Building secure software systems using security patterns

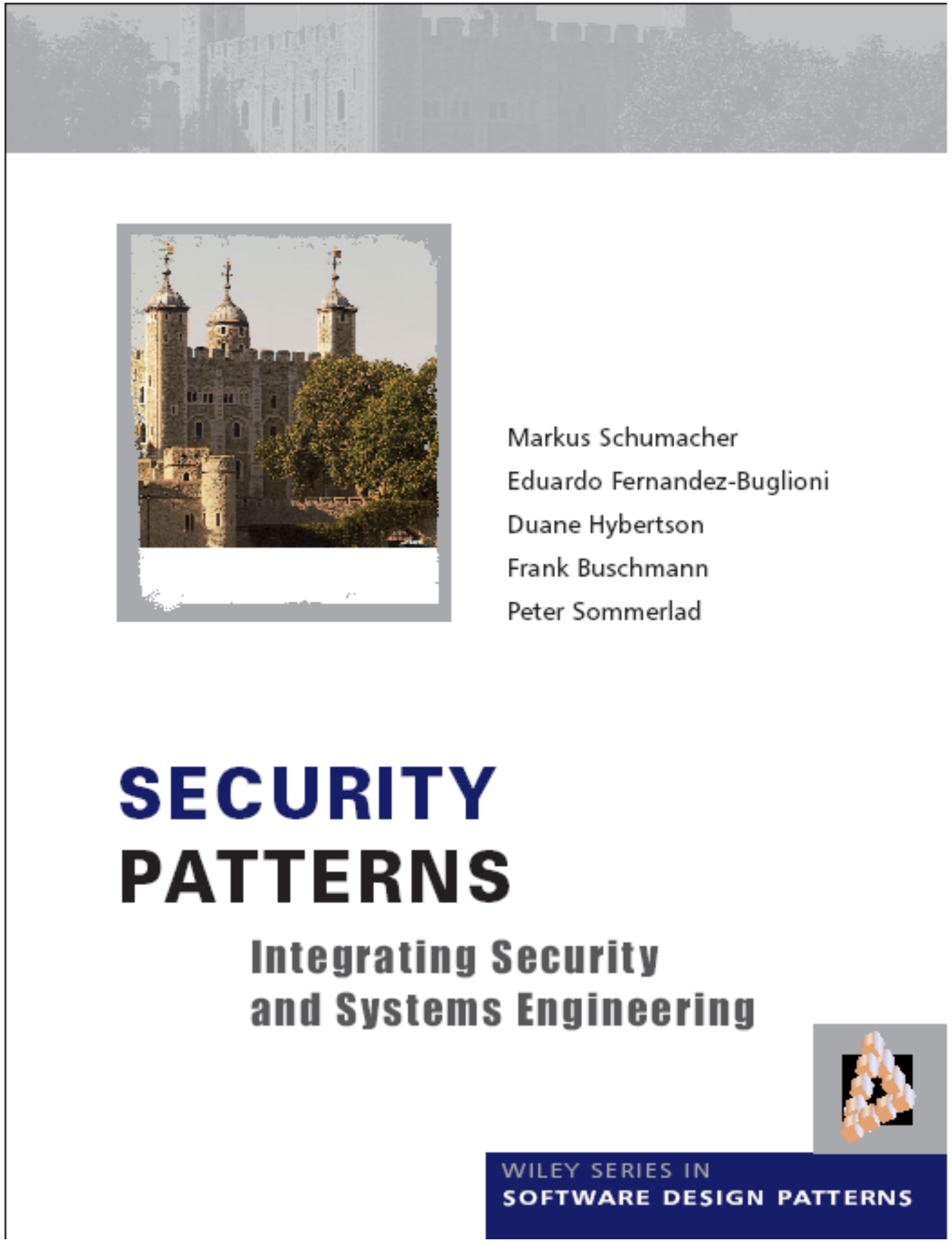
Eduardo B. Fernandez
Florida Atlantic University
Boca Raton, FL, USA
fernande@fau.edu

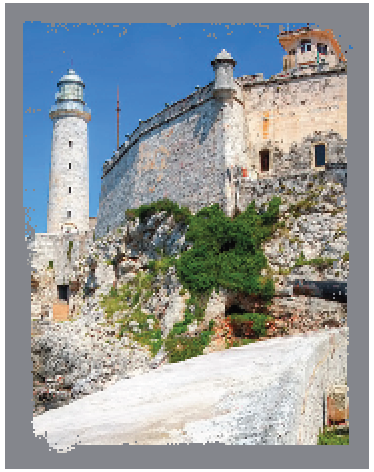
Outline

- ***Introduction- motivation***
- ***Security patterns and abstraction***
- ***Catalogs of security patterns***
- ***A methodology to build secure systems***
- ***Reference architectures: a Cloud architecture***
- ***Security verification***
- ***Conclusions***

About me

- Eduardo B. Fernandez (Eduardo Fernandez Buglioni)
- Professor of Computer Science at Florida Atlantic University, Boca Raton, FL., USA
- BSEE UTFSM, Chile, MS EE Purdue U., PhD CS UCLA
- Worked at IBM for 8 years (L.A. Scientific Center).
- Wrote the first book on database security (Addison-Wesley, 1981) and two books on security patterns (2006 and 2013)
- Author of many research papers (Google Scholar h-index 40, i10index 159)
- Consultant to IBM, Siemens, Lucent, Motorola, Huawei,…


Markus Schumacher
Eduardo Fernandez-Buglioni
Duane Hybertson
Frank Buschmann
Peter Sommerlad
SECURITY
PATTERNS
Integrating Security
and Systems Engineering
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS



Eduardo Fernandez-Buglioni

SECURITY PATTERNS IN PRACTICE

Designing Secure Architectures Using Software Patterns

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

The value of information

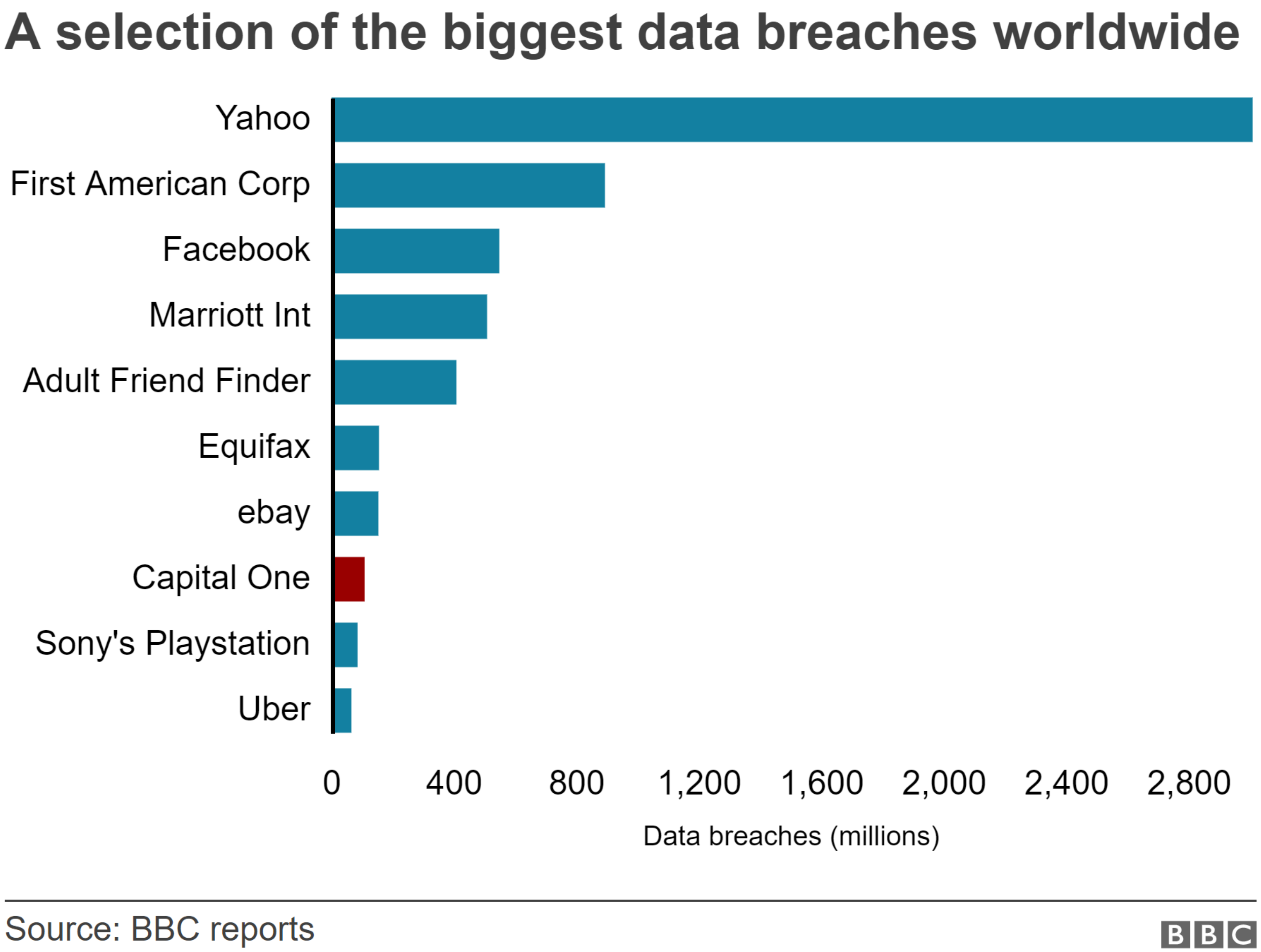
- We depend heavily on computers; most of the devices that make our life safe and convenient, e.g. cell phones, vehicle controls, and building controls, use some type of software to store and process information
- We also rely on institutions, public or private; we are born in hospitals, then go to schools, join clubs, get jobs in the government or private businesses, get married at some church or public office, travel using some agency, etc. All these institutions use computers to keep information about us.
- In general, the data of an institution has a great value; it may represent customers, orders, bills, business plans, course grades, etc. It might even be its product.
- Data corruption in a hospital may result in patients getting the wrong medication, leakage of military information could endanger an army in war, and erroneous aircraft maintenance information could compromise passenger safety, unauthorized access to a bank information may result in large money losses.

Motivation for security

- Data and other resources are *assets,* items that have value for individuals and institutions*;* **security** is the protection of these assets, including enterprise and individual information.
- We need this protection because there are people who intentionally try to read/copy or modify information either for their own gain, for political purposes, or for the sake of disruption
- In addition to the direct monetary cost there may be losses of productivity, and even endangering of lives

Do we have a problem?

