

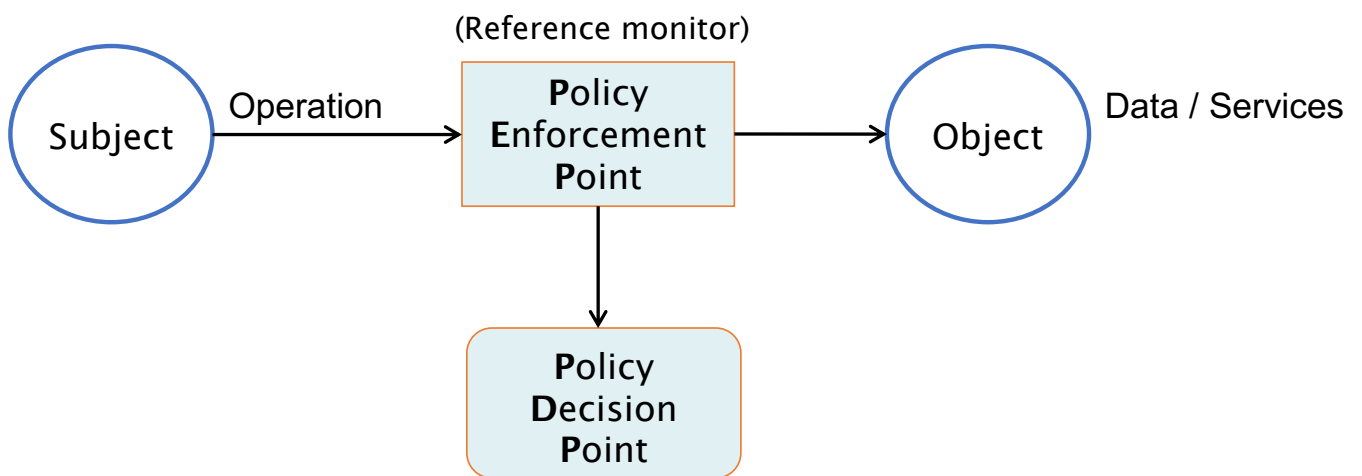
Modelos e políticas para controlo de acessos

- *Modelos Matriciais*
- *Modelos Role-based Access Control*
- *Biblioteca Casbin*

Agenda

- Modelos para controlo de acessos
- Controlo de acessos baseado em *roles*
- Família de modelos
 - Base: $RBAC_0$
 - Hierarquia de *roles*: $RBAC_1$
 - Restrições: $RBAC_2$
 - Unificação: $RBAC_3$
- Caso prático: Biblioteca Casbin (<https://casbin.org/>)

Monitor de referências



- Propriedades do PEP:
 - Não deve ser possível alterá-lo.
 - Não deve ser possível contorná-lo.
 - Deve ser pequeno e estar confinado ao núcleo de segurança do sistema por forma a facilitar a verificação da sua correcção.

Elementos do sistema de controlo de acessos

- Modelo de segurança: formalização da forma de aplicação das políticas de segurança
- Política de segurança: define as regras do controlo de acessos
- Mecanismos de segurança – software/hardware: funções de baixo nível que dão suporte à implementação de modelos e políticas de segurança
- PEP depende dos mecanismos de segurança
- PDP depende da política e modelo de segurança

Modelo Matriz de Acessos

- Matriz de acessos define um triplo (S,O,A) , onde:
 - S é o conjunto de sujeitos.
 - O é o conjunto de objectos.
 - A é o conjunto de operações
 - M_{so} é a matriz de operações, onde as linhas representam os sujeitos, as colunas os objectos e cada entrada $M[s,o]$ as permissões do sujeito s sobre o objecto o .

	File1	File2	File3	Program1
Alice	read write	read write		execute
Bob	read		read write	
Charlie		read		execute write

Proposta inicial por Lampson (Protection, In 5th Princeton Symposium Information Science and Systems, 1971)

Formalizado por Harrison, Ruzzo e Ullmann (Protection in Operating Systems, Communications of the ACM, 19(8), 1976)

Implementação da matriz de acessos (1)

Tabela de autorização

Sujeito	Permissão	Objecto
Alice	read	File1
Alice	write	File1
Alice	read	File2
Alice	write	File2
Alice	execute	Program1
Bob	read	File1
Bob	write	File3
Bob	read	File3
Charlie	read	File2
Charlie	execute	Program1
Charlie	write	Program1

Implementação da matriz de acessos (2)

Capabilities:

