all

How to Prevent

Preventing injection requires keeping data separate from commands and queries.
* The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).
Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
* Use positive or “whitelist” server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
* For any residual dynamic queries, escape special characters using the

Injection - “Little Bobby Tables”

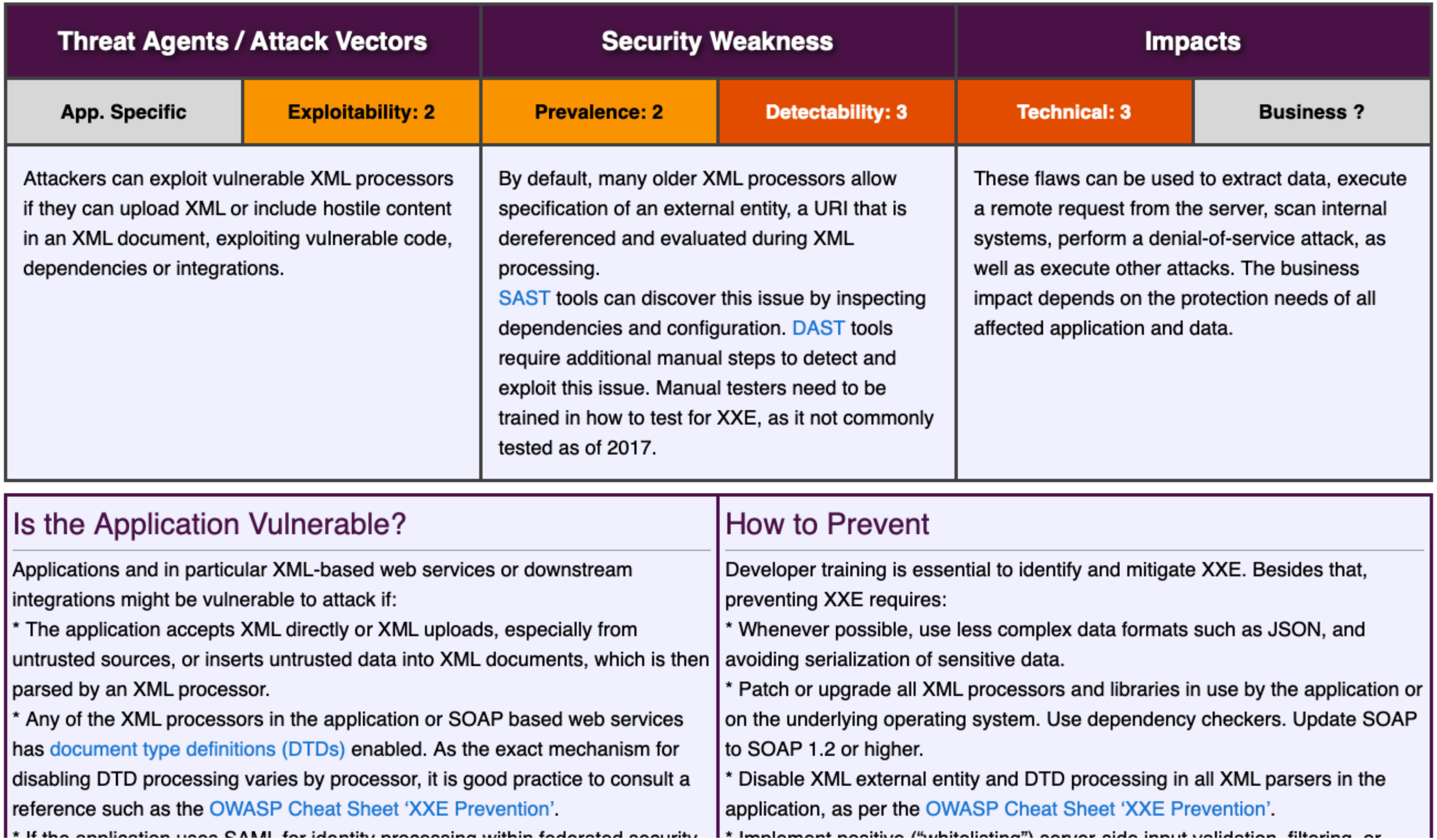


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

- https://xkcd.com/327/
- https://www.explainxkcd.com/wiki/index.php/Little_Bobby_Tables

XML External Entities

[OWASP Page]



Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.		By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. SAST tools can discover this issue by inspecting dependencies and configuration. DAST tools require additional manual steps to detect and exploit this issue. Manual testers need to be trained in how to test for XXE, as it not commonly tested as of 2017.		These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.	

Is the Application Vulnerable?

Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if:
* The application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.
* Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the OWASP Cheat Sheet 'XXE Prevention'.
* If the application uses SAML for identity processing within federated security

How to Prevent

Developer training is essential to identify and mitigate XXE. Besides that, preventing XXE requires:
* Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.
* Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system. Use dependency checkers. Update SOAP to SOAP 1.2 or higher.
* Disable XML external entity and DTD processing in all XML parsers in the application, as per the OWASP Cheat Sheet 'XXE Prevention'.
* Implement positive ("whitelisting") server-side input validation, filtering, or

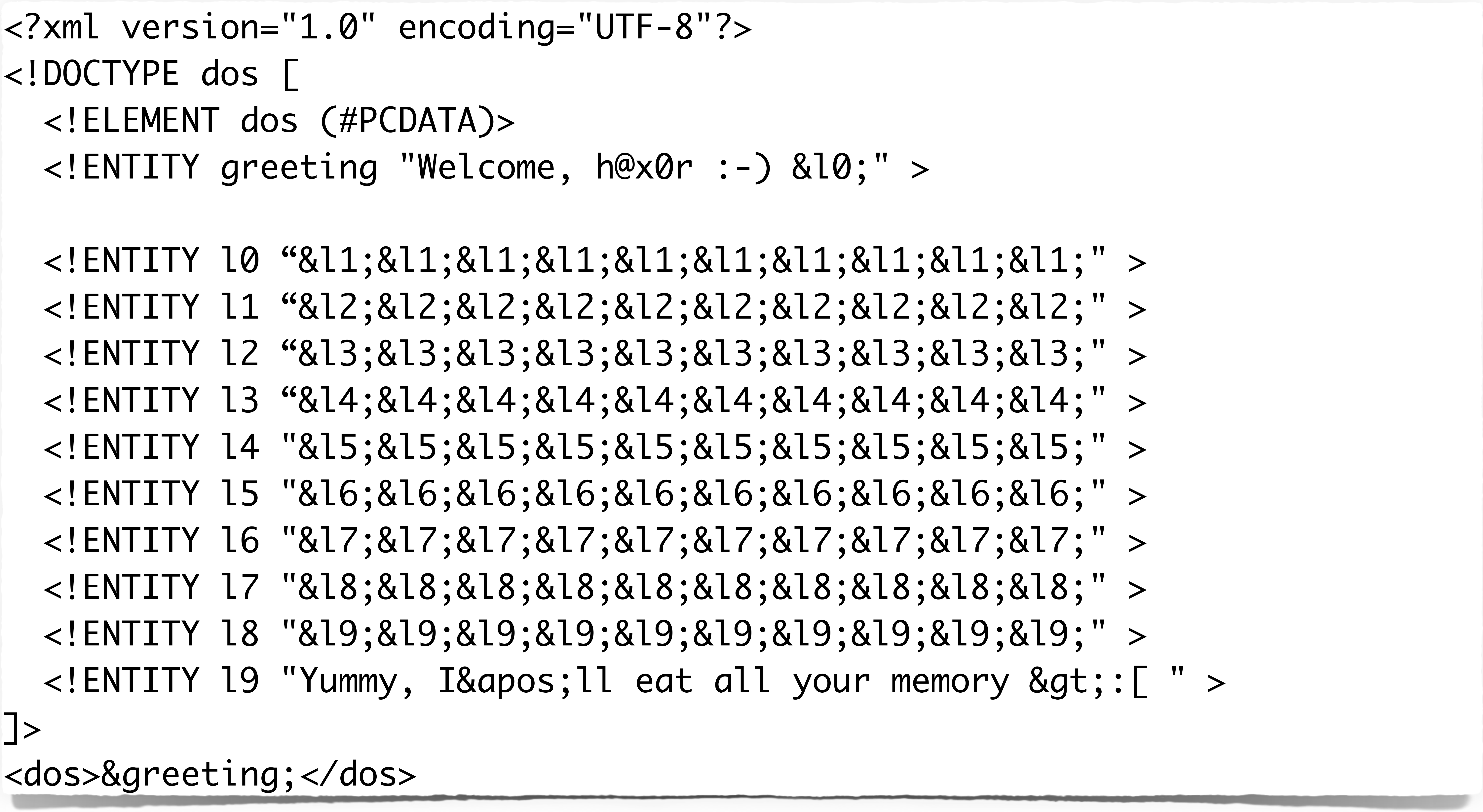
Example: XML DOS (Denial Of Service)

“XML bomb”

- DOS attack on XML processors
- DTD processing expands entities due to standard specification
  - Entities can contain other entity references
  - Expansion creates huge DOM tree
  - High CPU load; system runs out of memory

- Possible mitigations
  - Don’t forward user (attacker) provided XML to vulnerable processors
  - Disable DTD processing (technology is outdated anyway)
  - Use XML schema language instead