- Almost every month we have a major security incident.
- Companies: Target, Sony (twice), Home Depot, Goodwill, JP Morgan, Chick-fil-A, Neiman Marcus, Michaels, Yahoo (twice), Equifax,Uber, Marriott, British Airways, Facebook, Under Armour,…
- Government: IRS, DOE, OPM,…
- Cyber-Physical systems: German steel mill, Aramco, Ukraine Grid…
- Medical systems and devices: Anthem
- Point of sale systems: Target, Michaels, Home Depot, …


A selection of the biggest data breaches worldwide
Yahoo
First American Corp
Facebook
Marriott Int
Adult Friend Finder
Equifax
ebay
Capital One
Sony's Playstation
Uber
0
400
800
1,200
1,600
2,000
2,400
2,800
Data breaches (millions)
Source: BBC reports

Software complexity is constantly increasing

Embedded systems

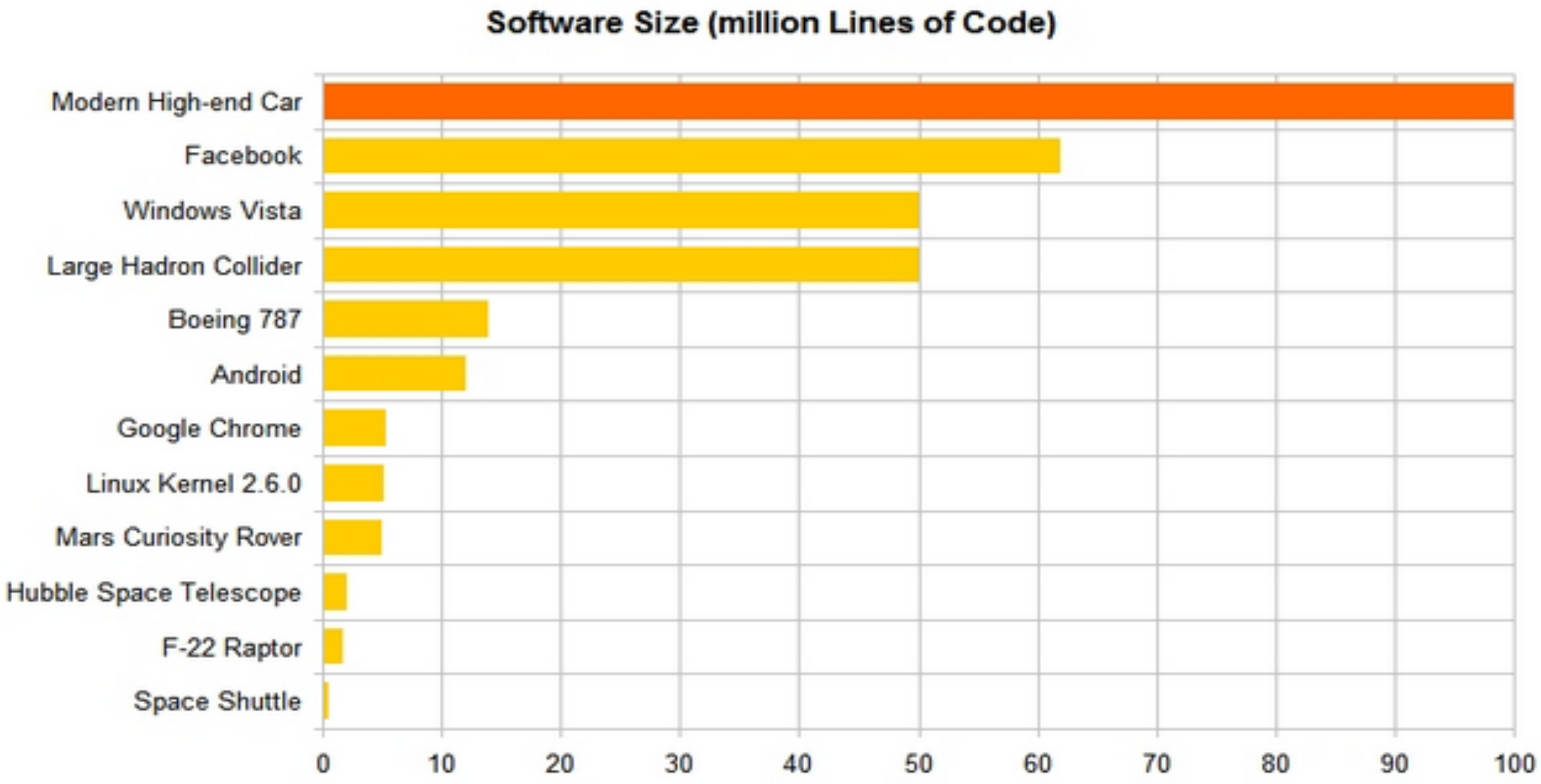
- The average device now has one million lines of code, and that number is doubling every two years.

Vehicles

- A modern passenger jet, such as a Boeing 777, uses about 4 million lines of code. Older planes such as a Boeing 747 had only 400,000 lines of code.
- A car uses 30-50 electronic control units (ECUs) that altogether include as much as 100 M lines of code.

Code size

https://www.linkedin.com/pulse/20140626152045-3625632-car-software-100m-lines-of-code-and-counting



Reasons for system vulnerabilities

- Other than complexity, another important reason for systems weakness is that **security is built as an add-on, in piecemeal fashion**, parts of the system are secured using specific mechanisms but there is rarely a global security analysis of the complete system
- If done, different models may be used in different parts, e.g., one for the databases and another for wireless devices
- However, **security requires a holistic approach** to block all possible ways of attack or at least control their effects
- **Security is not composable**: combining secure units does not produce a secure system, securing separate code components is not enough

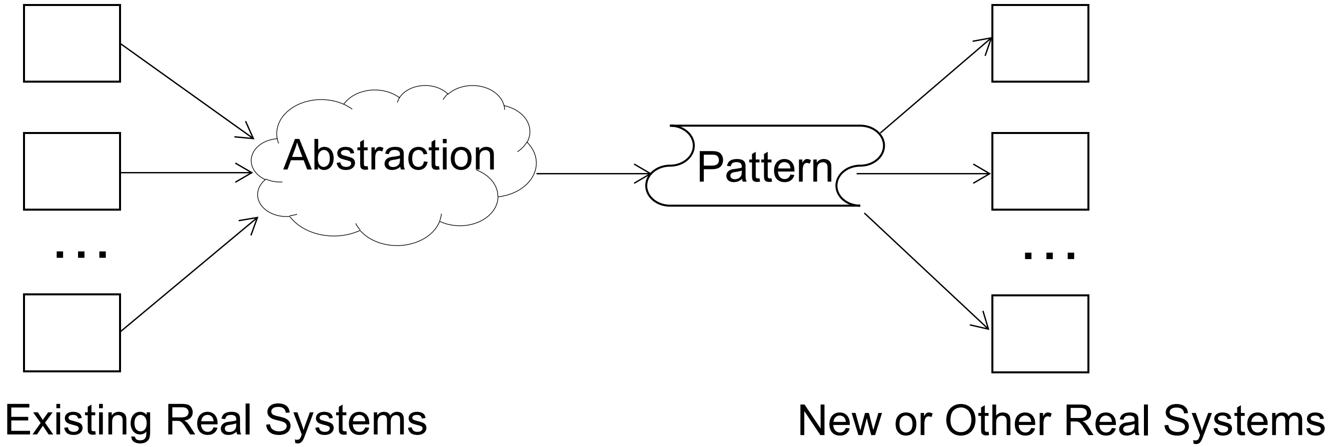
Use of abstraction

- The only way to provide a unification in the presence of myriad implementation details of the component units is to use abstraction. In particular, **we can apply abstraction through the use of *patterns*.**
- **The description of architectures and mechanisms using patterns makes them easier to understand, provides guidelines for design and analysis, and can define a way to make their structure more secure.**
- **Their abstraction properties make them ideal for dealing with highly complex systems and for holistic views.**

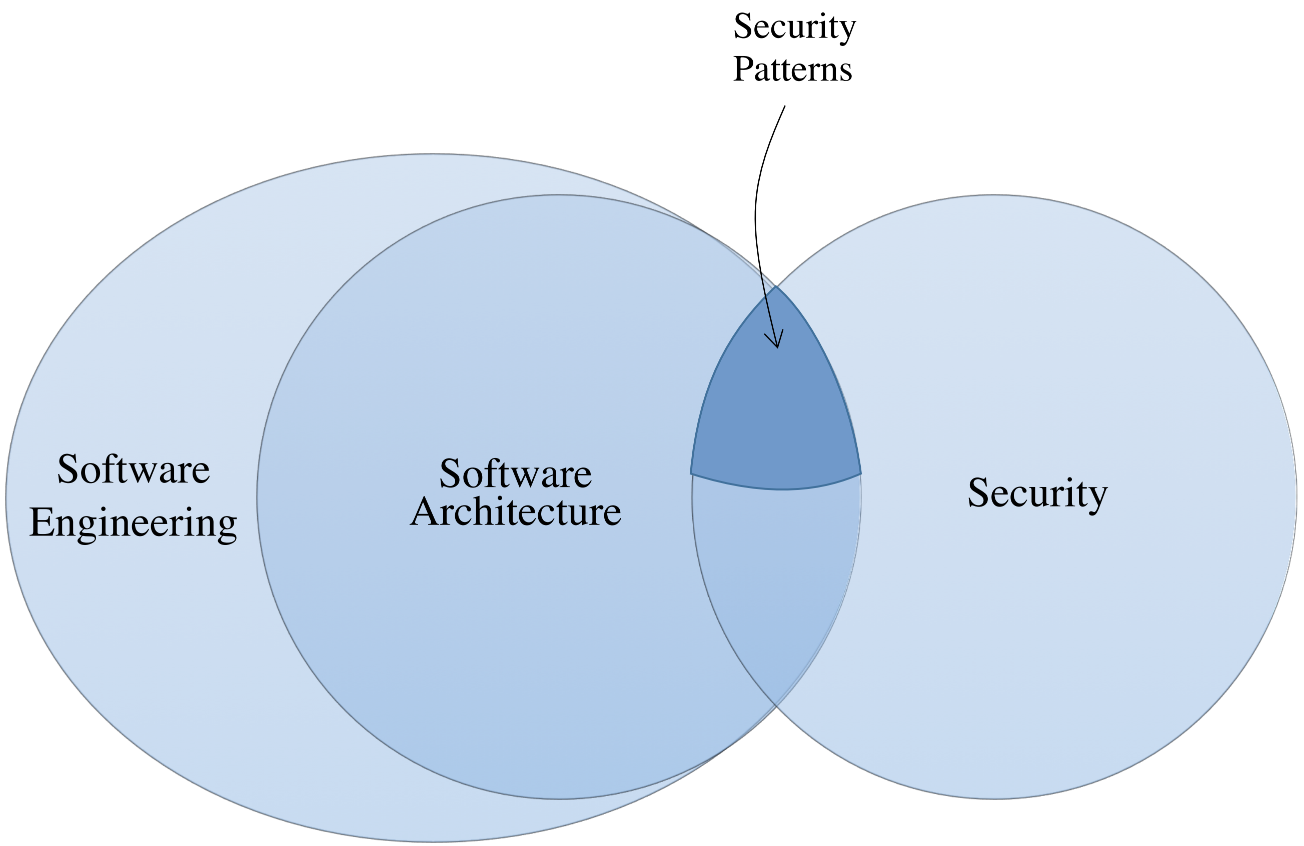
Patterns

- *A pattern is a solution to a recurring problem in a specific context*
- *The idea comes from the architecture of buildings (Christopher Alexander)*
- *It was applied initially to software but it has been extended to other aspects.*
- *It appeared in 1994 and it is slowly being accepted by industry.*
- *A security pattern solves a security problem, usually how to control a threat*

Derivation and validation of patterns



Where patterns fit in software design



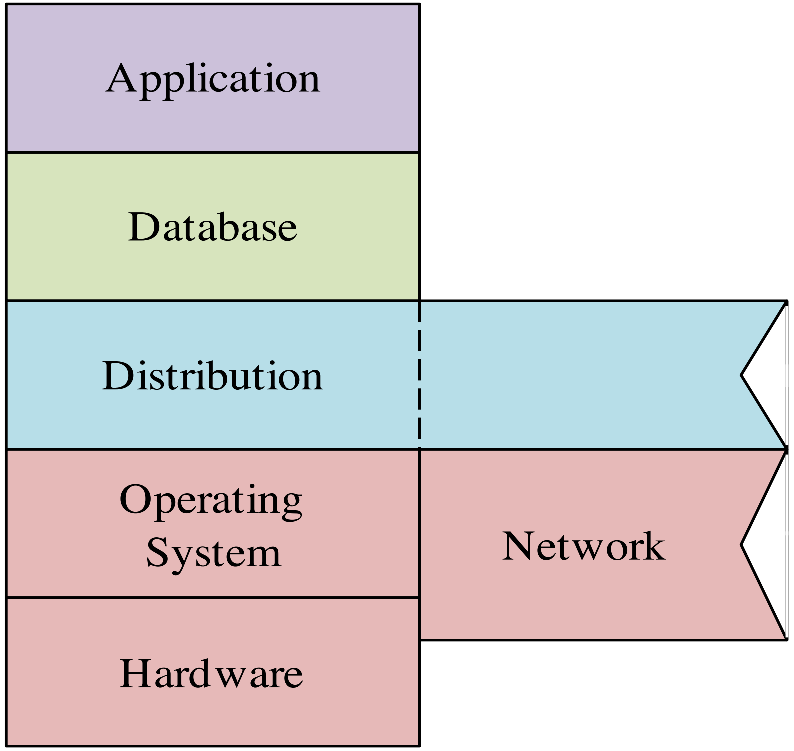
Value of security patterns

- Can **apply security principles** (Least privilege) or describe security mechanisms able to stop specific threats(Firewalls) in all architectural layers
- Can **guide the design and implementation** of the security mechanism itself
- Can **guide the use of security mechanisms** in an application (stop specific threats)
- Can **help understanding and use of complex standards (**XACML, WiMax)
- **Convenient for teaching** security principles and mechanisms

Patterns can be defined at all architectural levels

- At the conceptual model we can define abstract security patterns
- These patterns can be mapped to the lower architectural layers
- The lower-level patterns add aspects specific to their layer, e.g., a database pattern will use database concepts such as views and database items (columns, tuples,…)
- By doing this, **we can obtain a holistic view** of the system security
- An **Abstract Security Pattern** (ASP), describes a conceptual security mechanism that realizes one or more security policies able to control (stop or mitigate) a threat or comply with a security-related regulation or institutional policy (no implementation aspects).