- As permissões são guardadas junto dos sujeitos
- A capacidade (*capability*) de cada sujeito corresponde à sua linha na matriz:
 - *Alice capability:* File1: read, write; File2: read, write; Program1: execute
 - *Bob capability:* File1: read; File3: read, write
 - *Charlie capability:* File2: read; Program1: execute, write
- Vantagens:
 - Facilidade na obtenção das permissões associadas a um sujeito
 - Em ambientes distribuídos elimina a necessidade de múltiplas autenticações
- Desvantagens:
 - Para obter lista de acessos a objetos obriga a pesquisar todas as capacidades
 - Possibilidade de cópia e uso fraudulento

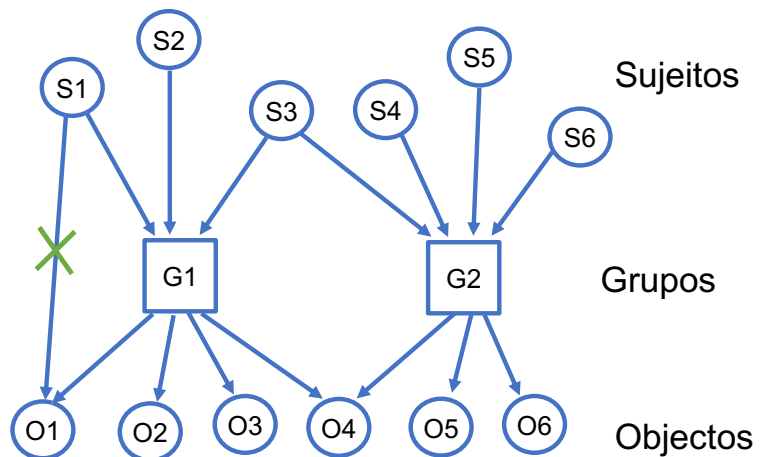
Implementação da matriz de acessos (3)

Access Control List (ACL):

- As permissões são guardadas junto dos objectos.
- A ACL de cada objecto corresponde à sua coluna na matriz:
 - ACL para File1: *Alice: read, write; Bob: read*
 - ACL para File2: *Alice: read, write; Charlie: read*
 - ACL para File3: *Bob: read, write*
 - ACL para Program1: *Alice: execute, Charlie: execute, write*
- Vantagens:
 - Facilidade na obtenção das permissões associadas a um objecto
 - Ao eliminar um objeto elimina-se todas as permissões a ele associadas
- Desvantagens:
 - Para saber todas as permissões de um sujeito é necessário pesquisar todas as ACLs

Permissões para grupos

- Motivação: facilitar a gestão de sujeitos, agrupando sujeitos com permissões semelhantes num grupo.
- Os grupos funcionam como uma camada intermédia na definição de controlos de acesso.
- As permissões podem ser associadas a grupos ou individualmente aos sujeitos.
- A verificação de controlo de acesso passa a ser feita também em função do sujeito ser membro ou não de um grupo.
- É possível usar permissões negativas para um determinado sujeito dentro de um grupo.