The code - you may try it in your browser…

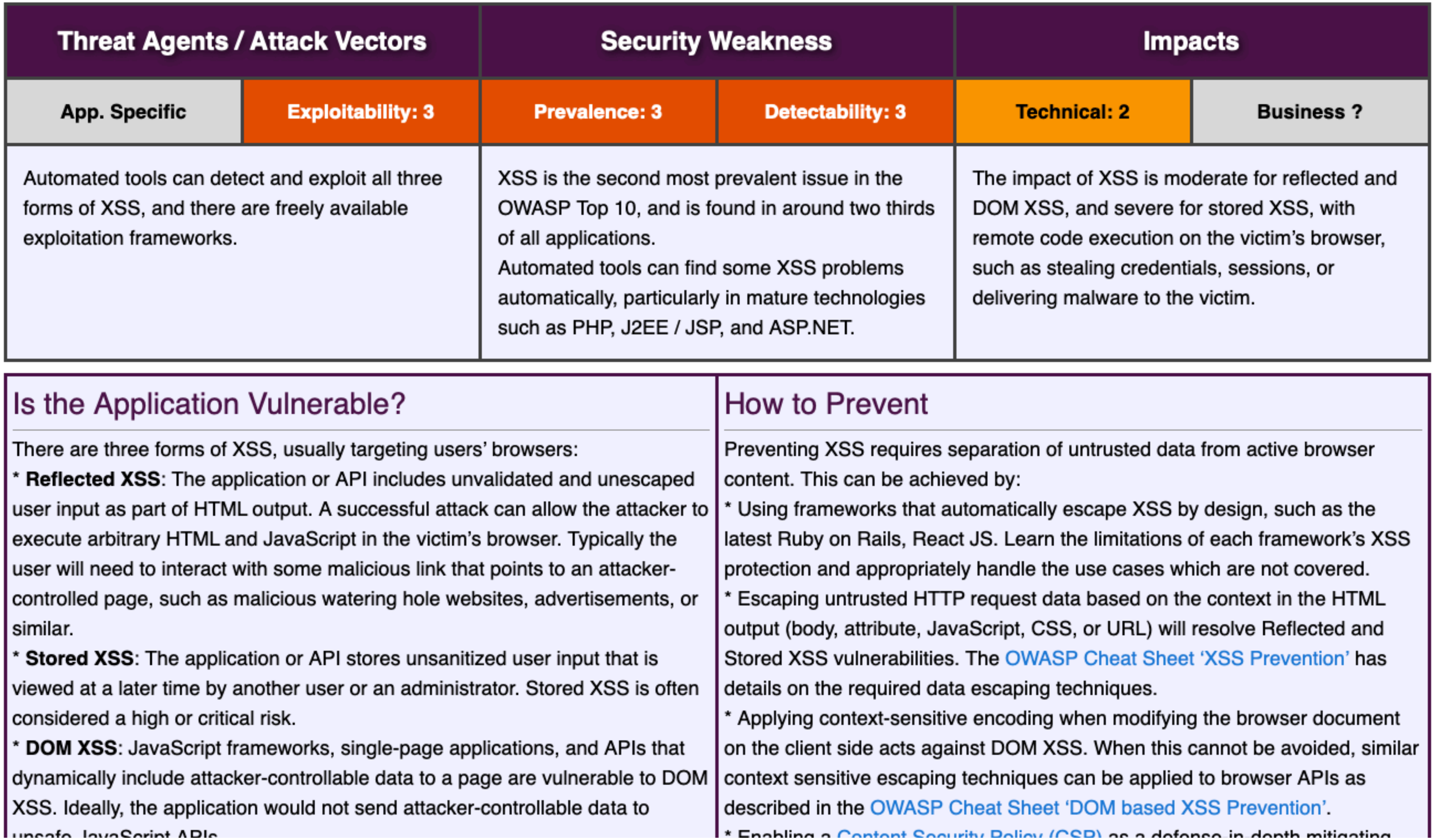


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dos [
  <!ELEMENT dos (#PCDATA)>
  <!ENTITY greeting "Welcome, h@x0r :-) &l0;" >