Architectural layers



Conceptual security

- Security is a quality aspect that constrains the semantic behavior of applications (by imposing access restrictions), so the requirements stage is the right development stage to start addressing security
- However, we only want to indicate at this stage which specific security controls are needed, not their convenient or optimal implementation.
- For example, in bank applications we only want to specify the semantic aspects of accounts, customers, and transactions with their corresponding restrictions.

Security and application semantics

- In the bank case, we need to specify that customers are the only ones who can perform transactions on their own accounts and similar type of constraints.
- The constraints come from the semantics of the application and from the necessity to defend against expected threats.
- At this stage, it appears useful to provide a set of patterns (or other artifacts) which define abstract security mechanisms that can describe these restrictions, these are ASPs

An ASP example: Authenticator

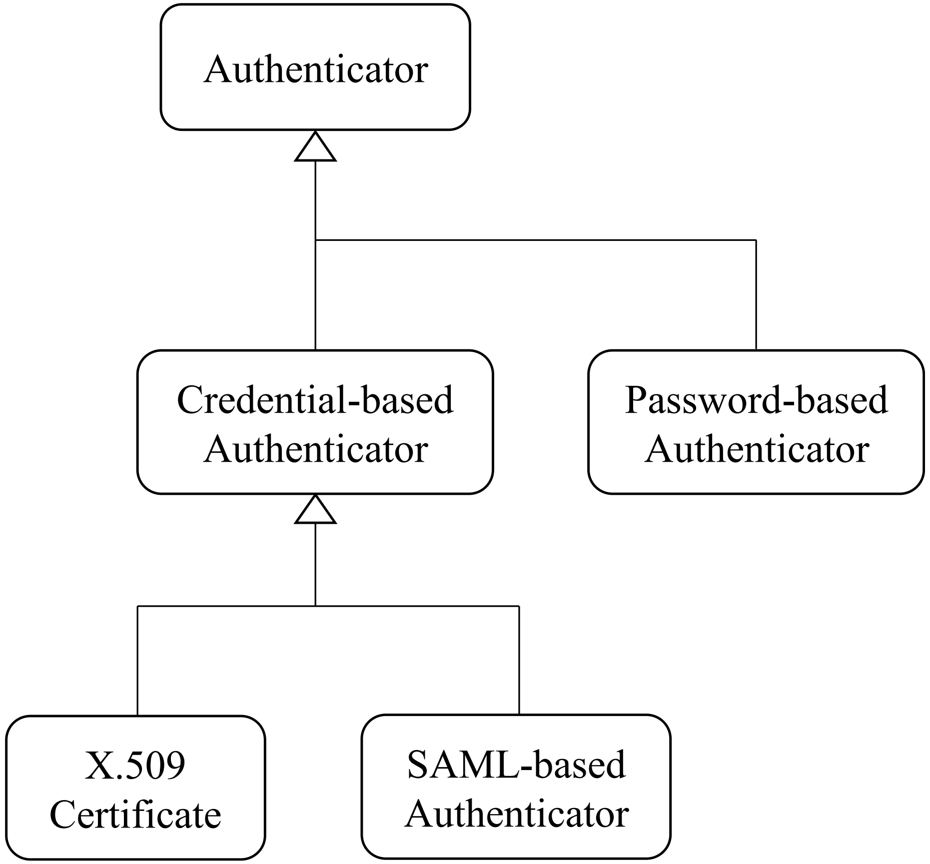
- This is the Intent section of an Authenticator pattern: "When a user or system (subject) identifies itself to the system, how do we verify that the subject intending to access the system is who it says it is? Present some information that is recognized by the system as identifying this subject. After being recognized, the requestor is given some proof that it has been authenticated."
- Authentication restricts access to a system to only registered users; it handles the threat where an intruder enters a system and may try to perform unauthorized access to information
- It is clear that there are many ways to perform this authentication, that go from manual ways, as done in voting places, to purely automatic ways, as when accessing a web site, but all of them must include the requirements of the abstract Authenticator

Abstract authentication

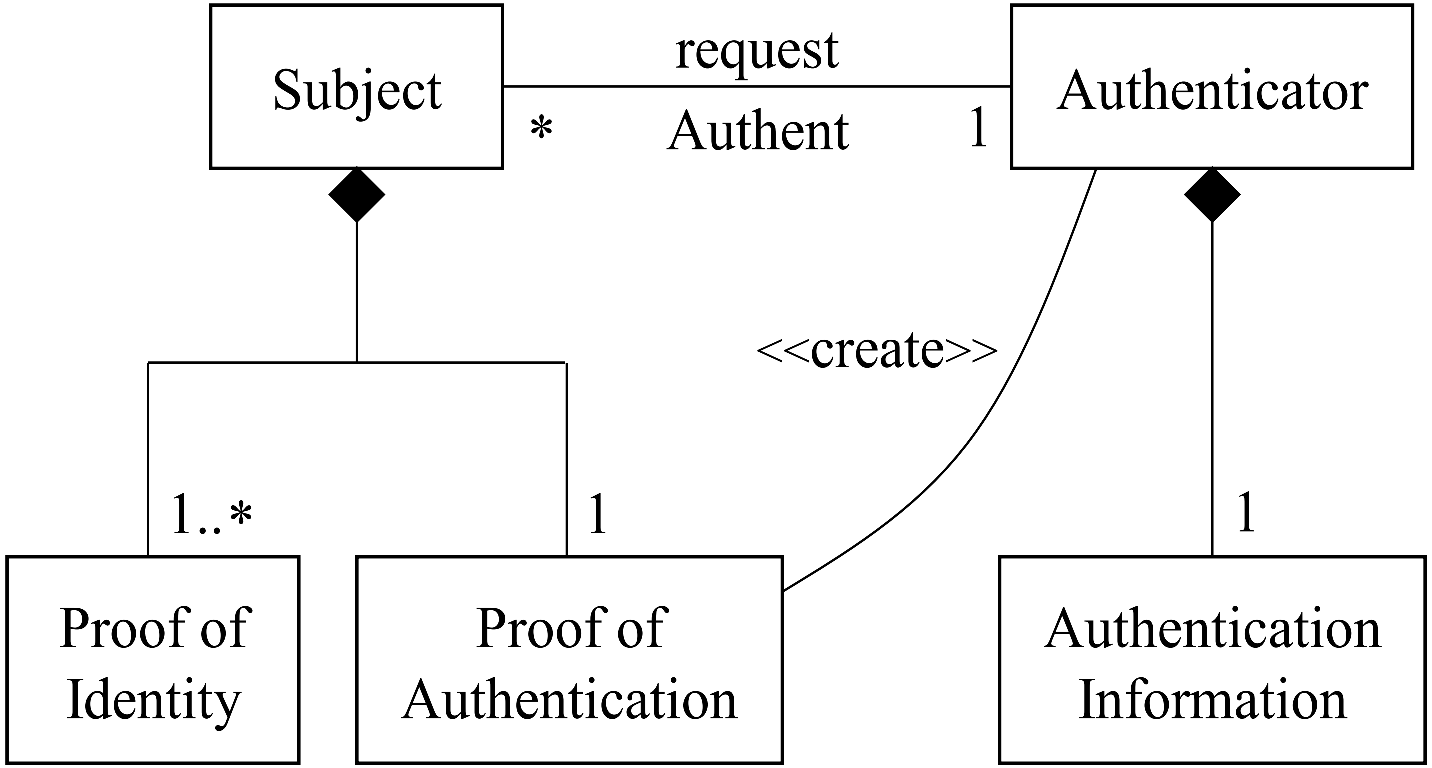
Authentication as an abstract function requires a basic sequence of activities. Concrete realizations of this sequence implement these steps in different ways but all must perform these two steps:

- The subject requests to enter a system indicating its identity and presenting some proof of identity.
- If the system recognizes the subject using its identity information, it grants her entrance to the system and provides her with a proof of authentication for further use. If not, the request is denied.
- We can define a hierarchy of authentication patterns starting from the abstract Authenticator

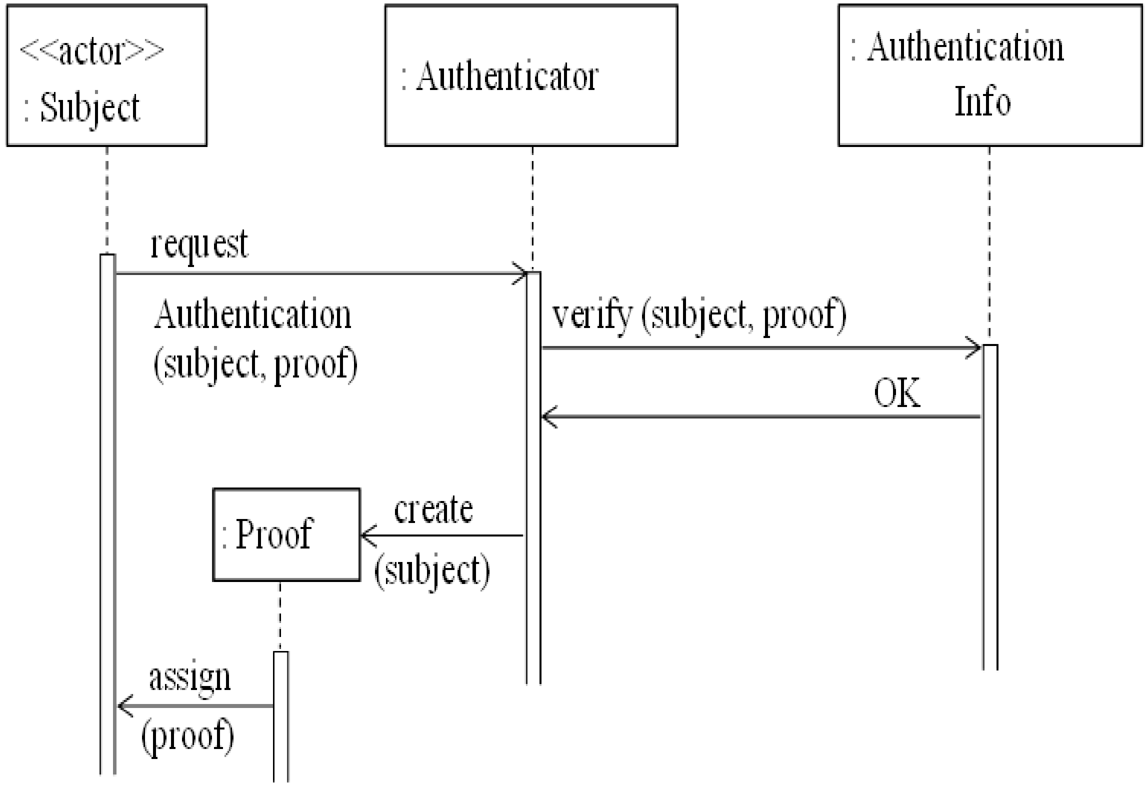
An authentication hierarchy



Class diagram of Abstract Authenticator



Sequence diagram for the use case “Authenticate a subject”



