APISEC interactive workshop:
Application-level access control
for API-based cloud applications

Policy-driven access control for multi-tenant cloud applications

Bert Lagaisse, Martijn Sauwens 2021/09/14

Bert.lagaisse@kuleuven.be, Martijn.sauwens@kuleuven.be

KU LEUVEN DistriNet

T10 OWASP API Security Top 10 - 2019

API1:2019 - Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.
API2:2019 - Broken User Authentication	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising system's ability to identify the client/user, compromises API security overall.
API3:2019 - Excessive Data Exposure	Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.
API4:2019 - Lack of Resources & Rate Limiting	Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.
API5:2019 - Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.

Broken
Application-level
access control
(Authentication,
Authorization)
=
Root of
Many problems
In API Security

Overall 3-phase approach

Application-level access control for API-based cloud applications

Application-driven requirements analysis

- Example case studies
- Example architectures and functional decompositions
- Example security requirements and their variations
- Feedback and refinement based on your case studies

Possible architectural solutions and their trade-offs

- Security architecture: tactics, solutions and trade-offs
- Their support in OAUTH and IdM systems.
- OAUTH token acquisition flows
- Support in actual technologies and implementations

Advanced server-side access control

- Overview of server-side access control models
- ABAC, PBAC and multi-tenancy support
- State of practice and state of the art
- State-of the art research based on state-of-practice tech

Quick recap

Token acquisition from several types of apps:

consumer tier

SPA

server-side web app

Mobile App

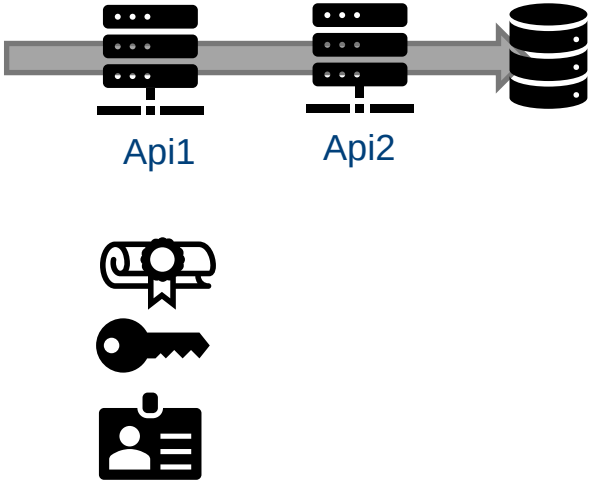
Browserless app

Desktop app

IOT device

tenant-side daemon

› With a signed-in-user



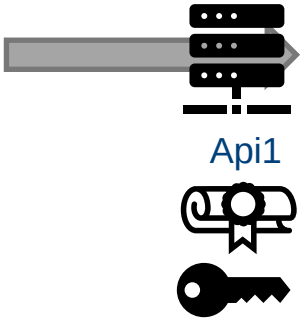
consumer tier

› As application

server-side daemon

tenant-side daemon

daemon web app



Flow/Grant overview

And their implementations in (some) technologies and managed services

Technology	KeyCloak	IdentityServer	AzureAD	Cognito	Auth0	Okta
implicit	V	V	V	V	V	V
Authz code	V	V	V	V	V	V
Authz code+PKCE	V	V	V	V	V	V
Hybrid flow	V	V	V		V	
Client Credentials	V	V	V	V	V	V
Token Exchange	V (loosely)	V(delegation)	V (OBO)			
ROPC	V	V	V		V	V
Device code		V	V		V	
RT rot. (single use)	+/- (revoke)	V	X	X	V	V

So we got all the tokens and their claims to the server …

Now what ?

Server-side access control

- Decide if the operation is allowed
  - assuming the token is correct
    - Integrity
    - Issuer and claim verification
  - given the user claims/attributes
  - given the app claims/attributes
  - given the request context
- Based on the access control policy

Access control policies for reading documents

```
doc.sending_tenant
doc.receiving_tenant
user.tenant
 [application-level]
```

Application Provider

- A user can only access documents sent by or sent to the tenant to which he/she belongs

Small Bank

- Only document managers can read documents

```
user.role
```

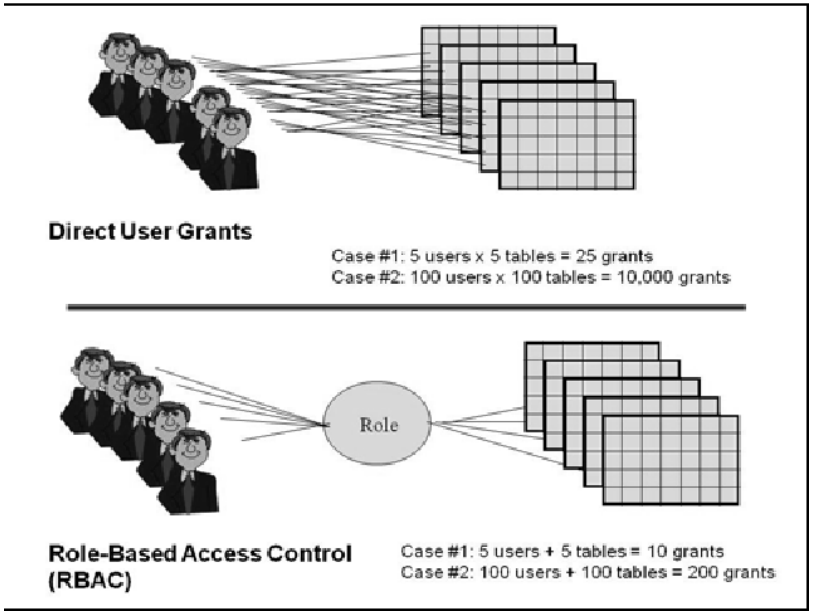
Leasing Company

- Account managers can only read documents that were sent by a tenant to which they were assigned

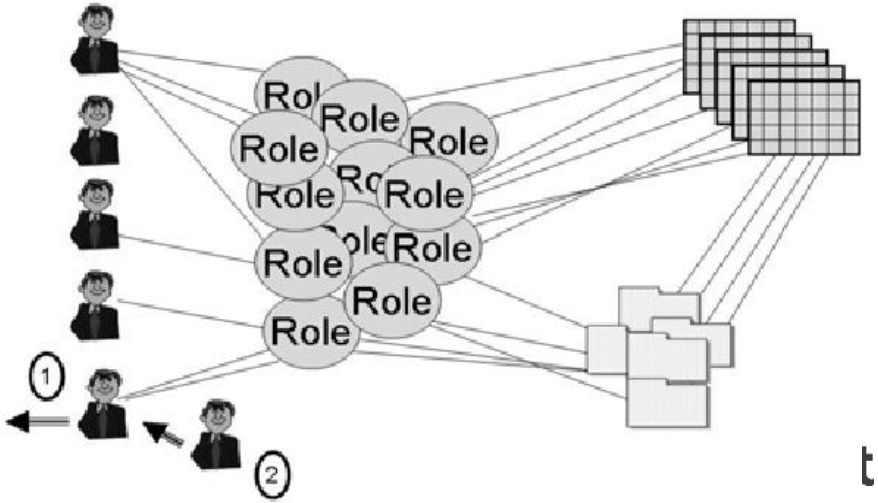
```
user.role
user.manages
```

What's wrong with RBAC

- The promise of RBAC



- Static
- No context
- Too coarse-grained
- Role-explosion



Separation of concerns in secure software engineering

- for the sake of modularity:
  - the right person doing the right task at the right moment in the right artifact.
  - Separate security logic and business logic in separate software artifacts
  - Specified by different kind of people
    - Security administrator
    - Developer
- for adaptability
  - build-time: custom access control logic in a dedicated build for a customer
  - deploy-time: custom access control at deploy time in a dedicated deployment for a customer
  - run-time (concurrent adaptations)