  <!ENTITY l0 “&l1;&l1;&l1;&l1;&l1;&l1;&l1;&l1;&l1;&l1;" >
  <!ENTITY l1 “&l2;&l2;&l2;&l2;&l2;&l2;&l2;&l2;&l2;&l2;" >
  <!ENTITY l2 “&l3;&l3;&l3;&l3;&l3;&l3;&l3;&l3;&l3;&l3;" >
  <!ENTITY l3 “&l4;&l4;&l4;&l4;&l4;&l4;&l4;&l4;&l4;&l4;" >
  <!ENTITY l4 "&l5;&l5;&l5;&l5;&l5;&l5;&l5;&l5;&l5;&l5;" >
  <!ENTITY l5 "&l6;&l6;&l6;&l6;&l6;&l6;&l6;&l6;&l6;&l6;" >
  <!ENTITY l6 "&l7;&l7;&l7;&l7;&l7;&l7;&l7;&l7;&l7;&l7;" >
  <!ENTITY l7 "&l8;&l8;&l8;&l8;&l8;&l8;&l8;&l8;&l8;&l8;" >
  <!ENTITY l8 "&l9;&l9;&l9;&l9;&l9;&l9;&l9;&l9;&l9;&l9;" >
  <!ENTITY l9 "Yummy, I&apos;ll eat all your memory &gt;:[ " >
]>
<dos>&greeting;</dos>
```

Proper entity expansion would need approx. 70GB of memory and a lot of CPU power.

Cross Site Scripting (XSS)

[OWASP Page]



Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.		XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.		The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.	

Is the Application Vulnerable?

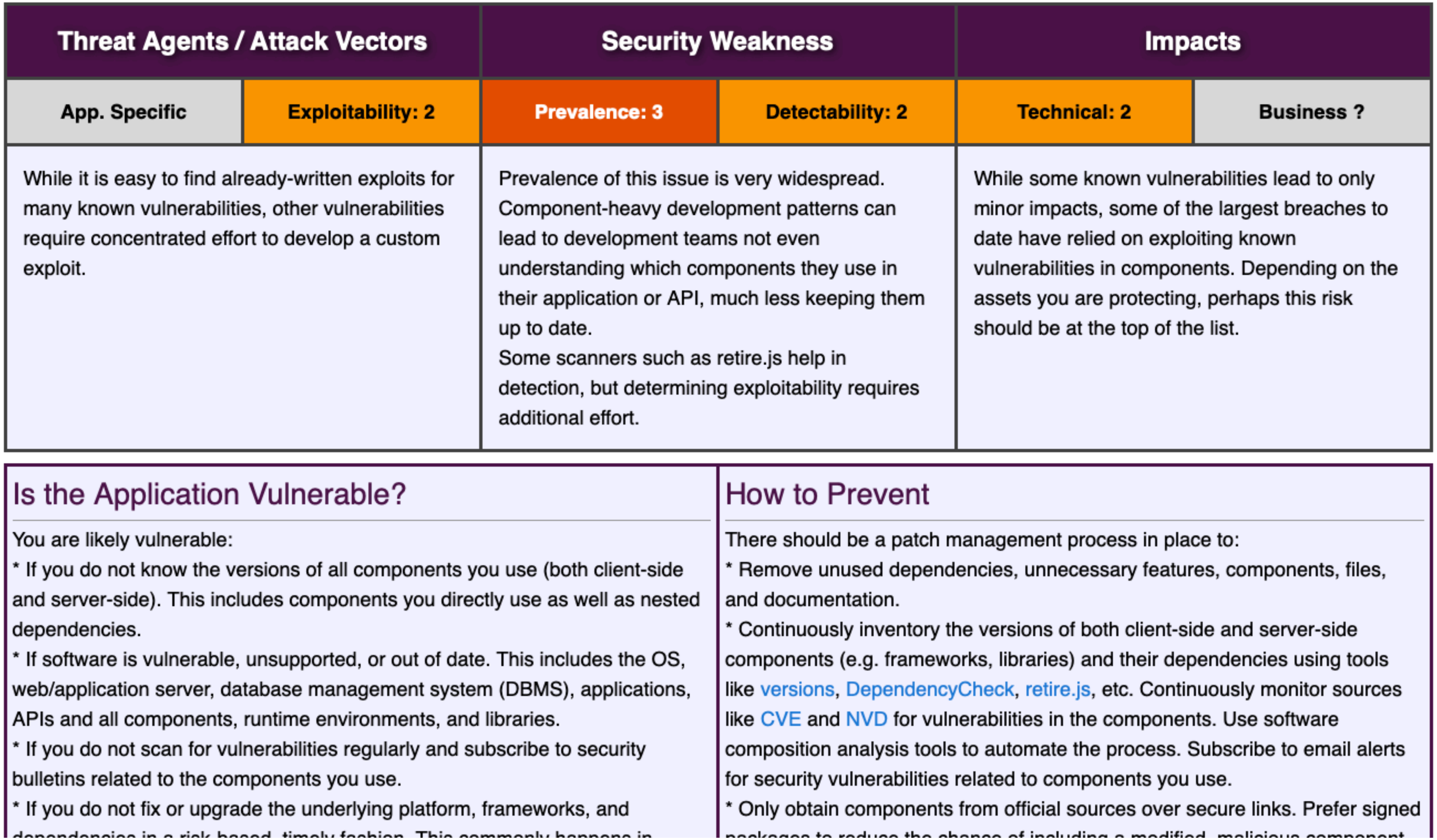
There are three forms of XSS, usually targeting users' browsers:
* **Reflected XSS**: The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.
* **Stored XSS**: The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.
* **DOM XSS**: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs.

How to Prevent

Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:
* Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
* Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP Cheat Sheet 'XSS Prevention' has details on the required data escaping techniques.
* Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the OWASP Cheat Sheet 'DOM based XSS Prevention'.
* Enabling a Content Security Policy (CSP) as a defense-in-depth mitigating

Vulnerable Components

[OWASP Page]



Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.		Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date. Some scanners such as retire.js help in detection, but determining exploitability requires additional effort.		While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.	

Is the Application Vulnerable?

You are likely vulnerable:
* If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
* If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
* If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
* If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in

How to Prevent

There should be a patch management process in place to:
* Remove unused dependencies, unnecessary features, components, files, and documentation.
* Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
* Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component

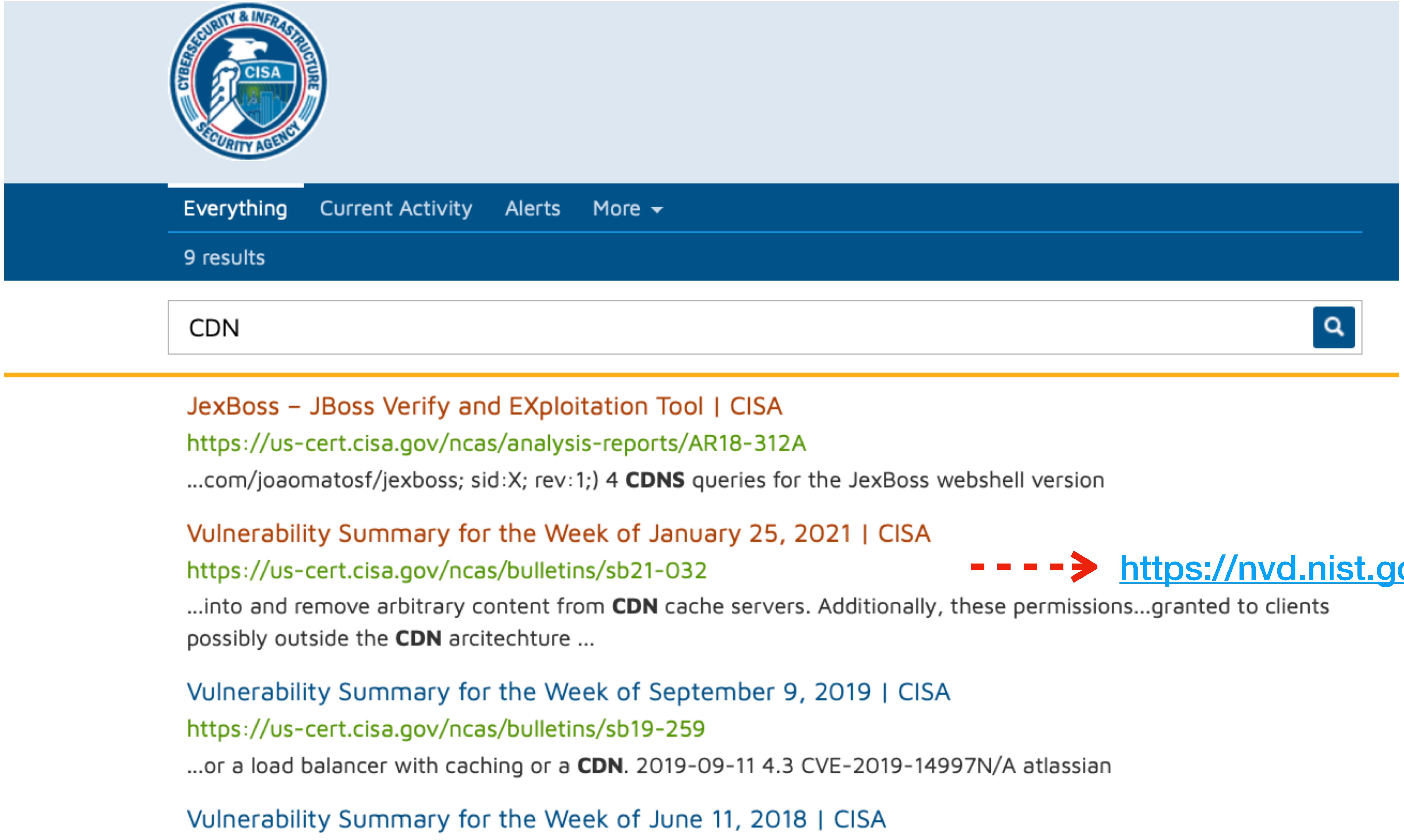
Security Risk: CDN

CDN = Content Delivery Network

- Distribution of popular CSS/JS libraries and other resources
- Advantages
  - Static content delivered by nodes close to the client
  - Saves local server load and bandwidth
- Disadvantages
  - Compromised CDN servers
  - Manipulated libraries
  - Malicious code injection
  - Trojans
  - …
- Possible Mitigations
  - Don’t use CDNs at all
  - Use `integrity` attribute for `<script>` tags, CSS libraries, and other active content
  - Use fixed version numbers
  - Regularly check for changes/updates

Recent Reports (visited 2021-02-11)

https://search.us-cert.gov/search?utf8=✓&affiliate=us-cert&query=CDN



Everything | Current Activity | Alerts | More ▾

9 results

CDN

JexBoss – JBoss Verify and EXploitation Tool | CISA
https://us-cert.cisa.gov/ncas/analysis-reports/AR18-312A
...com/joaomatosf/jexboss; sid:X; rev:1;) 4 **CDNS** queries for the JexBoss webshell version

Vulnerability Summary for the Week of January 25, 2021 | CISA
https://us-cert.cisa.gov/ncas/bulletins/sb21-032
...into and remove arbitrary content from **CDN** cache servers. Additionally, these permissions...granted to clients possibly outside the **CDN** arcitechture ...

Vulnerability Summary for the Week of September 9, 2019 | CISA
https://us-cert.cisa.gov/ncas/bulletins/sb19-259
...or a load balancer with caching or a **CDN**. 2019-09-11 4.3 CVE-2019-14997N/A atlassian

Vulnerability Summary for the Week of June 11, 2018 | CISA

- - - -> https://nvd.nist.gov/vuln/detail/CVE-2020-17522

6.2 Access Control

Access Control

- **Access control** means enforcing which subject has which type of access to an object
  - subject = person, machine, process, …
  - object = any asset, e.g., piece or group of information, in a system
- Access control comprises
  - configuration (management),
  - authentication
  - and authorization.
- Usually provided by an IdM/IAM system (identity and access management)

- **Configuration**
  - defines subjects, groups, roles, …
  - permissions, rules, policies, ….
- **Authentication**
  - ensures identity of a subject
  - subject provides credentials, e.g. user id and password
  - authentication system checks validity and assigns groups, roles, and other attributes
- **Authorization**
  - permits or denies operations on objects

Access Control Decision