Forces of Abstract Authenticator

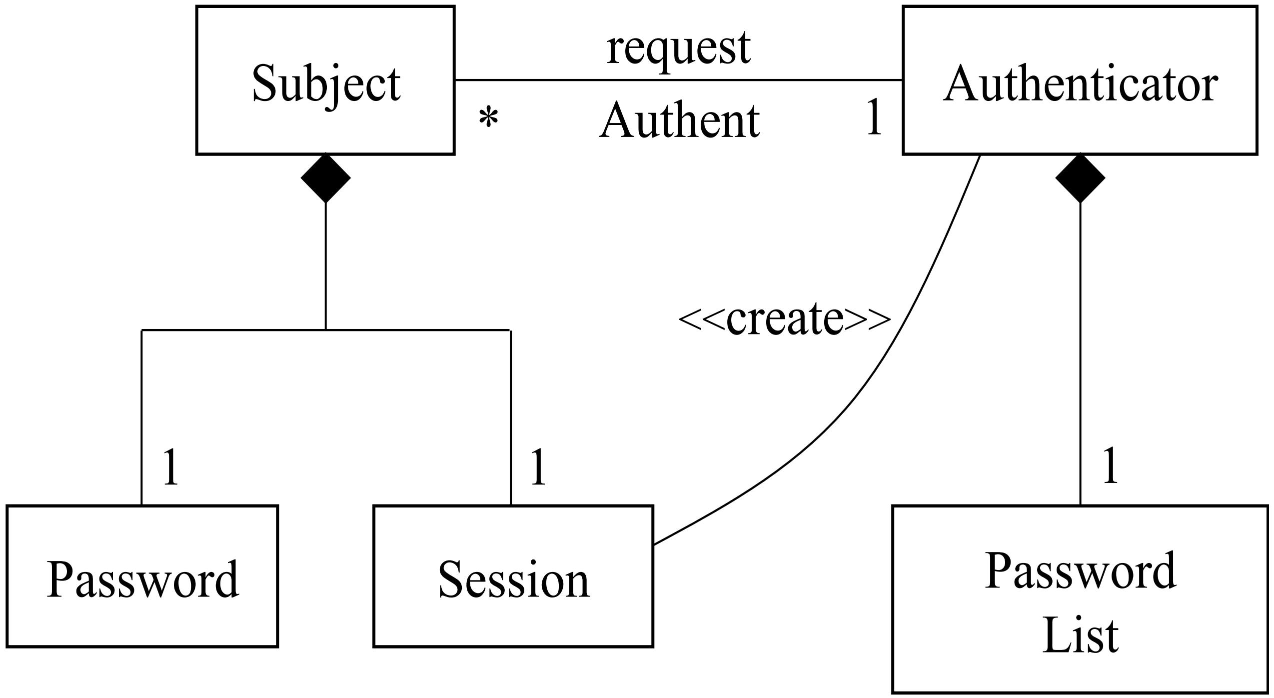
- ***Closed system***. If the authentication information presented by the user is not recognized, there is no access. In an open system all subjects would have access except some who are blacklisted for some reason.
- ***Registration.*** Users must register their identity information so that the system can recognize them later.
- **Flexibility**. There may be a variety of individuals (users) who require access to the system and a variety of system units with different access restrictions. We need to be able to handle all this variety appropriately or we risk security exposures.
- ***Dependability****.* We need to authenticate users in a reliable and secure way. This means a robust protocol and a high degree of availability. Otherwise, users may fool the authentication process or enter when the system authentication is down.
- ***Protection of authentication information***. Users should not be able to read or modify the authentication information. Otherwise, they can give themselves access to the system.

Forces II

- ***Simplicity***. The authentication process must be relatively simple or the users or administrators may be confused. User errors are annoying to them but administrator errors may lead to security exposures.
- ***Reach***. Successful authentication only gives access to the system, not to any specific resource in the system. Access to these resources must be controlled using other mechanisms, typically authorization.
- ***Tamper freedom****.* It should be very difficult to falsify the proof of identity presented by the user.
- ***Cost****.* There should be tradeoffs between security and cost, more security can be obtained at a higher cost.
- ***Performance****.* Authentication should not take a long time or users will be annoyed.
- ***Frequency****.* We should not make users authenticate frequently. Frequent authentications waste time and annoy the users.

All these properties must be present in the lower-level ways of performing authentication, e.g. in a Password Authenticator (see next slide). A Password Authenticator needs tomake concrete its Authentication Information (list of passwords) and its proof of authentication (a session)

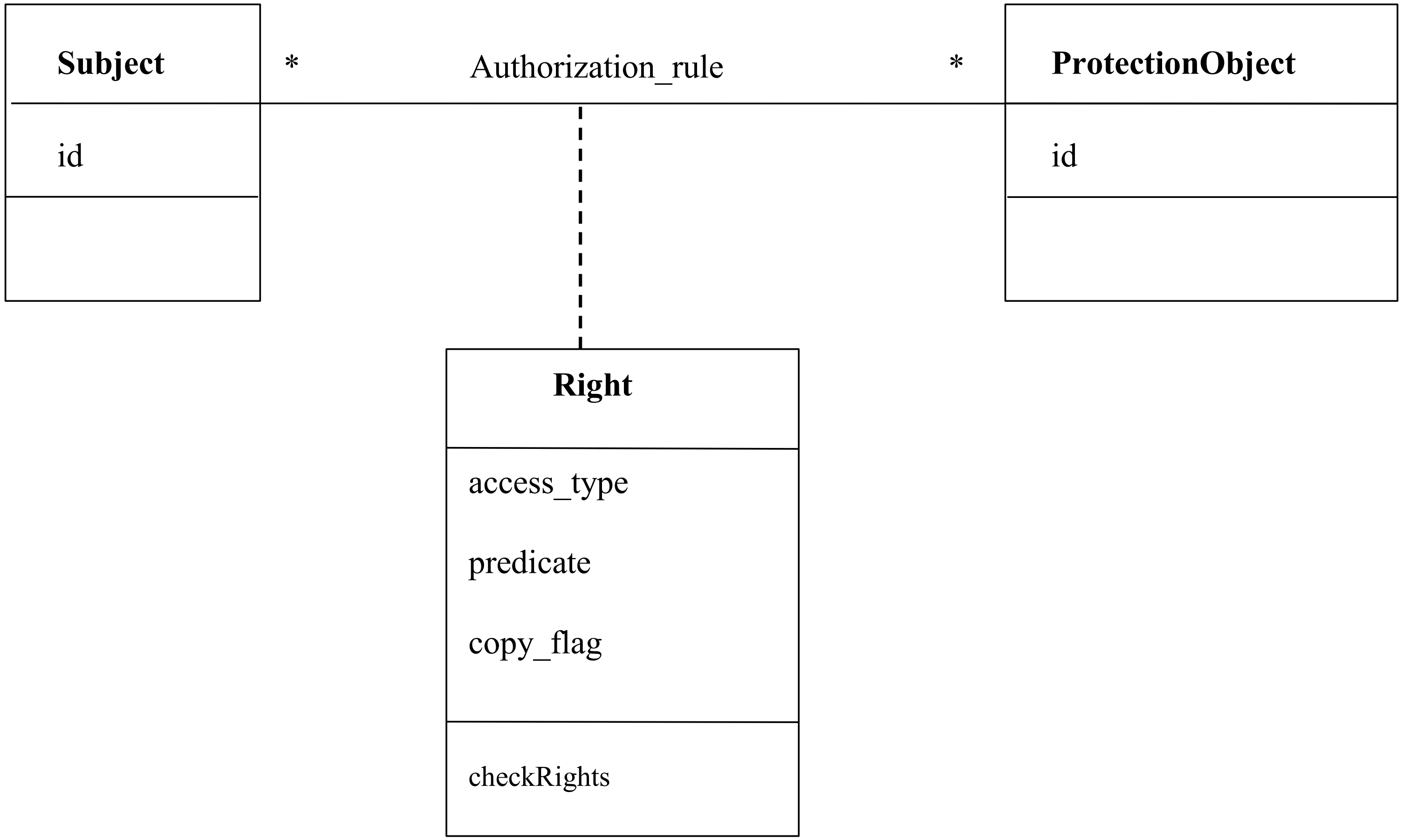
A concrete pattern: Password-based authenticator



Application Layer: Access control models

- **Authorization.** How do we describe who is authorized to access specific resources in a system? A list of authorization rules describes who has access to what and how.
- **Role-Based Access Control (RBAC).** How do we assign rights to people based on their functions or tasks? Assign people to roles and give rights to these roles so they can perform their tasks.
- **Multilevel Security**. Users and data are classified into levels