From modular programming artifact to declarative access control policy

e.g custom Amazon s3 policy

allows access only to objects
with a name that includes cognito,
the name of the application,
and the federated user's ID,
represented by the
${cognito-identity.amazonaws.com:sub}
variable.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "ListYourObjects",
         "Effect": "Allow",
         "Action": "s3:ListBucket",
         "Resource": ["arn:aws:s3:::bucket-name"],
         "Condition": {
            "StringLike": {
               "s3:prefix": ["cognito/application-name/${cognito-identity.amazonaws.com:sub}"]
            }
         }
      },
      {
         "Sid": "ReadWriteDeleteYourObjects",
         "Effect": "Allow",
         "Action": [
            "s3:GetObject",
            "s3:PutObject",
            "s3:DeleteObject"
         ],
         "Resource": [
            "arn:aws:s3:::bucket-name/cognito/application-name/${cognito-identity.amazonaws.com:sub}",
            "arn:aws:s3:::bucket-name/cognito/application-name/${cognito-identity.amazonaws.com:sub}/*"
         ]
      }]}
```

Basic technologies

1. Policy-based access control
2. Attribute-based access control
   - Generalizes popular models such as ACL and RBAC
   - Attributes assigned to
     - subjects, actions, resources and environment
   - Express rules based on key-value properties
     - Example: roles:

       ```
       Deny if "manager" not in subject.roles
       ```

     - Example: ownership:

       ```
       Permit if object.owner_id == subject.id
       ```

     - Example: time:

       ```
       Permit if environment.now > 14:00
       ```

PBAC: Specifying access control rules

- **Externalize policies from application code**
  - Policies are evaluated by an evaluation engine
  - Application sends evaluation request to the engine
  - Evaluation engine may fetch additional information (e.g., roles of a subject) from an attribute repository if required for evaluation

→ Increased modularity

→ Better separation of concerns

→ Run-time reconfiguration

→ Concurrent adaptation in Multi-tenancy

A trip back into memory lane (20 years)
Example of a XACML policy:

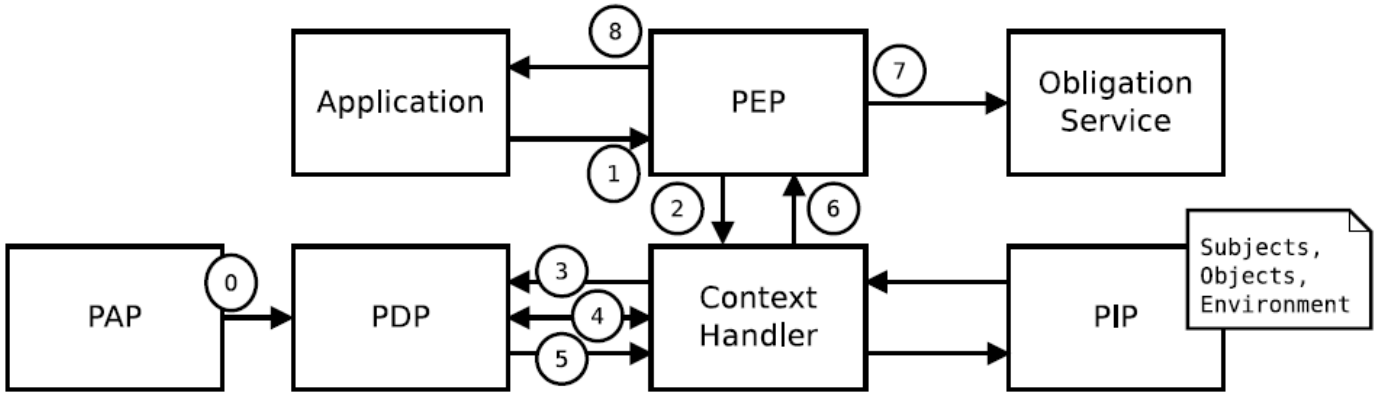
```
<Policy … PolicyId="policy:1“ RuleCombiningAlgId=" deny-overrides">
  <Description>Users can only act on objects owned by their tenant organization</Description>
…
<Rule RuleId="rule:1" Effect="Deny">

      <Condition>
            <Apply FunctionId="not">
            <Apply FunctionId="string-equal">
            <Apply FunctionId="string-one-and-only">
                  <ResourceAttributeDesignator AttributeId="object:creating-tenant" .. />
            </Apply>
            <Apply FunctionId="string-one-and-only">
                  <SubjectAttributeDesignator AttributeId="subject:tenant" …/>
            </Apply></Apply></Apply>
      </Condition>

      </Rule>
</Policy>
```

XACML

- Attribute-based expressions
  - Attributes have types
- Tree-structured policies
  - PolicySets > Policies > Rules
  - Targets (e.g., when resource.id == “doc123”)
  - Policy references for modularity
  - Combination logic: permit-overrides, deny-overrides
- Obligations (e.g., log(“John Smith accessed doc123”), appendAttribute(“history”, “John Smith”, “doc123”))

XACML reference architecture for access control

- Policy Enforcement Point
- Policy Decision Point
- Policy Information Point
- Policy Administration point



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>
```

20 years of access control research
Research tracks

Modularity

- Separations of concerns and modularity of AC with AOP
  - Access control with AspectJ(Bart Dewin)
  - Advanced access control with CaesarJ (Tinne Verhanneman)

Expressive power in policies

- XACML++
  - STAPL: simple tree-based access control (ease of use and readability)
  - EBAC: entity-based access control (OO domain concepts in policy)

Efficient Middleware For multi-tenancy

- Access control middleware for contemporary software architectures
  - AMUSA, ACE: combining policies in multi-tenant applications
  - Sequoia: data query rewriting with policy constraints