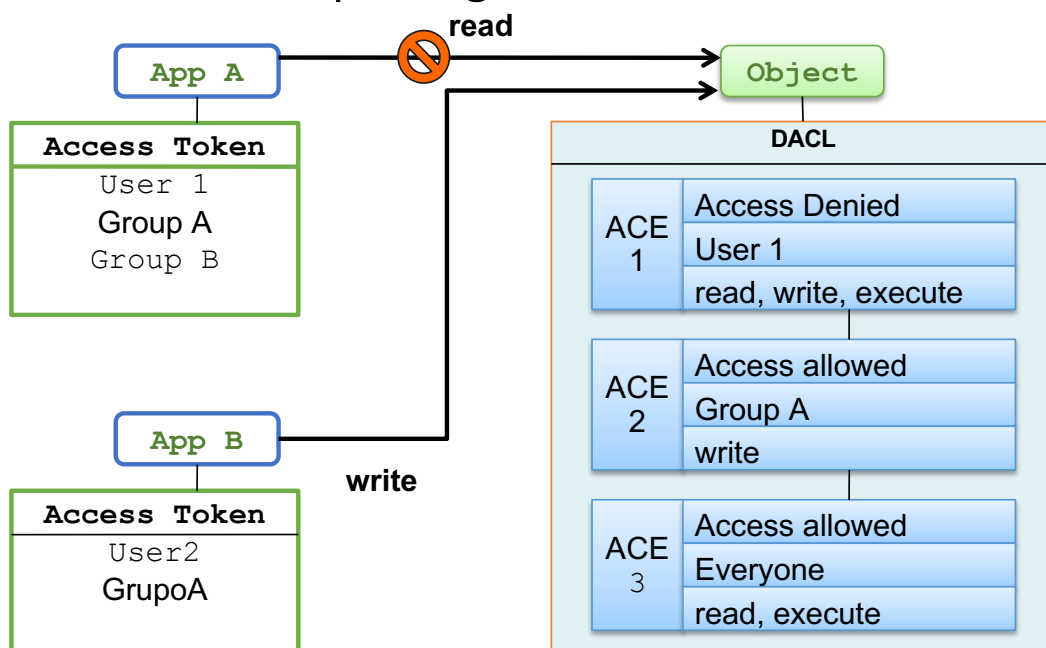


Caso prático: Controlo de acessos em Windows

- Após *login* é atribuído ao utilizador um *access token*
 - Em cada *access token* estão presentes *security identifiers (SID)* com a identificação do utilizador e dos grupos a que pertence
- Após a criação de um objecto (recurso) é-lhe associado um *security descriptor* com:
 - O SID do seu dono
 - Discretionary Access Control List (DACL)
 - System Access Control List (SACL) com a política do sistema para auditar o acesso ao objecto e o “nível de integridade” do mesmo
- Uma ACL é uma lista de Access Control Entry (ACE), onde consta:
 - SID (utilizador ou grupo), Permissão ou Negação, Acções

Controlo de acessos através de DACL

- Para determinar se o acesso é autorizado ou não, a DACL é percorrida até à negação de uma das acções ou permissão de todas as acções requeridas
- À cabeça ficam as ACE que negam

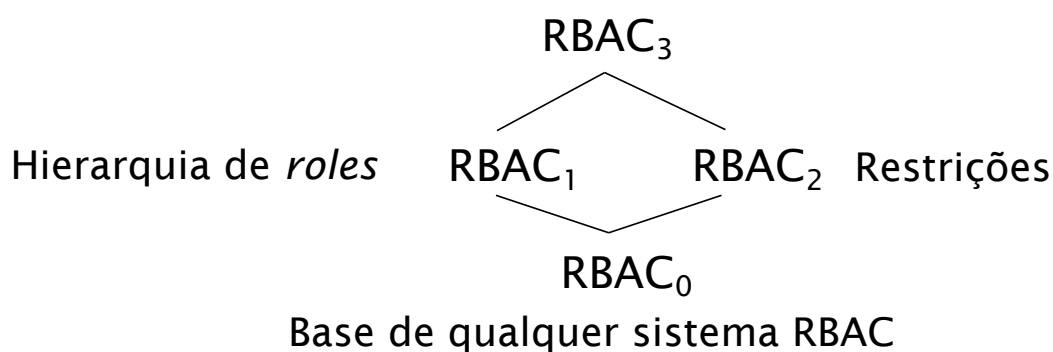


Modelos RBAC – Motivação

- Para efeitos de controlo de acessos é mais relevante saber as responsabilidades do utilizador do que quem ele é
- Nas organizações, as permissões estão tipicamente associadas a roles e não a pessoas
 - Funcionário da tesouraria; Director de departamento
- As permissões associadas a roles mudam com menos frequência do que a associações entre utilizadores e roles
- As organizações valorizam princípios como o da separação de poderes
 - e.g. Quem pede material para projetos não deve ser a mesma pessoa que autoriza o pagamento