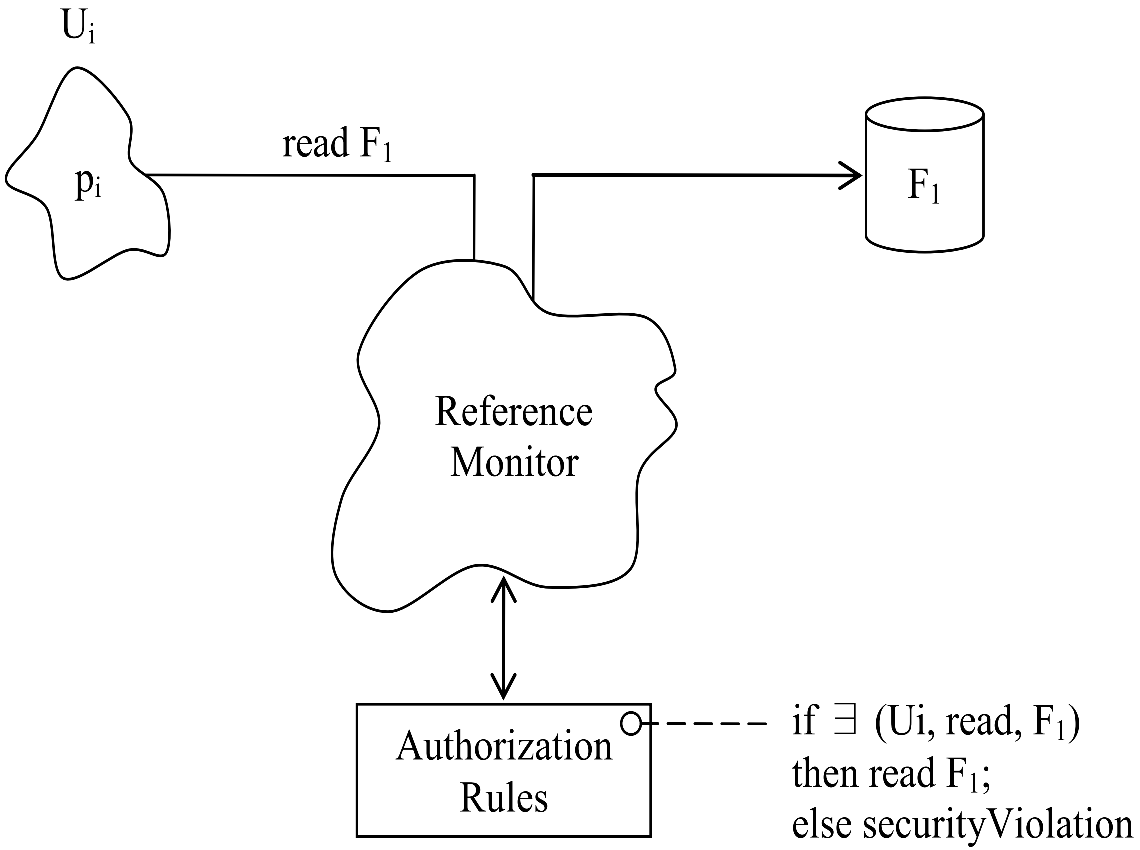
Access Matrix



Reference Monitor

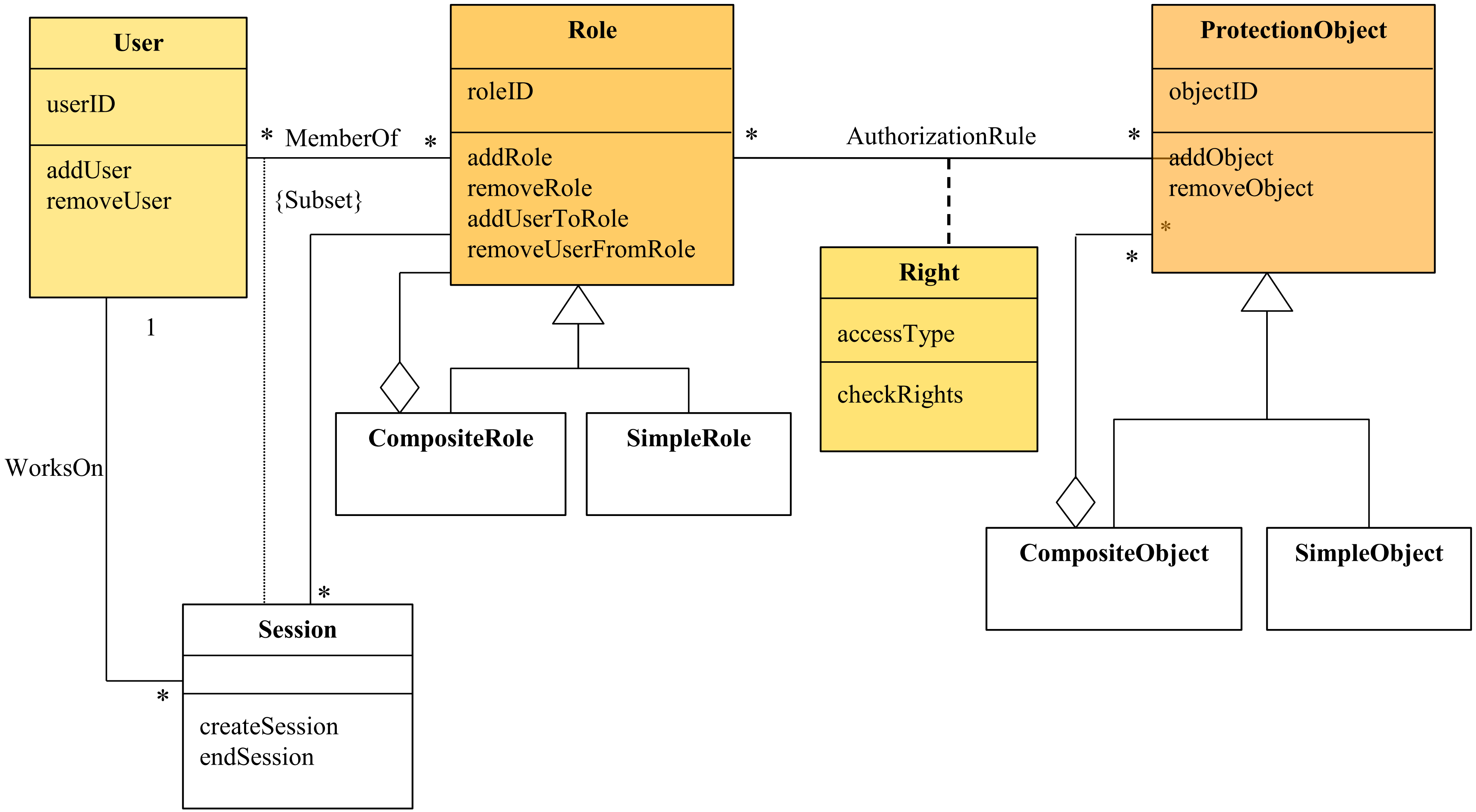
- ***Authorization rules define who has access to what and how. They must be enforced when a process request a resource***
- ***Each request for resources must be intercepted and evaluated for authorized access; this is the concept of Reference Monitor***
- ***An abstract concept, implemented as memory access manager, file permission checks, CORBA adapters, etc.***

Abstract Reference Monitor



Role-Based Access Control

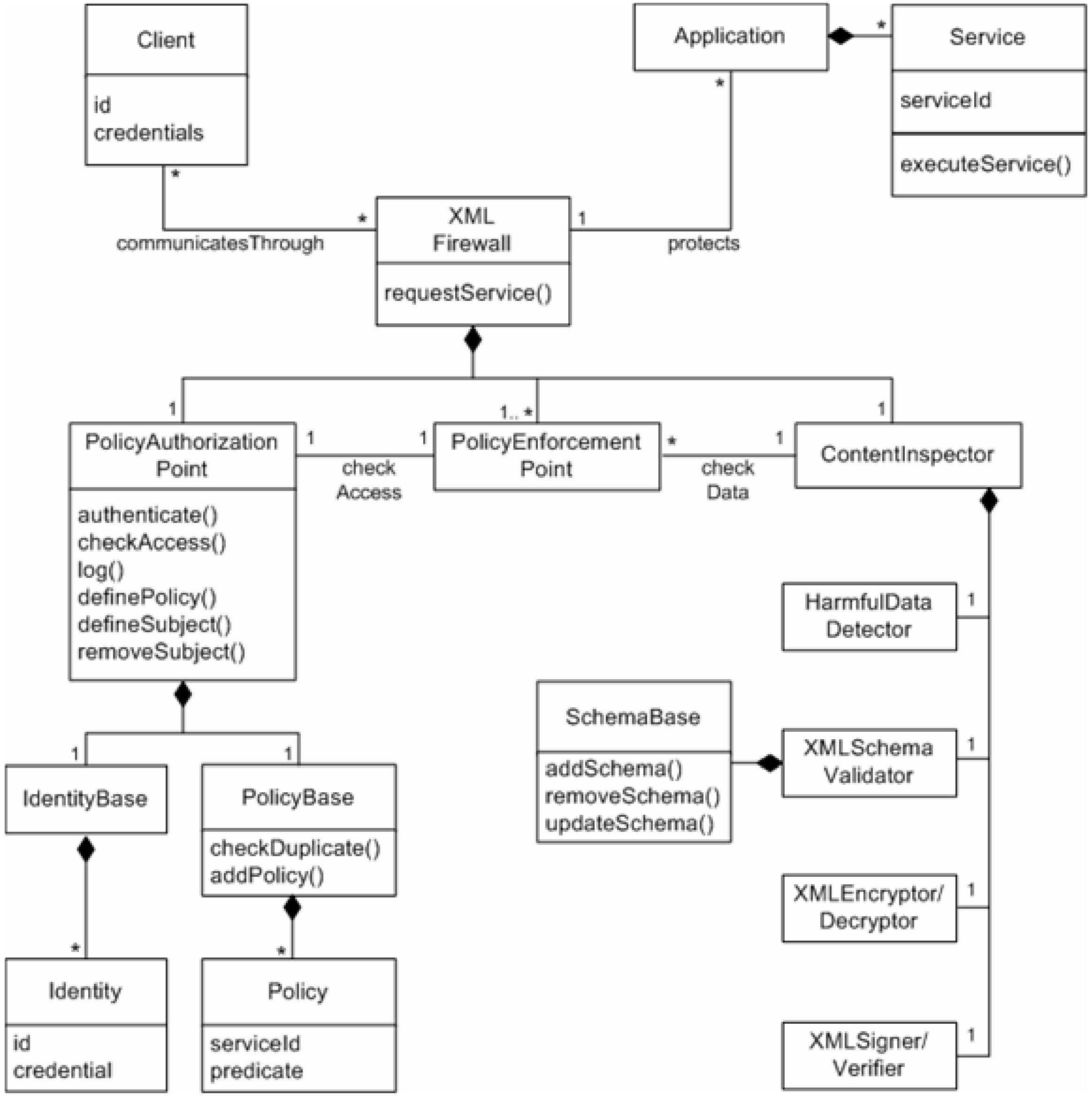
- ***Users are assigned roles according to their functions and given the needed rights (access types for specific objects)***
- ***When users are assigned by administrators, this is a mandatory model***
- ***Can implement least privilege and separation of duty policies***



User
userID
addUser
removeUser
*
MemberOf
*
{Subset}
1
WorksOn
*
*
Session
createSession
endSession
Role
roleID
addRole
removeRole
addUserToRole
removeUserFromRole
CompositeRole
SimpleRole
*
AuthorizationRule
*
Right
accessType
checkRights
ProtectionObject
objectID
addObject
removeObject
*
*
CompositeObject
SimpleObject

XML firewall

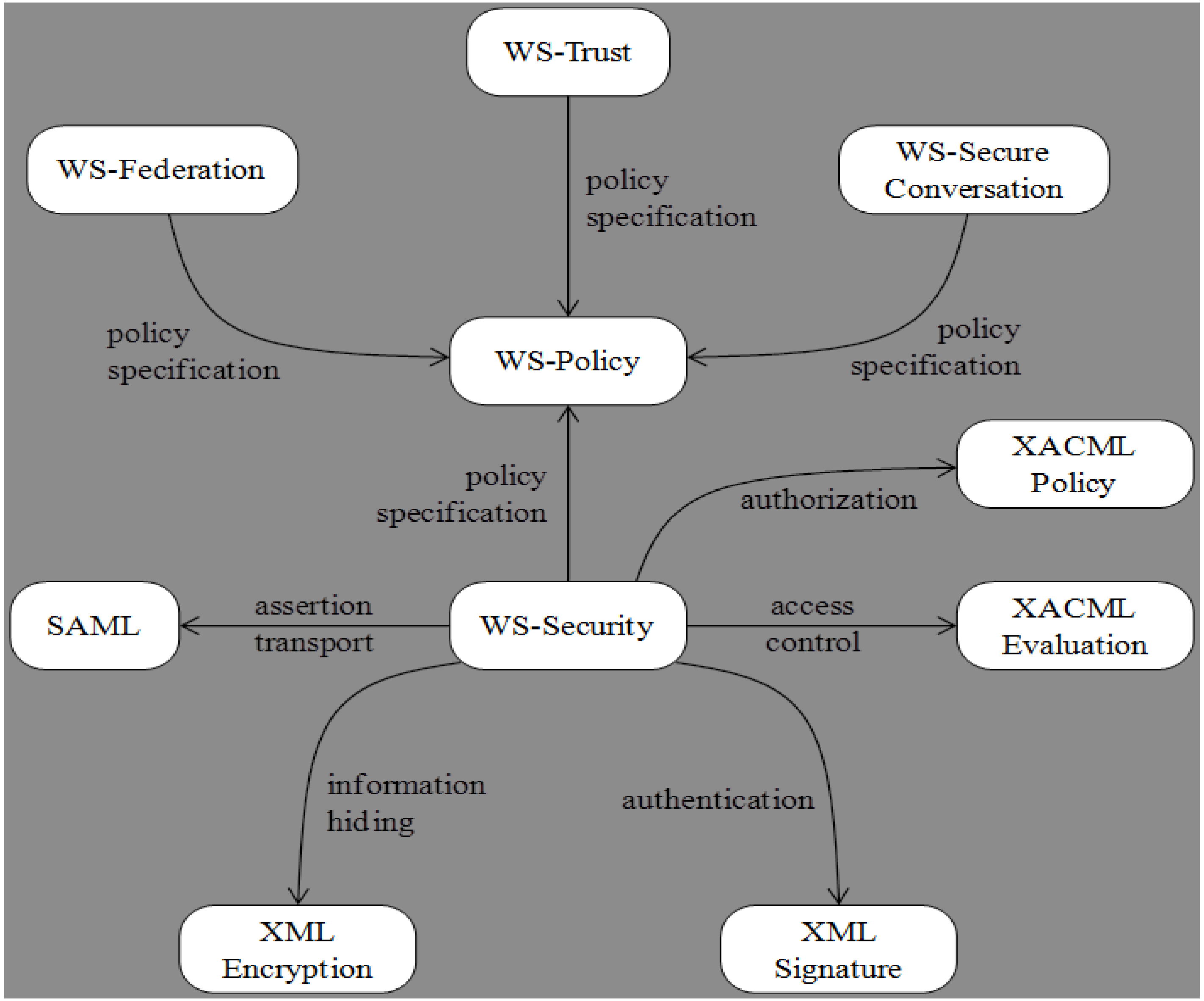
- ***Controls input/output of XML applications***
- ***Well-formed documents (schema as reference)***
- ***Harmful data (wrong type or length)***
- ***Encryption/decryption***
- ***Sign and verify signatures in documents***


