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
## OWASP API Security Top 10 - 2019

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API1:2019 - Broken Object Level Authorization</td>
<td>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.</td>
</tr>
<tr>
<td>API2:2019 - Broken User Authentication</td>
<td>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.</td>
</tr>
<tr>
<td>API3:2019 - Excessive Data Exposure</td>
<td>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.</td>
</tr>
<tr>
<td>API4:2019 - Lack of Resources &amp; Rate Limiting</td>
<td>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.</td>
</tr>
<tr>
<td>API5:2019 - Broken Function Level Authorization</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

- **With a signed-in-user**
- **As application**

---

**SPA**
- Server-side web app
- Mobile App
- Browserless app
- Desktop app
- IOT device
- Tenant-side daemon
- Daemon web app

---

**Api1**
**Api2**

---

**Consumer tier**
Flow/Grant overview
And their implementations in (some) technologies and managed services

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

**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

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

```json
{
    "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}/*"]
        }
    ]
}
```

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.
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>User 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>
      </Apply>
      <Apply FunctionId="string-one-and-only">
        <SubjectAttributeDesignator AttributeId="subject:tenant" …/>
      </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
<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>
  </Target>
  <Obligations>
    <Obligation ObligationId="append-attribute" FulfillOne="Permit">
      <AttributeAssignment AttributeId="value" DataType="string"/>
      <SubjectAttributeDesignator AttributeId="resource:id" DataType="string"/>
    </AttributeAssignment>
    <AttributeAssignment AttributeId="attribute-id" DataType="string">subject:history</AttributeAssignment>
  </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
Expressive power in policies
STAPL
The Simple Tree-structure
Attribute-based Policy Language
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

<table>
<thead>
<tr>
<th>E-health</th>
<th>Attr. def.</th>
<th>Obl. def.</th>
<th>Pol. spec.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>XACML</td>
<td>-</td>
<td>-</td>
<td>706</td>
<td>706 (100%)</td>
</tr>
<tr>
<td>ALFA</td>
<td>168</td>
<td>3</td>
<td>259</td>
<td>430 (60.9%)</td>
</tr>
<tr>
<td>STAPL</td>
<td>27</td>
<td>4</td>
<td>84</td>
<td>115 (16.3%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E-docs</th>
<th>Attr. def.</th>
<th>Obl. def.</th>
<th>Pol. spec.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>XACML</td>
<td>-</td>
<td>-</td>
<td>1332</td>
<td>1332 (100%)</td>
</tr>
<tr>
<td>ALFA</td>
<td>175</td>
<td>3</td>
<td>514</td>
<td>692 (52.0%)</td>
</tr>
<tr>
<td>STAPL</td>
<td>31</td>
<td>4</td>
<td>196</td>
<td>231 (17.3%)</td>
</tr>
</tbody>
</table>
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 ... {
  val 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
Entity Based Access Control
“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 = ?
“Physicians can view medical records if the corresponding patient had a consultation with them in the 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 (...)

resource.consultation_patient_id ∈ subject.patients_of_last_year
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
resource.consultation.physician.trainee
Comparison with ABAC

Attribute-Based Access Control

\[
\text{subject.affiliation_id} \in \text{resource.cons_patient_enroll_id}
\]

\[
\text{SELECT Facility.id FROM MedRec}
\]
\[
\text{JOIN Consultation ON MedRec.consultation = Consultation.id}
\]
\[
\text{JOIN Patient ON Consultation.patient = Patient.id}
\]
\[
\text{JOIN Facility ON Patient.enrollment = Facility.id}
\]
\[
\text{WHERE MedRec.id = ?}
\]

\[
\text{resource.consulation.patient_id} \in \text{subject.patients_of_last_2_years}
\]

\[
\text{SELECT Patient.id FROM Physician}
\]
\[
\text{JOIN Consultation ON Physician.consultations = Consultation.id}
\]
\[
\text{JOIN Patient ON Consultation.patient = Patient.id}
\]
\[
\text{WHERE Subject.id = ? AND Consultation.date BEFORE (NOW – 2y)}
\]

\[
\text{resource.consulation.physician} \in \text{subject.all_supervisors}
\]