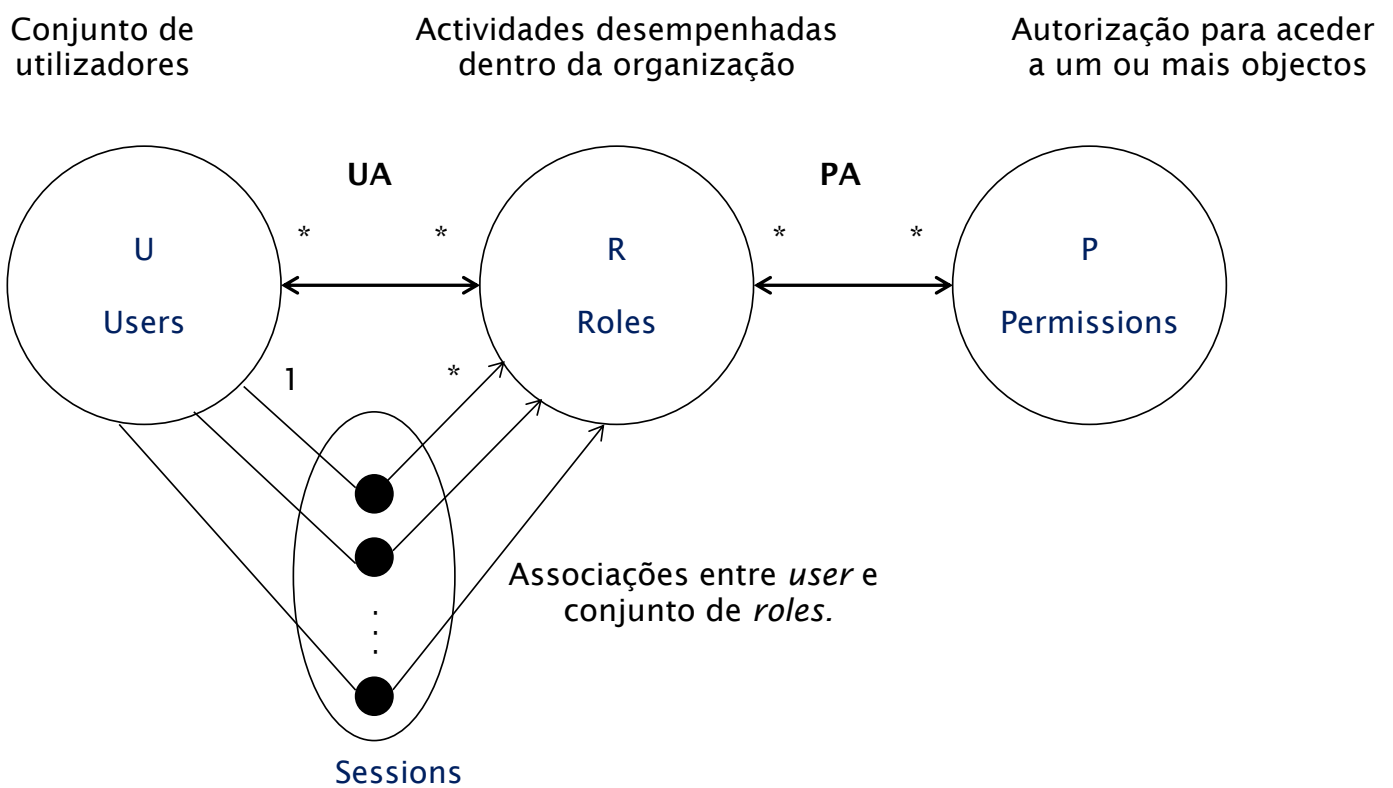
Família de modelos RBAC

- São quatro os modelos da família RBAC



- Servem de referência para caracterizar os sistemas que usam este tipo de controlo de acessos

RBAC₀



RBAC₀

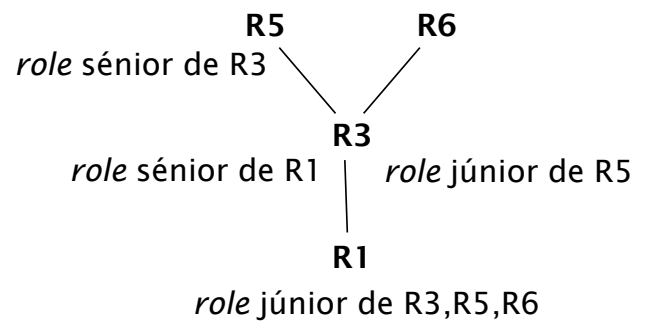
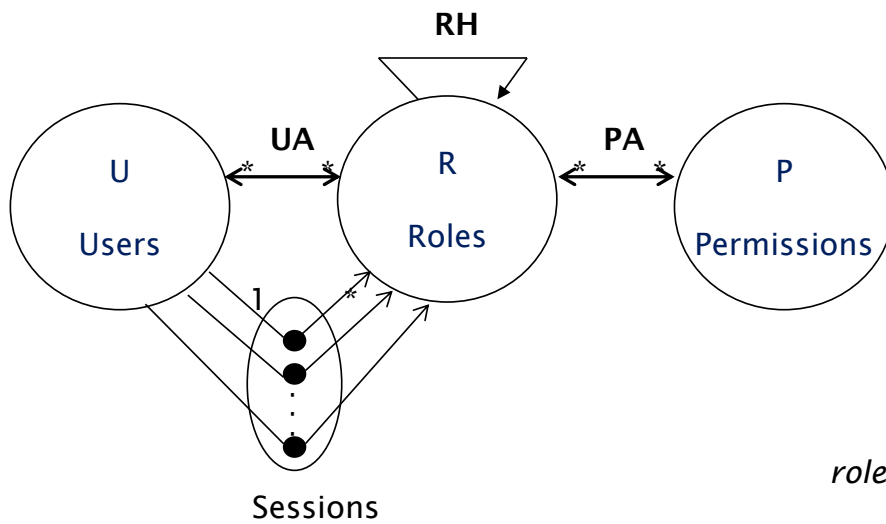
- As relações user assignment (UA) e permission assignment (PA) são a base do modelo
 - Ambas são relações de muitos para muitos
- As políticas são expressas pela concretização destas relações
- As permissões são sempre positivas
- As permissões de um utilizador são a reunião das permissões dos roles activos na sessão
 - Relação de um para um entre utilizadores e sessões
 - Relação de um para muitos entre sessões e *roles*
 - Cada utilizador activa apenas o conjunto de roles que lhe interessa

