Access Control

SecAppDev 2016

Maarten Decat


maarten.decat@kuleuven.be


@maartendecat

DistriNet
iMinds
KU LEUVEN

What is access control?

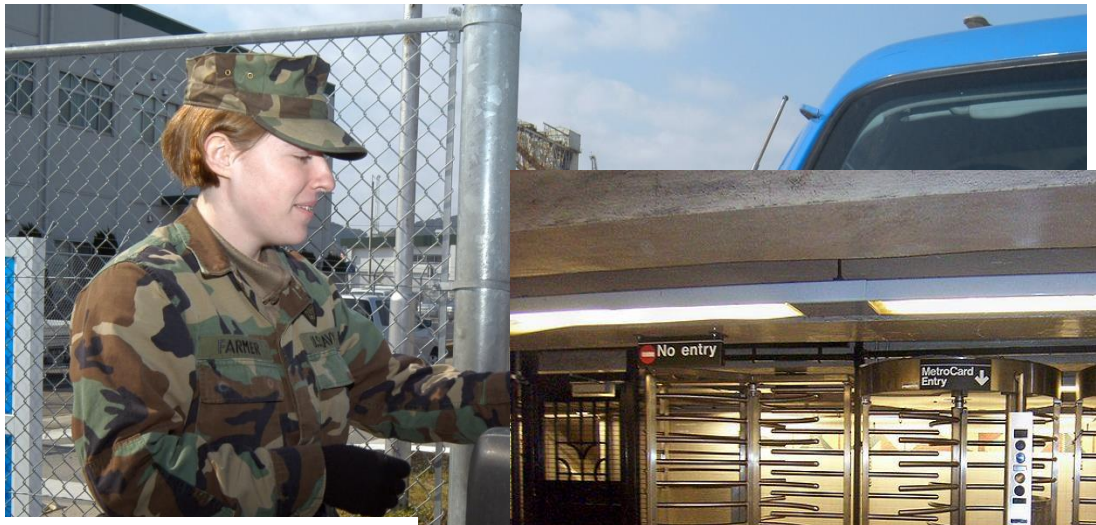
Access control is the part of *security* that constrains the *actions* that are performed in a system based on *access control rules*.

- As any security: confidentiality, integrity, availability
- Layer in between (malicious) users and the protected system
- Part of the Trusted Computing Base

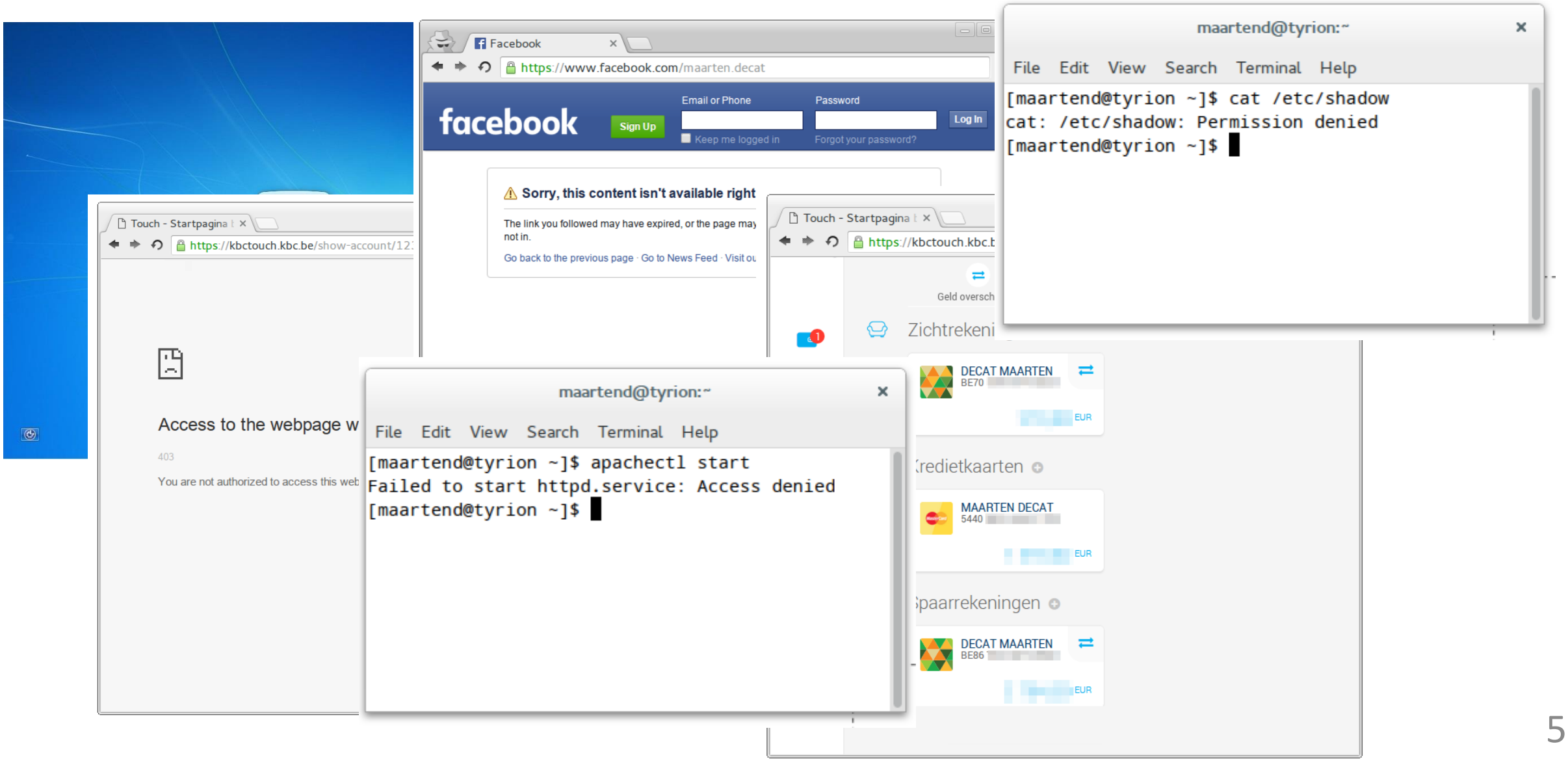
What is access control?

1. Not easy to get right,
   e.g., what about windows?
2. Difference between access
   rules and mechanism
3. Different mechanisms have
   different properties
4. Different mechanisms support
   different rules

Access control



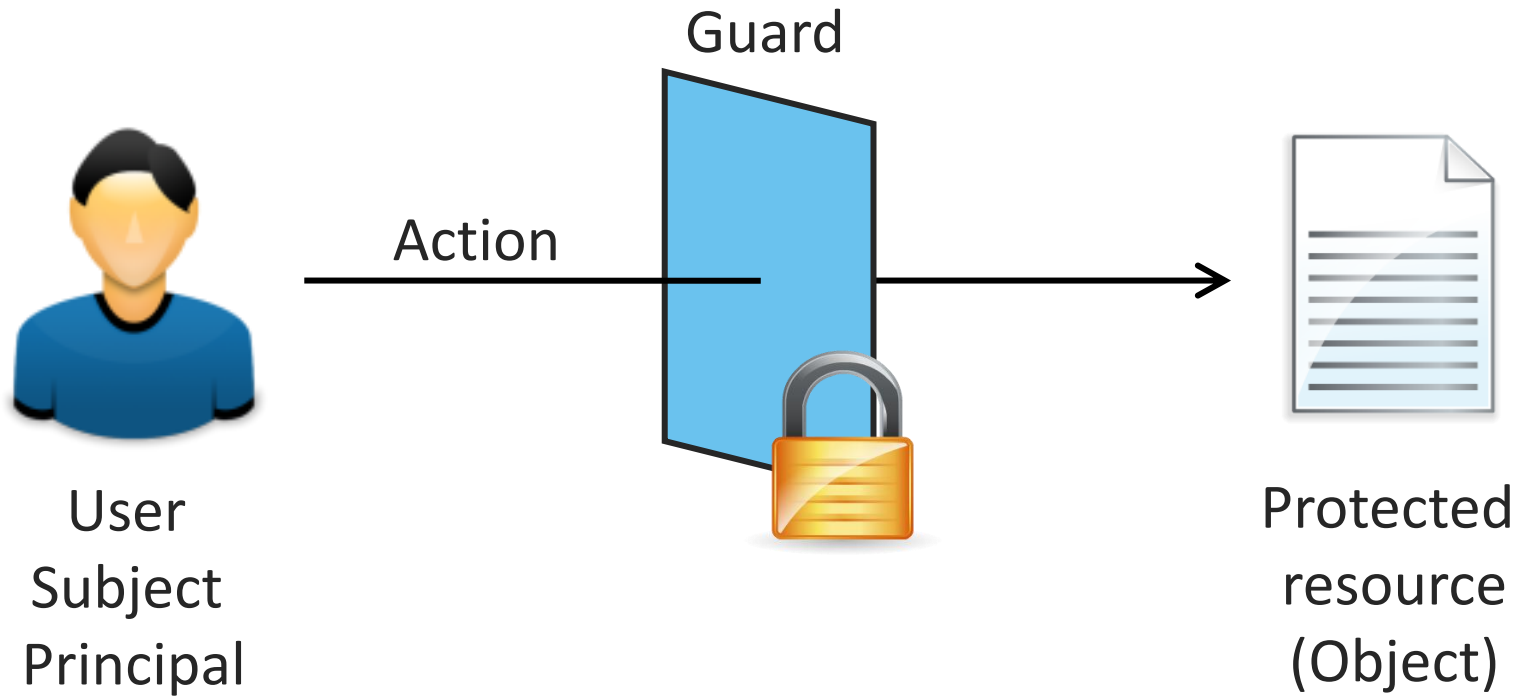
Access control in software



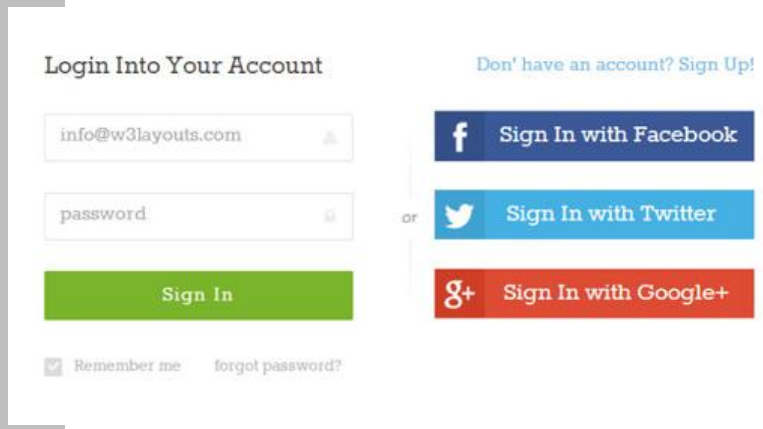
Outline

- Introduction
- Positioning access control
- Access control models
- How to enforce access control
- The bigger picture
- Some important technologies in practice
- Recap and conclusion

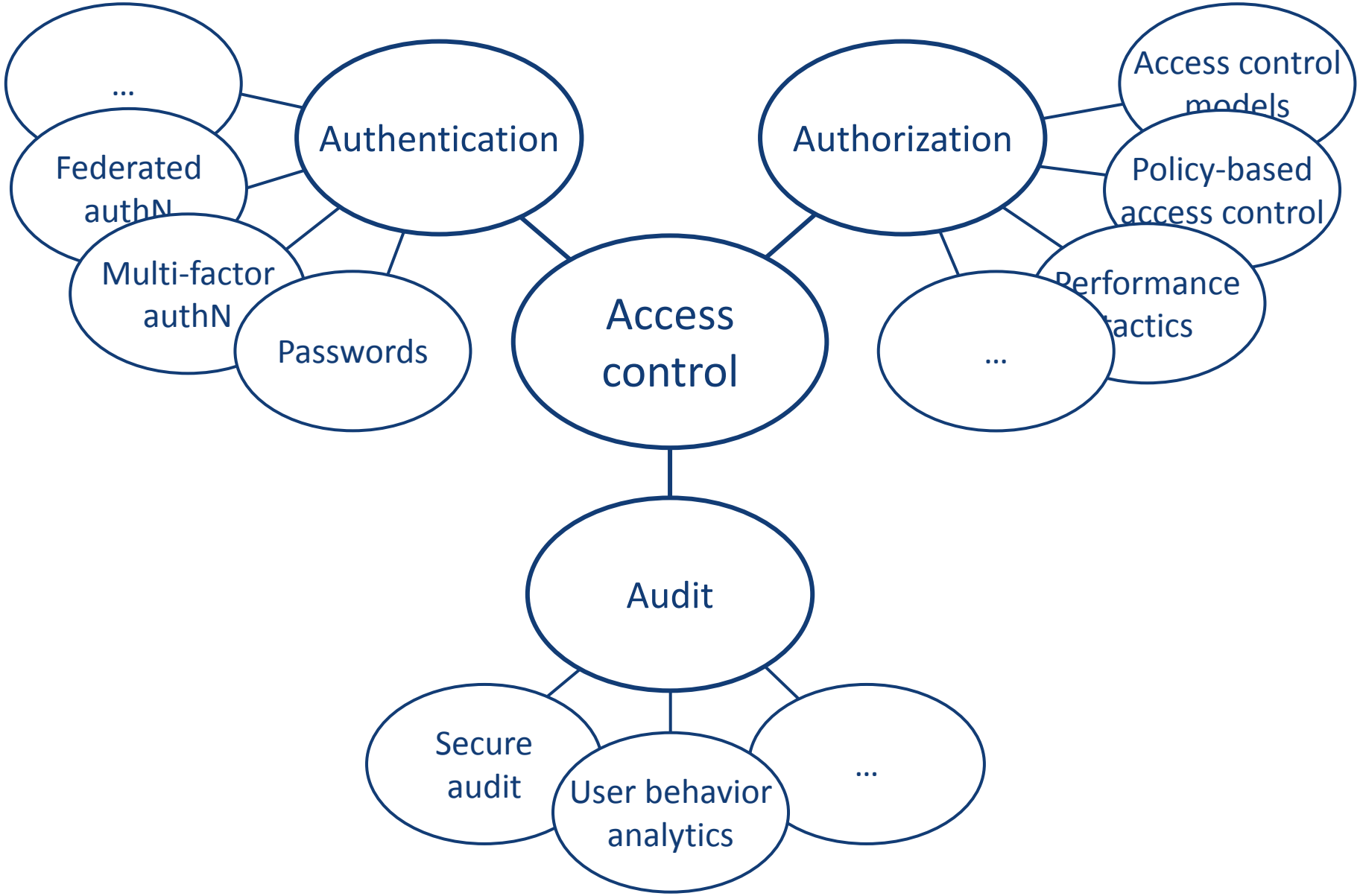
10,000m point of view



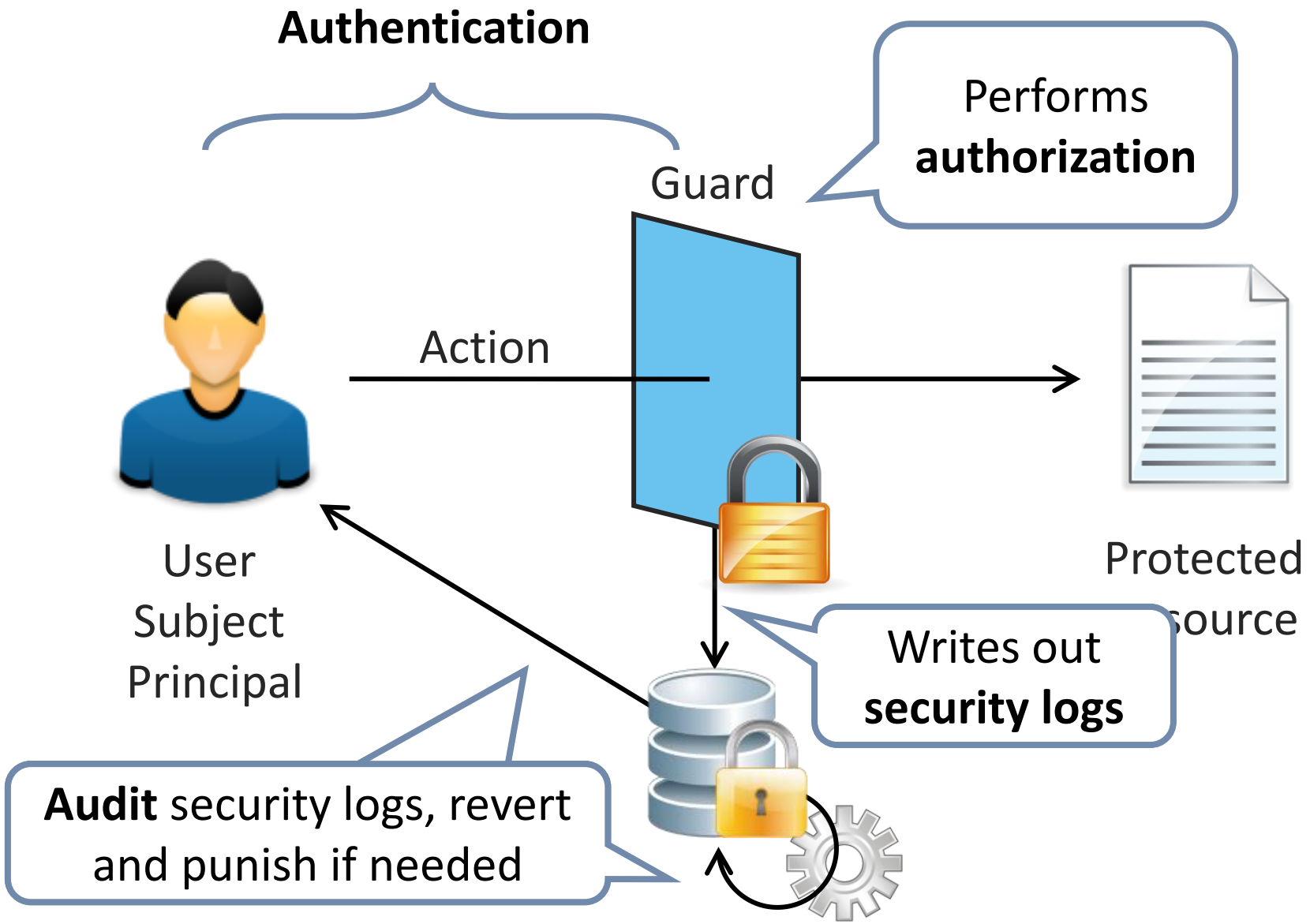
But there is more to it



But there is more to it

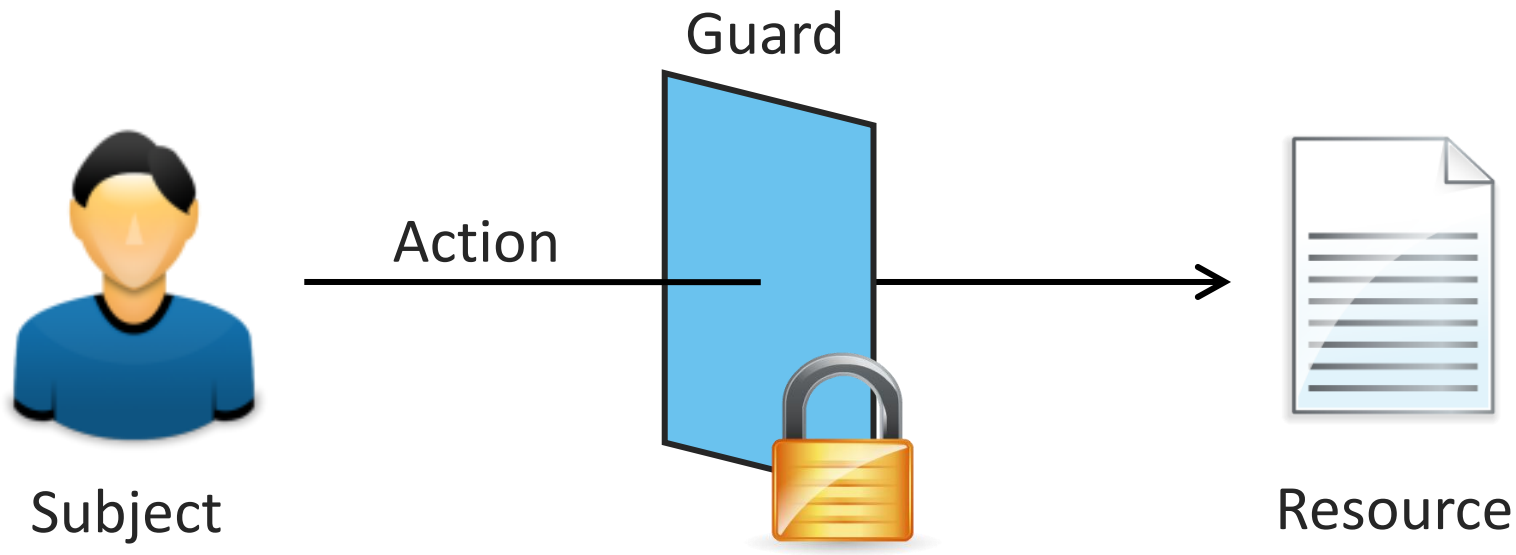


5000m point of view



For the rest of this presentation

“Access control” = “authorization”



Authorization exists on multiple levels

Level	Subject	Action	Guard	Protected System
Hardware	OS Process	Read memory	CPU	CPU and Memory
Network	Host	Send packets	Firewall	Intranet
Database	User	SELECT query	DBMS	User database
OS	User	Open file	OS Kernel	Filesystem
OS	Java Program	Open file	Java Security Manager	Filesystem
Application	User	Read patient file	Application code	Application data

Models, policies and mechanisms

- Guard is responsible for mediating access
  - Authorize specific actions
  - *Mechanism* that enforces a specific *security policy*

- Rules, policies, models and mechanisms
  - **Access rules**: the logical access rules, independent of representation
  - **Policy**: an explicit representation of the desired rules in a SW artifact
  - **Model**: (formal) representation of how rules can be expressed
  - **Mechanism**: low-level implementation of controls

- Access control seems straightforward… but is it?