Market-driven evolution of the programme

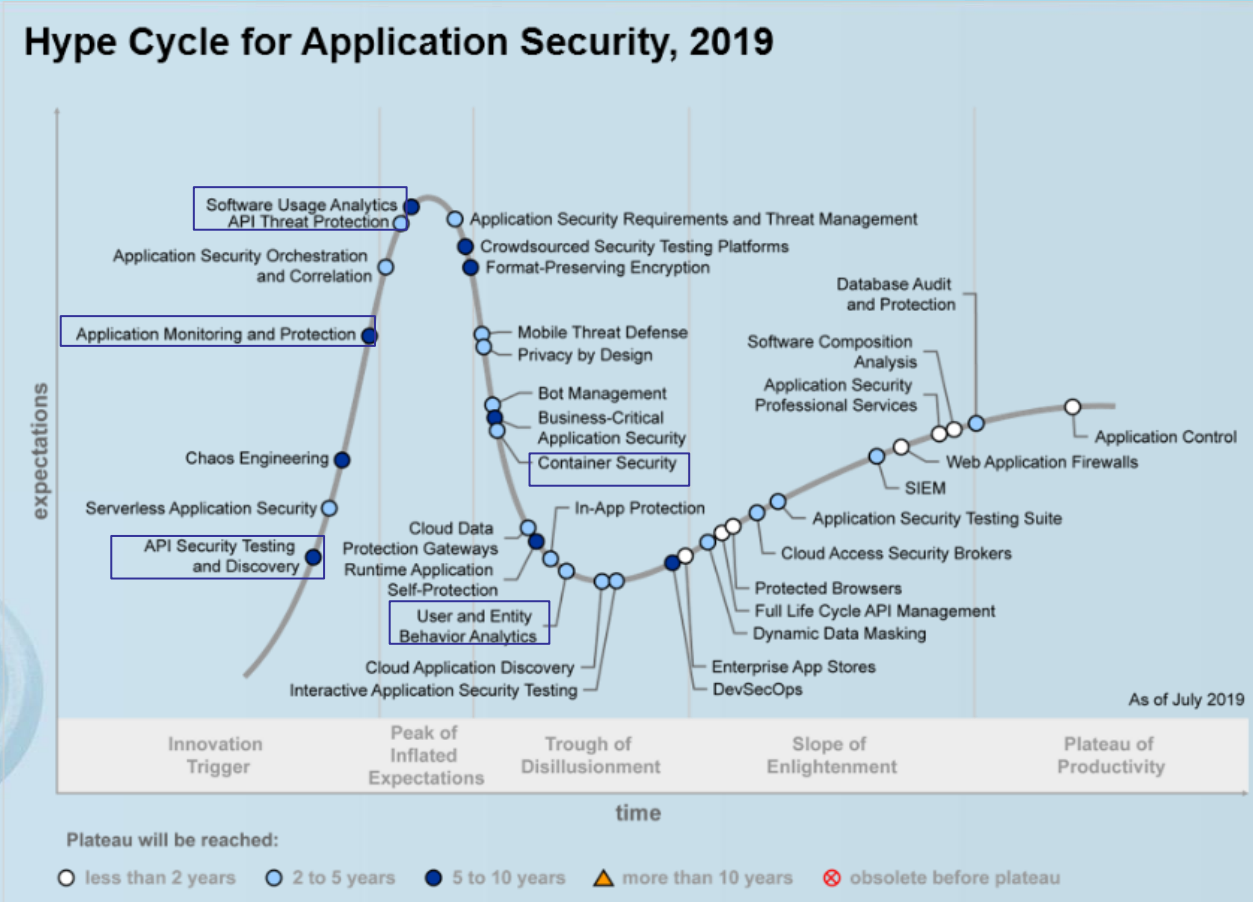
More focus:

Adaptive Application Security

- SaaS → API security
- From:Customer-manageable security
- To:
  - Self-adaptive security
  - Audit-driven security
  - Closing the loop

Distributed data management

- From: Distributed data system
- To: more security tactics
  - Data privacy tactics
  - Data protection tactics



Focus on prototype-driven intensive collaborations with Flemish Industry, and EU

KATHOLIEKE UNIVERSITEIT LEUVEN

Vlaio-O&O

xenit return on content

noesis

egemin AUTOMATION

inventivedesigners

ICON

verizon

UnifiedPost

pwc

televic rail

esas

proximus

EU

FOKKER AEROSTRUCTURES

ERICSSON

DRÄXLMAIER

Expressive power in policies

STAPL

The Simple Tree-structure Attribute-based Policy Language

STAPL

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

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

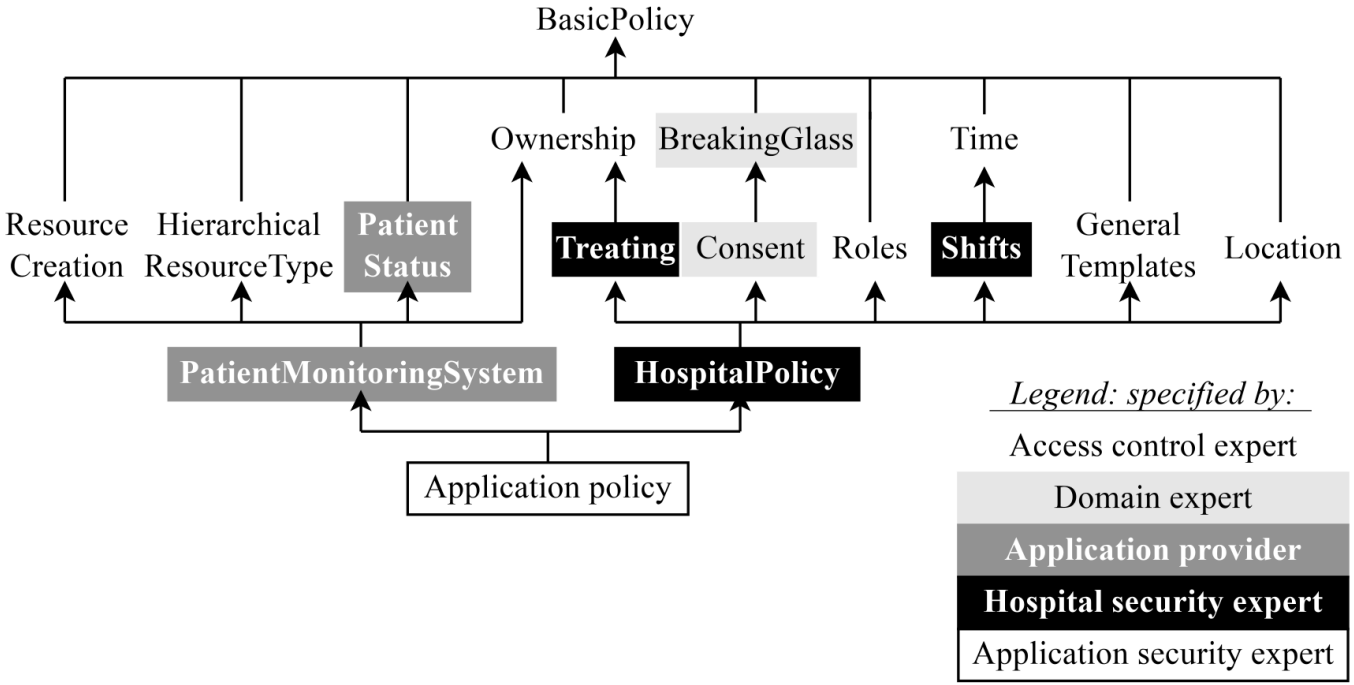
```
Rule(“time") := deny iff (env.currentDateTime > (resource.created + 5.days))
```

```
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)
```

Ease of specifying policies

		Attr. def.	Obl. def.	Pol. spec.	Total
E-health	**XACML**	-	-	706	706 (100%)
	ALFA	168	3	259	430 (60.9%)
	STAPL	27	4	84	115 (**16.3%**)
E-docs	**XACML**	-	-	1332	1332 (100%)
	ALFA	175	3	514	692 (52.0%)
	STAPL	31	4	196	231 (**17.3%**)