RBAC₀ – Definições

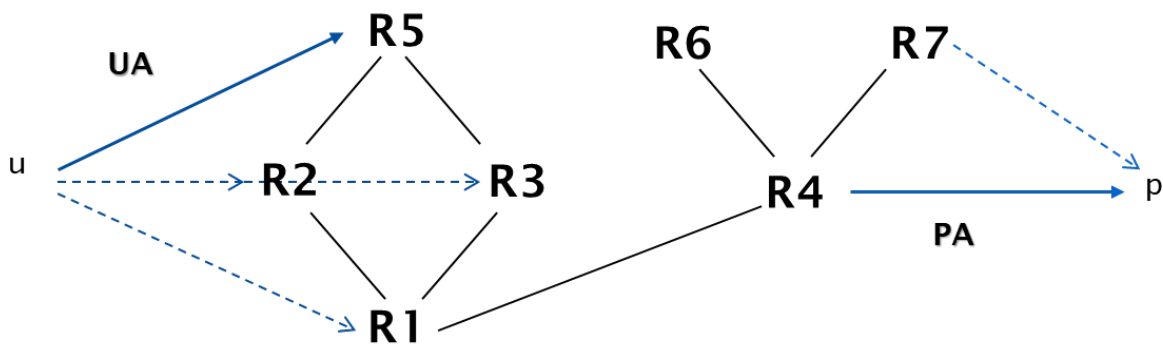
- U, R, P e S (*users, roles, permissions e sessions*)
- $UA \subseteq U \times R$, relação de muitos para muitos entre *users* e *roles*
 - $U = \{u_1, u_2\} \quad R = \{r_1, r_2\}$
 - $U \times R = \{ (u_1, r_1), (u_1, r_2), (u_2, r_1), (u_2, r_2) \}$
- $PA \subseteq P \times R$, relação de muitos para muitos entre *permissions* e *roles*
- $user : S \rightarrow U$, função que associa cada sessão s_i a um utilizador $user(s_i)$ (constante ao longo da sessão)
- $roles : S \rightarrow 2^R$, função que associa cada sessão s_i a um conjunto de *roles*
 $2^R = \text{power set de } R, \text{ conjunto dos subconjuntos de } R \{ \{\}, \{r_1\}, \{r_2\}, \{r_1, r_2\} \}$
- $roles(s_i) \subseteq \{ r \mid (user(s_i), r) \in UA \}$ (pode mudar ao longo da sessão)
- Cada sessão s_i tem as permissões $\cup_{r \in roles(s_i)} \{ p \mid (p, r) \in PA \}$