Challenges for access control

- **Expressiveness**: can the high-level rules be expressed in terms of the access control model?
- **Performance**: access control decisions are frequent, and must be dealt with quickly
- **Full mediation**: does the guard check *every* action? Does your policy cover every action?
- **Safety**: does the access control mechanism match the policy?

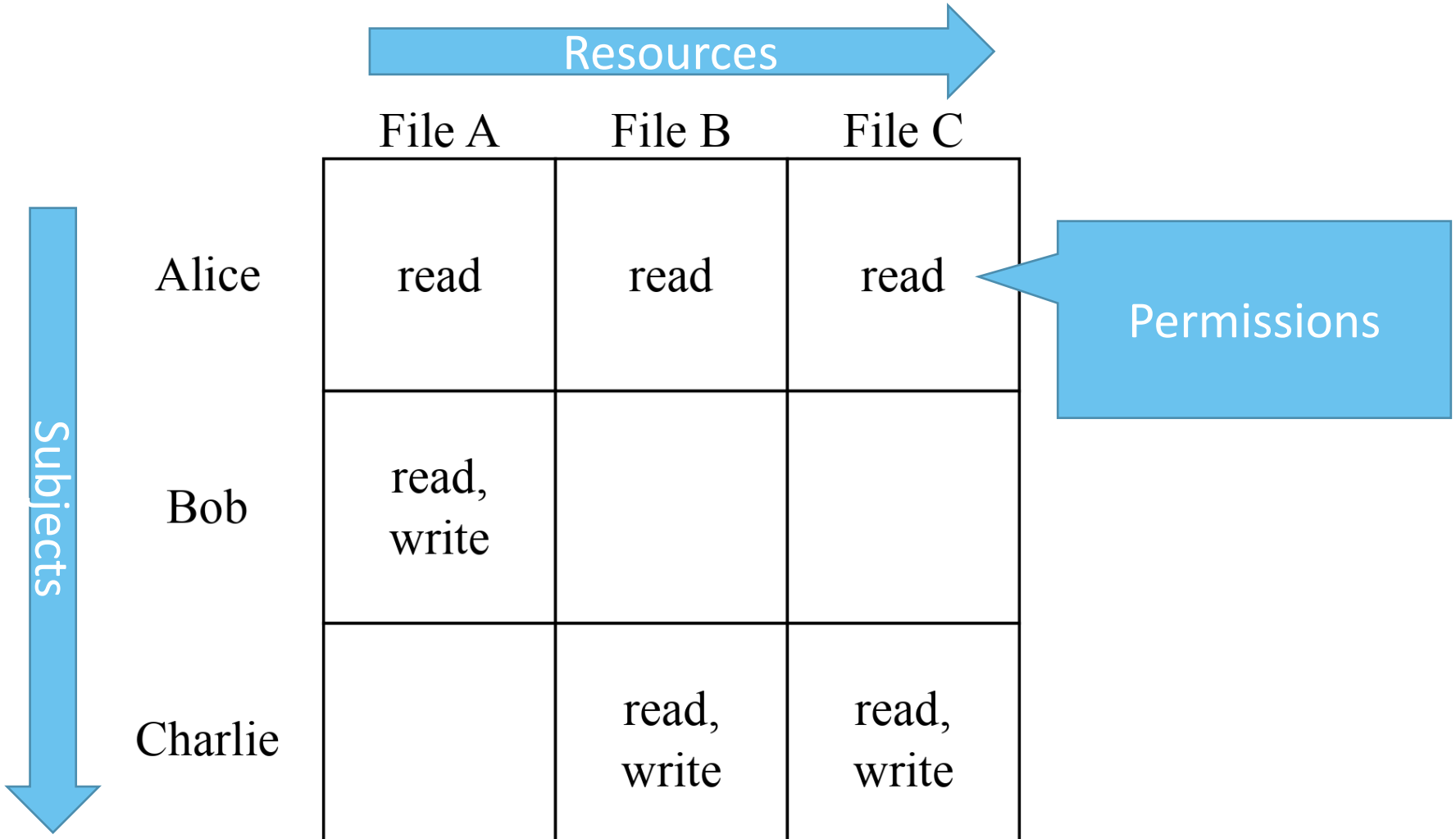
CWE/SANS Top 25 Software Errors

Rank	Description
5	Missing authentication for critical function
6	Missing authorization
7	Use of hard-coded credentials
8	Missing encryption of sensitive data
10	Reliance on untrusted inputs in a security decision
11	Execution with unnecessary privileges
15	Incorrect authorization
17	Incorrect permission assignment for critical resource
19	Use of a broken or risky cryptographic algorithm
21	Improper restriction of authentication attempts
25	Use of a one-way hash without a salt

Outline

- Introduction
- Positioning access control
- Access control models
  - The basics
  - Who can assign permissions
  - How permissions are assigned
  - Advanced topics
- How to enforce access control
- The bigger picture
- Some important technologies in practice
- Recap and conclusion

The basics: the access control matrix



[Lampson1971]

Who can assign permissions?

Who can assign permissions?

- In general, two approaches:

1. Mandatory access control (MAC)
   - By central authority

2. Discretionary access control (DAC)
   - By subjects themselves

Mandatory access control (MAC)

- Permissions are assigned by a central authority according to a central policy
  - Good fit within organizations with a strong need for central controls
  - Low flexibility and high management overhead

- Mandatory Access Control in use
  - Often linked to multi-level security systems -> see later on
    - E.g. Government-regulated secrecy systems, military applications
  - Modern operating systems, to separate applications and processes
    - E.g. Windows' *Mandatory Integrity Control*, SELinux, TrustedBSD

SELinux

- Security-Enhanced Linux
  - “A set of patches to the Linux kernel and some utilities to incorporate a strong, flexible MAC architecture into the major subsystems of the kernel [for] confidentiality and integrity”
  - Activated by default in Fedora, Red Hat Enterprise Linux, etc
- Enforce MAC policy to processes in order to limit access to files and network resources
  - Least privilege
- Policy-based (see later on)
  - Separation of policy from enforcement with well-defined policy interfaces
  - Changing a policy does not require a reboot

SELinux

```
~]$ ls -Z /usr/bin/passwd
-rwsr-xr-x. root root system_u:object_r:passwd_exec_t:s0 /usr/bin/passwd
~]$ ls -Z /etc/shadow
----------. root root system_u:object_r:shadow_t:s0 /etc/shadow
```

SELinux policies:

- applications running in the `passwd_t` domain can access files labeled with the `shadow_t` type
- the `passwd_t` domain can be entered from the `passwd_exec_t` type

Discretionary access control (DAC)

- Permissions are set *at the discretion* of the resource owner
  - Highly flexible policy, where permissions can be transferred
  - Lack of central control makes revocation or changes difficult

- Discretionary acces control in use
  - Controlling access to files
    - E.g., Windows Access Control Lists (ACL), UNIX file handles
  - Controlling the sharing of personal information
    - E.g., Social networks

The Graham-Denning Model

- Extends the access control matrix with control and ownership
  - Objects have an owner
  - Subjects have a controller
  - Permissions can be made transferrable

	Alice	Bob	File 1	File 2	File 3
Alice	control	owner	owner read write	owner	read*
Bob		control		read write	owner read

- Matrix can be modified by 8 commands
  - Creating and destroying subjects and objects
  - Granting, transferring and revoking permissions
  - Inspecting the authorization state

[Graham1972]

The Graham-Denning Model

1. Subject Alice creates object File 1

	Alice	File 1
Alice	control	owner

2. Subject Alice creates subject P1

	Alice	P1
Alice	control	owner
P1		control

3. Subject Alice destroys object File 1

➡ Alice must own File 1

	Alice	File 1
Alice	control	owner

4. Subject Alice destroys subject P1

➡ Alice must own P1

	Alice	P1
Alice	control	owner
P1		control

The Graham-Denning Model

5. Subject Alice grants a right read/read on File 1 to P1*

➡ Alice must be owner of File 1

	Alice	P1	File 1
Alice	control	control owner	owner
P1			read

6. Subject Alice transfers a right read/read on File 1 to P1*

➡ Alice must have a right read* on File 1

Only rights with a * are transferrable

	Alice	P1	File 1
Alice	control	control owner	read*
P1			read

7. Subject Alice deletes a right read/read on File 1 from P1*

➡ Alice must control P1 or Alice must own File 1

	Alice	P1	File 1
Alice	control	control owner	read*
P1			read

The Principle of Least Privilege

	Alice	File 1	File 2
Alice	control	owner read write	owner read write

- Processes run on user's behalf
  - No privilege separation
  - Alice's program would be able to write File 1
- How can *Alice* run a process *P1* that can only read *File 1*?

1. Subject Alice creates object File 1

2. Subject Alice creates subject P1

3. Subject Alice destroys object File 1
➡ Alice must own File 1

4. Subject Alice destroys subject P1
➡ Alice must own P1

5. Subject Alice grants a right read/read on File 1 to P1*
➡ Alice must be owner of File 1

6. Subject Alice transfers a right r/r on File 1 to P1*
➡ Alice must have a right read* on File 1

7. Subject Alice deletes a right r/r on File 1 from P1*
➡ Alice must control P1 or Alice must own File 1

The Principle of Least Privilege

- Starting state

	Alice	File 1	File 2
Alice	control	owner read write	owner read write

- Subject *Alice* creates subject *P1*

	Alice	P1	File 1	File 2
Alice	control	owner	owner read write	owner read write
P1		control		

- Subject *Alice* grants a right *read* on object *File 1* to subject *P1*

	Alice	P1	File 1	File 2
Alice	control	owner	owner read write	owner read write
P1		control	read	

The Principle of Least Privilege

1. *Subject Alice creates object File 1*

2. *Subject Alice creates subject P1*

3. *Subject Alice destroys object File 1*
   ➡ Alice must own File 1

4. *Subject Alice destroys subject P1*
   ➡ Alice must own P1

5. *Subject Alice grants a right read/read* on File 1 to P1*
   ➡ Alice must be owner of File 1

6. *Subject Alice transfers a right r/r* on File 1 to P1*
   ➡ Alice must have a right read* on File 1

7. *Subject Alice deletes a right r/r* on File 1 from P1*
   ➡ Alice must control P1 or Alice must own File 1

- Could *Alice* read *File 1*?
- Could *Bob* read *File 1*?

	Alice	Bob	File 1	File 2
Alice	control		owner	owner read write
Bob		control		

The question of safety

- The access control matrix implements a security policy
  - But DAC allows users to specify the access control policy
  - Given a specific starting state of the matrix and a given set of commands, can we prove any properties of all reachable states?
    - E.g. (Bob, Passwords File, Read) will never be granted

- Harrison-Ruzzo-Ullman model
  - Simple framework, with six commands to manipulate the matrix
  - Impossible to build a security argument for the general case
  - Safety can be checked for some models

[Harrison1976]

Recap: MAC vs DAC

- Two dual approaches
- In practice: combine both
  - Provide some form of discretionary self-management within the constraints of mandatory access rules
    - For example, delegate administration of team resources to an administrator
  - Options:
    - Enforce mandatory policy
    - Audit mandatory policy
    - Trust subjects to enforce mandatory policy

How permissions are assigned

Existing models

- Identity-based access control
- Multi-level access control
- Role-based access control (RBAC)
- Attribute-based access control (ABAC)

Identity-based access control

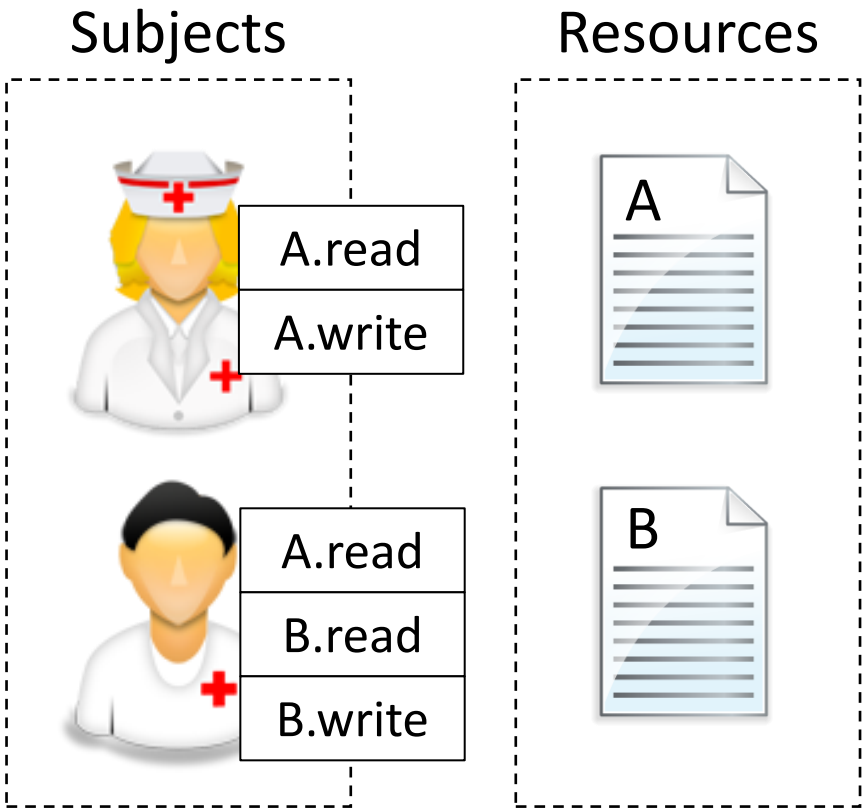
- Assign permissions to individual subjects and resources
  - This is actually again the Access Control Matrix

	File A	File B	File C
Alice	read	read	read
Bob	read, write		
Charlie		read, write	read, write

Identity-based access control

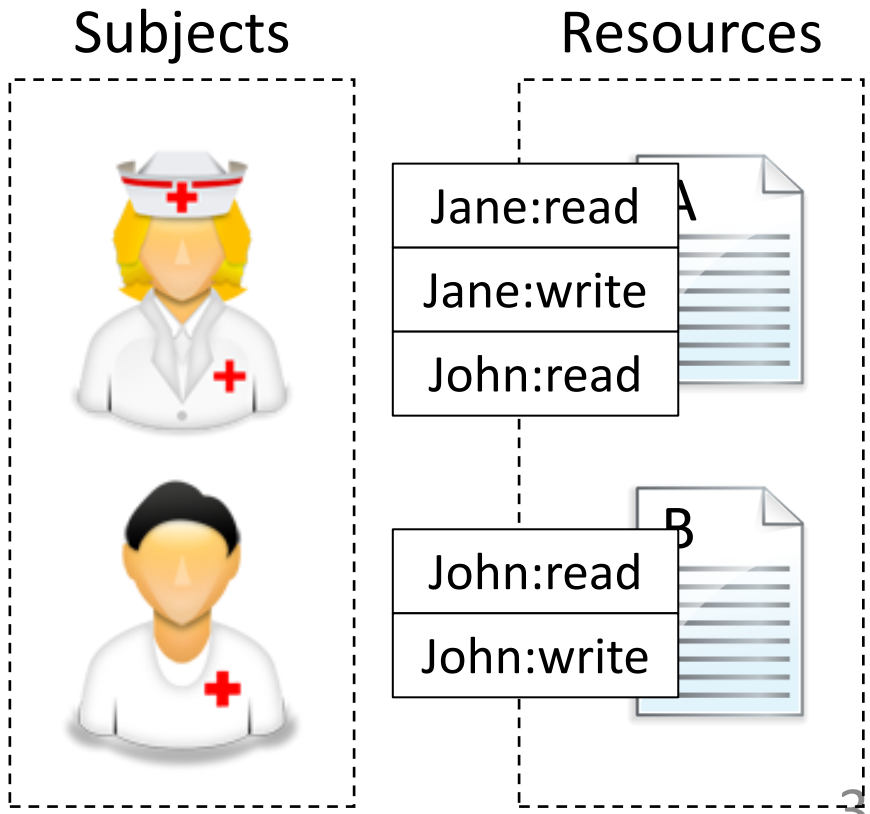
- Possible implementations: store 1 big matrix (not efficient) or:

Access Control Lists



	File A	File B
Jane	Read Write	
John	Read	Read Write

Capability Lists

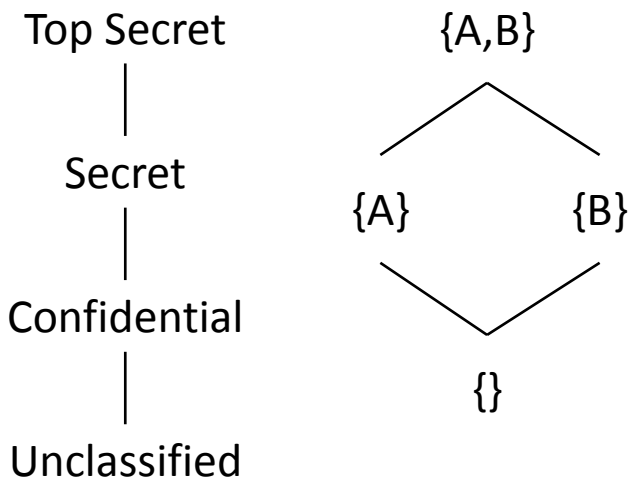


Identity-based access control

- Disadvantages:
  - Large management effort
    - E.g., “all nurses can read patient files” -> repeat for all nurses
    - E.g., “patients can read their own patient files” -> repeat for all patients
  - Susceptible to Trojans
    - Because programs run in name of a user
    - To address this: control access of code
    - Common model for this: multi-level access control