$$acd : subject \times object \times operation \times context \rightarrow \{\texttt{allow}|\texttt{deny}\}$$

- An **access control decision** is computed by a boolean function that can take various inputs and returns `allow` or `deny`.
- Inputs to decisions (among others)
  - subject and it's attributes, e.g. identity, roles, groups, trust level, clearance, …
  - object and it's attributes, e.g. type, sensitivity level, confidentiality level, label, …
  - requested operation, e.g. create, read, update, delete, execute, access, connect, transmit, …
  - context information, e.g. location, time, ip address, protection level, trust level, …
- In general, the access permission check is done before the operation is executed.
- With long-running operations, it can be necessary to re-check permissions periodically

DAC - Discretionary Access Control

- Access control based on subject and object properties
- Subjects may transfer permissions to other subjects
- Usually has a concept of an owner of an object
- Owners can define or change permissions at their own discretion

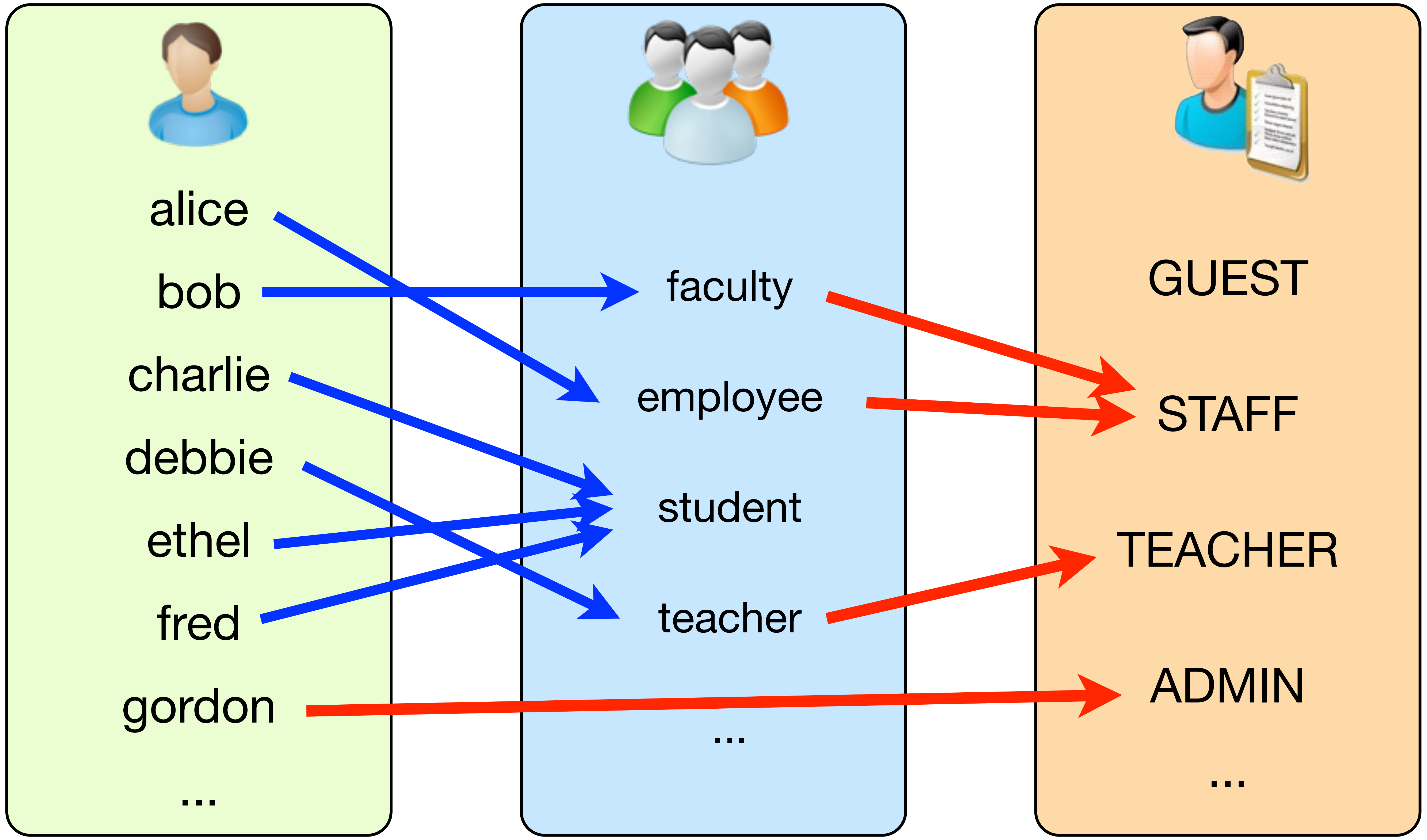
- e.g. traditional file/directory permissions in UNIX systems
  - entries have an *owner* and a *group*
  - permissions *read*, *write*, *execute* for owner, group, and others
  - in early systems, transferring ownership of a program to another user (e.g., “root”…) could be used execute with escalated privileges
- File systems were extended by *access control lists* (ACLs) with more fine-grained control and inherited permissions

MAC - Mandatory Access Control

- Strict access control enforced in an organization, usually based on
  - sensitivity levels
  - confidentiality levels
  - trust levels
  - security clearance of individuals
  - need-to-know principle
- Fine-grained constrained permissions for access
- Distinct individual permissions to modify access rules, in contrast to DAC
- MAC evolved and was used mainly in military/intelligence and public/ health administration context

RBAC - Role Based Access Control

- Permissions based on roles, groups and individual subjects, as well as object properties
- Subjects can have many roles and groups, roles can be assigned to many subjects
- Groups defined usually by organizational assignment
- Role definition can be application specific
- Same subject can interact with a system in different roles



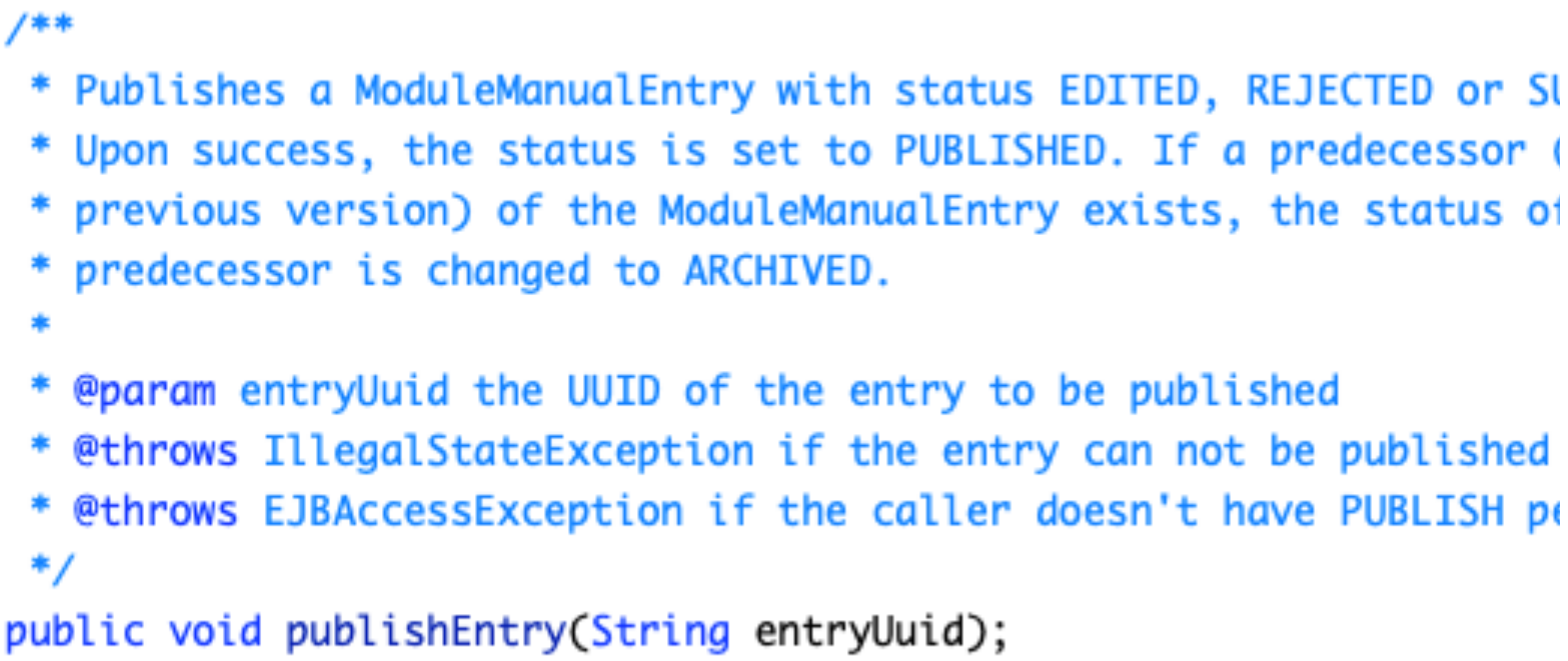
RBAC - Example Permission Definition

Subject Role	Administrator	Besitzer	(Studien-)Dekan	Rechtsabteilung	Beauftragter	Leiter	QS-Beauftragter	Lehrperson	Öffentlich
Data Object	ADMIN	OWNER	DEAN	LEGAL_ DEPARTMENT	DELEGATE	HEAD	QA_DELEGATE	TEACHER	(anonymous)
Masterdata	CRaUDX	r	r	r	r	r	r	r	r
OrganizationalUnit	CRaUDX	r	r	r	r	r	r	r	r
ProgramGroup	CRaUDPAX	RU	RUPA	R	RU	Rp	Rp	Rp	Rp
ProgramOfStudy	CRaUDPAX	RU	RUPA	R	RU	Rp	Rp	Rp	Rp
ModuleGroup	CRaUDPX	RU	CRUDPA	R	CRUD	Rp	Rp	Rp	Rp
ModuleImport	CRaUDPX	RU	CR*UDPA	R*	CR*UD	R*	Rp	Rp	Rp
Module	CRaUDPX	RU	CRUDPA	R*	CRUD	CRUD	Rp	Rp	Rp
Course	CRaUDPX	RU	CRUDPA	R*	CRUD	CRUD	Rp	Rp	Rp
CourseImport	CRaUDPX	RU	CR*UDPA	R*	CR*UD	CR*UD	Rp	Rp	Rp
Role ADMIN	O	-	-	-	-	-	-	-	-
Role HEAD (Org.Unit)	O	-	-	-	-	-	-	-	-
Role DEAN (Org.Unit/ProgramOfStudy)	O	-	-	-	-	-	-	-	-
Role DELEGATE (Org.Unit/ProgramOfStudy	O	-	O	-	O	O	-	-	-
Role QA_DELEGATE (Org.Unit)	O	-	O	-	O	O	-	-	-
Role TEACHER (Course)	O	O	O	-	O	O	-	-	-
Role LEGAL_DEPARTMENT (Org.Unit/ProgramOfStudy)	O	-	O	-	O	-	-	-	-
Role OWNER (all entries)	O	-	O	-	O	O	-	-	-

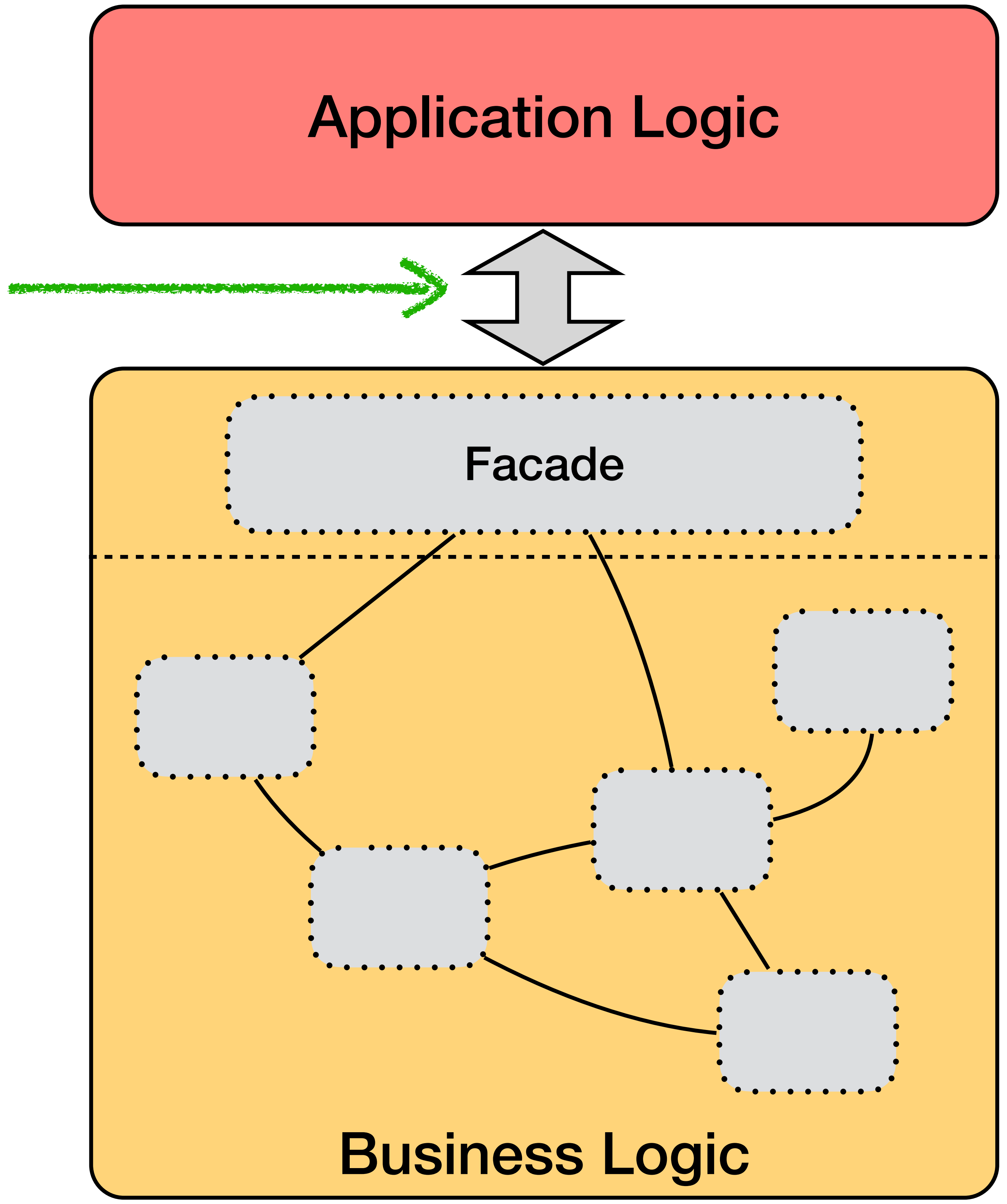
C	Create	create a data object inside another entity that is related to the role
r	Read	read master data entities that are not deleted
R	Read	read of PUBLISHED entities that are not deleted, if the role is assigned, also hidden entities
R*	Read	same as R, but also read moduleimports/courseimports that import entities that are related to the role
Ra	Read ALL	entails also deleted entries („trash“)
Rp	Read PUBLISHED	read PUBLISHED entities that are not deleted and not hidden
U	Update/Edit	implies access to EDITED/REJECTED/SUBMITTED entites
D	Delete	move to trash
P	Publish	
A	Archive	
X	Undelete / Remove	Remove = ultimately DELETE from Database
O	Assign/Remove Role	Admins can not remove their own admin role.
		Owner can be changed to any person, since access is still possible via DEAN, DELEGATE, ADMIN, HEAD

RBAC example (from Module Manual System)

Facade Interface



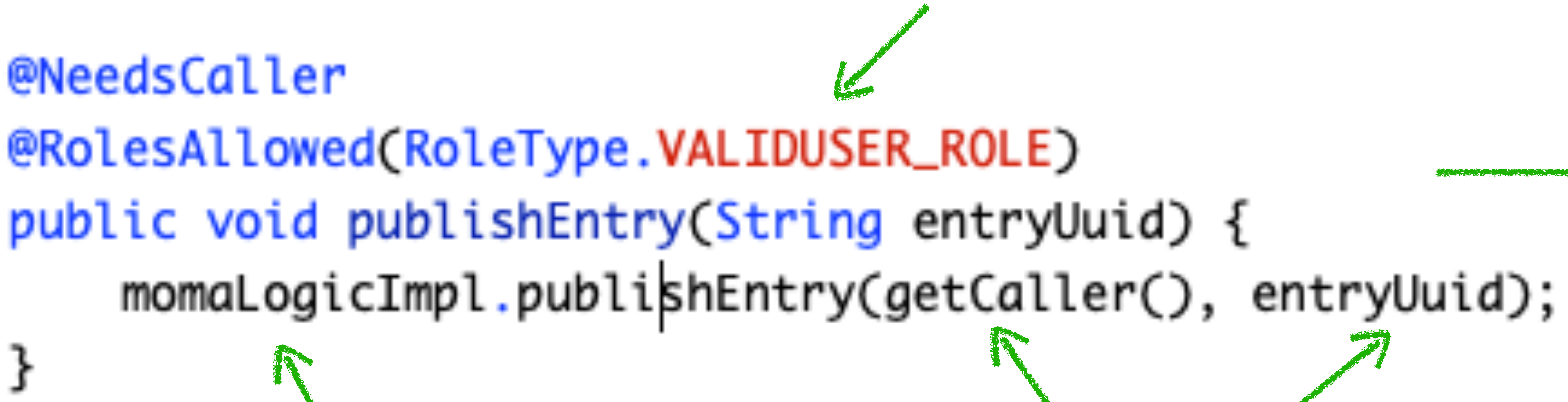
```
/**
 * Publishes a ModuleManualEntry with status EDITED, REJECTED or SU
 * Upon success, the status is set to PUBLISHED. If a predecessor (
 * previous version) of the ModuleManualEntry exists, the status of
 * predecessor is changed to ARCHIVED.
 *
 * @param entryUuid the UUID of the entry to be published
 * @throws IllegalStateException if the entry can not be published
 * @throws EJBAccessException if the caller doesn't have PUBLISH pe
 */
public void publishEntry(String entryUuid);
```



RBAC example (from Module Manual System)

Facade Implementation

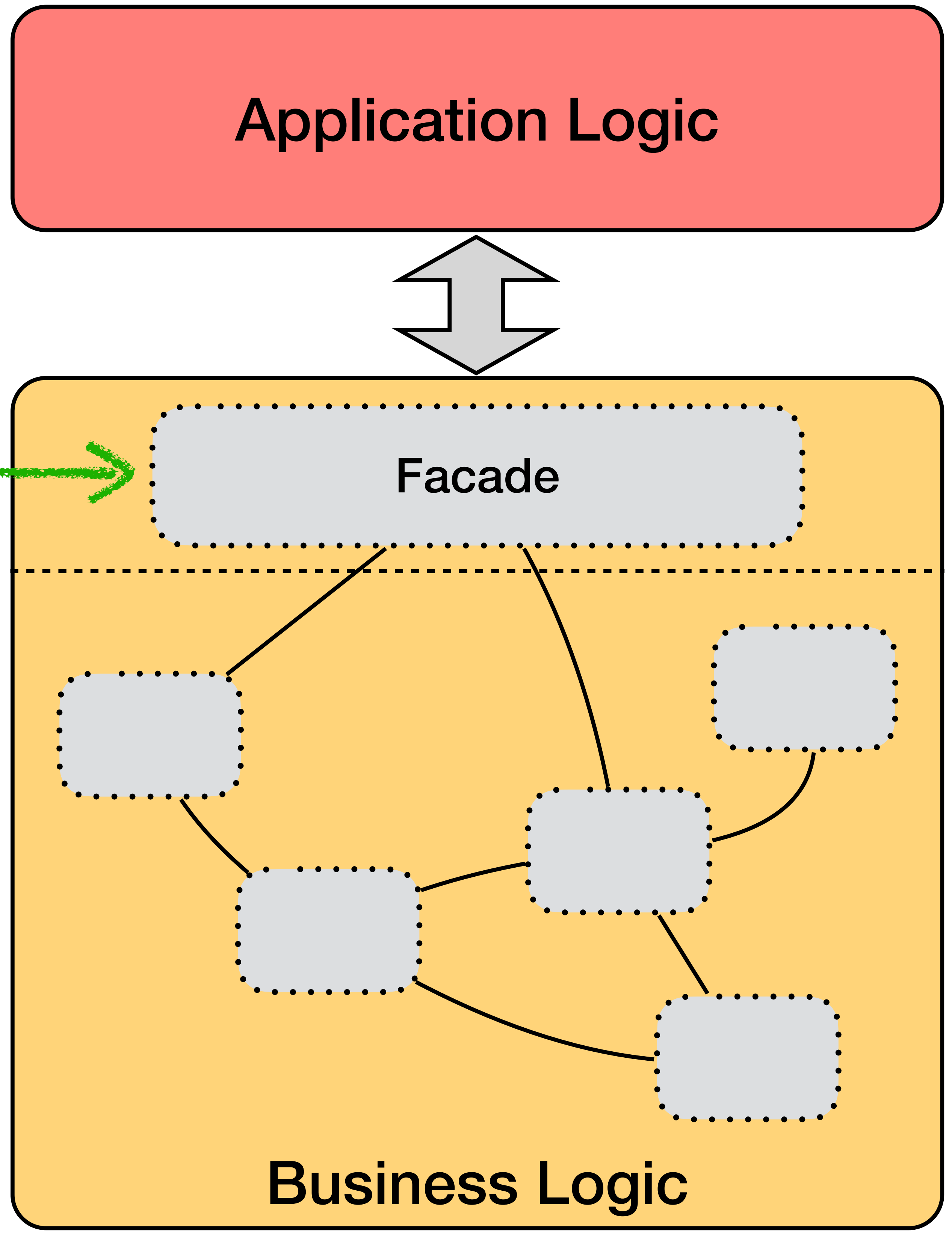
declarative role restriction,
check done by EJB framework