RBAC₁

- Introduz o conceito de hierarquia de *roles*



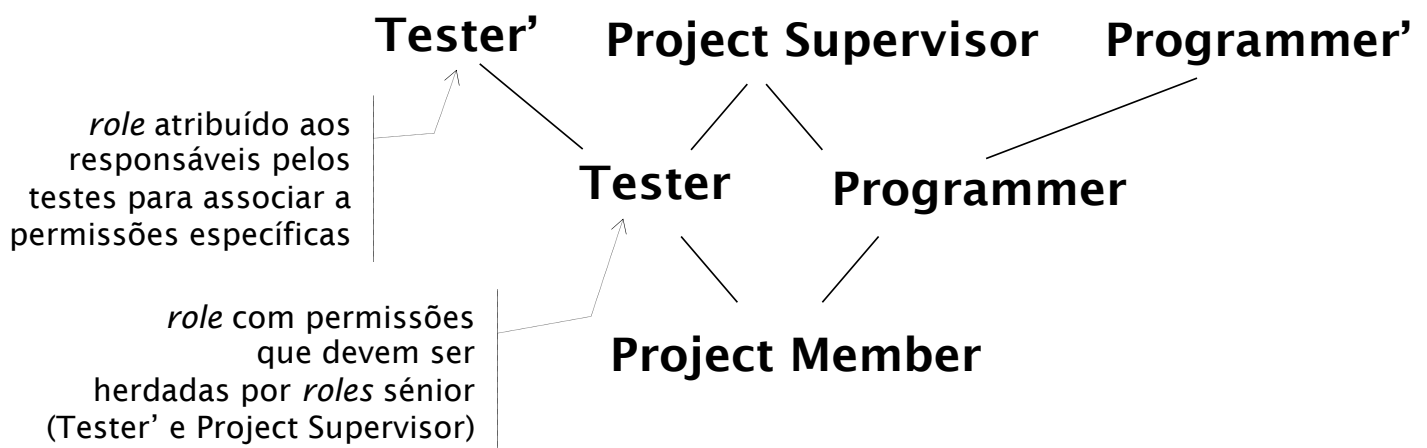
RBAC₁



- O utilizador escolhe qual o *role* que quer activar, herdando os *roles* júnior desse
- As permissões são as directamente associadas ao *role* do utilizador mais as dos *roles* júnior

RBAC₁ – exemplo

- Pode ser útil limitar os *roles* herdados (*private roles*)
 - e.g. Resultados intermédios de trabalho em progresso podem não fazer sentido serem analisados pelo responsável de projecto



RBAC₁ – Definições

- U, R, P, S, UA, PA
- $RH \subseteq R \times R$, relação de dominância entre roles, representada por \geq
- $roles : S \rightarrow 2^R$, função modificada de RBAC₀ para ter como requisito
- $roles(s_i) \subseteq \{ r \mid (\exists r' \geq r) [(user(s_i), r') \in UA] \}$ (pode mudar ao longo da sessão)
- Cada sessão s_i tem as permissões
$$\bigcup_{r \in roles(s_i)} \{ p \mid (\exists r'' \leq r) [(p, r'') \in PA] \}$$

RBAC₂

- Restrições são um mecanismo para impôr regras da organização
 - Podem ser aplicadas às relações UA e PA, e às funções *user* e *roles*
- Têm a forma de predicados, retornando “aceite” ou “não aceite”
- Exemplos de restrições
 - Separação de deveres
 - Pode ser garantido de forma estática (relação UA) ou dinâmica (função *roles*)
 - Cardinalidade
 - Pré-requisitos

RBAC₃

- Combina RBAC₁ e RBAC₂ suportando hierarquia de *roles* e restrições
- Num cenário onde a administração é delegada em terceiros pode ser necessário impor restrições
 - E.g. Dois ou mais *roles* sénior não podem ter em comum determinados *roles* júnior

Resumo

- RBAC \neq grupos
 - Um *role* representa um conjunto de permissões e não um conjunto de utilizadores
 - Os utilizadores escolhem qual o role que pretendem desempenhar
- O modelo suporta cenários simples e complexos
- Dada a natureza neutral quanto ao tipo de políticas impostas pelo modelo, este pode ser usado para a sua própria gestão

Legislação nacional

- A resolução do Conselho de Ministros n.º 41/2018 [2] relaciona o RGPD e aspetos tecnológicos
- Criação de perfis com privilégios mínimos
 - Cada tipo de perfil é definido em função do Tipo de Dado Pessoal a que acede e Ação que pode efetuar sobre o Dado Pessoal
- Processo de registo de tentativas de acesso a dados excluídos dos privilégios associados ao perfil (qualquer perfil, incluindo o dos administradores)
 - Alarmística a partir de um determinado número de tentativas (por exemplo, 3 tentativas), a notificar ao encarregado da proteção de dados da organização
 - Deve ser guardado registo de atividade (*log*) de todas as ações que um utilizador efetue sobre dados pessoais, independentemente do seu perfil e função

RBAC na prática

- Biblioteca node.js
 - <https://www.npmjs.com/package/accesscontrol>
- Security-Enhanced Linux
 - [https://wiki.gentoo.org/wiki/SELinux/Role-based access control](https://wiki.gentoo.org/wiki/SELinux/Role-based_access_control)
- Cloud-based Access Control
 - Ex: <https://www.parkmycloud.com/blog/cloud-user-management/>
- Kubernetes
 - <https://learnk8s.io/rbac-kubernetes>