Multi-level access control

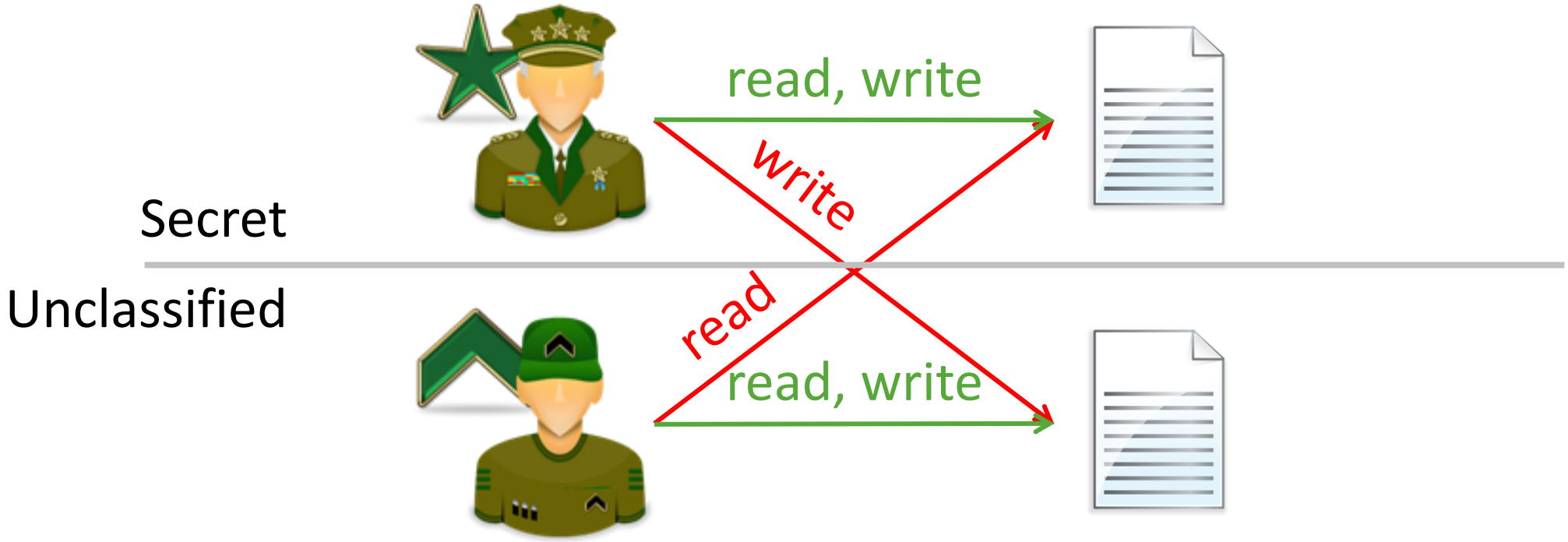
- Sometimes also called Lattice-Based Access Control
- Strict control over information flow
  - Resources are assigned *security classifications*
  - Subjects (and their programs) are assigned *security clearances*
  - Labels are organized in a lattice
- Two well-known rule sets:
  - Bell-LaPadula (confidentiality)
  - Biba (integrity)



Multi-level access control

- Model of Bell-LaPadula:
  - No read up
  - No write down (“*-property”)

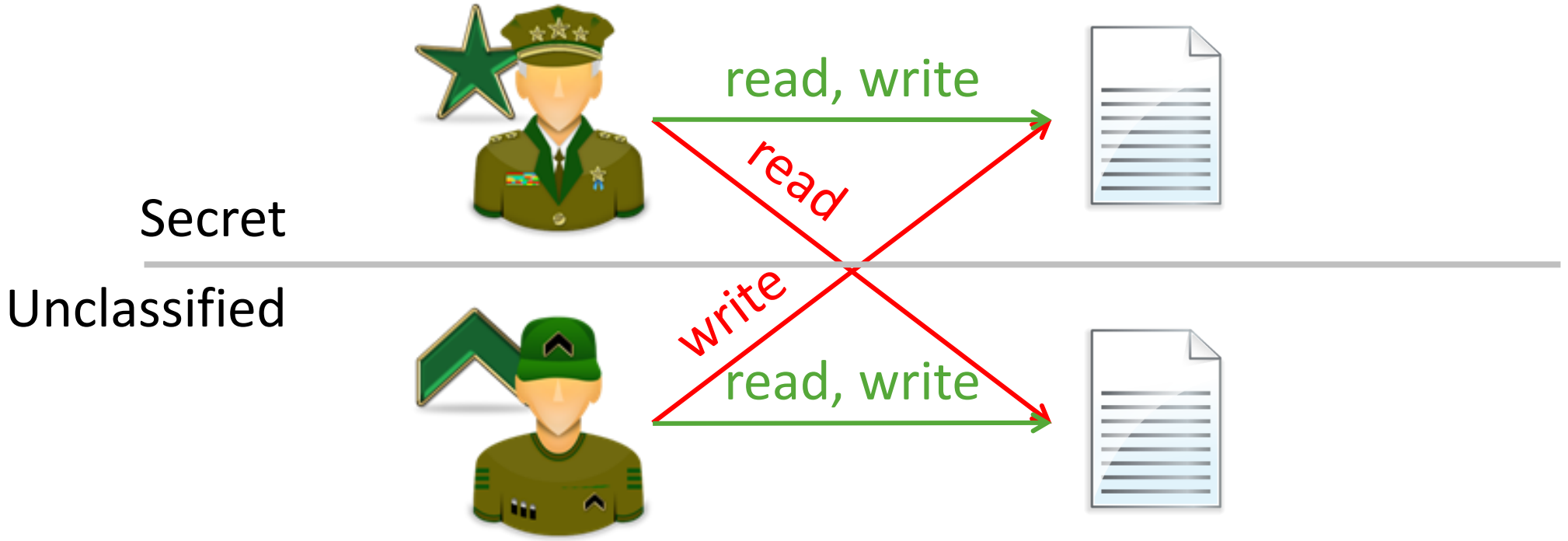
Confidentiality



Multi-level access control

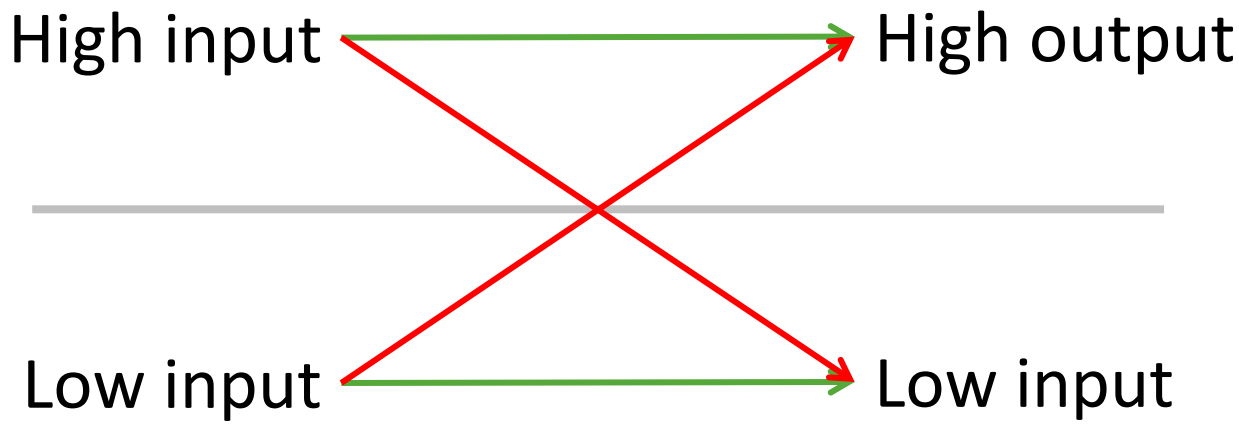
- Model of Biba:
  - No write up
  - No read down

  } Integrity



Multi-level access control

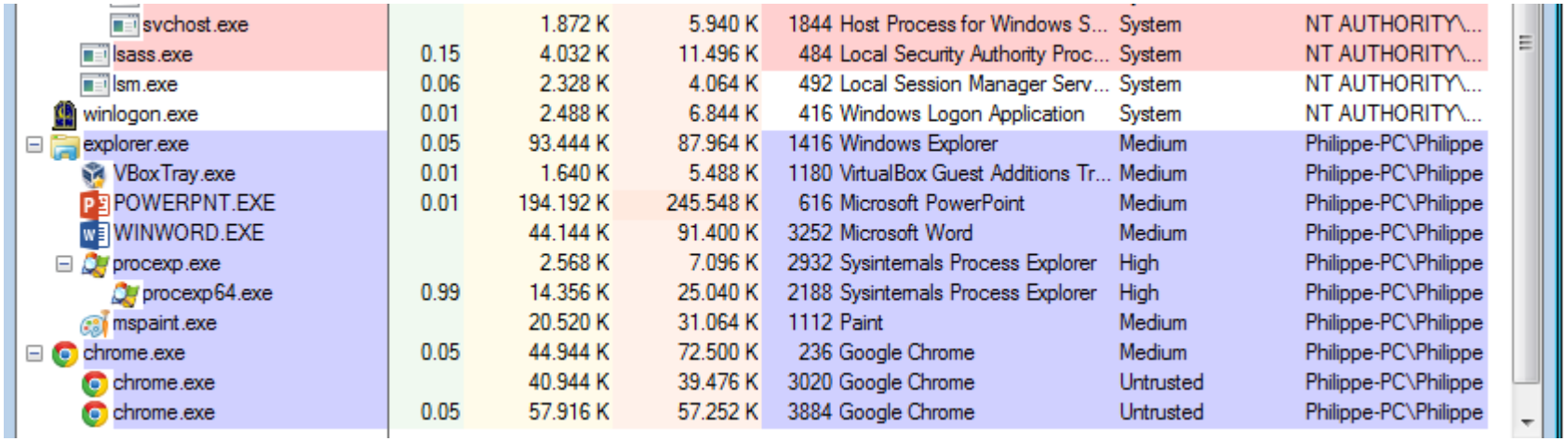
- You want both Bell-LaPadula and Biba
- However, this is not workable in practice
- => Refinement: Information flow control, taint tracking




```
var low, high
if check(high) then
  low := declassify(high)
```

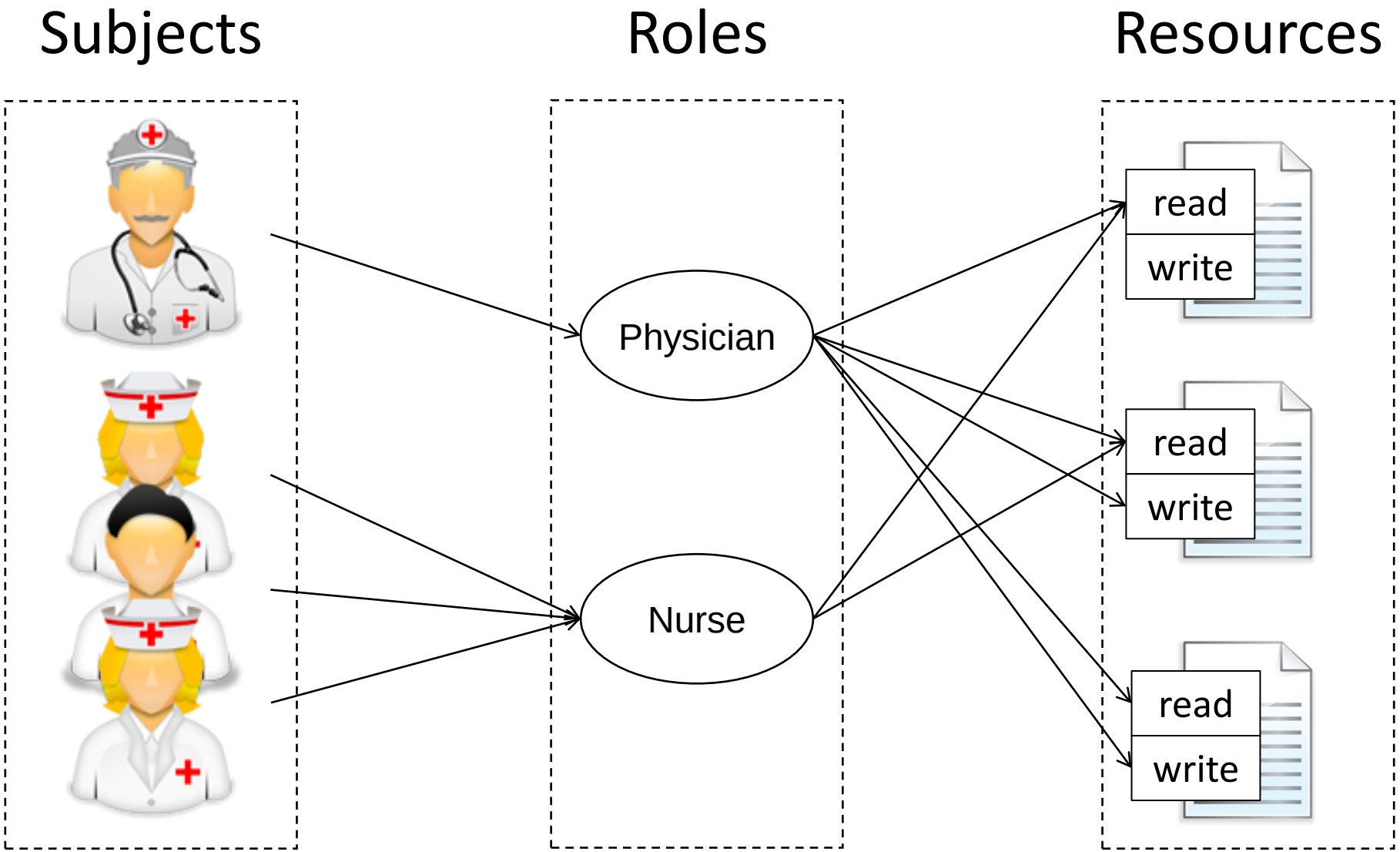
Multi-level access control in the wild

- Core security feature of Windows Vista and newer
  - Complementary to discretionary access control
  - Control access to securable objects based on integrity level
  - Define the minimum integrity level required to access an object
- Isolate potentially untrustworthy contexts within the OS
  - Used by Google Chrome and Adobe Reader



svchost.exe		1.872 K	5.940 K	1844 Host Process for Windows S...	System	NT AUTHORITY\...
lsass.exe	0.15	4.032 K	11.496 K	484 Local Security Authority Proc...	System	NT AUTHORITY\...
lsm.exe	0.06	2.328 K	4.064 K	492 Local Session Manager Serv...	System	NT AUTHORITY\...
winlogon.exe	0.01	2.488 K	6.844 K	416 Windows Logon Application	System	NT AUTHORITY\...
explorer.exe	0.05	93.444 K	87.964 K	1416 Windows Explorer	Medium	Philippe-PC\Philippe
VBoxTray.exe	0.01	1.640 K	5.488 K	1180 VirtualBox Guest Additions Tr...	Medium	Philippe-PC\Philippe
POWERPNT.EXE	0.01	194.192 K	245.548 K	616 Microsoft PowerPoint	Medium	Philippe-PC\Philippe
WINWORD.EXE		44.144 K	91.400 K	3252 Microsoft Word	Medium	Philippe-PC\Philippe
procexp.exe		2.568 K	7.096 K	2932 Sysinternals Process Explorer	High	Philippe-PC\Philippe
procexp64.exe	0.99	14.356 K	25.040 K	2188 Sysinternals Process Explorer	High	Philippe-PC\Philippe
mspaint.exe		20.520 K	31.064 K	1112 Paint	Medium	Philippe-PC\Philippe
chrome.exe	0.05	44.944 K	72.500 K	236 Google Chrome	Medium	Philippe-PC\Philippe
chrome.exe		40.944 K	39.476 K	3020 Google Chrome	Untrusted	Philippe-PC\Philippe
chrome.exe	0.05	57.916 K	57.252 K	3884 Google Chrome	Untrusted	Philippe-PC\Philippe

Role-based access control (RBAC)

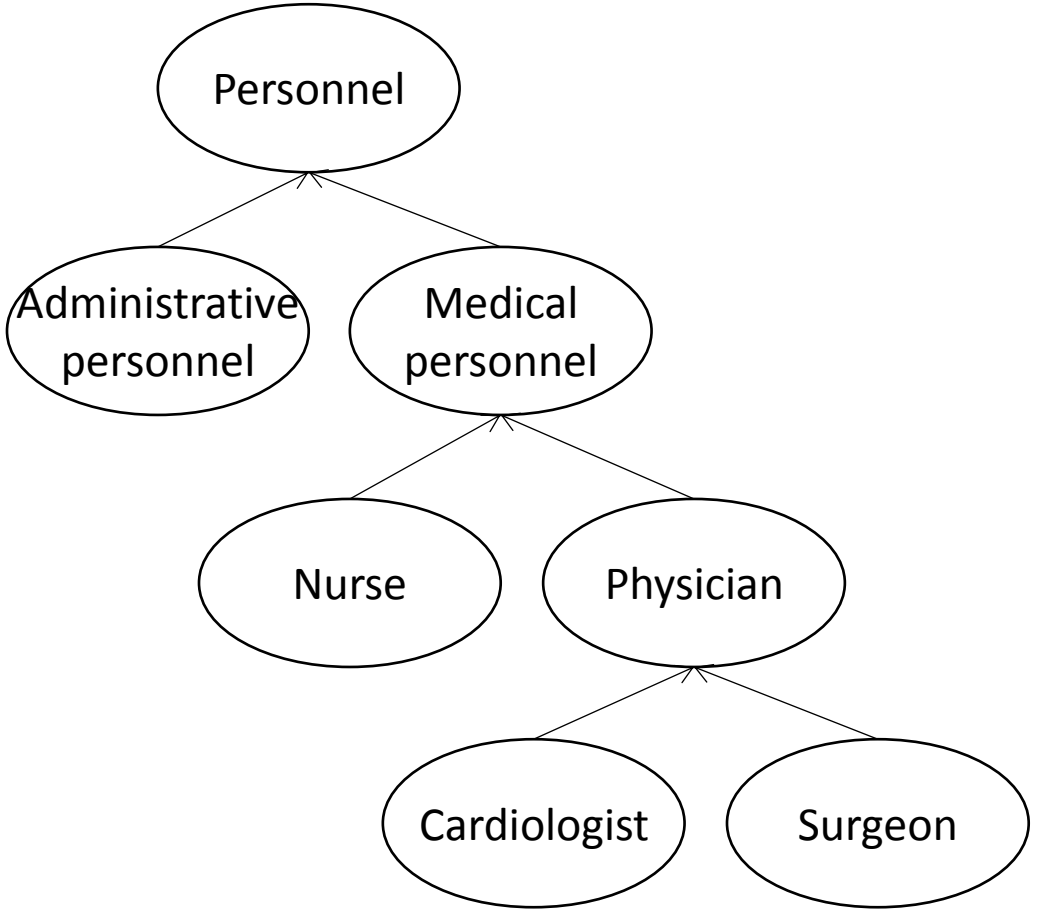


Role-based access control (RBAC)

- Permissions assigned to roles, roles adopted by users
  - Goal: reduce large number of permissions to limited number of roles
  - Fits well onto the organizational structure of an enterprise
- Originated in research in 1992, NIST standard in 2004
- Immense research field
  - Role mining, administrative models, delegation, constraints, ...