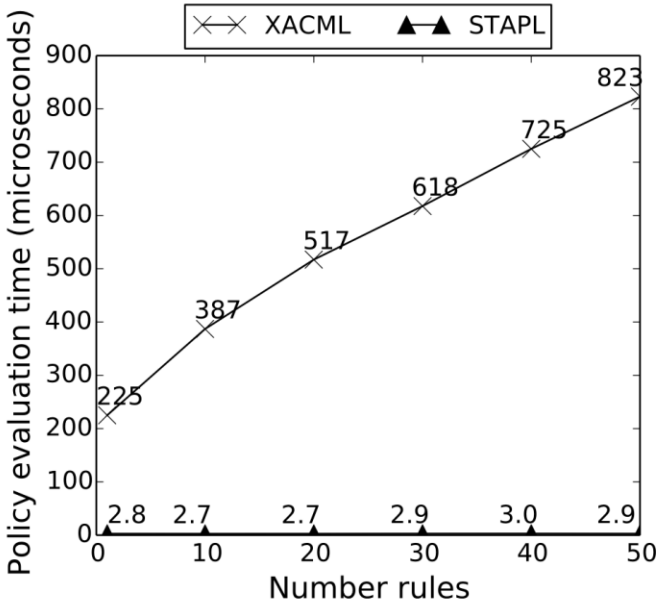
Modularization

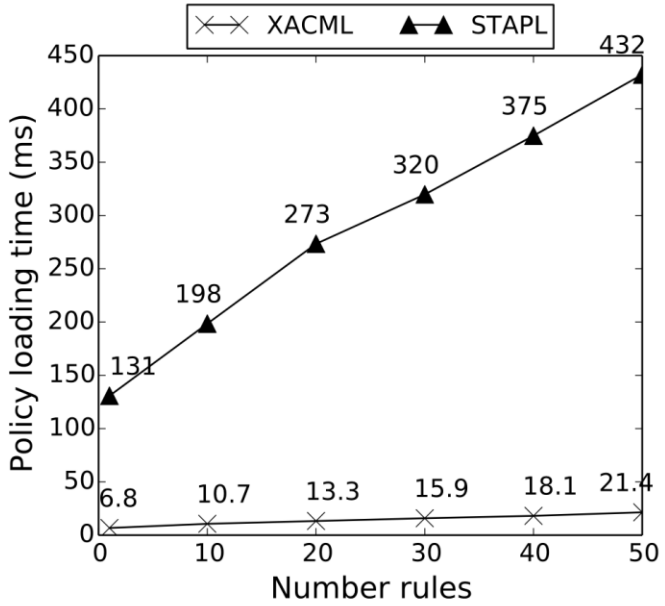


Modularization

```
trait Shifts extends BasicPolicy {
    env.time = SimpleAttribute(Time)
    def denyIfNotOnShift(start: Time, stop: Time) =
        Rule := deny iff (!(env.time ≥ start & env.time stop))
}
object example extends Shifts with Treating with … {
    Policy := when (action.id === "view") apply  PermitOverrides to (
        Policy := when ("nurse" in subject.roles) apply  DenyOverrides to (
            denyIfNotTreating,
            denyIfNotOnShift(09:00, 17:00),
            Rule := permit),
        Rule := permit iff (subject.triggered_breaking_glass)
            performing (log(subject.id + " broke the glass")))
}
```

Performance evaluation



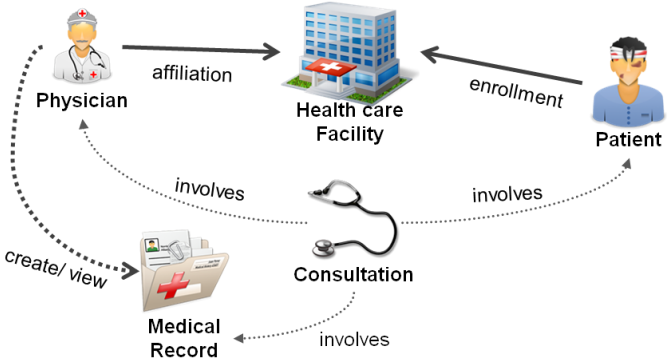


KU LEUVEN DistriNet

Entity Based Access Control

Problem, revisited

"Physicians can only create medical records for patients enrolled to the same facility as them"



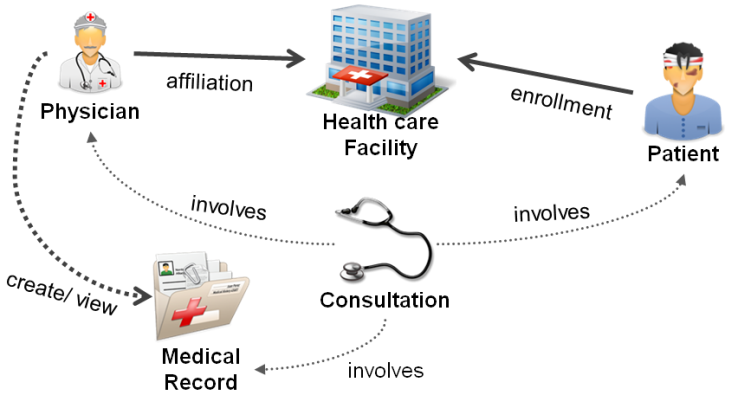
subject.affiliation_id = resource.consultation_patient_enrollment_id

```
SELECT Facility.id FROM MedRec
    JOIN Consultation ON MedRec.consultation = Consultation.id
    JOIN Patient ON Consultation.patient = Patient.id
    JOIN Facility ON Patient.enrollment = Facility.id
    WHERE MedRec.id = ?
```


DistriNet

Problem, revisited

"Physicians can view medical records if the corresponding patient had a consultation with them in the last year"



resource.consultation_patient_id ∈ subject.patients_of_last_year

```
SELECT Patient.id FROM Physician
        JOIN Consultation ON Physician.consultations = Consultation.id
        JOIN Patient ON Consultation.patient = Patient.id
        WHERE Subject.id = ? AND Consultation.date BEFORE (...)
```