```
@NeedsCaller
@RolesAllowed(RoleType.VALIDUSER_ROLE)
public void publishEntry(String entryUuid) {
    momaLogicImpl.publishEntry(getCaller(), entryUuid);
}
```

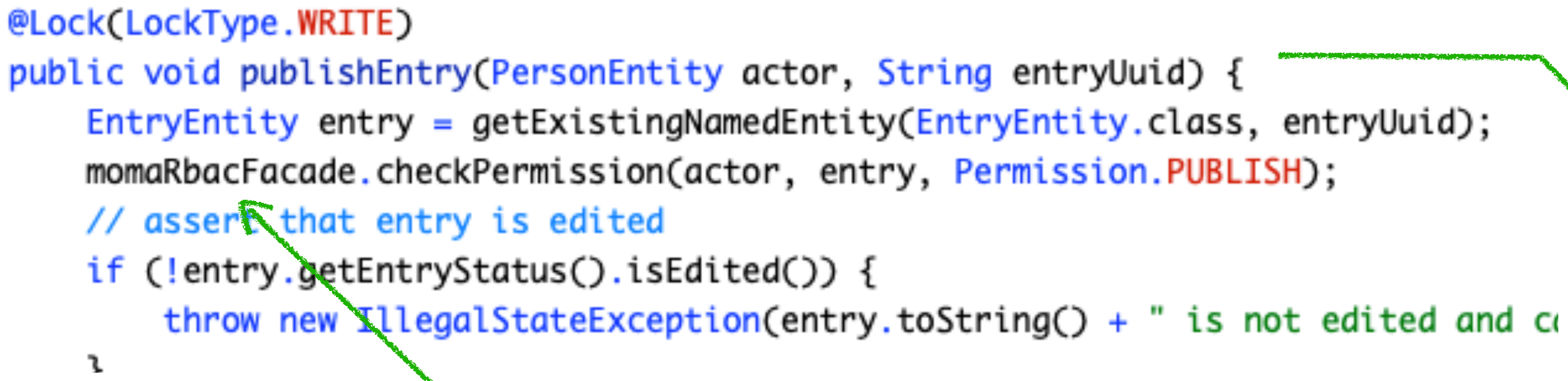
delegate to business
function implementation

determine
subject and
object

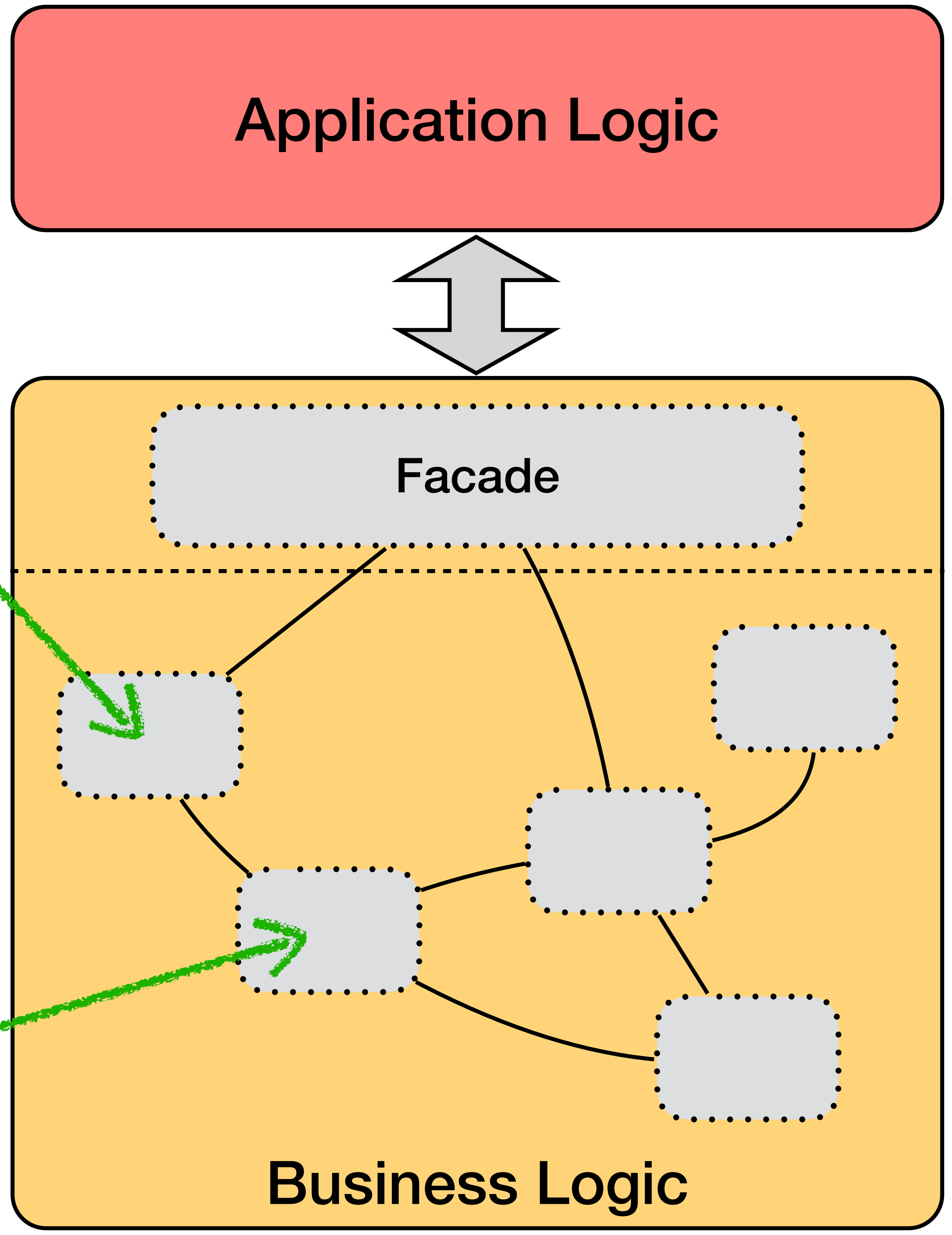


RBAC example (from Module Manual System)

Business Function Implementation



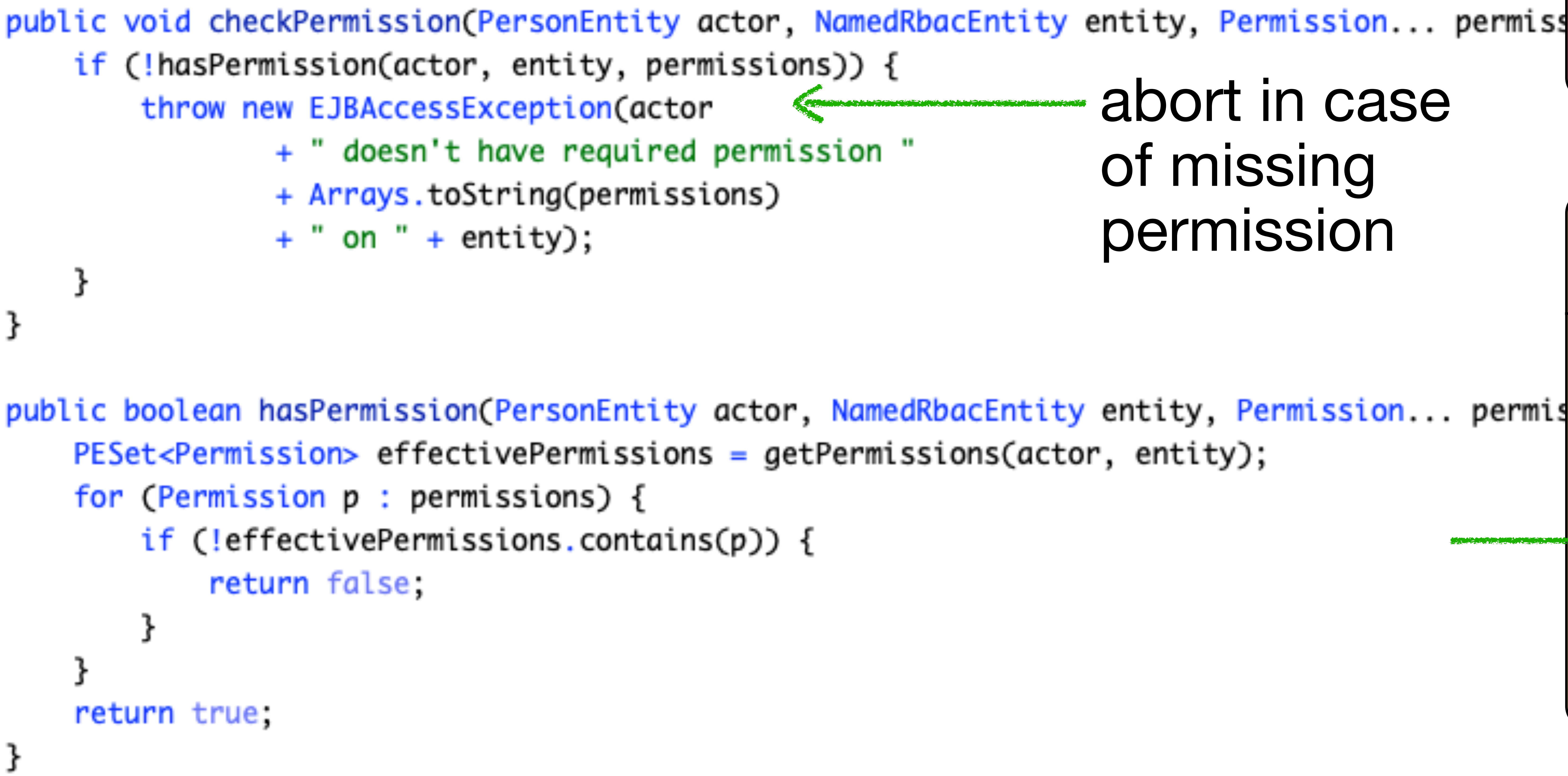
```
@Lock(LockType.WRITE)
public void publishEntry(PersonEntity actor, String entryUuid) {
    EntryEntity entry = getExistingNamedEntity(EntryEntity.class, entryUuid);
    momaRbacFacade.checkPermission(actor, entry, Permission.PUBLISH);
    // assert that entry is edited
    if (!entry.getEntryStatus().isEdited()) {
        throw new IllegalStateException(entry.toString() + " is not edited and c
    }
```



permission check relies on
RBAC subsystem

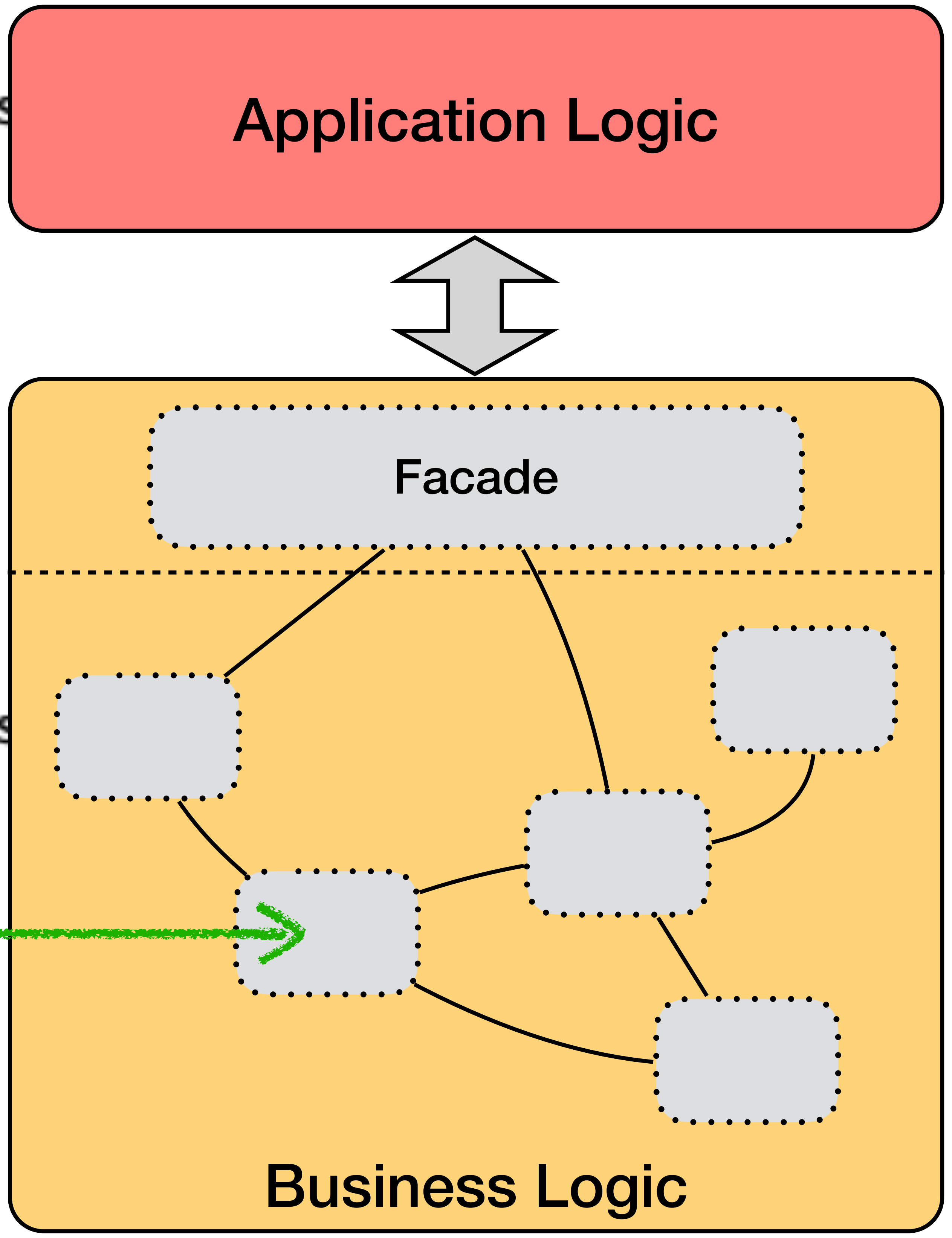
RBAC example (from Module Manual System)

Access Decision Implementation (excerpt)



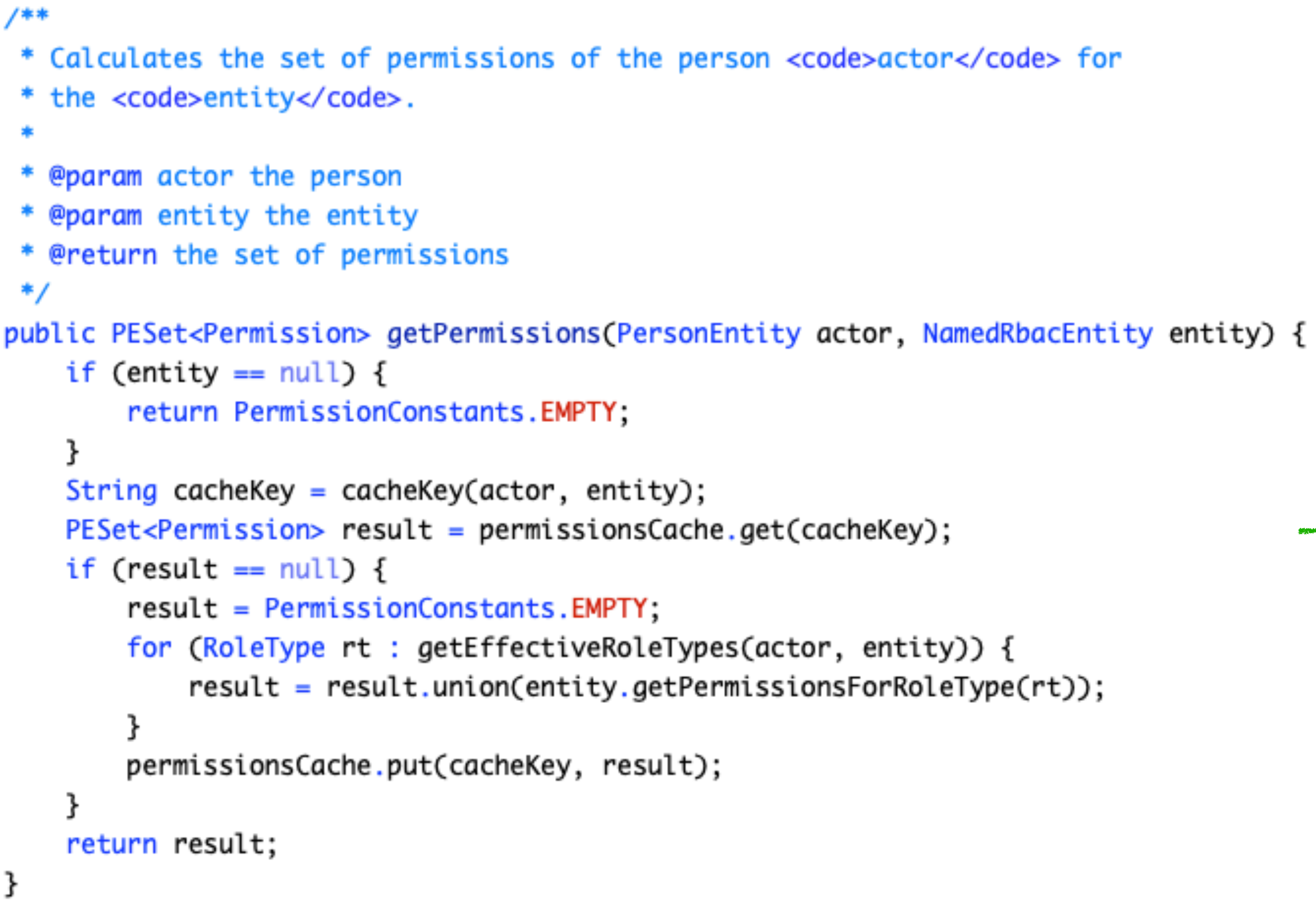
```
public void checkPermission(PersonEntity actor, NamedRbacEntity entity, Permission... permiss
    if (!hasPermission(actor, entity, permissions)) {
        throw new EJBAccessException(actor
                + " doesn't have required permission "
                + Arrays.toString(permissions)
                + " on " + entity);
    }
}