Entity-Based Access Control

\[
\text{subject.affiliation_id} \in \text{resource.consultation.patient.enrollment.id}
\]

\[
\exists c \in \text{subject.consultations}:
\]
\[
(c.\text{patient.id} = \text{resource.consultation.patient.id} \land
\]
\[
c.\text{date} \leq (\text{environment.now} – 2\text{y})
\]

\[
\exists p \ s \in \text{subject.supervisor}:
\]
\[
(\text{resource.consultation.physician.id} = s.\text{id})
\]

recursive method!
Auctoritas: Extension of STAPL that supports EBAC

Example:

```plaintext
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
        )
    ))
)"
Access control policies for reading documents

- **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

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

<table>
<thead>
<tr>
<th>Tenants</th>
<th>Large Bank</th>
<th>Press Agency</th>
</tr>
</thead>
<tbody>
<tr>
<td>subj.assigned_customers</td>
<td>subj.region</td>
<td></td>
</tr>
<tr>
<td>eDocs</td>
<td>subj.email, subj.tenant_credit, res.sender</td>
<td></td>
</tr>
<tr>
<td>Amusa</td>
<td>subj.id, res.id, subj.tenant, res.tenant, res.owner, subj.roles</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tenants</th>
<th>Large Bank</th>
<th>Press Agency</th>
</tr>
</thead>
<tbody>
<tr>
<td>subj.assigned_customers</td>
<td>subj.region</td>
<td></td>
</tr>
<tr>
<td>eDocs</td>
<td>subj.tenant_credit &lt; action.cost</td>
<td></td>
</tr>
<tr>
<td>Amusa</td>
<td>Override isolation if res.owner in subj.reseller_tenants</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decisions</th>
<th>Large Bank</th>
<th>Press Agency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tenant</td>
<td>Deny if not res.owner in subj.assigned_customers</td>
<td></td>
</tr>
<tr>
<td>eDocs</td>
<td>Deny if subj.tenant_credit &lt; action.cost</td>
<td></td>
</tr>
<tr>
<td>Amusa</td>
<td>Default tenant isolation policy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Override isolation if subj.region != &quot;Europe&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Deny if subj.region != "Europe"
Authorization model: 4 levels of policies

- **Built-in policies in the framework**
  - Tenant-isolation

- **Provider-specific policies (application domain)**
  - Only read document if your organization is the destination

- **Tenant-specific policies (by provider)**
  - Tenant 1: post-paid
  - Tenant 2: pre-paid

- **Tenant-specific policies (by tenant)**
  - Tenant 1: by region
  - Tenant 2: ownership
  - Tenant 3: roles

- **Tenant isolation**
Goal

› Combine policies securely

› Enforce at run-time
Secure policy combination

Provider policies about tenants

- any; DenyOverrides
- any; PermitOverrides
- Deny if subj.tenant_credit < action.cost
- subj.tenant == “Large Bank”; FirstApplicable
- subj.tenant == “Press Agency”; FirstApplicable

Provider exceptions to tenant isolation

- [built-in policy for strict tenant isolation]
- Permit if res.owner in subj.reseller_tenants
- res.owner == “Large Bank”; FirstApplicable
- Permit if subj.tenant == “Partner A”

Tenant policies about their own users

- Deny if not res.owner in subj.assignedCustomers
- Deny if subj.region != “Europe”

Tenant exceptions to tenant isolation

Legend:
- Leaf: Effect if condition
- Intermediate node: target; combination algorithm
- Colors: defined by Amusa; defined by the tenants; defined by the provider
Performance

![Graph showing performance evaluation time for different requests.](image)
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()
    {
        ...
    }

    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";
    }
```

```
[Authorize]
public class BillController : ApiController
{
    [PBAC]
    public string Get(int id)
    {
        ...
    }
```
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: secure queries on internet APIs

"Only see bills for customers you are managing"

"Only see bills for the tenant you belong to"

See open bills of customer X

Business Logic

Shared Services

Data Tier

Shared Storage

Leasing, inc
Tenant Policies
Telecom, inc
eDocs
Provider Policies
Tenant Employee
Leasing, inc

Bills

Query

storage
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