Client
id
credentials
Application
Service
serviceId
executeService()
XML Firewall
requestService()
communicatesThrough
protects
PolicyAuthorization Point
authenticate()
checkAccess()
log()
definePolicy()
defineSubject()
removeSubject()
check Access
PolicyEnforcement Point
check Data
ContentInspector
HarmfulData Detector
SchemaBase
addSchema()
removeSchema()
updateSchema()
XMLSchema Validator
XMLEncryptor/ Decryptor
XMLSigner/ Verifier
IdentityBase
PolicyBase
checkDuplicate()
addPolicy()
Identity
id
credential
Policy
serviceId
predicate

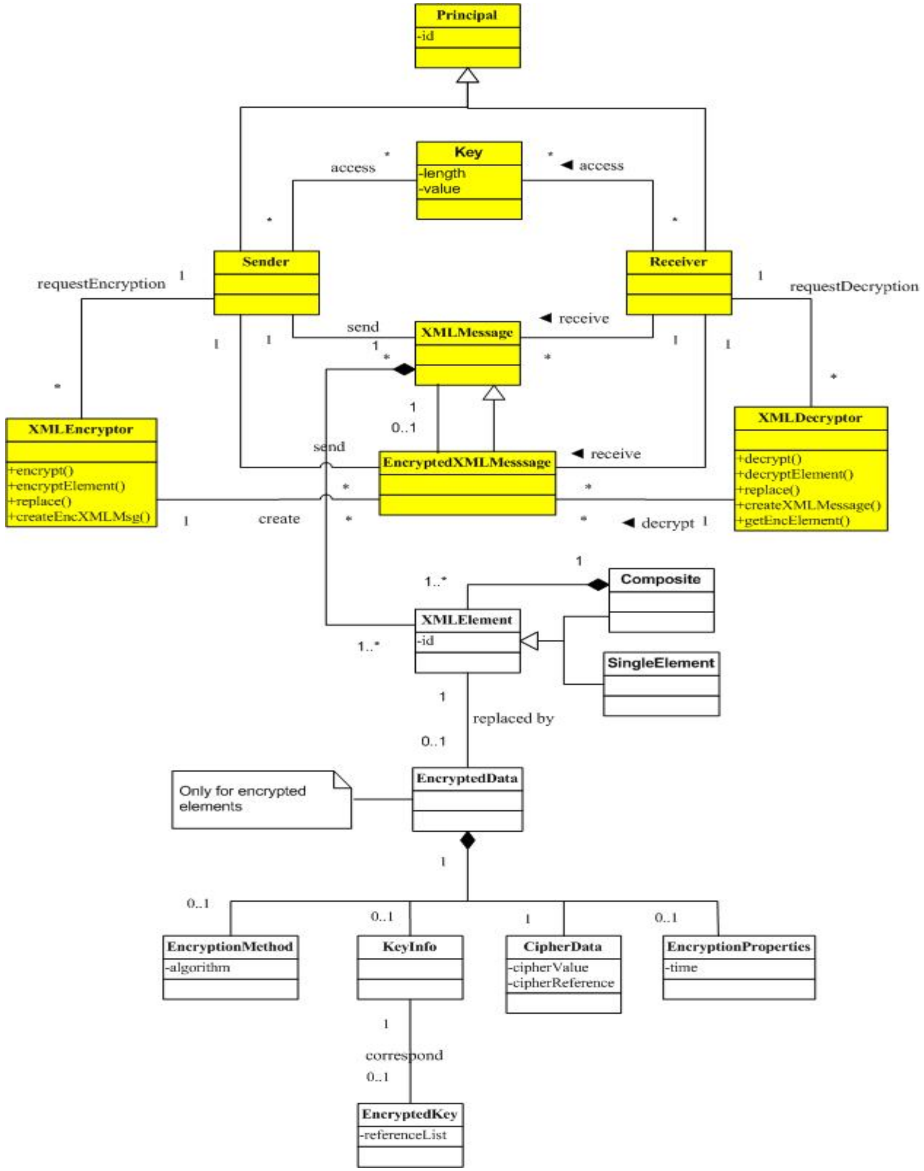
Description of standards

- Some standards, e.g., those for XML web services security, are very complex and described in verbose documents (50-100 pages each)
- By describing those standards as patterns we have made them much easier to understand and apply
- We modeled most of the standards for XML web services
- The next slide describes the set of standards we modeled and the following slide after it a specific standard

Web services standards



XML Encryption standard



Building secure systems

- **Secure systems need to be built in a systematic way** where security is an integral part of the lifecycle, and the same applies to safety.
- The platform should match the type of application, and **all compliance, safety and security constraints should be defined at the application level, where their semantics are understood and propagated to the lower levels**.
- The lower levels must provide the assurance that the constraints are being followed, i.e., they implement these constraints and enforce that there are no ways to bypass them.
- Following these ideas, we developed **a secure systems development methodology,** which considers all lifecycle stages and all architectural levels. We expanded its architectural aspects, and recently expanded it with process aspects. We are now extending it to CPSs. We use reference architectures as guidelines.

What is a security methodology?

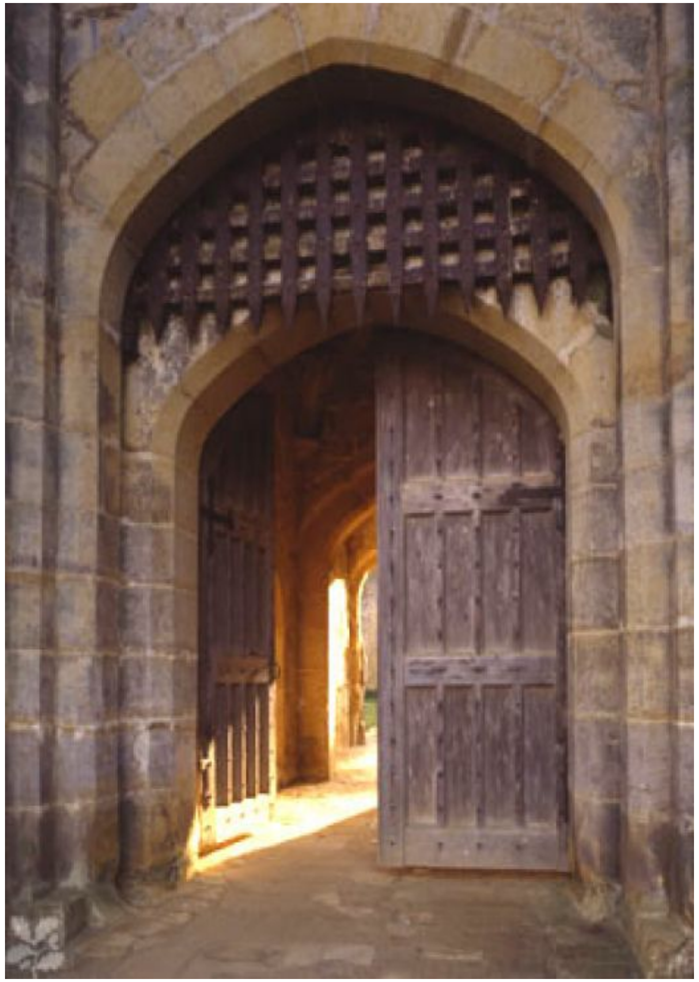
- *Methodology*: systematic way of doing something
- *Security methodology*:

  systematic way of introducing security into a software system during the development life-cycle

- Advantages analogous to those of software engineering process vs. ad-hoc development
- Partial or comprehensive; covering early phases of the development life-cycle especially important
- Consists of two aspects/facets: **security process** (SP) and **conceptual security framework** (CF)
- Can be specific (e.g. Web services, CPS) or generic (distributed systems)

ASE: a comprehensive security methodology for distributed systems

- Many methodologies exist with different paradigms
- Very important class is methodologies that use security patterns
- ASE: a security methodology using patterns and related constructs designed specifically for general distributed systems



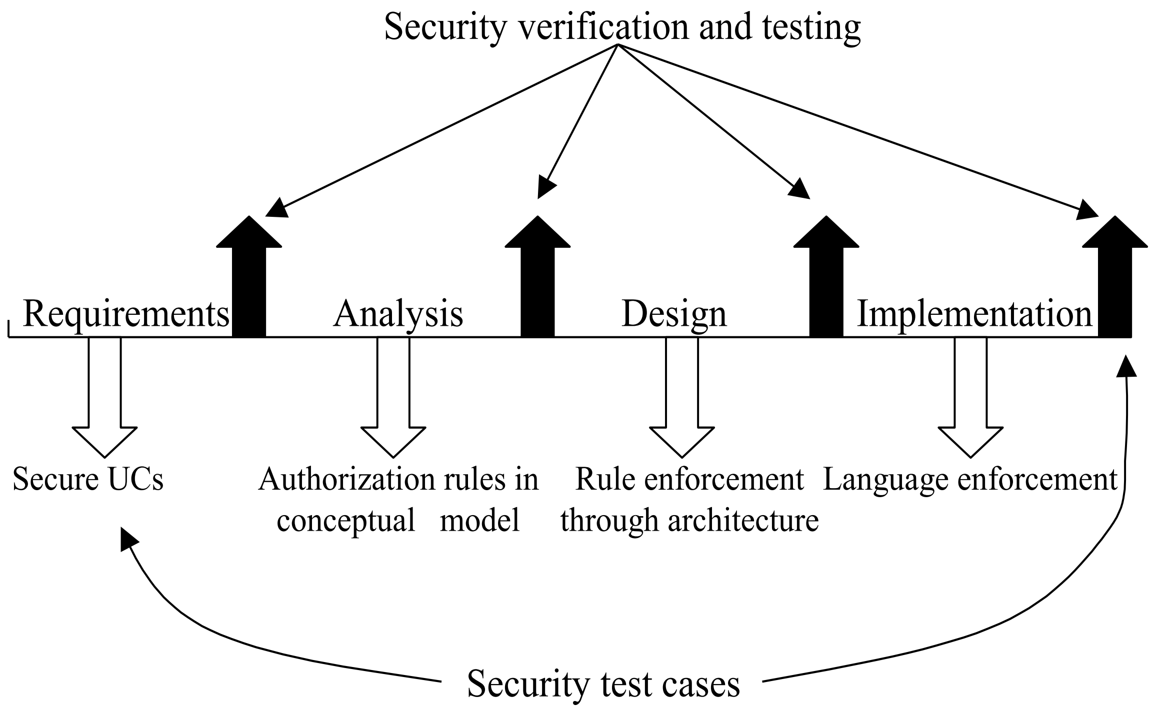
Major elements of CF: Threat taxonomies/libraries

- *Threat taxonomies/libraries* consist of *threat patterns*, which can be customized and instantiated in different architectural contexts to define specific threats to a system.
- Allow developers to quickly and efficiently consider a range of relevant threats during threat modeling.