public boolean hasPermission(PersonEntity actor, NamedRbacEntity entity, Permission... permis
    PESet<Permission> effectivePermissions = getPermissions(actor, entity);
    for (Permission p : permissions) {
        if (!effectivePermissions.contains(p)) {
            return false;
        }
    }
    return true;
}
```

abort in case of missing permission

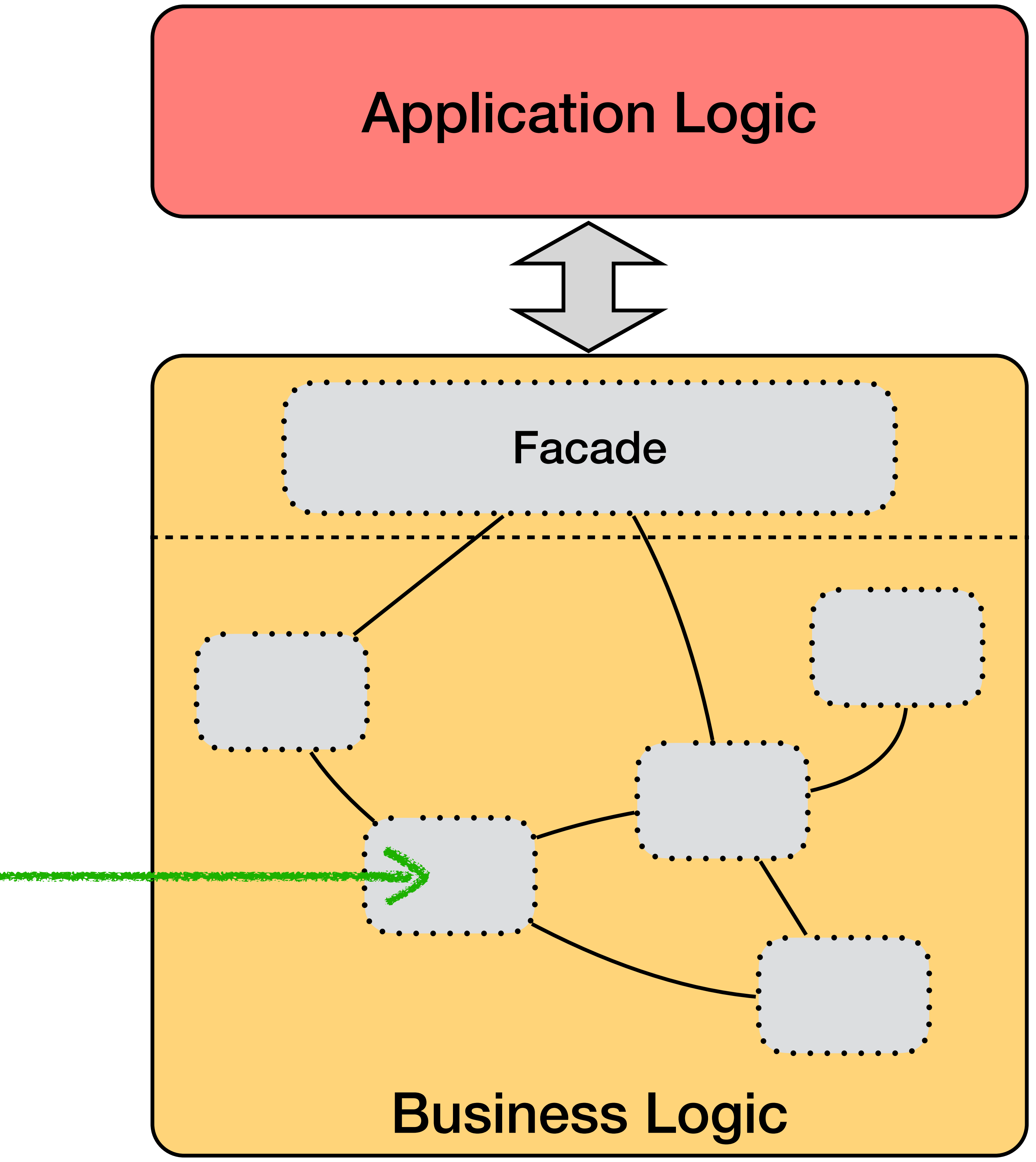


RBAC example (from Module Manual System)

Access Decision Implementation (excerpt)

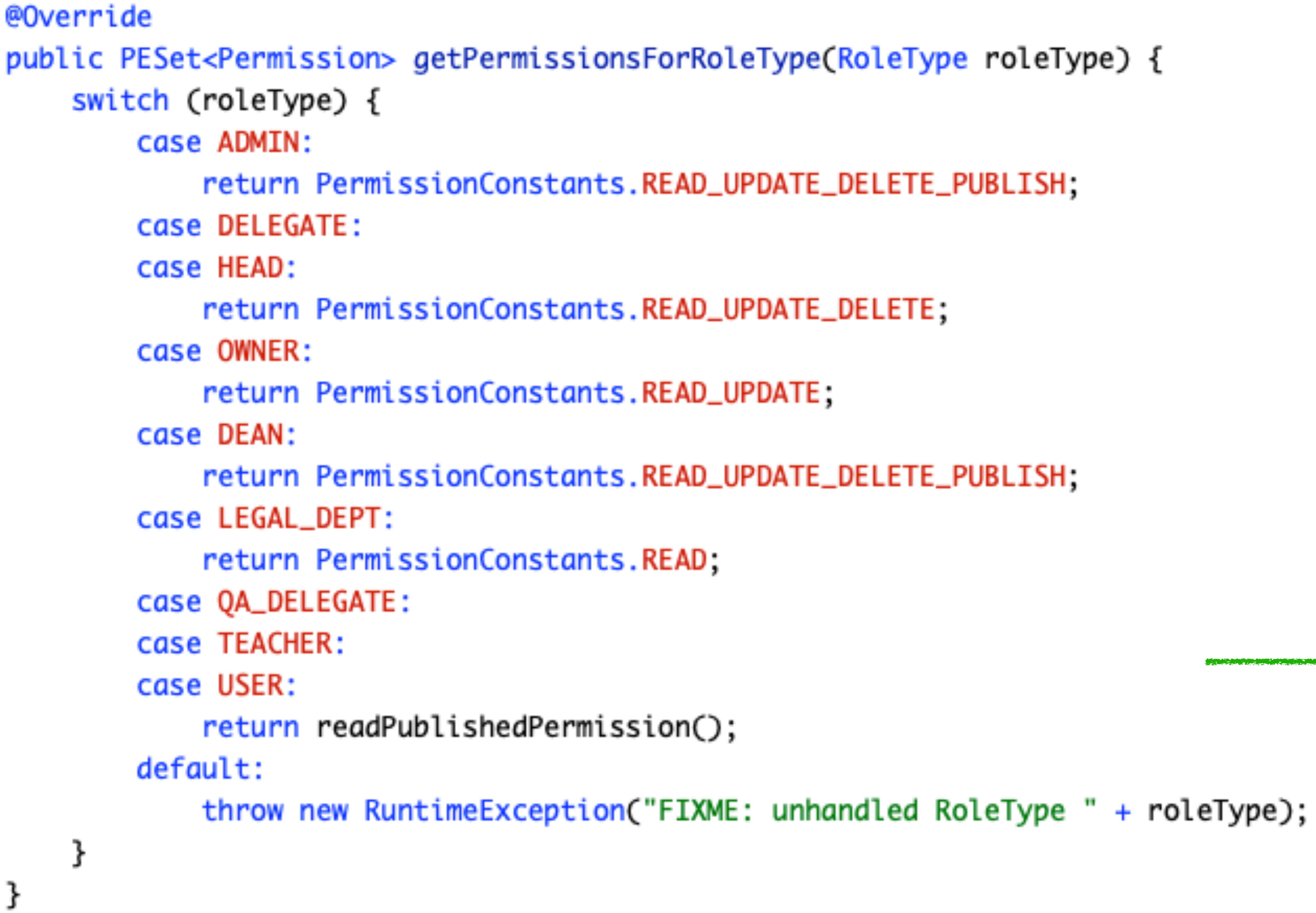