Role-based access control (RBAC)

- Additional features in the NIST standard:
  - Role hierarchies
  - Least privilege through sessions
  - Static separation of duty through meta-rules
  - …



RBAC in the wild

- Database systems often use and support RBAC
  - E.g. Oracle Enterprise Server
- Application development frameworks
  - Apache Shiro, Spring Security, …
  - E.g., Java Spring Security:

```
@PreAuthorize("hasRole('manager')")
public void create(Contact contact);


@PreAuthorize("hasPermission('delete_contact')")
public void deleteContact(Contact contact);
```

Role-based access control (RBAC)

- Major disadvantage: role explosion
- Reasons:
  - Roles cannot express ownership and time
    - Requires roles like “owns_docA”, “owns_docB”, etc
  - Reality is too fine-grained
    - Often small differences between different persons *in the same job*, leading to yet another role (e.g., “secretary_with_colorprint”)
  - Cross-product of multiple hierarchies
    - E.g., “sales_manager_for_belgium_with_colorprint_owns_docA”
- To address this:
  - In practice: pragmatic choice for RBAC + ownership

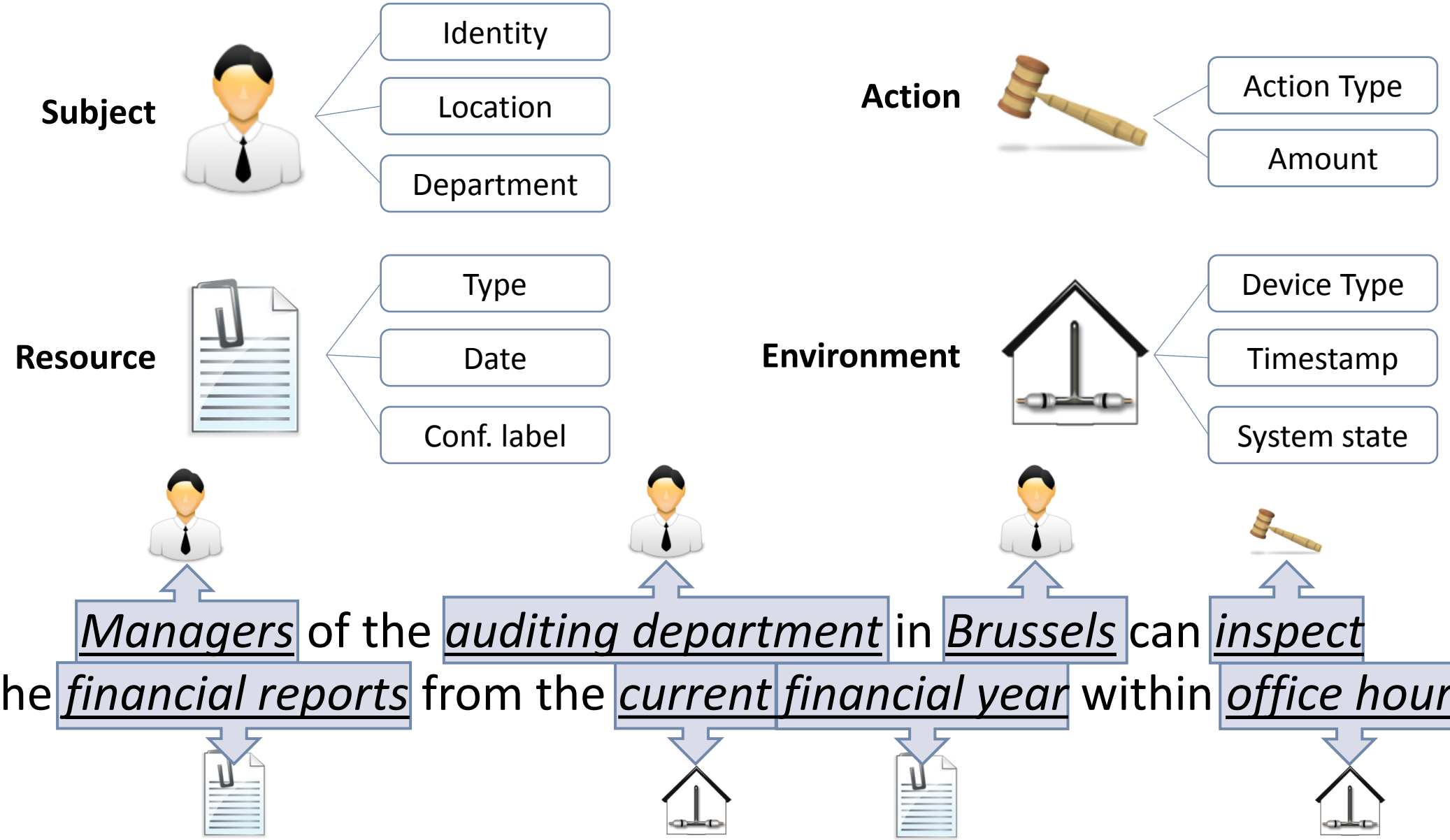
Role-based access control (RBAC)

- Major disadvantage: role explosion
- Reasons:
  - Roles cannot express ov
    - Requires roles like "own
  - Reality is too fine-graine
    - Often small differences
      to yet another role (e.g.
  - Cross-product of multip
    - E.g., "sales_manager_fo

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
Comment			
Administer comments and comment settings	☐	☐	☑
View comments	☐	☑	☑
Post comments	☐	☑	☑
Skip comment approval	☐	☑	☑
Edit own comments	☐	☑	☑

- To address this:
  - In practice: pragmatic choice for RBAC + ownership
  - In research: large number of extensions proposed

Attribute-based Access Control (ABAC)



Attribute-based Access Control (ABAC)

permit if

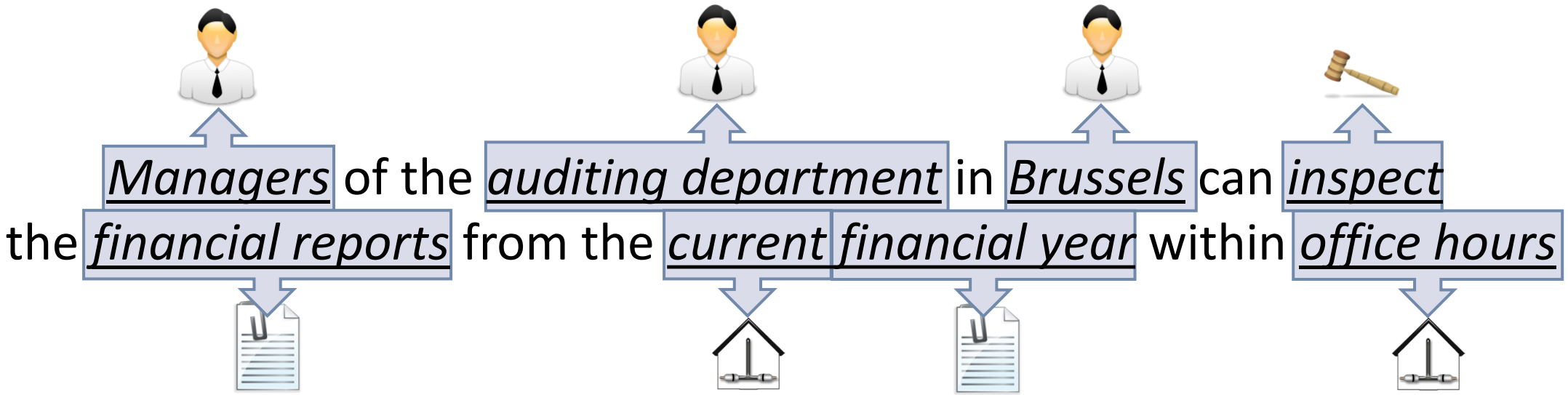
"manager" in subject.roles and subject.department == "auditing"

and subject.location == "Brussels" and action == "inspect"

and resource.type == "financial report"

and resource.year == environment.current_year

and 8h00 < environment.time < 17h00



Attribute-based Access Control (ABAC)

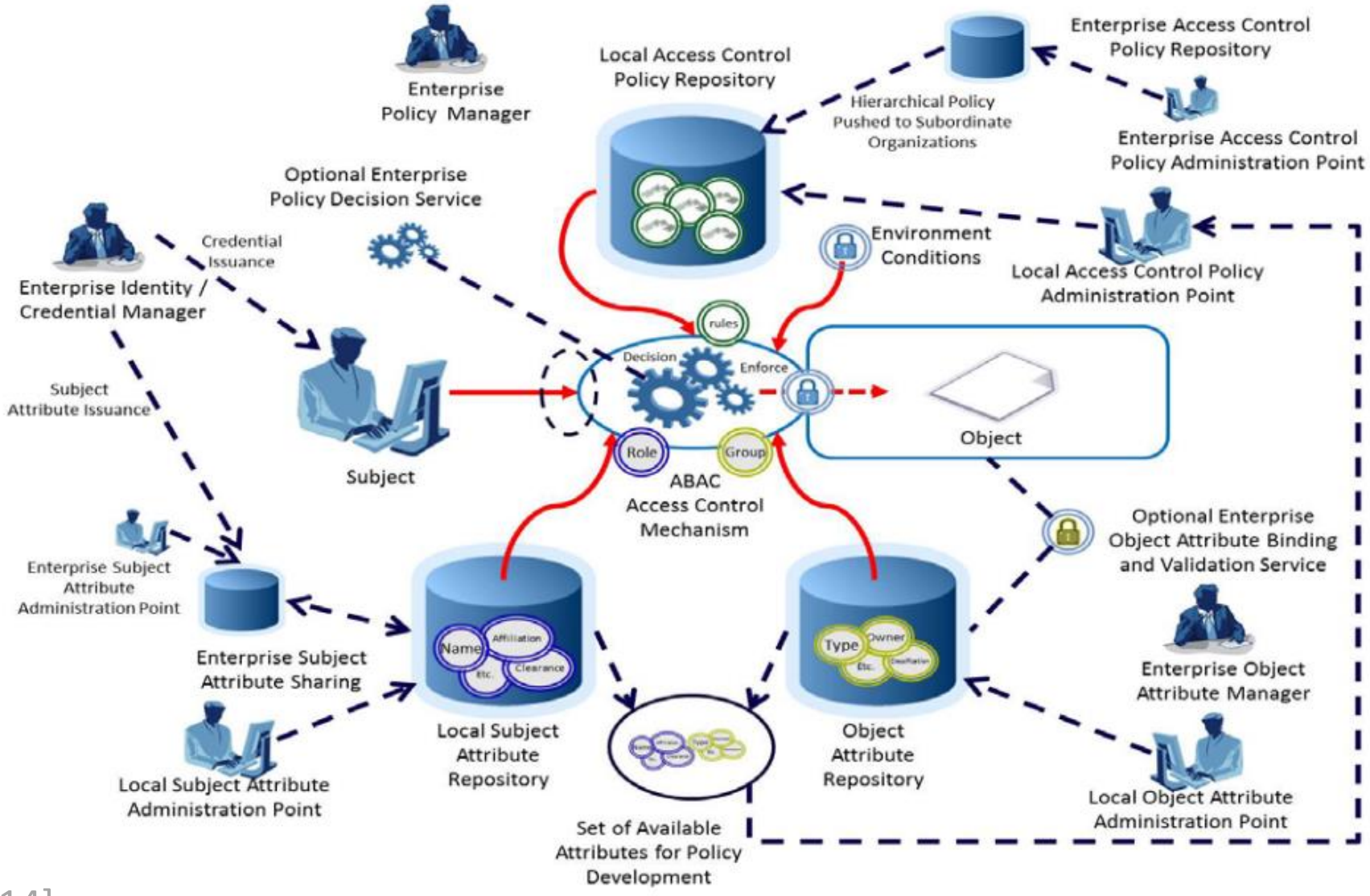
- Access decisions are made based on attributes
  - Attributes are key-value properties of the subject, the resource, the action or the environment
  - Results into dynamic and context-aware access control
- Attributes can express many different access control concepts
  - Permissions, roles, groups, departments, time, location, ownership, domain-specific ownership, ...

Migrating from RBAC to ABAC

In general, three approaches:

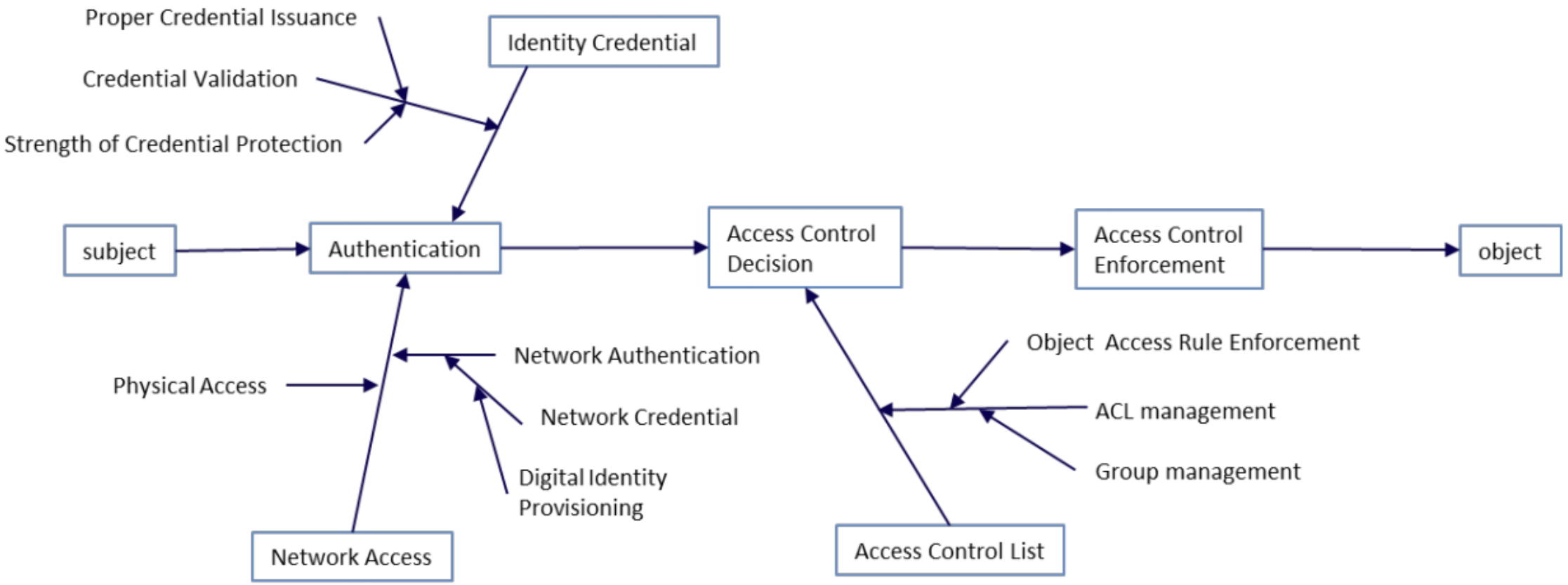
1. Dynamic roles
   - Determine a subject's roles based on attributes, e.g., time
   - Advantage: backwards compatible with existing systems
2. Attribute-centric
   - A role is just one of many attributes
   - Advantage: can reuse existing roles in attribute-based rules
3. Role-centric
   - Use attributes to constrain roles, i.e., reduce permissions of a role
   - Essentially an extension to RBAC

ABAC is more than expressing rules



Source: [NIST2014]

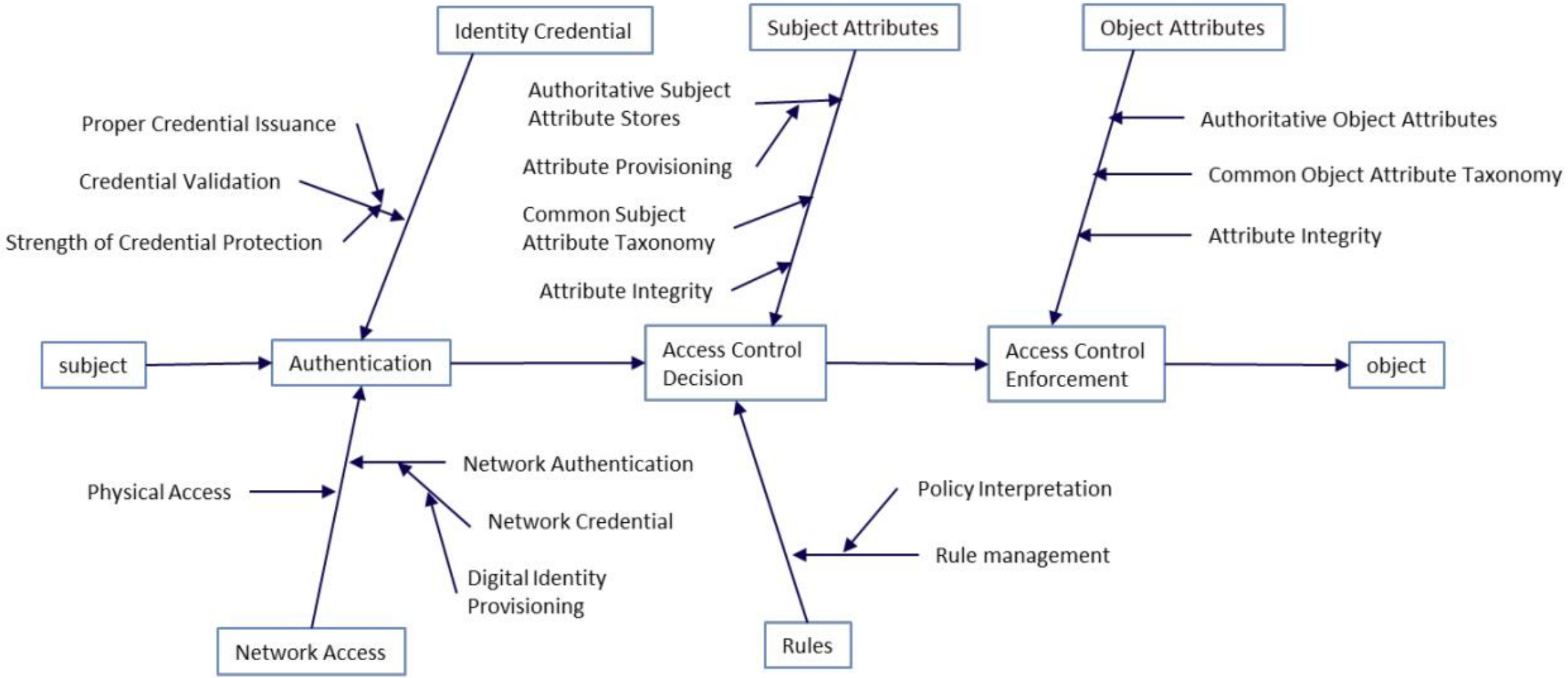
Not all rainbows and unicorns



Trust chain for Access Control Lists

Source: [NIST2014]

Not all rainbows and unicorns



Trust chain for ABAC

Source: [NIST2014]

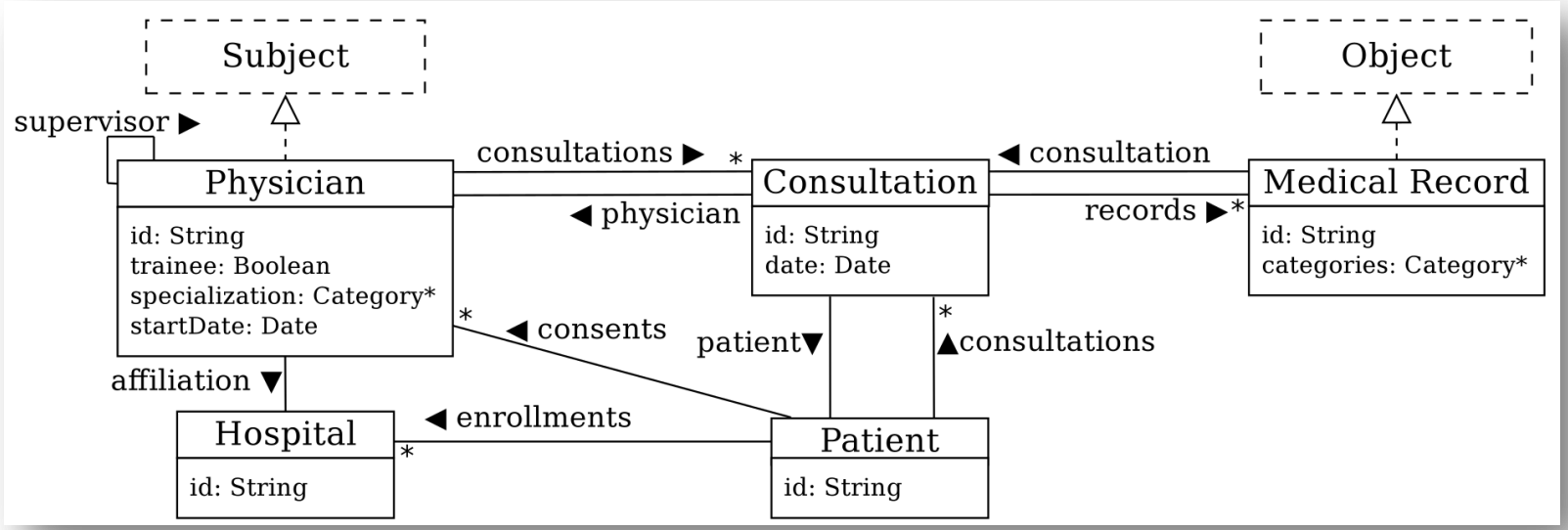
ABAC: Conclusion

- ABAC brings many interesting improvements compared to previous models
- ABAC is seen by many as the next step in access control
- => Definitely something you should consider, but not a small step to take
- Further reading: [NIST2014]
  - Overview of ABAC, challenges and enterprise considerations

Advanced topics

Advanced topics

- Relationship-Based Access Control
  - Originated from social networks
  - Further reading: [Cheng2012, Fong2011]
- Entity-Based Access Control
  - Express access rules in terms of the entities in your application
    - Attributes + relationships
  - Fixes limitations of ABAC
  - I expect a lot of this, but still a long way to go
  - Further reading:
    - [Crampton2014]
    - [Bogaerts2015]