DistriNet

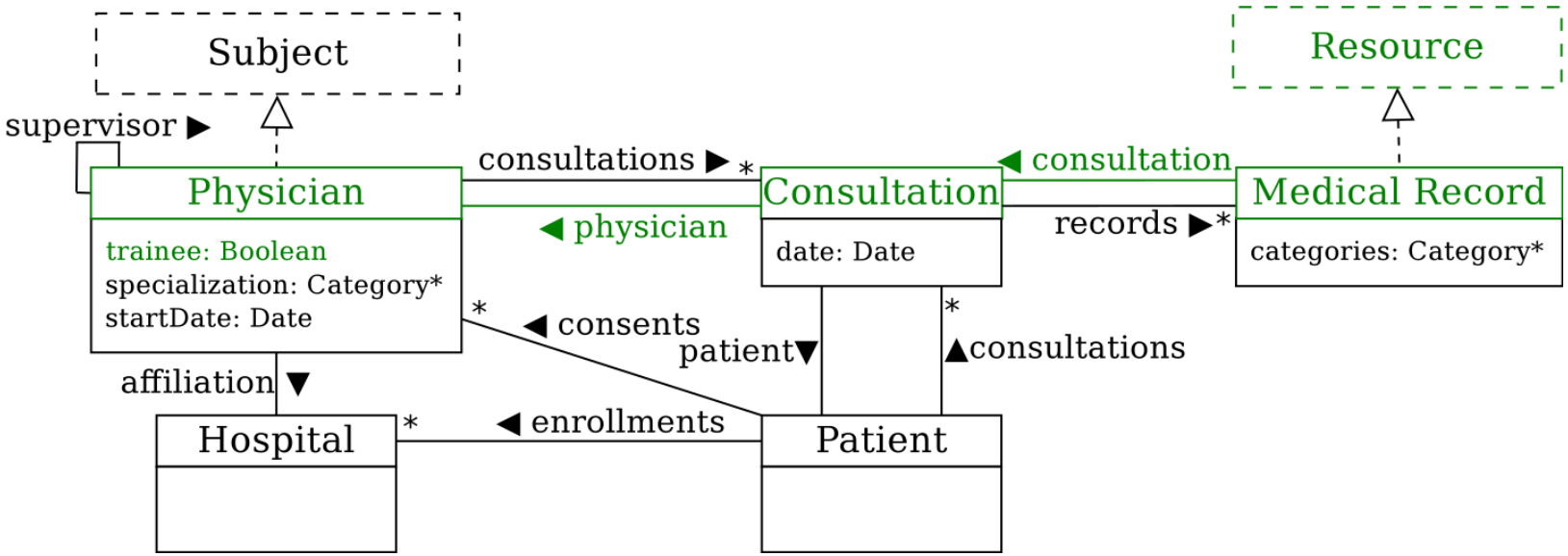
Problem, revisited

- ABAC does not support expression of relationships
  - Attributes are assigned to subject, resource, action and environment
  - Does not seamlessly apply to application domain!
  - Also, multiple attributes over the relationship may be relevant!

Entity-Based Access Control (EBAC)

- First-class citizen: Entity
  - cfr. Entity-Relationship Model
  - Entities have both relationships and attributes
- Like ABAC, attributes compared in logical expressions
  - Addressed starting from subject, resource, action or environment
  - Unlike ABAC, attributes of auxiliary entities can be addressed through relationships

Entity model



resource.consultation.physician.trainee

DistriNet

Comparison with ABAC

Attribute-Based Access Control

subject.affiliation_id ∈
resource.cons_patient_enroll_id

```
SELECT Facility.id FROM MedRec
    JOIN Consultation ON MedRec.consultation = Consultation.id
    JOIN Patient ON Consultation.patient = Patient.id
    JOIN Facility ON Patient.enrollment = Facility.id
    WHERE MedRec.id = ?
```

resource.consultation_patient_id ∈
subject.patients_of_last_2_years

```
SELECT Patient.id FROM Physician
    JOIN Consultation ON Physician.consultations = Consultation.id
    JOIN Patient ON Consultation.patient = Patient.id
    WHERE Subject.id = ? AND Consultation.date BEFORE (NOW – 2y)
```

resource.consultation_physician ∈
subject.all_supervisors

recursive method!

Entity-Based Access Control

subject.affiliation.id ∈
resource.consultation.patient.enrollment.id

∃ c ∈ subject.consultations:
(c.patient.id = resource.consultation.patient.id &
c.date <= (environment.now – 2 years))

$\exists_{\rho}$ s ∈ subject.supervisor:
(resource.consultation.physician.id = s.id)

Auctoritas: Extension of STAPL that supports EBAC

- Example:

```
Policy(“example”) := apply DenyOverrides to (
  Rule(“Only enrolled”) := permit iff (action.id === “create” &
           subject.affiliation in resource.consultation.patient.enrollments
         ),
  Rule(“Recent consultation”) := permit iff (action.id === “view” &
           resource.consultation.patient.consultations.exists(
              consultation => consultation.physician.id === subject.id &
                              environment.now >= (consultation.date + 2.years)
           )
         ),
  Rule(“Indirect supervisor”) := permit iff (action.id === “view” &
           subject.supervisor.existsOnPath(
             supervisor => resource.consultation.physician.id === supervisor.id
           )
         )
)
```

AMUSA

Access control policies for reading documents

```
doc.sending_tenant
doc.receiving_tenant
user.tenant
 [application-level]
```

Application Provider

- A user can only access documents sent by or sent to the tenant to which he/she belongs

Small Bank

- Only document managers can read documents

```
user.role
```

Leasing Company

- Account managers can only read documents that were sent by a tenant to which they were assigned

```
user.role
user.manages
```

Authorization middleware for multi-tenant applications

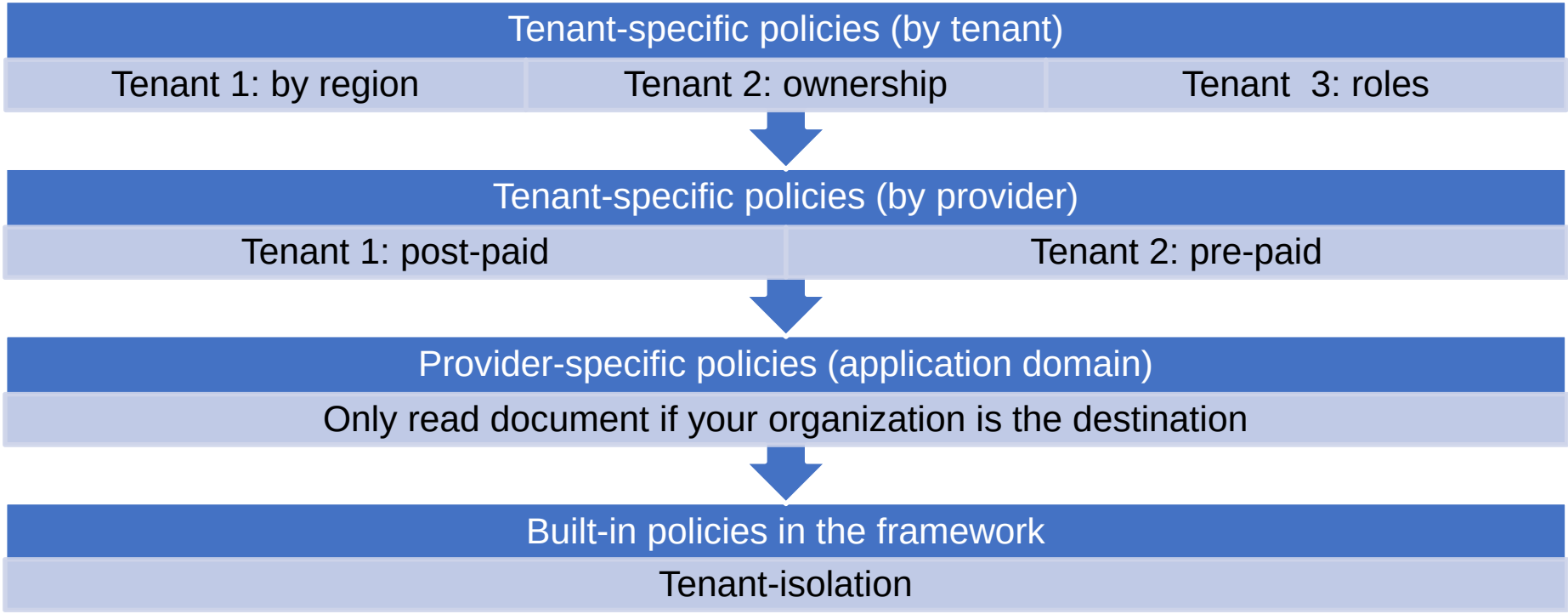
AMUSA middleware:

1. Multi-tenancy out-of-the-box
2. Provider-specific policies
3. Tenant-specific attributes
4. Tenant-specific policies

	Large Bank	*Press Agency*
Tenants	subj.assigned_customers	subj.region
eDocs	subj.email, subj.tenant_credit, res.sender	
Amusa	subj.id, res.id, subj.tenant, res.tenant, res.owner, subj.roles	

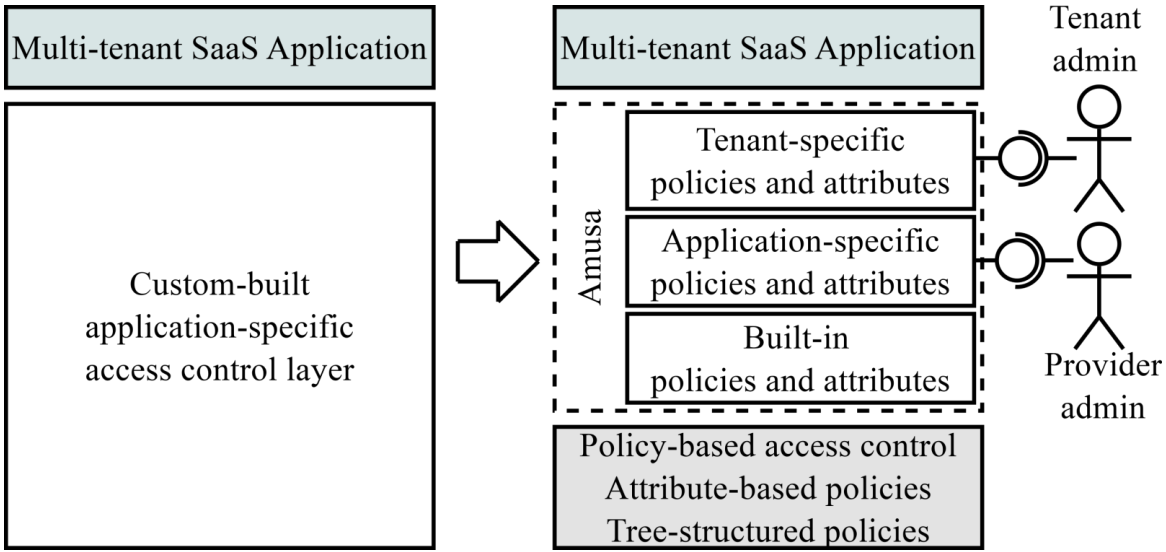
	Large Bank	*Press Agency*
Tenants	Deny if not res.owner in subj.assigned_customers Override isolation if subj.tenant == "PartnerA"	Deny if subj.region != "Europe"
eDocs	Deny if subj.tenant_credit < action.cost Override isolation if res.owner in subj.reseller_tenants	
Amusa	Default tenant isolation policy	