Threat classes

Functionality decomposition layer	Relevant threat classes
User interaction	Identity attacks, Passing illegal data, Remote information inference, Repudiation, Uncontrolled operations
Data / storage management	Passing illegal data, Stored data attacks, Remote information inference, Uncontrolled operations
Resource management	Uncontrolled operations
Distribution control	Identity attacks, Passing illegal data, Remote information inference, Uncontrolled operations
Communication	Network communication attacks, Network protocol attacks, Repudiation
Addressing	Network communication attacks, Network protocol attacks, Repudiation

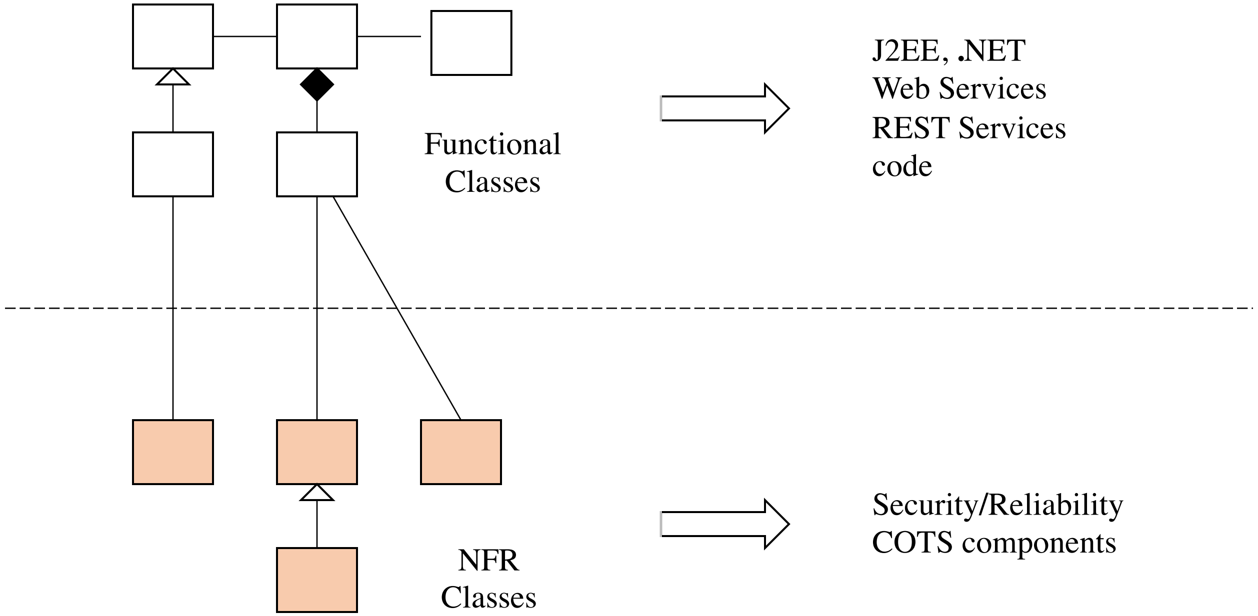
Secure software lifecycle



Basic security principles for system design

- Security constraints must be defined at the highest layer, where their semantics are clear, and propagated to the lower levels, which enforce them.
- **All the layers of the architecture must be secure**.
- We can define patterns at all levels. This allows a designer to make sure that all levels are secured, and also makes easier propagating down the high-level constraints.
- **We must apply security in all development stages**
- A two-dimensional approach: time and space

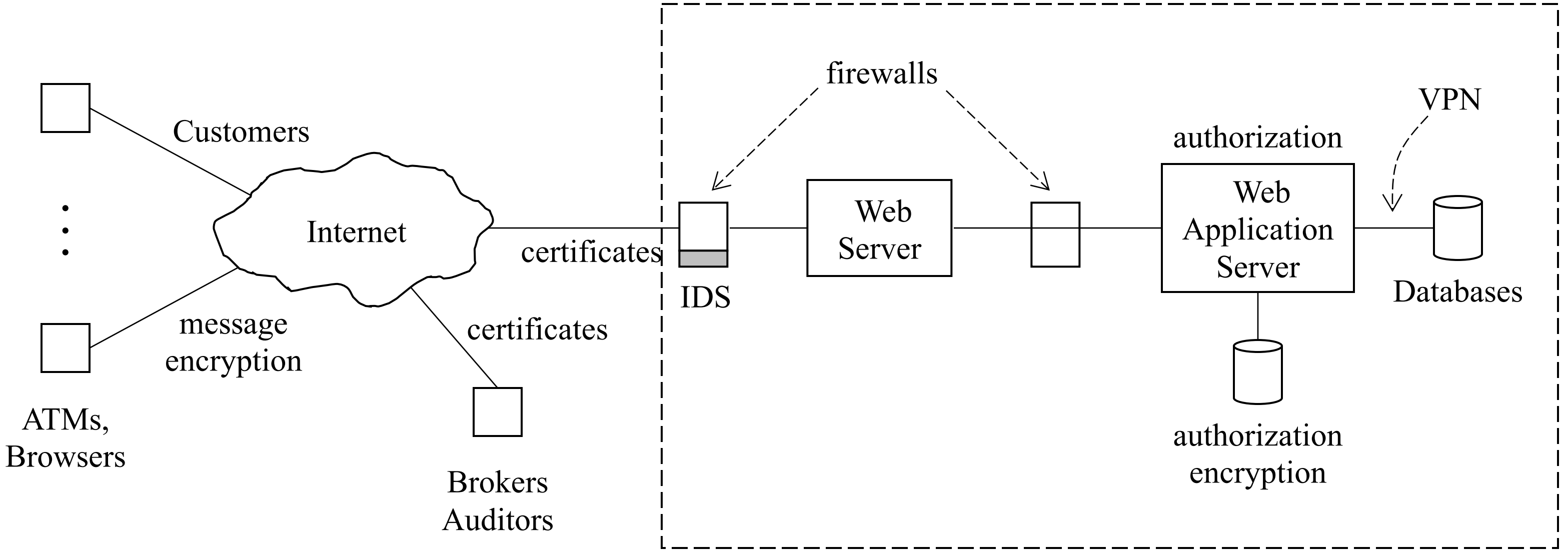
Secured system



J2EE, .NET
Web Services
REST Services
code

Security/Reliability
COTS components

Deployment for secured financial institution

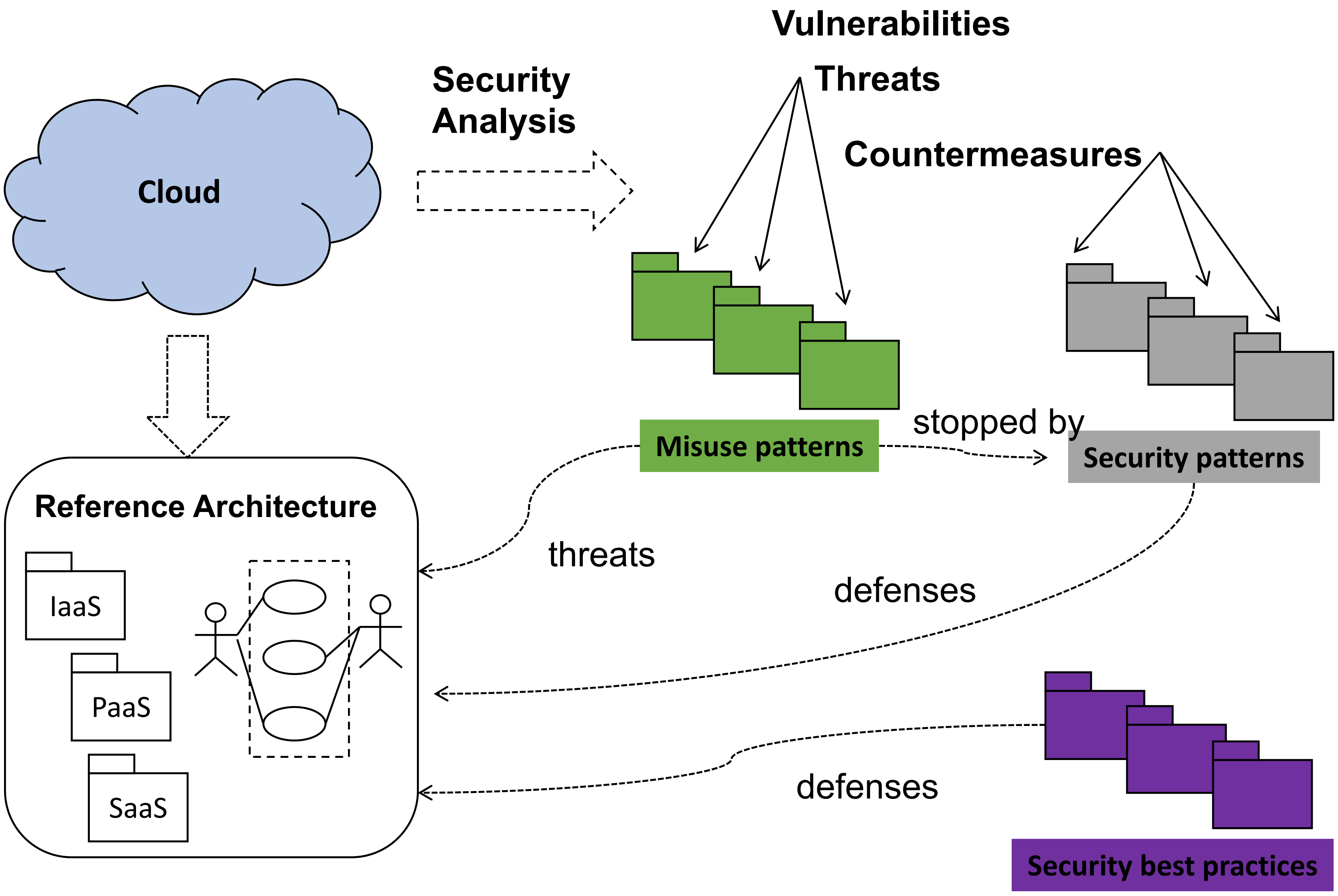


Reference Architecture (RA)

- A **Reference Architecture** (RA) is a generic software architecture, based on one or more domains, with no implementation aspects
- An RA is reusable, extendable, and configurable.
- It specifies the components of the system, their individual functionalities and their mutual interaction.
- An RA can be considered as **a compound pattern** and its components described as patterns.
- In addition to domain models, an RA may include a set of use cases (UC), and a set of Roles (R) corresponding to its stakeholders (actors).

Securing an RA

- We start from a list of use cases which describe the typical cloud uses and their associated roles
- We analyze each use case looking for vulnerabilities and threats. This implies checking each activity in the activity diagram of the use cases to see how it can be attacked. This approach results in a systematic enumeration of threats.
- We use lists of threats from repositories to confirm these threats and to find possible further vulnerabilities and threats.
- These threats are expressed in the form of misuse patterns. We developed some misuse patterns for Clouds.
- We apply policies to handle the threats and we identify security patterns to realize the policies. There are some defenses that come from best practices and others that handle specific threats. There are also regulatory policies which are realized as security patterns.