Advanced topics

- Advanced policy pattern: breaking the glass
  - Enable users to override a deny by “breaking the glass”
  - Common pattern in e-health
    - “A physician should be able to override a deny when a patient is in critical condition”
  - More general application: “The End of Default Deny” (Gartner)
  - Challenge: *controlled* override
    - Limit who can override a deny (e.g., only physicians of emergency department), limit for which actions a deny can be overridden (e.g., only for reads)
    - Audit these overrides later on, e.g., by writing out logs at override

Advanced topics

- Advanced policy pattern: separation of duty
  - Separate duties within an organization
  - Statically:
    - E.g., “a manager can never also be a secretary”
    - E.g., “a manager cannot approve his own funding requests”
  - Dynamically:
    - E.g., “if a user has had access to documents of Bank A, he or she is not allowed to access documents of Bank B”
    - Originally described in 1989 as the “Chinese wall policy”, a “commercial security policy” in contrast to “Bell-LaPadula-style policies” [Brewer1989]
  - Very relevant because of Sarbanes-Oxley, but still a hard problem
    - Hard to apply to an organization
    - Hard to implement well (performance issues)

Advanced topics

- History-based access control
  - E.g., dynamic separation of duty
  - E.g., limit the number of accesses
    - “a user cannot watch more than 10 movies per month”
- Implementation options:
  - Use log files in the policy evaluation
  - Use provenance data in the policy evaluation [Nguyen2012, Nguyen2013]
  - Explicitly update history attributes [Decat2015]

Advanced topics

- History-based access control
  - E.g., dyna
  - E.g., limit
    - "a user
- Implemen
  - Use log files in the policy evaluation
  - Use provenance data in the policy evaluation [Nguyen2012, Nguyen2013]
  - Explicitly update history attributes [Decat2015]

When resource.owner == "Bank B",
apply DenyOverrides **to**

Obligations

Deny if
"Bank A" **in** subject.history

Permit performing
append("Bank B", subject.history)

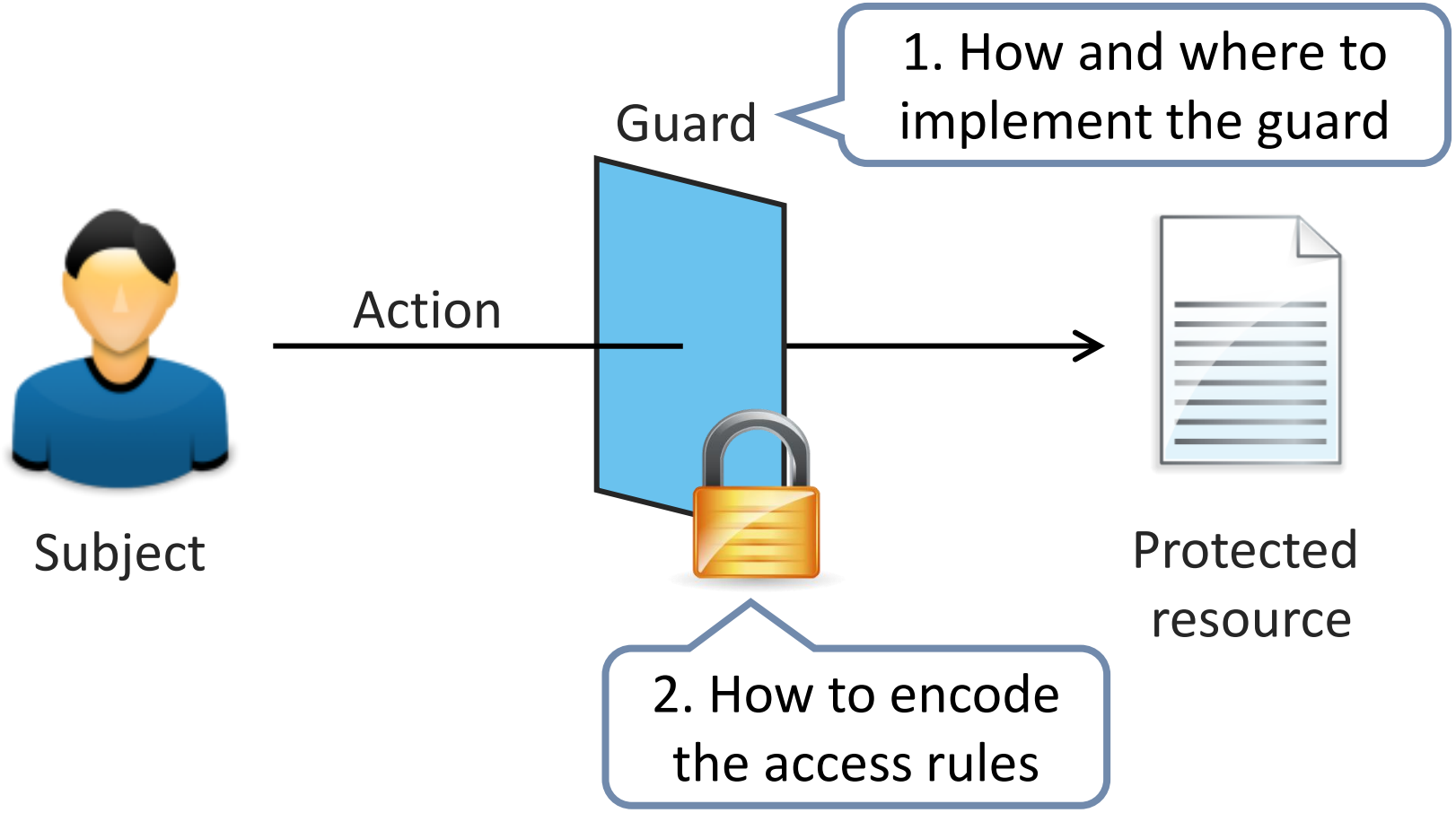
Advanced topics

- Obligations
  - Early definition: “predicates that verify mandatory requirements a subject has to perform before or during a usage exercise” [Park2004]
    - Pre-obligations, ongoing-obligations
    - Examples:
      - User has to agree to terms and conditions (pre)
      - User has to be shown an ad during watching the requested movie (ongoing)
  - More pragmatic definition: action that should be performed with permitting/denying the action
    - Send an e-mail to an administrator on deny to a confidential document
    - Write out log
    - Update attribute

Outline

- Introduction
- Positioning access control
- Access control models
- How to enforce access control
  - Reference monitors
  - Access control in application code
    - Policy-based access control
- The bigger picture
- Some important technologies in practice
- Recap and conclusion

How to enforce access control

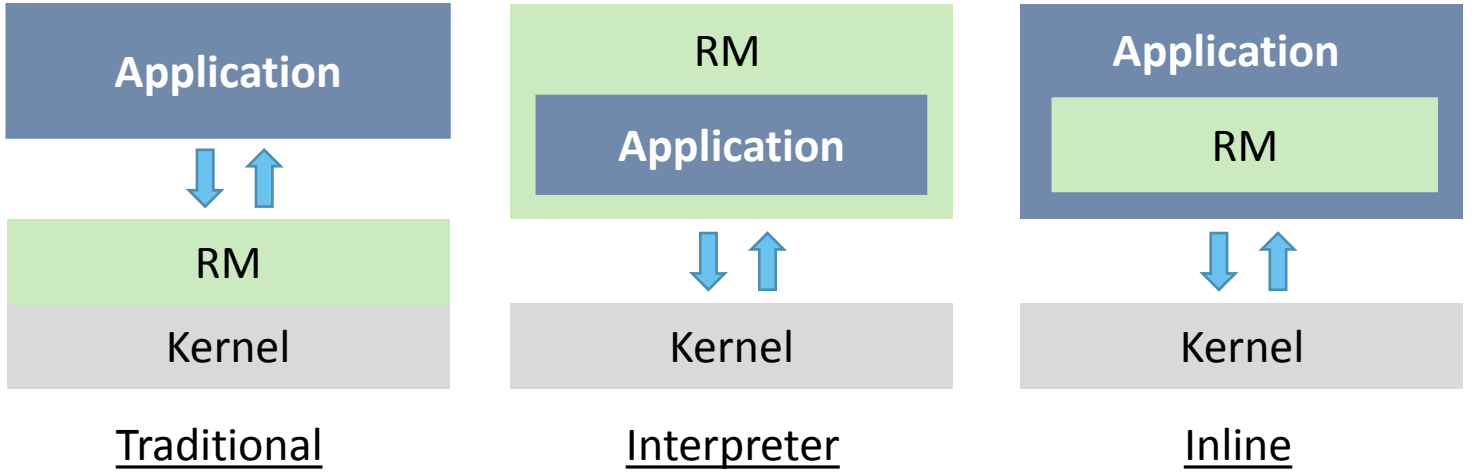


Access control exists on multiple levels

Level	Subject	Action	Guard	Protected System
Hardware	OS Process	Read memory	CPU	CPU and Memory
Network	Host	Send packets	Firewall	Intranet
Database	User	SELECT query	DBMS	User database
OS	User	Open file	OS Kernel	Filesystem
OS	Java Program	Open file	Java Security Manager	Filesystem
Application	User	Read patient file	Application code	Application data

Reference monitors

- Reference monitors
  - Observe software execution
  - Take remedial action on operations that violate a policy
- Three important security properties
  - Full mediation
  - Tamper proof
  - Verifiable

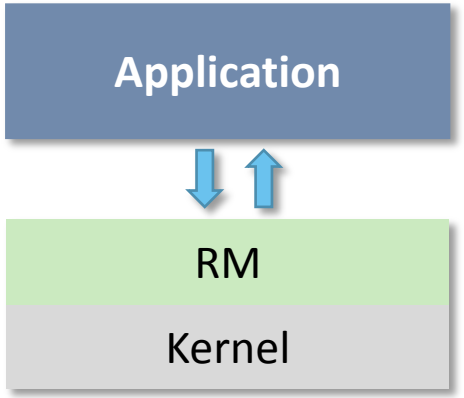


[Erlingsson2004]

Example of a reference monitor

- Antivirus software is implemented as reference monitor
  - Hooks into the OS's system calls to intercept application actions
  - E.g. inspects file contents upon read or write operations

- Good implementation strategy to meet security properties
  - Full mediation: requires coverage of all system calls
  - Tamper proof: requires strong process isolation
  - Verifiable: less straightforward, but possible



Example of a reference monitor

AVG, McAfee, Kaspersky Fix Common Vulnerability in Their Antivirus Products

By Catalin Cimpanu 9 Dec 2015, 22:50 GMT

AVG, Kaspersky, McAfee antiviruses fix securit...

The security vulnerability allowed attackers to compromise Windows computer...

A common security bug affected the antivirus engines of three major vendors, AVG, McAfee, and Kaspersky, as enSilo security researchers have discovered.

The problem was first detected back in March 2015, when one of enSilo's own products collided with an AVG antivirus on one of its client's workstations. After further investigation into the matter, enSilo's staff uncovered a security bug in the AVG antivirus as being the cause of the software incompatibility.

http://news.softpedia.com/news/avg-mcafee-kaspersky-fix-common-vulnerability-in-their-antivirus-products-497395.shtml

Application-level access control

- Reference monitors:
  - Constrain untrusted *code*
  - Can be applied to a program without having to modify it
  - Can only reason in terms of interface operations, e.g., system calls
- Application-level access control:
  - Rules reason about the concepts in your application
  - Add guard to code of your application
  - The same holds:
    - Full mediation
    - Tamper proof
    - Verifiable

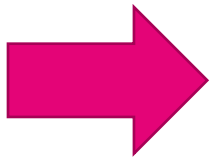
Option 1: encode guard and rules in app code

```
public  Document getDoc(docId) {
  Doc doc = db.getDoc(docId);
  if (! (“manager” in user.roles
        and doc.owner == user
        and 8h00 < now() < 17h00 )) {
    return null;
  } else {
    return doc;
  }
}
```

- \+ straightforward
- \+ you can encode almost anything
- \- no separation of concerns
- \- no modularity
  => hard for reviews
- \- what if rules change?
  - \- update application code
  - \- updates all over the place

Option 2: modularize

```
public  Document getDoc(docId) {
  Doc doc = db.getDoc(docId);
  if (! (“manager” in user.roles
        and doc.owner == user
        and 8h00 < now() < 17h00 )) {
    return null;
  } else {
    return doc;
  }
}
```



```
@authz(user, “read”, result)
public  Document getDoc(docId) {
  return db.getDoc(docId);
}
…
public boolean authz(
     subject, action, resource) {
  if (! (“manager” in user.roles and …)) {
    return true;
  } else {
    return false;
  }}
```

Option 2: modularize

+ more modularity: access control logic in 1 place
- no separation of concerns
± what if rules change?
  - update application code
  + updates in one place

```
@authz(user, “read”, result)
public Document getDoc(docId) {
  return db.getDoc(docId);
}
…
public boolean authz(
    subject, action, resource) {
  if (! (“manager” in user.roles and …)) {
    return true;
  } else {
    return false;
  }}
```

Option 2: modularize - Django

settings.py:

```
AUTHENTICATION_BACKENDS = [
   'mymodule.MyBackend'
]
```

mymodule/backends.py:

```
class MyBackend(object):

   ...
   def has_perm(self, user, perm, obj):
      if obj.owner == user.id:
         return True
      else:
         return False
```

https://docs.djangoproject.com/en/1.9/topics/auth/customizing

Option 2: modularize – Ruby on Rails

In the controller:

```
def show
  @article = Article.find(params[:id])
  authorize! :read, @article
end
```

In the view:

```
<% if can? :update, @article %>
  <%= link_to "Edit",
      edit_article_path(@article) %>
<% end %>
```

The access control code:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    if user.admin?
      can :manage, :all
    else
      can :read, :all
    end
  end
end
```

https://github.com/ryanb/cancan

Option 2: modularize – Java Spring Security

In the controller:

```
@PreAuthorize("hasPermission(#doc, 'view')")
public void getDocument(Document doc);
```

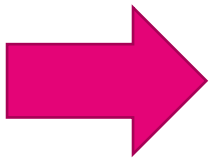
In the `PermissionEvaluator`:

```
boolean hasPermission(Authentication a,
                      Object resource, String permission) {
  User user = SecurityUtil.getUserCredential();
  if(permission == "view" and ...) {
    return true;
  } else {
    return false;
  }
}
```

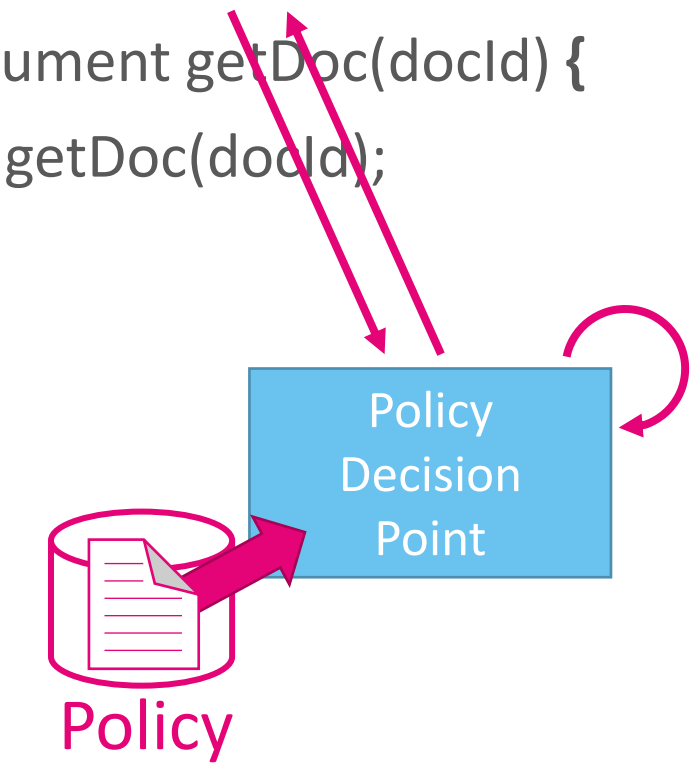
https://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html

Option 3: policy-based access control

```
@authz(user, “read”, result)
public  Document getDoc(docId) {
  return db.getDoc(docId);
}
…
public boolean authz(
     subject, action, resource) {
  if (! (“manager” in user.roles and …)) {
    return true;
  } else {
    return false;
  }}
```