Authorization model: 4 levels of policies

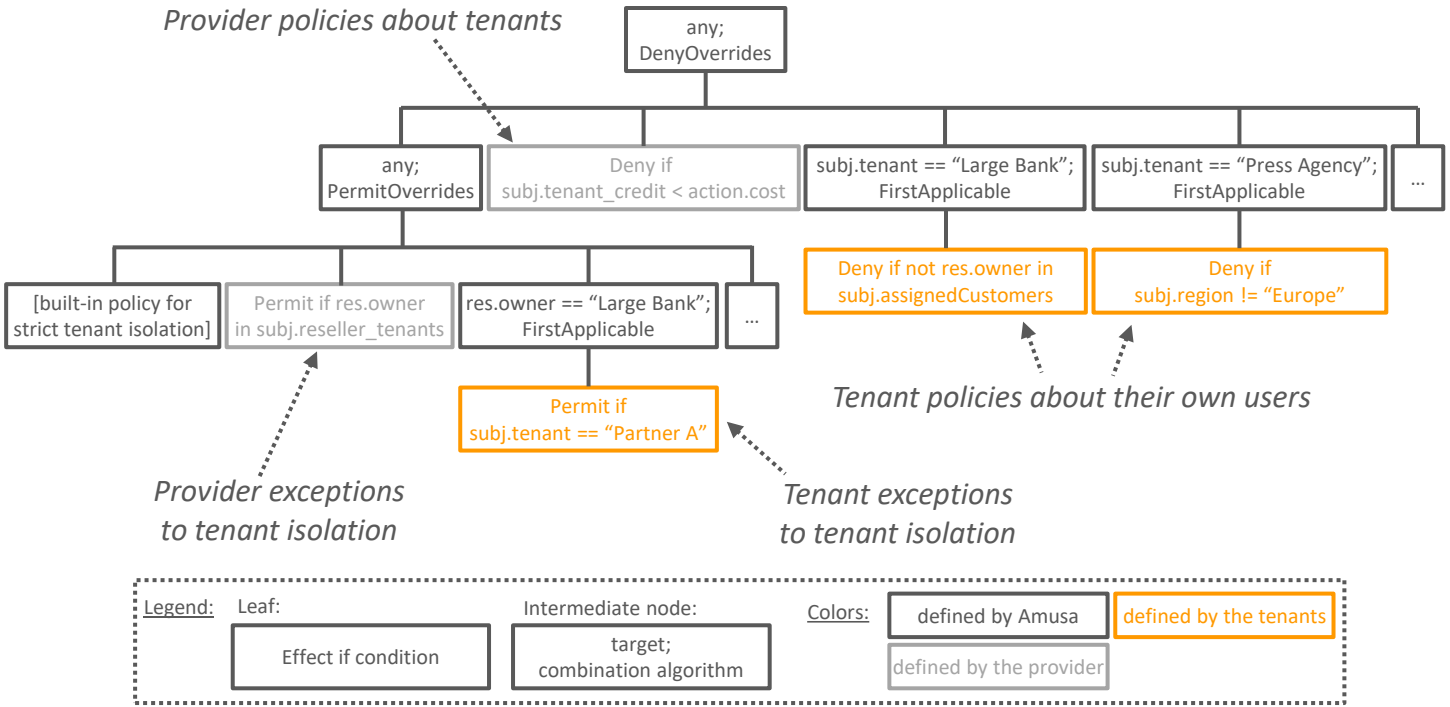


Goal

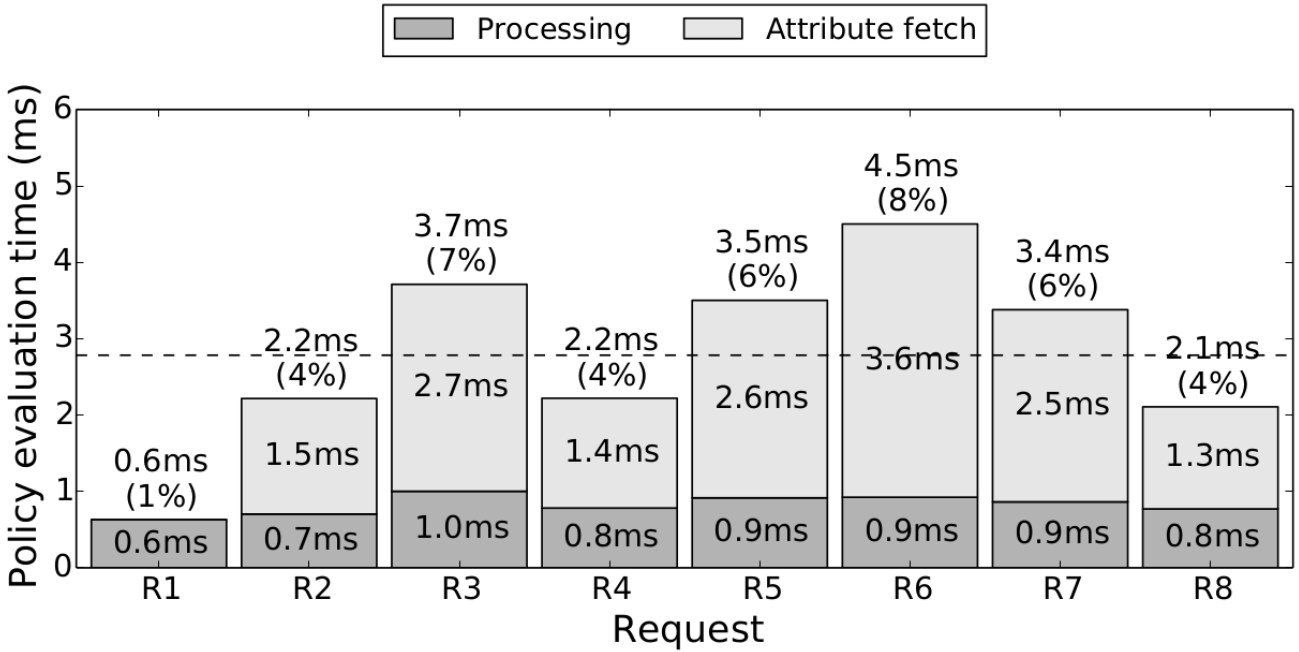
- Combine policies securely
- Enforce at run-time



Secure policy combination



Performance



Project ACE:
Multi-tenant PBAC in asp.net web stack

Out-of-the-box authentication, authorization and audit for

- SaaS: Out-of-the-box Multi-tenancy
- Flexible access control scripts
  - Per-tenant, by provider & by tenant
  - Constrain users, tenants, services
  - Dynamic customization, extension

Deep integration with .net stack

- Azure AD
- Asp.net MVC
- Asp.net Web API

```
[Authorize]
public class BillViewController : Controller
{
  [PBACMVC]
  public async Task<ActionResult> Index(){
  …
  }
```

```
[Authorize]
public class BillController : ApiController
{
  [PBAC]
  public string Get(int id){
      …
  }
```

```
if(request.TenantId != Tenants.SmallBuz)
 throw "This policy is for SmallBuz only";

if (request.Controller== “BillGenerator"
  && request.Action == "Put") {
   if (!isNight())
    if (request.AppId != Apps.WebPortal)
     throw "You can only upload at night";
}
```

KU LEUVEN DistriNet

Application-level Access Control: Configuration vs Policies vs Implementation

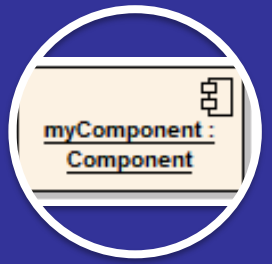
Configuration

- Role-based
- User-based
- Group--based
- Annotations
- Config file

Declarative Policy

- XACML
- STAPL
- JSON-based
- Constraints on ABAC
- **OPA**

Dynamic Externalized Access Scripts



Authorization components

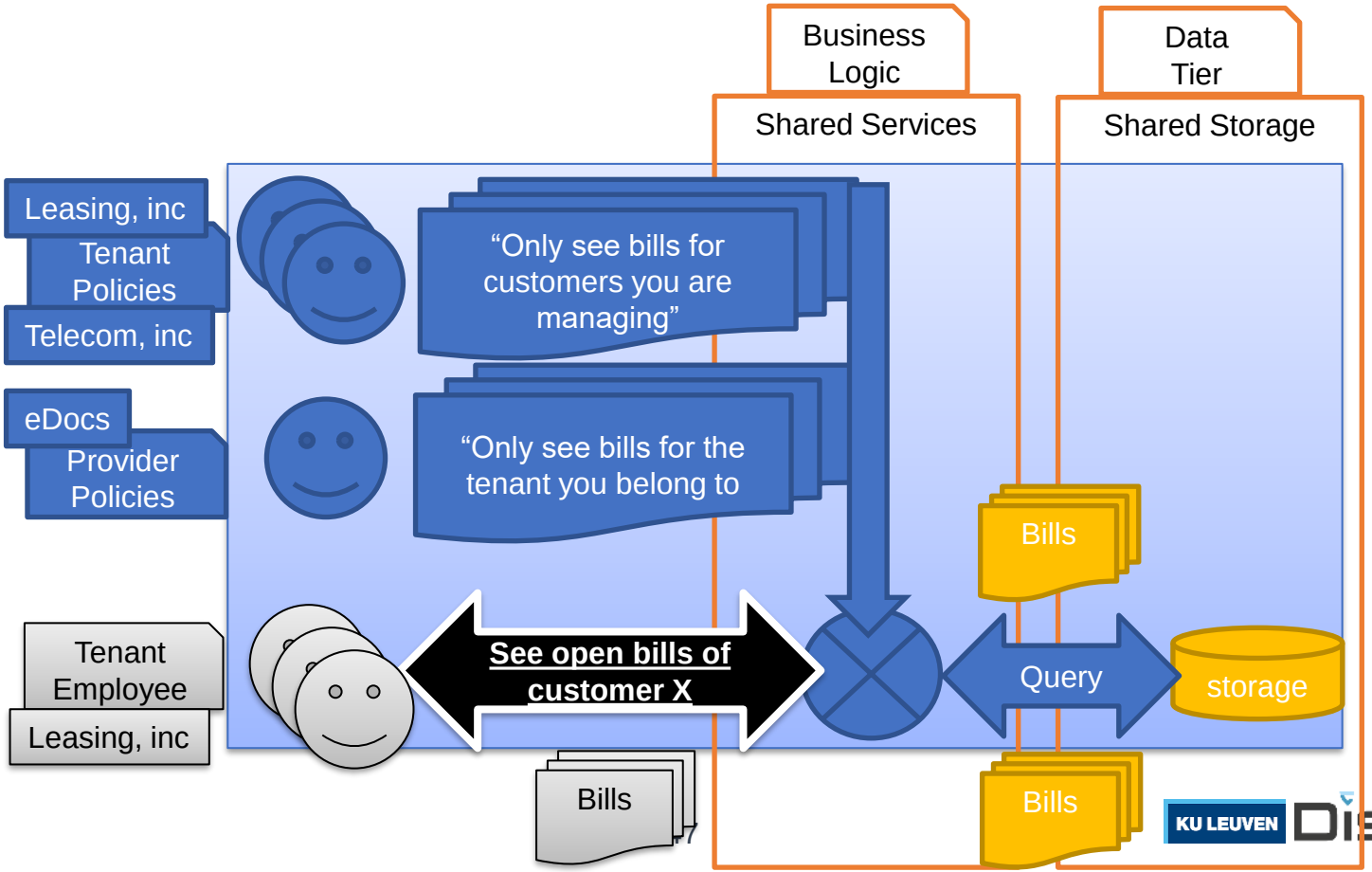
- Custom authorization module
- Policy implemented in component code
- Bound via config file or annotations