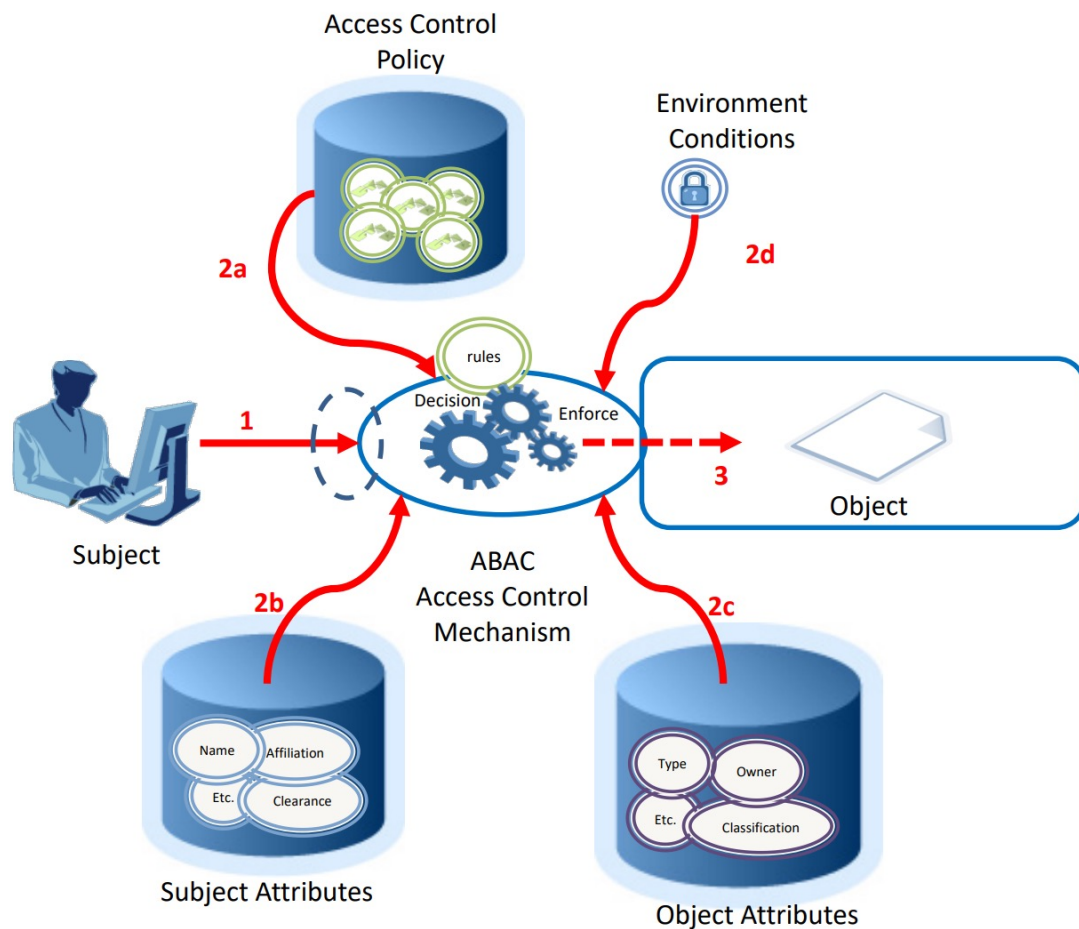
Algumas limitações

- “Explosão” de roles
 - As particularidades dos utilizadores dão origem a papéis com apenas alguns membros, contribuindo para o aumento de papéis
- Efeitos indesejados da herança de roles
 - Estruturar e gerir as hierarquias de papéis requer uma compreensão da herança de permissões para evitar efeitos colaterais inesperados que resultem em sub ou excesso de permissões
- Falta de interoperabilidade
 - O significado das funções em termos de terminologia e permissões deve ser partilhado entre diferentes departamentos, sucursais ou parceiros de negócios; chegar a um consenso pode não ser trivial
- Rigidez
 - Uma empresa pode não saber quais as permissões que os utilizadores devem ter até que a necessidade surja, especialmente em situações de emergência

Franqueira, Virginia N. L. and Roel Wieringa. “Role-Based Access Control in Retrospect.” *Computer* 45 (2012): 81-88.
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6133255&tag=1>

26

ABAC: Outro modelo (Attribute Based Access Control)