```
@authz(user, “read”, result)
public  Document getDoc(docId) {
  return db.getDoc(docId);
}
```



Option 3: policy-based access control

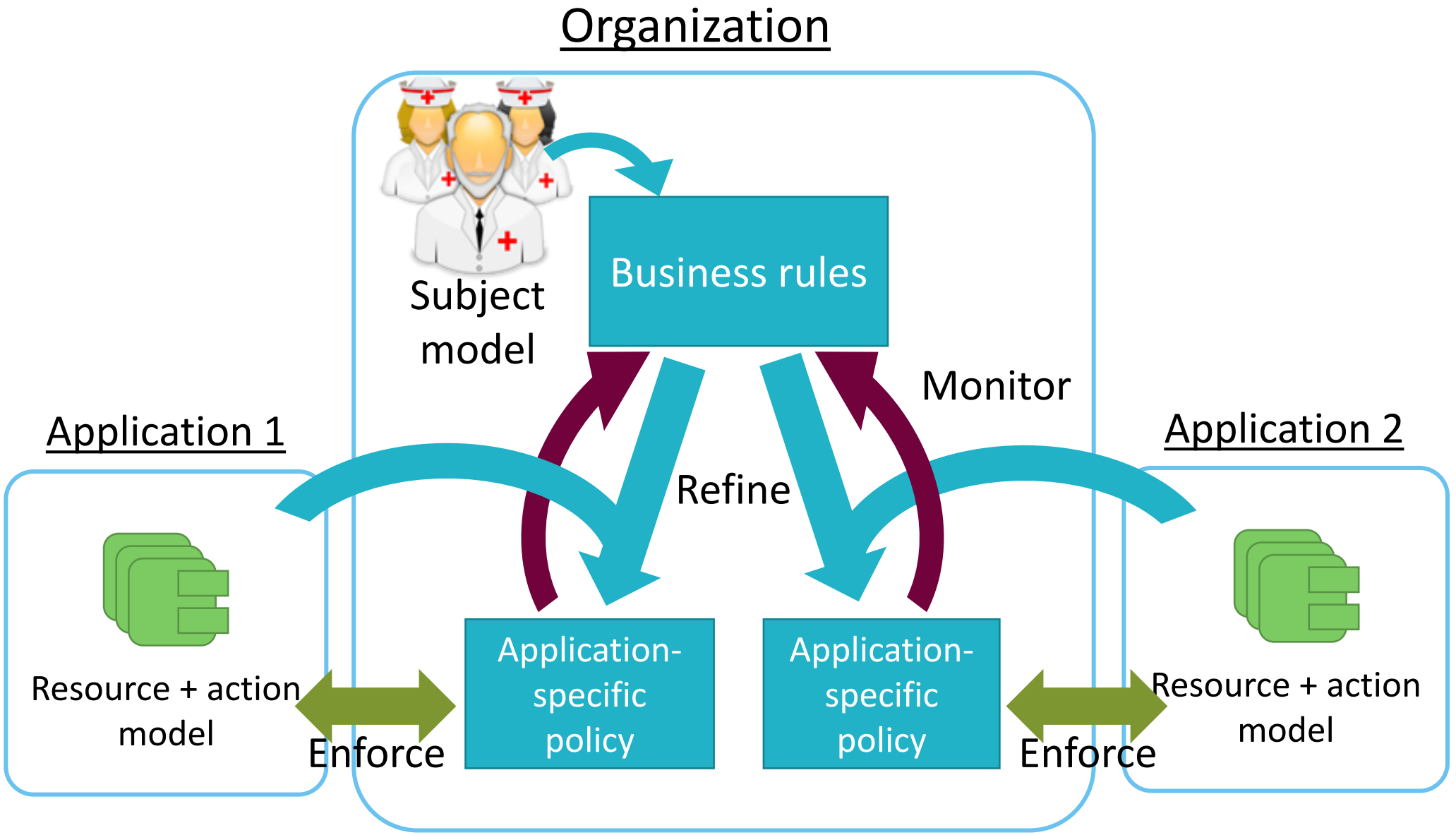
- Decouple access control rules from application code
  - Express access control rules in a format independent of your programming language
  - In application code: ask the generic question “can this subject perform this action on this resource”?
  - Policy evaluated by specialized component called the Policy Decision Point
  - If policy is stored in a file or a database: change policy at run-time

Advantages of PBAC

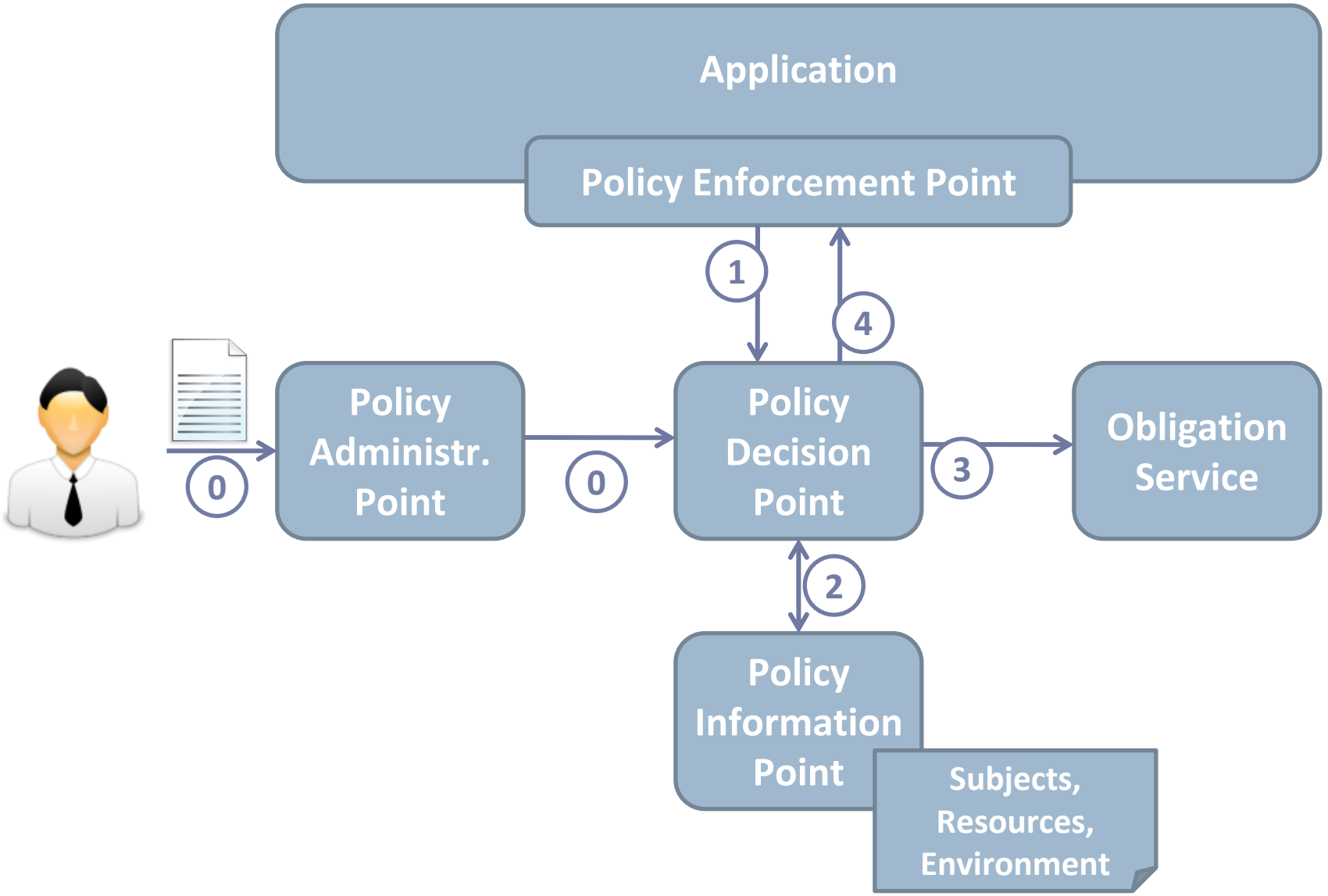
+ More modularity: access control logic in 1 place
+ Separation of concerns: policies can be written by non-developer
+ What if rules change?
  + no updates in application code
  + updates in a single place

  => enables highly-verified fixed policy engine and evolving access rules (though your rules should also be regarded as part of the TCB)
+ Enables your access control policies to easily evolve with your organization
+ Enables centralizing policies, explicitly managing policies across your organization, refining business policies, …

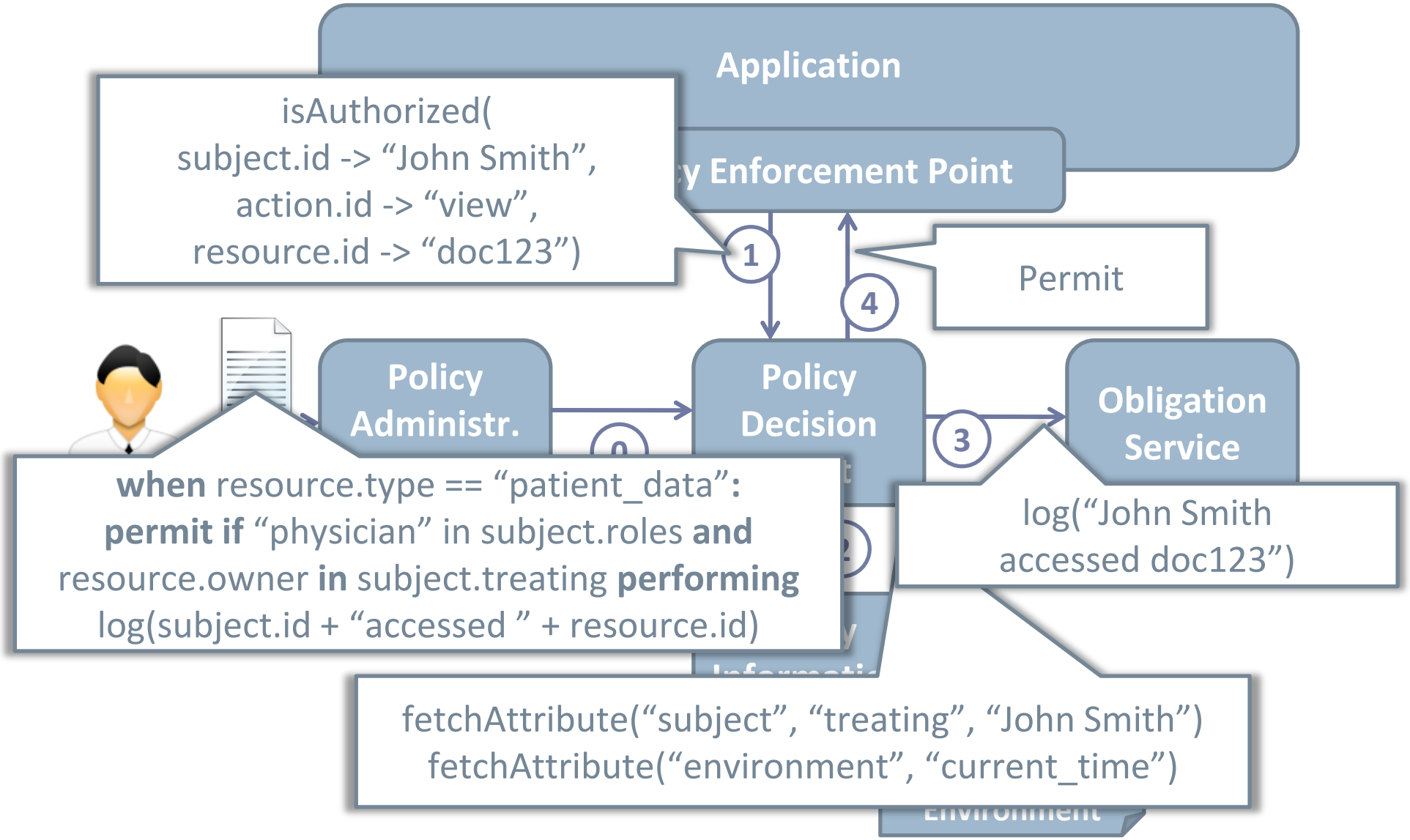
Vision


Organization
Subject model
Business rules
Monitor
Refine
Application 1
Application 2
Resource + action model
Application-specific policy
Application-specific policy
Enforce
Enforce
Resource + action model

XACML Reference architecture



XACML Reference architecture

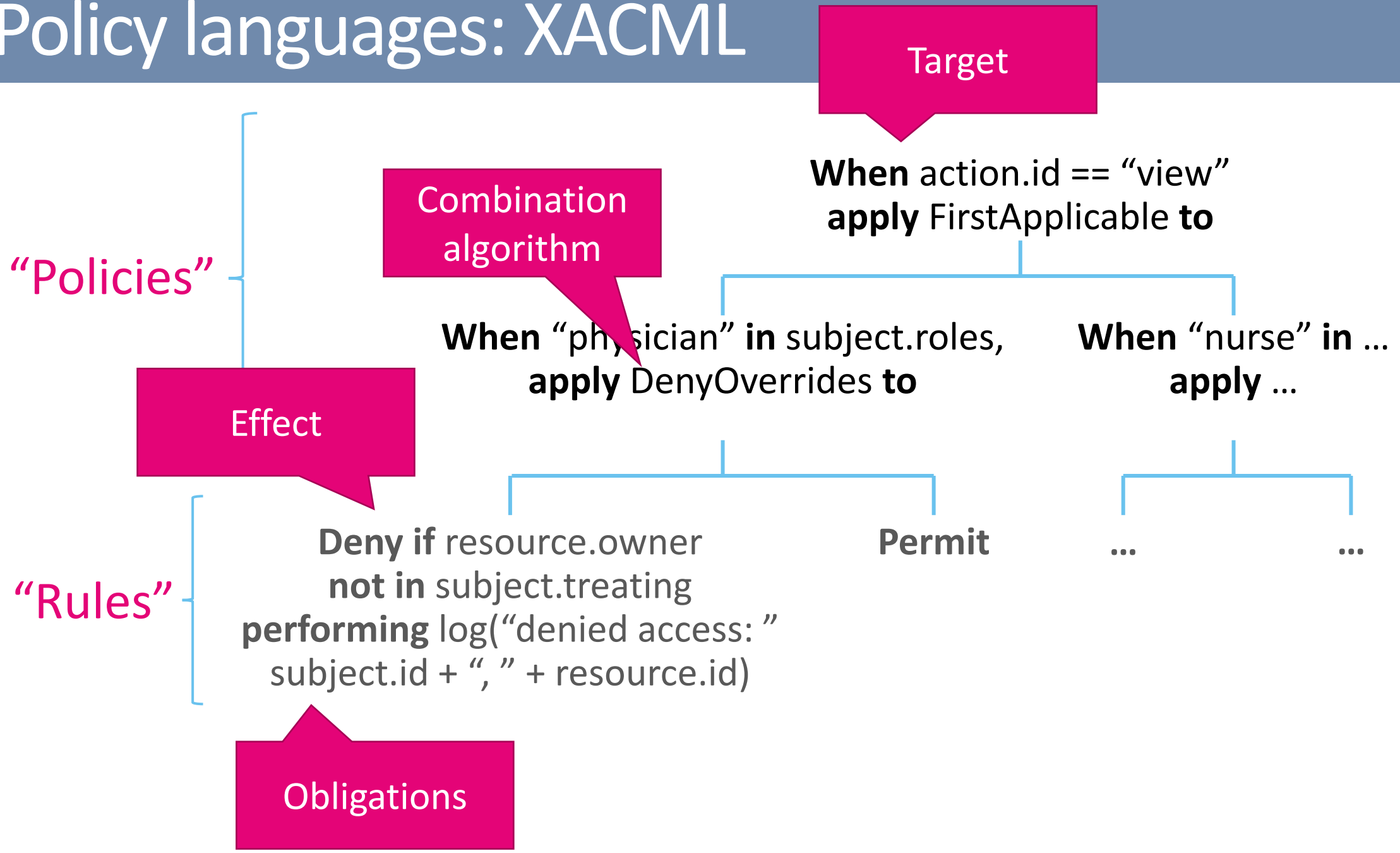


Policy languages

- A large number of domain-specific policy languages proposed in literature
  - E.g., SPL, Ponder, XACML, Cassandra, SecPAL, …
- Current major standard: XACML
  - Attribute-based, tree-structured, obligations
  - XML format
  - Standardized by OASIS
    - v1.0 ratified in 2003, v3.0 in 2013
  - Vendors: Axiomatics, WSO2, Oracle

http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

Policy languages: XACML



Policy languages: XACML

```
<Policy PolicyId="dynamic-separation-of-duty"
        RuleCombiningAlgId="deny-overrides">
  <Description>Dynamic separation of duty</Description>
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="string-equal">
          <AttributeValue DataType="string">doc123</AttributeValue>
          <ResourceAttributeDesignator AttributeId="resource:id" DataType="string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <Rule RuleId="deny" Effect="Deny">
    <Description>Deny if viewed other doc</Description>
    <Condition>
      <Apply FunctionId="string-is-in">
        <AttributeValue DataType="string">doc456</AttributeValue>
        <SubjectAttributeDesignator AttributeId="subject:history" DataType="string"/>
      </Apply>
    </Condition>
  </Rule>
  <Rule RuleId="default-permit" Effect="Permit"> </Rule>
  <Obligations>
    <Obligation ObligationId="append-attribute" FulfillOn="Permit">
      <AttributeAssignment AttributeId="value" DataType="string">
        <SubjectAttributeDesignator AttributeId="resource:id" DataType="string"/>
      </AttributeAssignment>
      <AttributeAssignment AttributeId="attribute-id"
DataType="string">subject:history</AttributeAssignment>
    </Obligation>
  </Obligations>
</Policy>
```

STAPL

```
Rule("roles") := permit iff (“physician" in subject.roles)
```

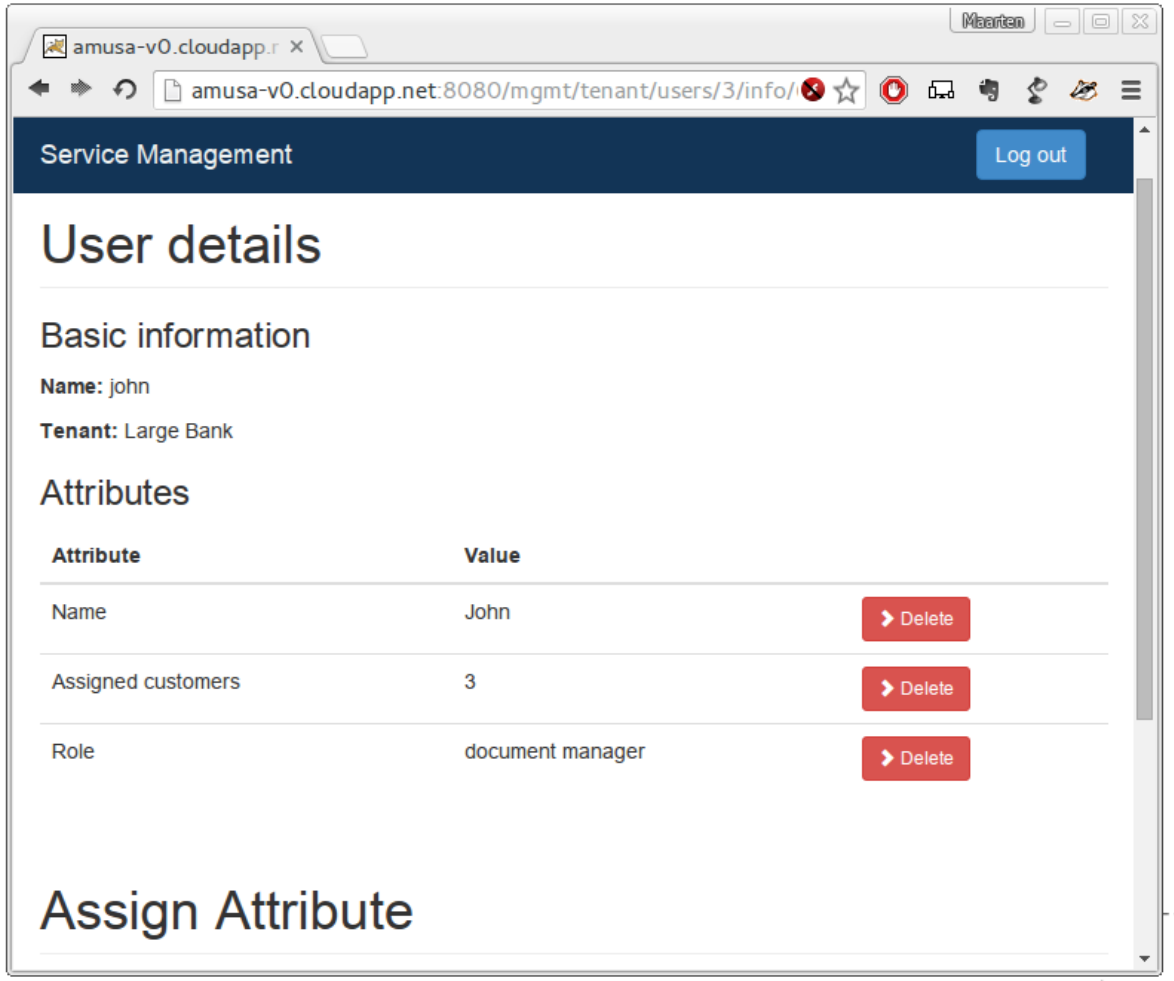
```
Rule(“ownership") := permit iff (resource.owner in subject.treating)
```

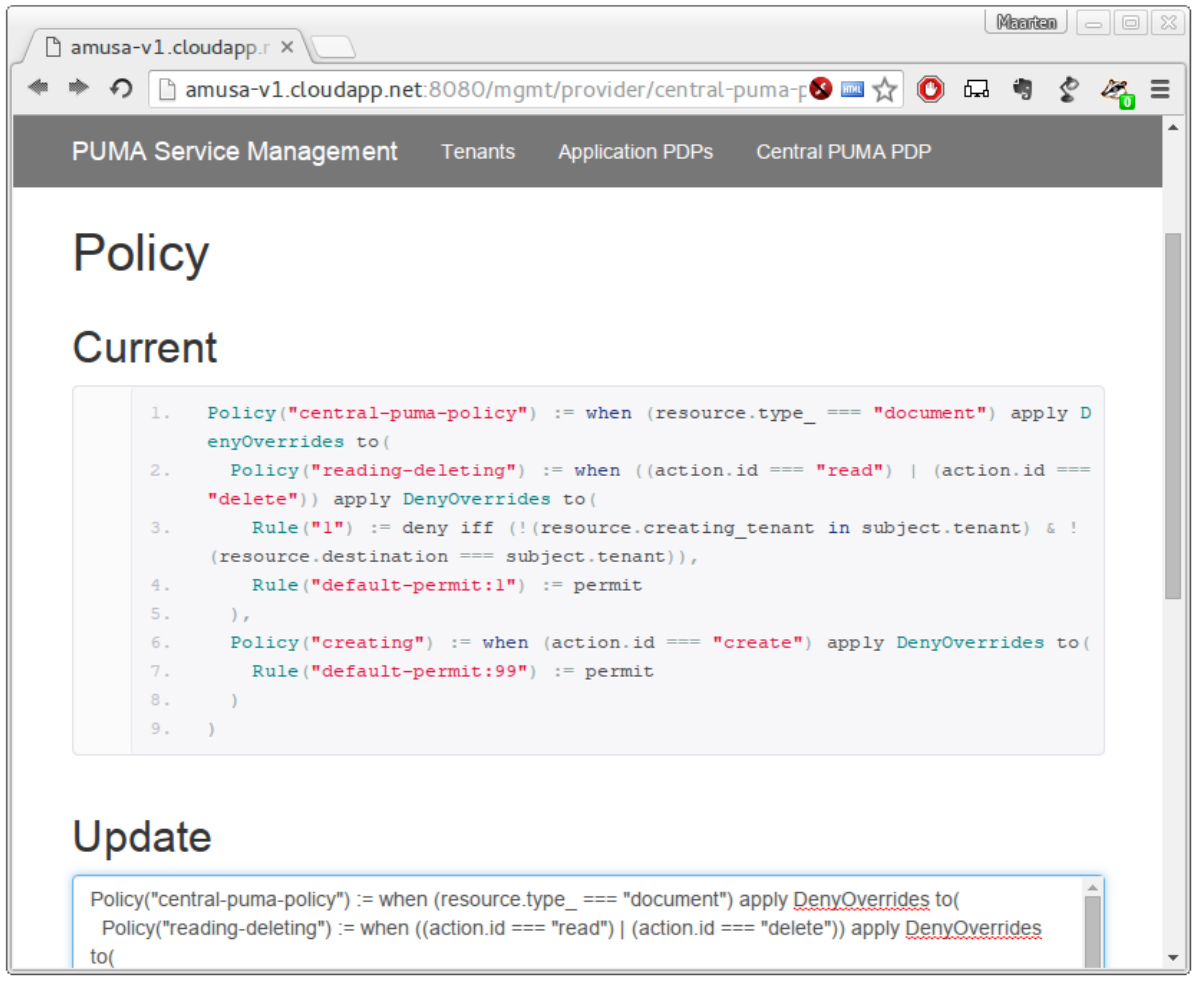
```
Rule(“time") := deny iff (env.currentDateTime > (resource.created + 5.days))
```