```java
/**
 * Calculates the set of permissions of the person <code>actor</code> for
 * the <code>entity</code>.
 *
 * @param actor the person
 * @param entity the entity
 * @return the set of permissions
 */
public PESet<Permission> getPermissions(PersonEntity actor, NamedRbacEntity entity) {
    if (entity == null) {
        return PermissionConstants.EMPTY;
    }
    String cacheKey = cacheKey(actor, entity);
    PESet<Permission> result = permissionsCache.get(cacheKey);
    if (result == null) {
        result = PermissionConstants.EMPTY;
        for (RoleType rt : getEffectiveRoleTypes(actor, entity)) {
            result = result.union(entity.getPermissionsForRoleType(rt));
        }
        permissionsCache.put(cacheKey, result);
    }
    return result;
}
```

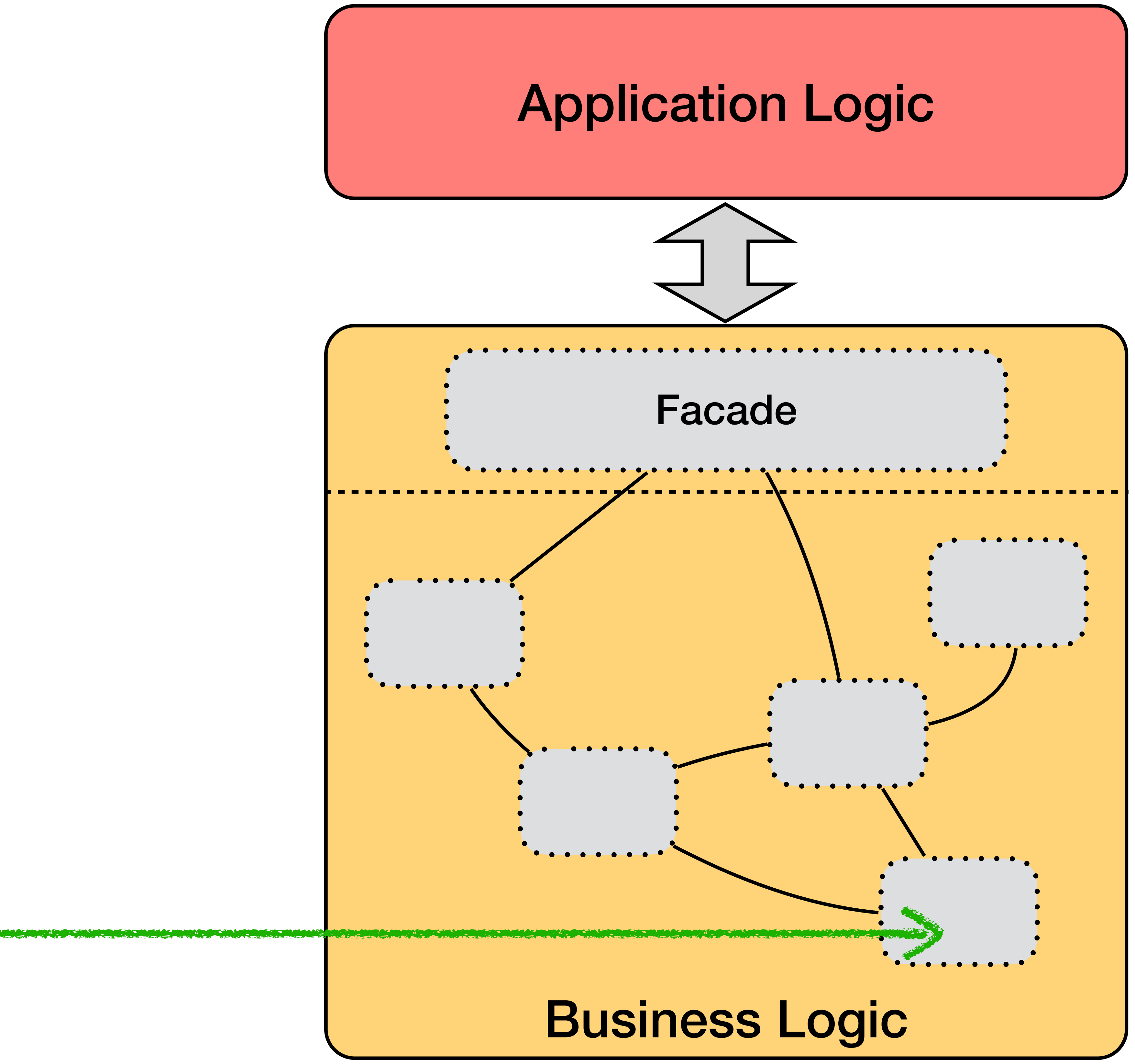


RBAC example (from Module Manual System)

Business Object Type Permission Definition