Vulnerabilities
Security
Analysis
Threats
Countermeasures
Cloud
Misuse patterns
stopped by
Security patterns
Reference Architecture
IaaS
PaaS
SaaS
threats
defenses
defenses
Security best practices

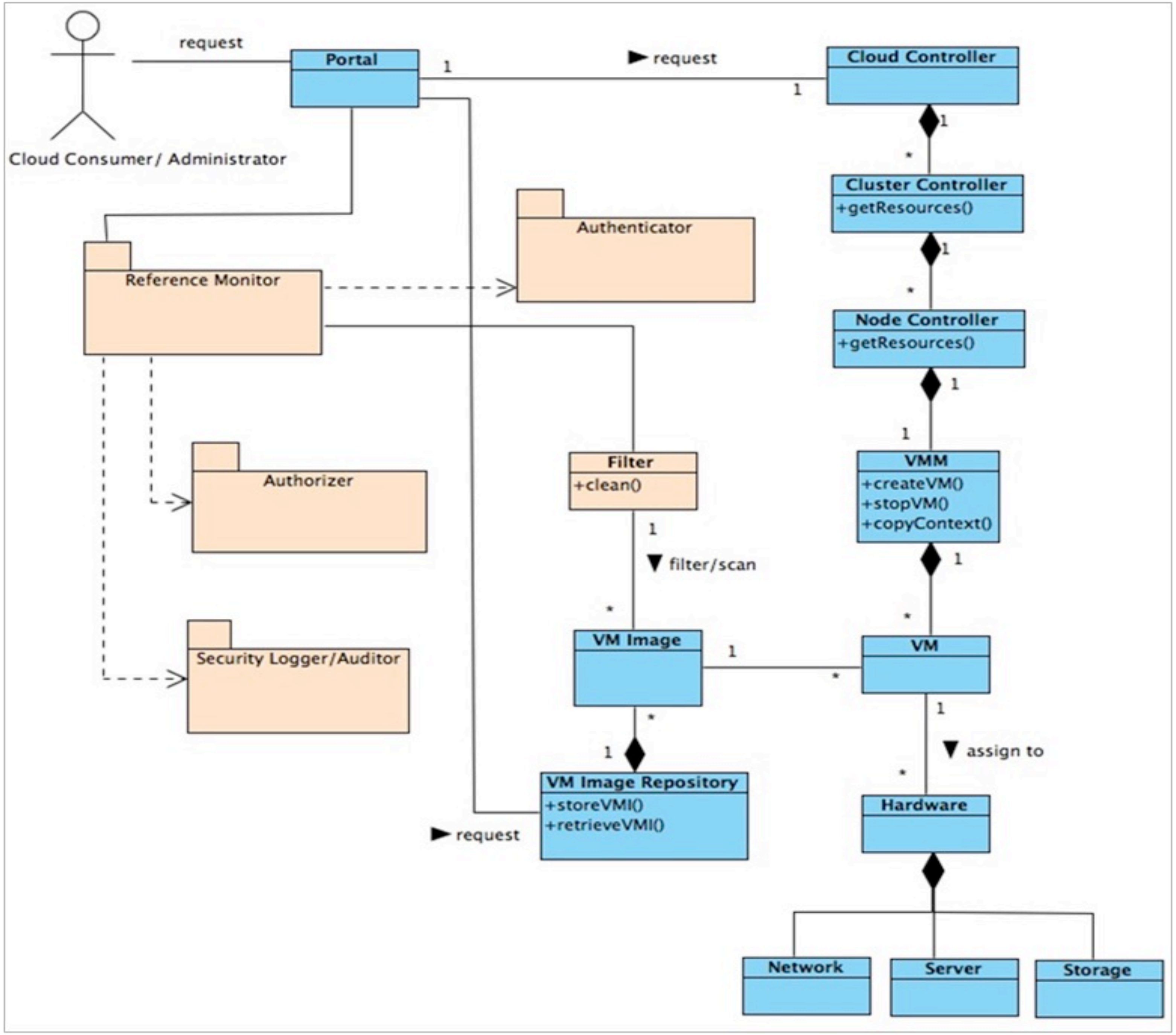
Threat enumeration and modeling

- In previous work we introduced an approach for **threat enumeration**
- This process is performed during the requirements and the design stages of the software development cycle and **analyzes each activity in the activity diagram of a use case** to see how it could be subverted by an attacker to reach her goals
- This analysis results in a set of threats and since the use cases are all the ways to interact with a system **we can enumerate threats systematically**
- We then consider which policies can mitigate these threats and we realize the policies with patterns; in fact, we incorporated this approach as part of a systematic methodology to build secure systems
- This process requires developers to conjecture possible attacks to different assets or parts of a system, to assess their impact and likelihood, and to determine how they could potentially be stopped or mitigated.
- We use the reference architecture (RA) as a reference framework, i.e., **each threat is related to a specific component of the architecture**

Threat List vs. Defenses for a cloud

ID	Threats	Defense
T11	The cloud consumer is malicious and inserts malicious code into the VMI	Authenticator - Authorizer
T21	An external attacker listens to the network to obtain information about the VMI	Secure Channel
T22	VMI may be modified while in transit	Secure Channel
T23	Disavows sending a VMI	Security Logger/Auditor
T31	The IaaS administrator is malicious and collects information within the VMI	Authenticator - Authorizer
T32	The IaaS disavows receiving a VMI	Security Logger/Auditor
T33	Insert malicious code in the image	Authenticator - Authorizer
T41	The IaaS administrator stores a malicious VMI	Authorizer – Authorizer Filter

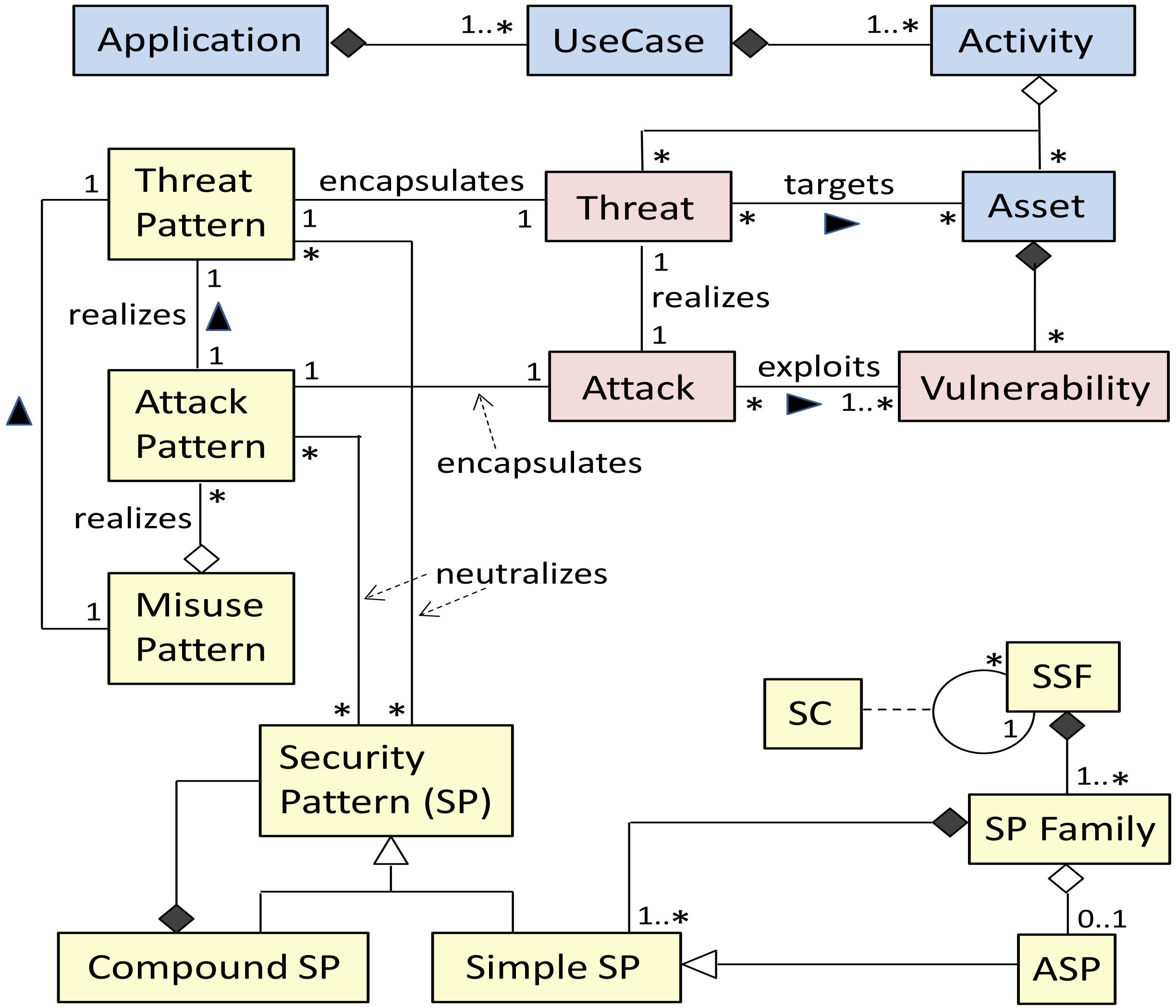
Partial SRA for Clouds



Misuse patterns

- A **misuse pattern** describes, from the point of view of an attacker, a generic way of performing an attack that takes advantage of the specific vulnerabilities of some environment or context
- A misuse is reading a list of credit card numbers, modifying a schedule,…
- Misuse patterns define the environment where the attack is performed, **countermeasures** to stop it, and indicate where to find forensic information in order to trace the attack once it happens
- This systematic and structured representation of attacks is important to classify and unify them as well as to find countermeasures against them
- We describe this type of patterns as well as our security patterns using a template based on the one used in the POSA book, which is commonly used for architectural patterns as well as security patterns

Metamodel for security concepts



Security verification

- Once all iterations of the security implementation stage are completed, the resulting software system must be carefully verified as to whether it really does satisfy the security architecture specifications (threats).
- This is accomplished by considering misuse pattern realizations of each of the threats found during the development phases, and performing penetration testing on the software system.
- We can measure security by **counting the threats that have been neutralized by using patterns**
- We can verify that a particular countermeasure has been implemented correctly, and also determine whether that countermeasure is effective against (corresponding) representative attacks.

Conclusions

- Security patterns are a useful tool to build secure architectures
- A strong system architecture can prevent the propagation of successful attacks to a part of the system (segmentation and gate checking), we have applied accepted design principles directly or through patterns
- We have written about **150 security patterns**, we intend to unify patterns from other authors
- They require appropriate methodologies to use them, good catalogs and tools
- They can handle **security in a holistic way**, necessary for complex systems
- Patterns are also valuable for evaluating existing systems and for teaching security concepts
- Reference architectures can simplify secure application development or can be used to build secure architectures that conform to some type of application, e.g. clouds

Conclusions II

- Patterns cannot prevent attacks that happen through code flaws but can make their effect much less harmful; in any case, they can be complemented with code analysis
- Patterns can be made more formal: Object Constraint Language (OCL)
- **Patterns emphasize architectural aspects**, keys to understand and prevent most attacks.
- Patterns can lead to building strong systems, a more effective and ethical approach than retaliation

References on security patterns and RAs

- E.B.Fernandez, *"Security patterns in practice: Building secure architectures using software patterns"*, Wiley Series on Software Design Patterns, 2013.
- E. B.Fernandez, Nobukazu Yoshioka, Hironori Washizaki, and Joseph Yoder, "Abstract security patterns for requirements specification and analysis of secure systems", *Procs. of the WER 2014 conference, a track of the 17th Ibero-American Conf. on Soft. Eng.(CIbSE 2014),* Pucon, Chile, April 2014
- E.B.Fernandez, Raul Monge, and Keiko Hashizume, "Building a security reference architecture for cloud systems", *Requirements Engineering*. Doi: 10.1007/s00766-014-0218-7, 2015
- E. B. Fernandez, Nobukazu Yoshioka, Hironori Washizaki and Madiha H. Syed, Modeling and security in cloud ecosystems, *Future Internet* **2016**, *8*(2), 13; doi:10.3390/fi8020013, (Special Issue Security in Cloud Computing and Big Data
- M. Schumacher, E. B.Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering",* Wiley Series on Software Design Patterns, 2006.

References on methodology

- E. B. Fernandez, M.M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", Chapter 5 in "*Integrating security and software engineering: Advances and future vision",* H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006, 107-126.
- Anton Uzunov and E.B.Fernandez, “An Extensible Pattern-based Library and Taxonomy of Security Threats for Distributed Systems”- Special Issue on Security in Information Systems of the *Journal of Computer Standards & Interfaces*. 2013. http://dx.doi.org/10.1016/j.csi.2013.12.008
- Anton Uzunov, E. B Fernandez, Katrina Falkner, “ASE: A Comprehensive Pattern- Driven Security Methodology for Distributed Systems”, *Journal of Computer Standards & Interfaces ,* Vol. 41, September 2015, 112-137,
- Anton Uzunov, E. B Fernandez, Katrina Falkner, “Security solution frames and security patterns for authorization in distributed, collaborative systems”, *Computers & Security*, 55, 2015, 193-234, doi: 10.1016/j.cose.2015.08.003