```
Policy(“dynamic SoD") := when (resource.id === "doc123") apply DenyOverrides to (
    Rule("deny") := deny iff ("doc456" in subject.history),
    defaultPermit
) performing (append(resource.id, subject.history) on Permit)
```

https://github.com/stapl-dsl/

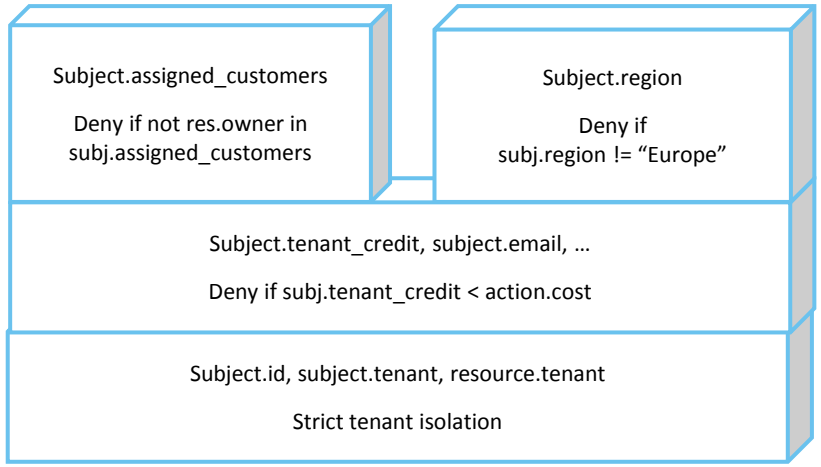
PBAC in research: Amusa



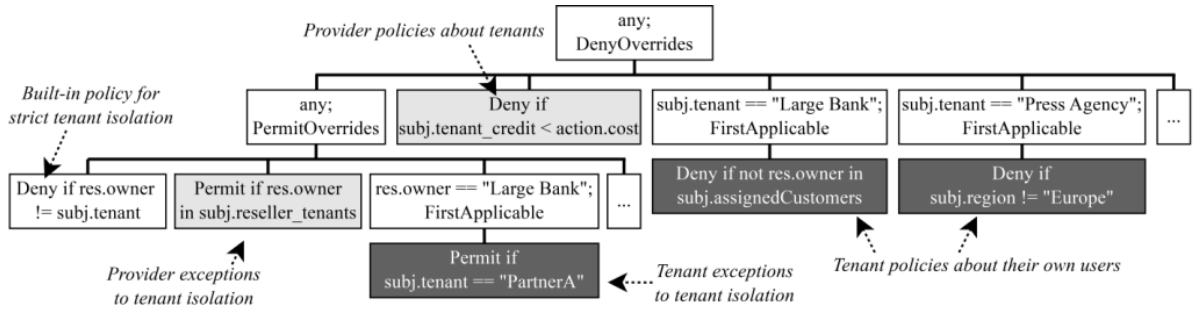


[Decat2015b]

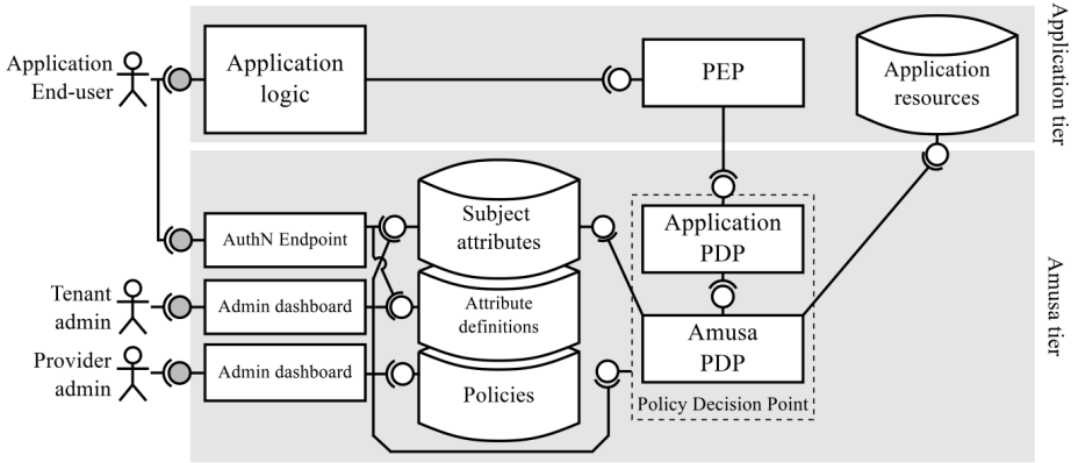
PBAC in research: Amusa



1. Three-layered mgmt



2. Secure policy combination



3. Supporting architecture

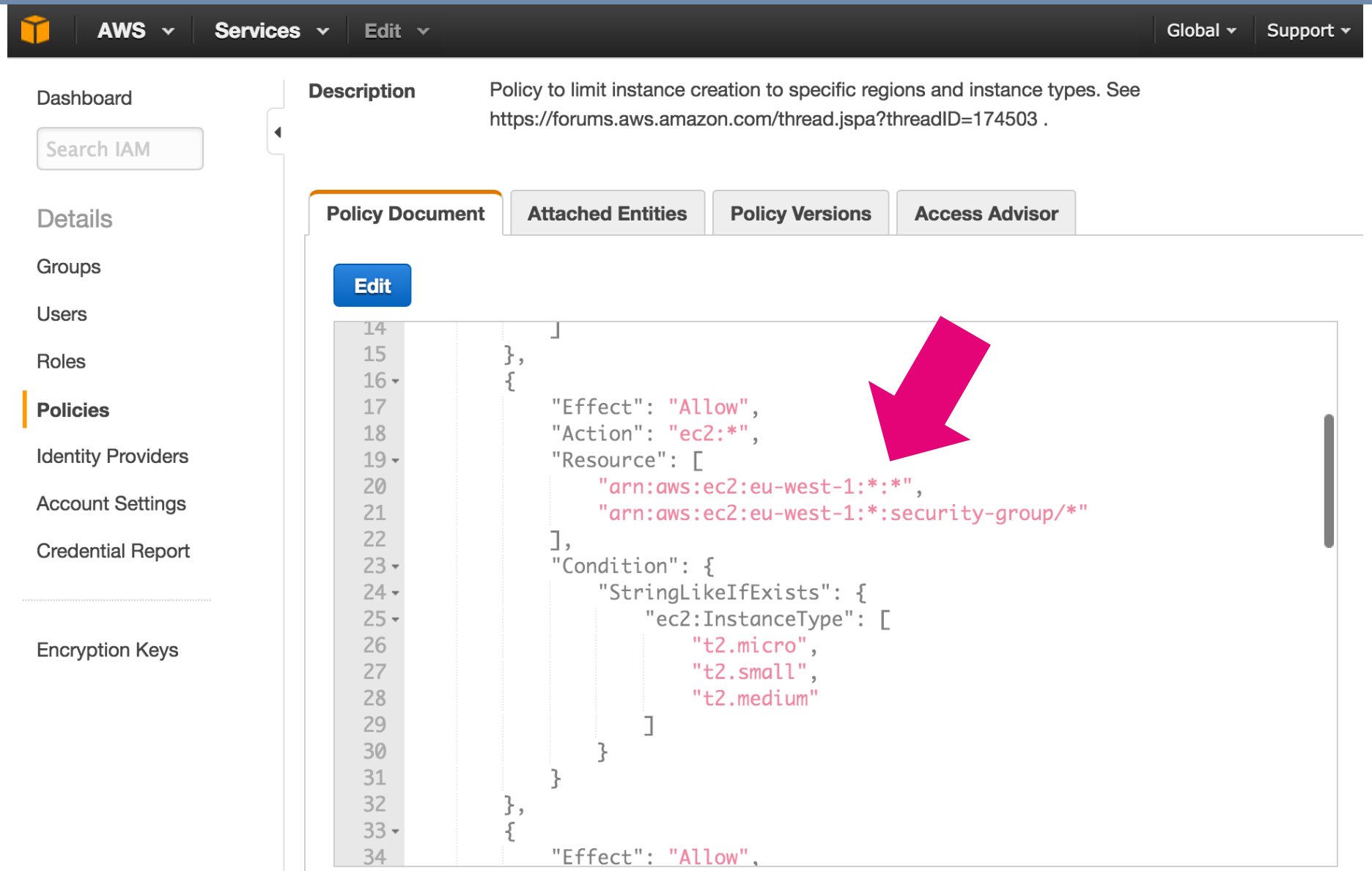
```
// an application method
public Document viewDoc(Document doc, HttpSession session) {
  Subject s = session.getSubject();
  Resource r = new Resource(doc.getId())
    .addAttribute("type", "document")
    .addAttribute("tenant", doc.getOwner());
  Action a = new Action("view");
  if(! pep.isAuthorized(s,r,a)) { return; }
  ... // application logic
}
```

Listing 3.1: Example usage of the basic authorization API in Java.

4. Low-effort API

[Decat2015b]

PBAC in the wild: Amazon EC2



http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html

PBAC in the wild: Amazon EC2

Policy Simulator

Amazon EC2 | 193 Action(s) se... | Select All | Deselect All | Reset Contexts | Clear Results | Run Simulation

▸ Global Settings ⓘ

Action Settings and Results [193 actions selected. 0 actions not simulated. 63 actions allowed. 130 actions denied.]

	Service	Action	Resource Type	Simulation Resource	Permission
▸	Amazon EC2	AcceptVpcPeeringConne...	vpc-peering-conn...	*	denied Implicitly denied (no matc...
▸	Amazon EC2	ActivateLicense	not required	*	denied Implicitly denied (no matc...
▸	Amazon EC2	AllocateAddress	not required	*	allowed 1 matching statements.
▸	Amazon EC2	AssignPrivateIpAddresses	not required	*	denied Implicitly denied (no matc...
▸	Amazon EC2	AssociateAddress	not required	*	allowed 1 matching statements.
▸	Amazon EC2	AssociateDhcpOptions	not required	*	denied Implicitly denied (no matc...
▸	Amazon EC2	AssociateRouteTable	not required	*	denied Implicitly denied (no matc...
▸	Amazon EC2	AttachClassicLinkVpc	instance,security-...	*	denied Implicitly denied (no matc...
▸	Amazon EC2	AttachInternetGateway	not required	*	denied Implicitly denied (no matc...
▸	Amazon EC2	AttachNetworkInterface	not required	*	denied Implicitly denied (no matc...
▸	Amazon EC2	AttachVolume	instance,volume	*	denied Implicitly denied (no matc...

Advantages of PBAC

- More modularity: access control logic in 1 place
- Separation of concerns: policies can be written by non-developer
- What if rules cl
  - no update
  - updates in
- Enables your access control policies to easily evolve with your organization
- Enables centralizing policies, explicitly managing policies across your organization, refining business policies, …

Not all rainbows and unicorns

- Very interesting technology, great vision to work towards
- But, externalizing authorization logic from an application is just very hard:
  - Different way of coding
  - Policy languages are not self-explanatory
  - Requires processes for managing policies within your organization
  - Requires supporting tools such as editors and correctness tests
  - Requires interoperability if you want to centralize authorization for multiple applications
  - Your trusted computing base and trust chains grow significantly
  - …
  - Plus, from my research experience: inherently hard to decouple authorization logic from an application because these rules should still say something about *this* application

PBAC: Conclusions

- PBAC:
  - A lot is expected of this technology
  - Enables exciting new stuff
  - But imho currently still too hard to apply in practice

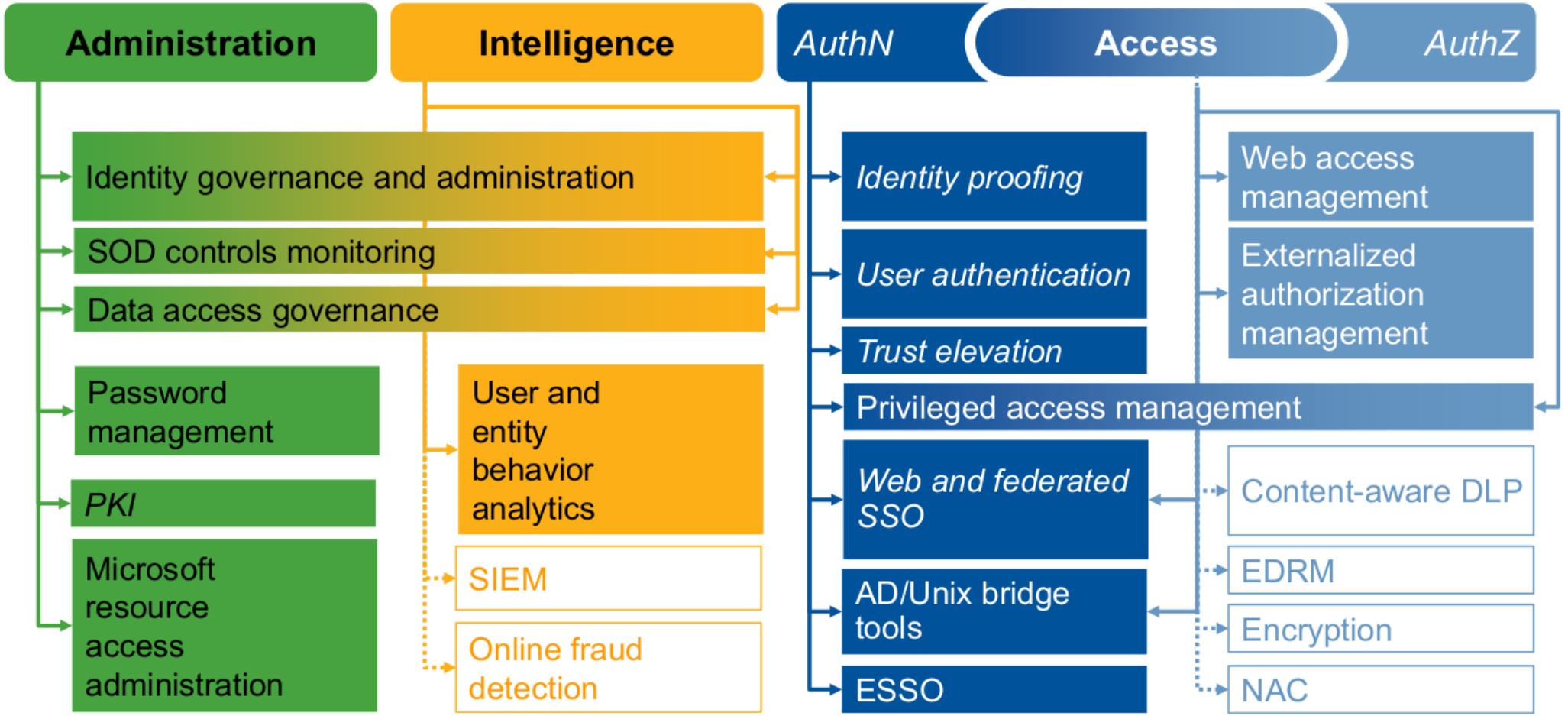
- My recommendation for now:
  - Modularize authorization in your application code (option 2)
    - Provides benefits by itself + future-proof

Outline

- Introduction
- Positioning access control
- Access control models
- How to enforce access control
- The bigger picture
- Some important technologies in practice
- Recap and conclusion

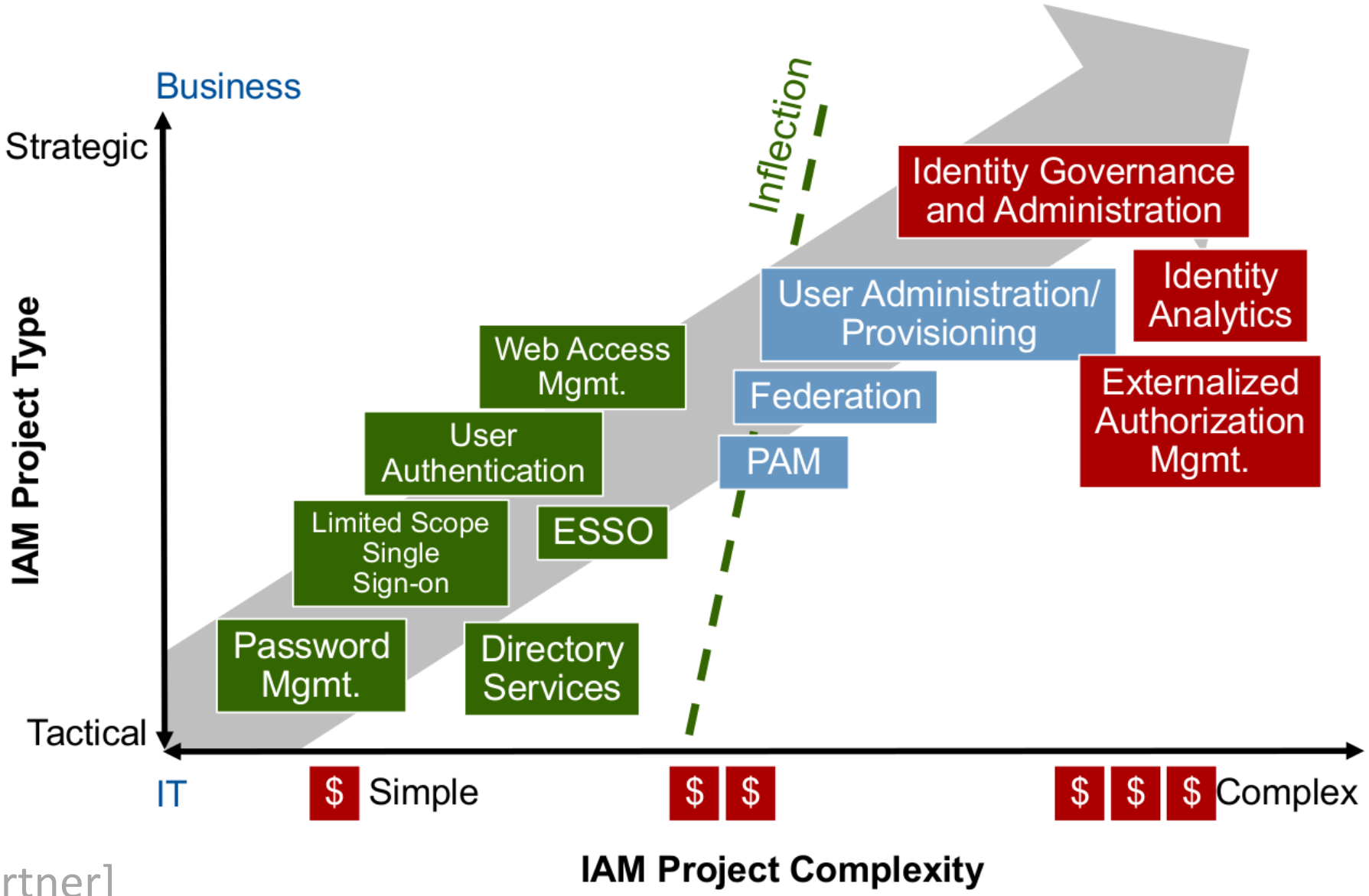
The bigger picture: IAM

IAM Technologies by Primary Function



[Source: Gartner]

The bigger picture: IAM

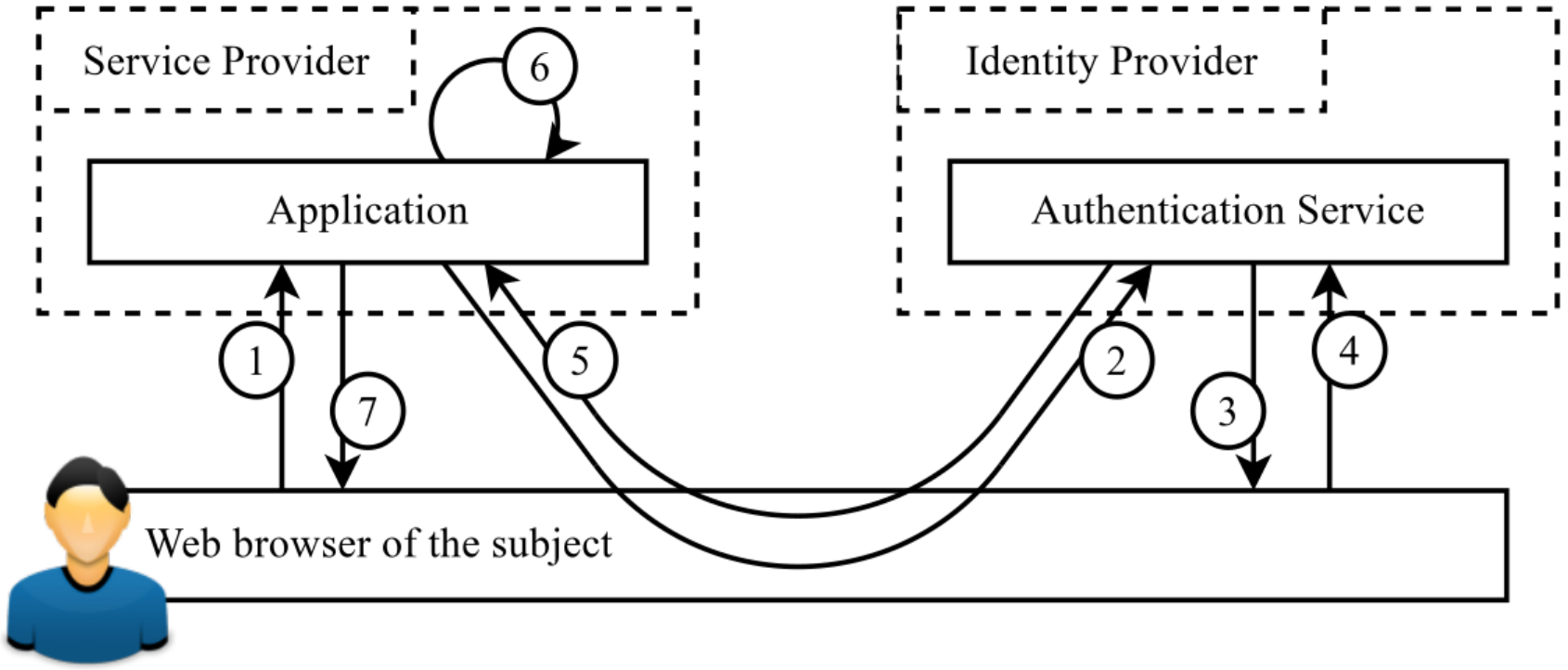


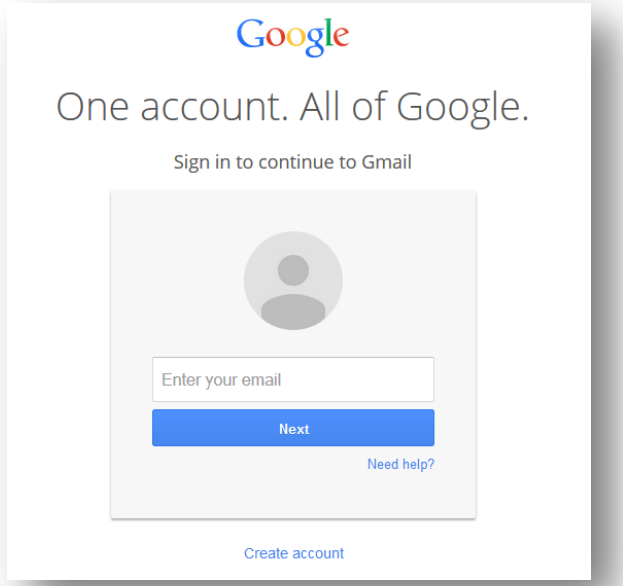
[Source: Gartner]

Outline

- Introduction
- Positioning access control
- Access control models
- How to enforce access control
- The bigger picture
- Some important technologies in practice
- Recap and conclusion

Federated authentication

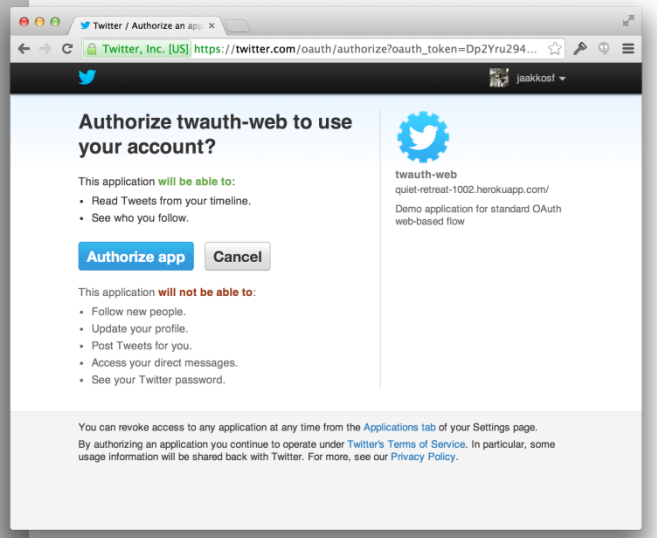


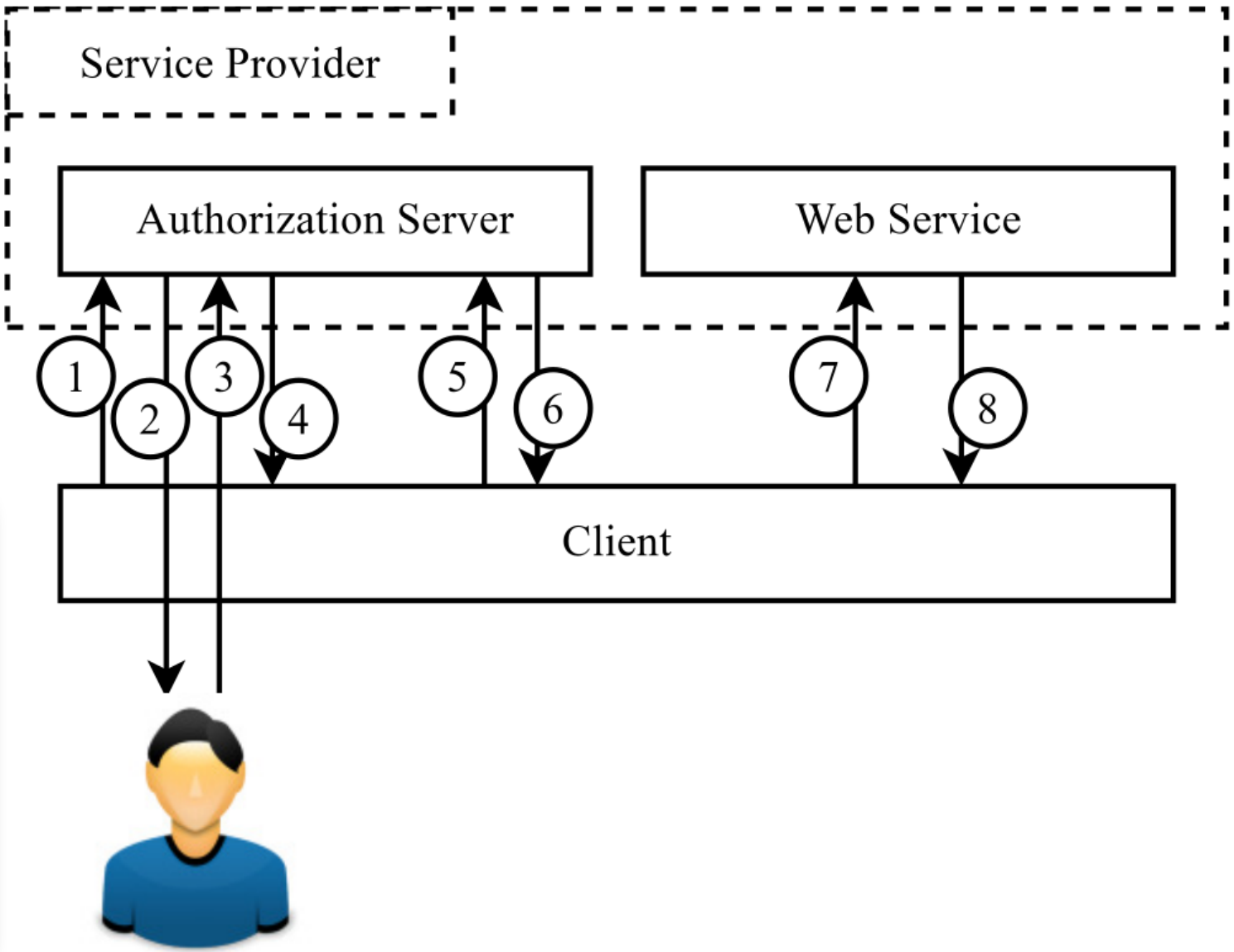


Federated authentication

- Externalizes authorization from a remote application
- Advantages:
  - Lowers the amount of passwords and therefore password reuse
  - Can be used to centralize user mgmt for an organization
  - Removes the need to store passwords in an application
- Standards:
  - OpenID: light-weight, fixed schema, mainly for consumer applications, deprecated
  - SAML: more heavy-weight, extensible, more suitable for enterprise scenarios

OAuth


Twitter / Authorize an app
Twitter, Inc. [US] https://twitter.com/oauth/authorize?oauth_token=Dp2Yru294...
jaakkosf
Authorize twauth-web to use your account?
This application will be able to:
• Read Tweets from your timeline.
• See who you follow.
Authorize app
Cancel
This application will not be able to:
• Follow new people.
• Update your profile.
• Post Tweets for you.
• Access your direct messages.
• See your Twitter password.
twauth-web
quiet-retreat-1002.herokuapp.com/
Demo application for standard OAuth web-based flow
You can revoke access to any application at any time from the Applications tab of your Settings page.
By authorizing an application you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.


Service Provider
Authorization Server
Web Service
1
2
3
4
5
6
7
8
Client

OAuth

- Constrained delegation of access, mostly to 3rd party applications
  - For example, grant a mobile client access to your Twitter stream
  - Also works well with web services and micro-service architectures
- A simplified form of federated authorization
- OAuth 1.0 (2010) was a protocol, OAuth 2.0 (2012) is more a framework
  - Interoperability suffers…

JSON Web Tokens (JWT)

- A recent standardized format to communicate key-value information between parties on the web
  - Commonly used for authentication or authorization info, e.g., in OAuth
  - Typically sent in the `Authorization` header using the `Bearer` schema
    - `Authorization: Bearer <token>`

- Format: encoded JSON
  - Properties: compact, URL-safe, digitally signed for integrity

https://jwt.io/

JSON Web Tokens (JWT) - structure

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Base64Url encode

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