Hard-coded

- Application-level implementation
- Inside the business logic

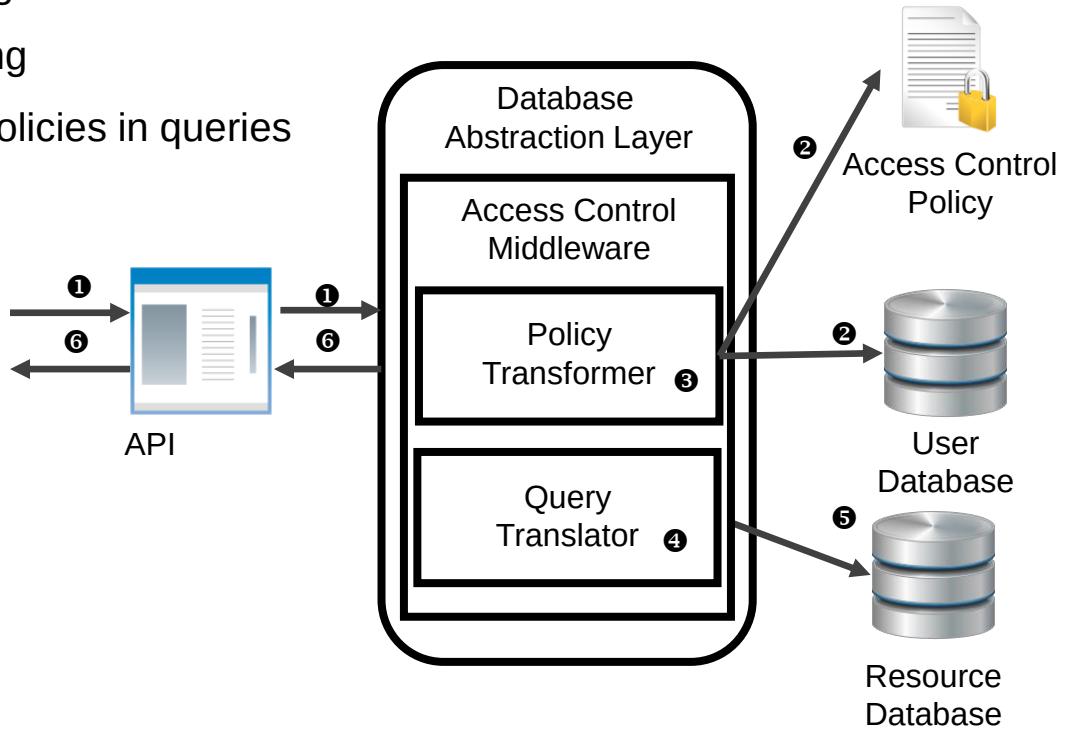
Sequoia

Sequoia: secure queries on internet APIs

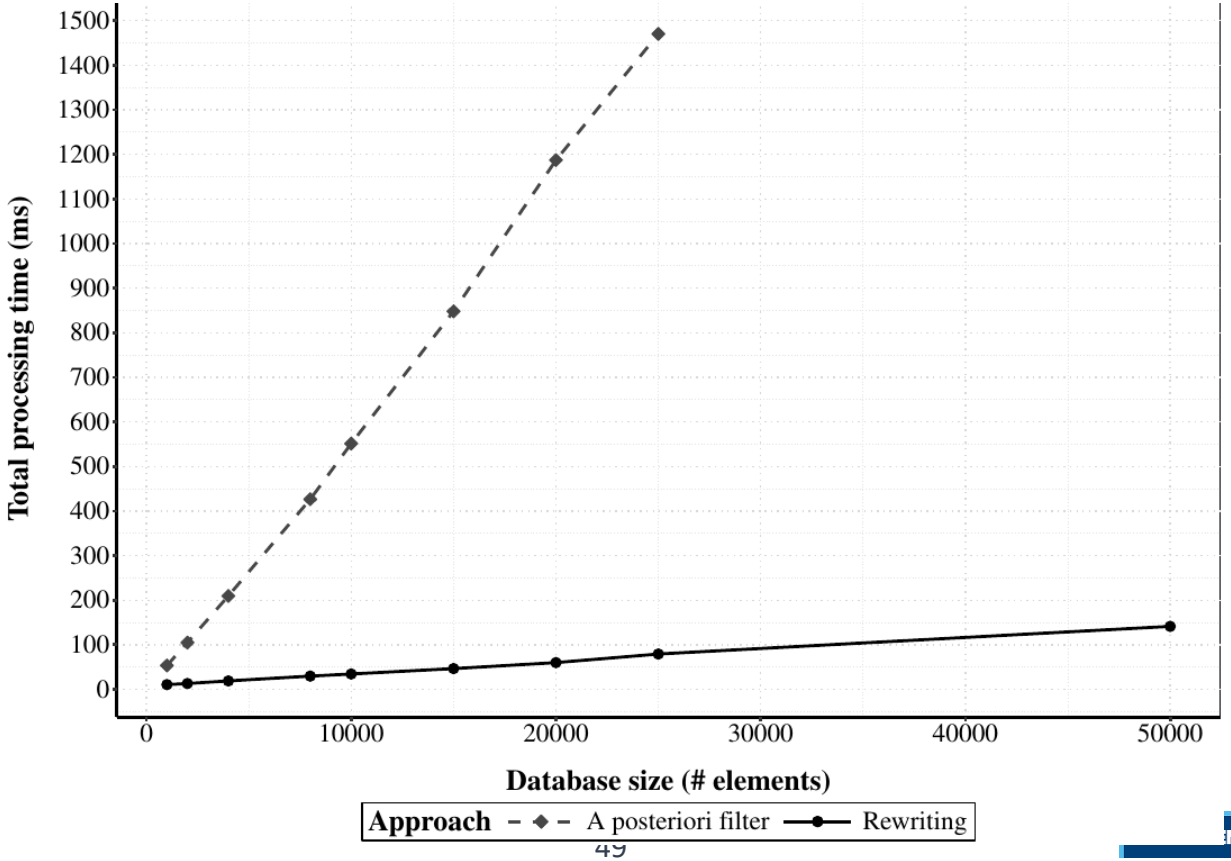


Sequoia: security framework solution

- Beyond evaluating policies on single resources
- Secure data querying and reporting
- Enforcing sophisticated security policies in queries



Scalability w.r.t. naïve approach



Processing overhead