Exemplo

(3) Considere a seguinte política definida sobre o modelo $RBAC_1$:

- $U = \{u_0, u_1, u_2, u_4\}$
- $R = \{r_0, r_1, r_2, r_3, r_4, r_5\}$
- $P = \{p_a, p_b, p_c, p_d\}$
- $UA = \{(u_0, r_0), (u_1, r_3), (u_1, r_4), (u_2, r_4), (u_4, r_5)\}$
- $\{r_0 \preceq r_1, r_0 \preceq r_2, r_1 \preceq r_3, r_2 \preceq r_4, r_1 \preceq r_5, r_2 \preceq r_5\} \subseteq RH$
- $PA = \{(r_0, p_a), (r_0, p_d), (r_3, p_b), (r_4, p_c)\}$

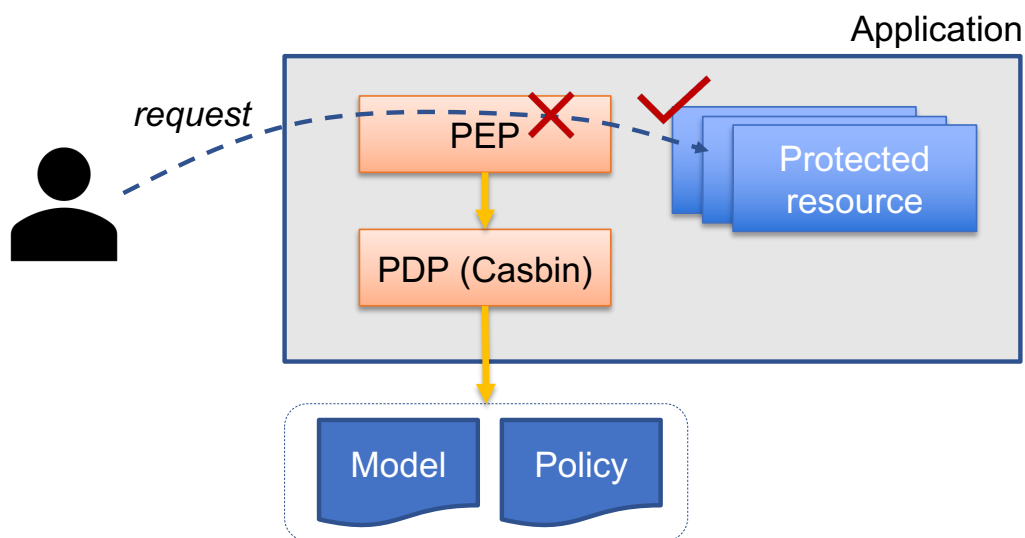
5.1. Justifique qual ou quais os utilizadores que podem aceder a um recurso que exija as permissões p_a e p_c .

Casbin

<https://casbin.org/>

Biblioteca Casbin

- Biblioteca para integração de controlo de acessos em aplicações
- Suporta vários modelos de controlo de acessos (ACL, RBAC, ...)
- Modelos e políticas definidos em ficheiros de configuração externos à aplicação



Exemplo - ACL

acl_model.conf

```
[request_definition]  
r = sub, obj, act
```




Formato do pedido

```
[policy_definition]  
p = sub, obj, act
```




Formato da política

```
[policy_effect]  
e = some(where (p.eft == allow))
```




Efeito da política, aceita qualquer regra "allow"

```
[matchers]  
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
```



policy_app.csv

```
p, alice, data1, read  
p, bob, data2, write
```



Regra do modelo com base no pedido e na definição da política

Políticas:

- 'alice' tem 'read' sobre o objeto 'data1'
- 'bob' tem 'write' sobre o objeto 'data2'

Exemplo - RBAC

rbac_model.conf

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

rbac_policy_app.csv

```
p, data2_admin, data2, read
g, alice, data2_admin
```



Políticas:

- 'data2_admin' tem 'read' sobre o objeto 'data2'
- 'alice' é membro do role 'data2_admin'

<https://casbin.org/docs/rbac>

<https://casbin.org/docs/rbac-96>

Demo

- Editor Casbin: <https://casbin.org/editor>
- Exemplo com ACL e RBAC

Referências

- [1] R. S. Sandhu, et al. “Role-Based Access Control Models”, IEEE Computer 29(2): 38-47, IEEE Press, 1996.
- [2] Resolução do Conselho de Ministros n.º 41/2018,
<https://dre.pt/home/-/dre/114937034/details/maximized>