```
header.payload.signature
```

Base64Url encode

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI
6IkpXVCJ9.eyJzdWIiOiIxMjM0N
TY3ODkwIiwibmFtZSI6IkpvaG4g
RG9lIiwiYWRtaW4iOnRydWV9.TJ
VA95OrM7E2cBab30RMHrHDcEfxj
oYZgeFONFh7HgQ
```

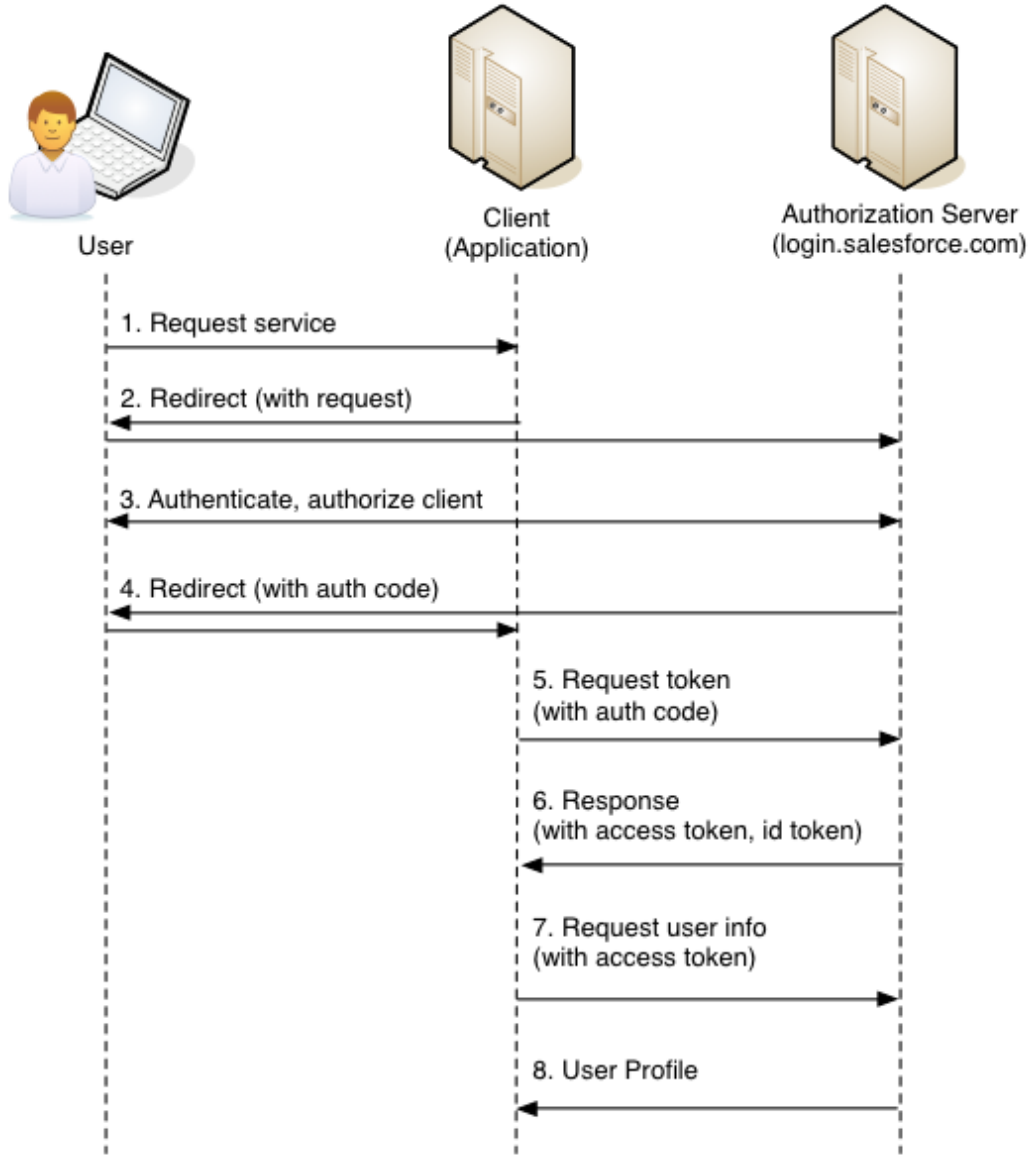

More information: https://jwt.io/

OpenID Connect

- Identity layer on top of the OAuth 2.0
- Achieves many of the *authentication* features of OpenID, but in a more API-friendly and app-friendly way
  - Get basic user info from AuthZ Server of OAuth, get more details from user mgmt API using the OAuth token
- OpenID is considered deprecated, OpenID Connect (OIDC) is considered the successor

http://openid.net/connect/

OpenID Connect



https://developer.salesforce.com/page/Inside_OpenID_Connect_on_Force.com

Outline

- Introduction
- Positioning access control
- Access control models
- How to enforce access control
- The bigger picture
- Recap and conclusion

Recap

- Prevent unauthorized access to protected information
  - AAA: authentication, authorization, audit
  - Often domain-specific enforcement and rules
- Properties of access control systems to take into account
  - Expressiveness, efficiency, full mediation and safety
- Different access control models available
  - Who can assign permissions:
    - MAC and/or DAC
  - How permissions are assigned:
    - identity-based, multi-level, RBAC, ABAC and beyond

Recap

- How to enforce access control in your application code:
  - Modularize!
  - The future: policy-based access control
- The bigger picture:
  - Identity and Access Management (IAM)
- Interesting technologies:
  - Federated authentication: SAML, ~~OpenID~~, OIDC
  - Federated authorization: OAuth

Some final words

- Modern software all depends on access control
- But:
  - Policies are complex to manage in a large organization
    - Choose the minimally complex model for your rules
  - Imperfect because of bugs in the mechanism
    - Make the mechanism as simple as possible
  - Imperfect due to mismatches between policy and mechanism
  - Access control depends on absence of other security bugs
    - Implement least privilege
- After all this, breaches will still occur so pepare and avoid being caught off guard

CWE/SANS Top 25 Software Errors

Rank	Description
5	Missing authentication for critical function
6	Missing authorization
7	Use of hard-coded credentials
8	Missing encryption of sensitive data
10	Reliance on untrusted inputs in a security decision
11	Execution with unnecessary privileges
15	Incorrect authorization
17	Incorrect permission assignment for critical resource
19	Use of a broken or risky cryptographic algorithm
21	Improper restriction of authentication attempts
25	Use of a one-way hash without a salt

References

- [Bogaerts2015] Bogaerts, J., Decat, M., Lagaisse, B., and Joosen, W. Entity-based access control: supporting more expressive access control policies. ACSAC 2015
- [Brewer1989] Brewer, D., and Nash, M. The Chinese Wall security policy. In IEEE Security and Privacy, 1989
- [Cheng2012] Cheng, Y., Park, J., and Sandhu, R. A User-to-User Relationship-Based Access Control Model for Online Social Networks. 2012
- [Crampton2014] Crampton, J., and Sellwood, J. Path Conditions and Principal Matching: A New Approach to Access Control. SACMAT 2014
- [Decat2015] Decat, M., Lagaisse, B., and Joosen, W. Scalable and secure concurrent evaluation of history-based access control policies. ACSAC '15
- [Decat2015b] Decat, M., Bogaerts, J., Lagaisse, B., and Joosen, W. Amusa: Middleware for Efficient Access Control Management of Multi-tenant SaaS Applications. SAC 2015
- [Erlingsson2004] Erlingsson, Úlfar. The inlined reference monitor approach to security policy enforcement. 2004.
- [Fong2011] Fong, P. W. Relationship-based Access Control: Protection Model and Policy Language, CODASPY 2011

References

- [Graham1972] G. Scott Graham and Peter J. Denning. Protection: principles and practice. AFIPS 1972
- [Harrison1976] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. 1976
- [Kuhn2010] Kuhn, D. R., Coyne, E. J., and Weil, T. R. Adding attributes to role-based access control. 2010
- [Lampson1971] Lampson, B. W. Protection. ACM SIGOPS Operating Systems Review, 1971
- [Nguyen2012] Park, J., Nguyen, D., and Sandhu, R. A provenance-based access control model. Privacy, Security and Trust (PST) 2012
- [Nguyen2013] Nguyen, D., Park, J., and Sandhu, R. A provenance-based access control model for dynamic separation of duties. Privacy, Security and Trust (PST) 2013
- [NIST2014] Hu, V., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST 2014.
- [Park2004] Park, Jaehong, and Ravi Sandhu. The UCON ABC usage control model. ACM Transactions on Information and System Security (TISSEC), 2004

Accreditation

- Red door: http://gomighty.com/user/meg/
- Banking application: https://kbctouch.kbc.be/
- Login form: https://w3layouts.com/wp-content/uploads/2014/01/facebook-twitter-google-login.jpg
- Policy man halt: https://pixabay.com/static/uploads/photo/2012/04/01/18/03/policeman-23796_960_720.png
- Policy man traffic fine: http://www.buyautoinsurance.com/wp-content/featured-content/seatbelt/images/traffic-ticket.png

Access Control

SecAppDev 2016

Maarten Decat

 maarten.decat@kuleuven.be

 @maartendecat

 /in/maartendecat

 http://maarten.decat.me/

DistriNet

iMinds KU LEUVEN