```
@Override
public PESet<Permission> getPermissionsForRoleType(RoleType roleType) {
    switch (roleType) {
        case ADMIN:
            return PermissionConstants.READ_UPDATE_DELETE_PUBLISH;
        case DELEGATE:
        case HEAD:
            return PermissionConstants.READ_UPDATE_DELETE;
        case OWNER:
            return PermissionConstants.READ_UPDATE;
        case DEAN:
            return PermissionConstants.READ_UPDATE_DELETE_PUBLISH;
        case LEGAL_DEPT:
            return PermissionConstants.READ;
        case QA_DELEGATE:
        case TEACHER:
        case USER:
            return readPublishedPermission();
        default:
            throw new RuntimeException("FIXME: unhandled RoleType " + roleType);
    }
}
```



ABAC - Attribute Based Access Control

- More comprehensive, flexible access control scheme
- Based on dynamic rules and policies instead of fixed predefined roles and permissions
- Various attributes (subject, object, context, dynamic values) can contribute to decisions
- Policy definition often based on rules represented in formal logic

- Permission checks can introduce substantial *overhead*, depending on
  - granularity and frequency of checks
  - complexity of policies
  - number of policies
  - size of authorization requests
- ABAC is applicable to arbitrary components of a system

ABAC Components

- **PAP** - Policy Administration Point
    - define rules and policies
- **PEP** - Policy Enforcement Point
    - protects objects, e.g., at calls to business functions or when accessing database entries
    - receives request and issues an authorization request to the policy decision point
    - based on outcome, permits or denies operation

- **PDP** - Policy Decision Point
    - computes decision for authorization request based on rule and policy checks
- **PIP** - Policy Information Point
    - enables access of PDP to external information about subjects, objects, and context, e.g. directories, sensor data, …

RBAC vs. ABAC

- ABAC is more flexible than RBAC
- Dynamic rules instead of hard-coded authorization logic
- Authorization has to be implemented at the PEP (calls to PDP and suitable reaction)
- Higher complexity of decisions requires careful performance evaluation
- Administration costs have to be considered
- Frameworks for creating and checking policies should be used
- System gets more complex due to additional critical components and dependencies

Consider security and privacy right from the start of your project, during planning, design, implementation, deployment, operation, maintenance, until end-of-life!

Retrofitting/fixing is often more expensive…

Use “Security/Privacy By Design” approaches!

(Repeatedly) verify at all phases that the security/privacy measures are applied thoroughly and are really working!

Constantly observe/monitor/patch your system!

What we have learned…

Security

- ✓ Web Application Security
- ✓ Access Control

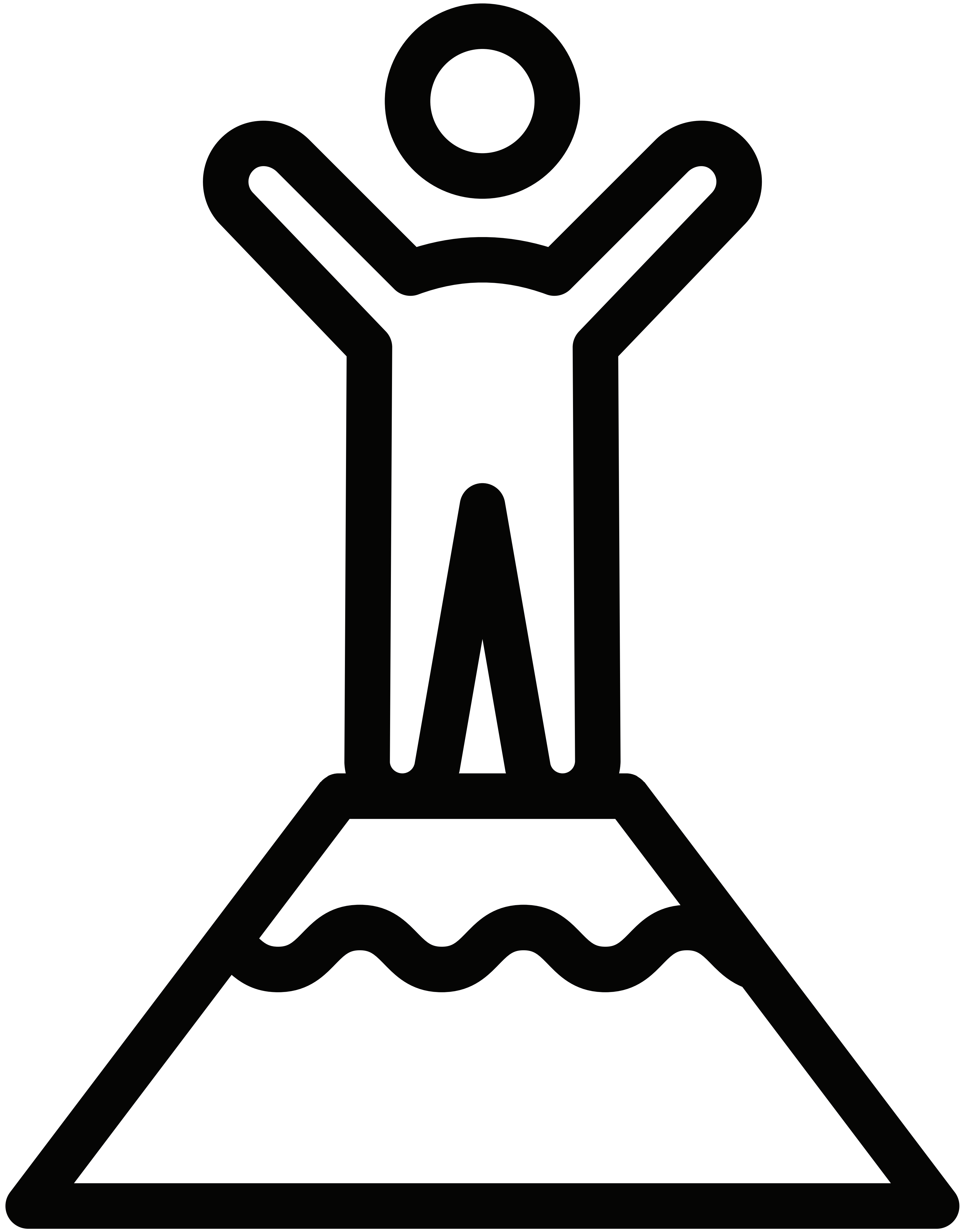
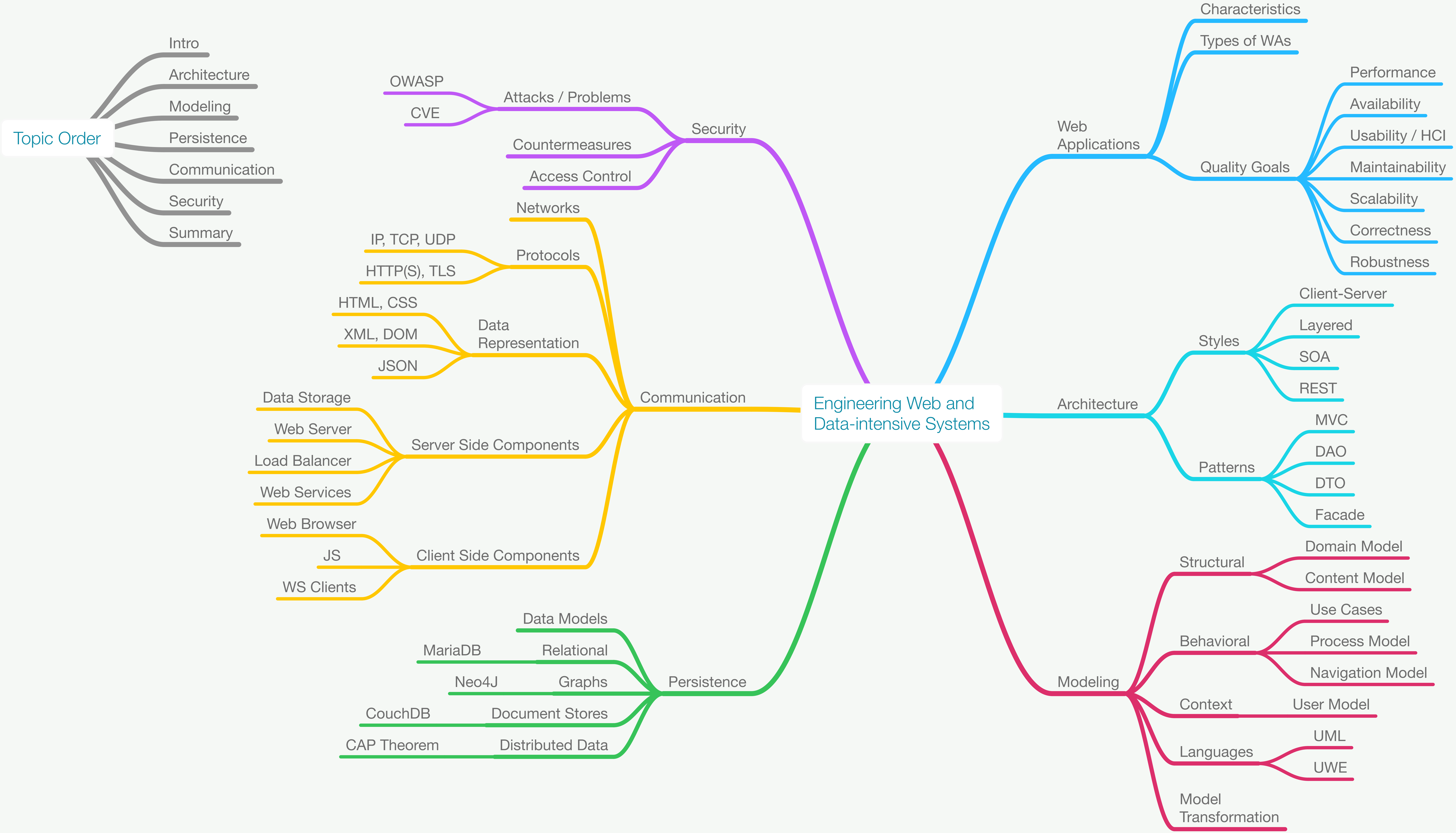


Image: colourbox.de



- Topic Order
  - Intro
  - Architecture
  - Modeling
  - Persistence
  - Communication
  - Security
  - Summary

- Engineering Web and Data-intensive Systems
  - Security
    - Attacks / Problems
      - OWASP
      - CVE
    - Countermeasures
    - Access Control
  - Communication
    - Networks
    - Protocols
      - IP, TCP, UDP
      - HTTP(S), TLS
    - Data Representation
      - HTML, CSS
      - XML, DOM
      - JSON
    - Server Side Components
      - Data Storage
      - Web Server
      - Load Balancer
      - Web Services
    - Client Side Components
      - Web Browser
      - JS
      - WS Clients
  - Persistence
    - Data Models
    - Relational
      - MariaDB
    - Graphs
      - Neo4J
    - Document Stores
      - CouchDB
    - Distributed Data
      - CAP Theorem
  - Web Applications
    - Characteristics
    - Types of WAs
    - Quality Goals
      - Performance
      - Availability
      - Usability / HCI
      - Maintainability
      - Scalability
      - Correctness
      - Robustness
  - Architecture
    - Styles
      - Client-Server
      - Layered
      - SOA
      - REST
    - Patterns
      - MVC
      - DAO
      - DTO
      - Facade
  - Modeling
    - Structural
      - Domain Model
      - Content Model
    - Behavioral
      - Use Cases
      - Process Model
      - Navigation Model
    - Context
      - User Model
    - Languages
      - UML
      - UWE
    - Model Transformation