Capítulo 6

Modelos de controle de acesso

Em um sistema computacional, o controle de acesso consiste em mediar cada solicitação de acesso de um usuário autenticado a um recurso ou dado mantido pelo sistema, para determinar se aquela solicitação deve ser autorizada ou negada [Samarati and De Capitani di Vimercati, 2001]. Praticamente todos os recursos de um sistema operacional típico estão submetidos a um controle de acesso, como arquivos, áreas de memória, semáforos, portas de rede, dispositivos de entrada/saída, etc.

6.1 Terminologia

Esta seção define alguns termos usuais na área de controle de acesso:

Sujeito: são todas aquelas entidades que executam ações no sistema, como processos, *threads* ou transações. Normalmente um sujeito opera em nome de um usuário, que pode ser um ser humano ou outro sistema computacional externo.

Objeto: são as entidades passivas sujeitas às ações dos sujeitos, como arquivos, áreas de memória, registros em um banco de dados ou outros recursos. Em alguns casos, um sujeito pode ser visto como objeto por outro sujeito (por exemplo, quando um sujeito deve enviar uma mensagem a outro sujeito).

Acesso: ação realizada por um sujeito sobre um objeto. Por exemplo, um acesso de um processo a um arquivo, um envio de pacote de rede através de uma porta UDP, execução de um programa, etc.

Autorização: é a permissão para que um sujeito realize uma determinada ação sobre um objeto. Existem autorizações positivas (que permitem uma ação) e autorizações negativas (que negam uma ação).

Tanto sujeitos quanto objetos e autorizações podem ser organizados em grupos e hierarquias, para facilitar a gerência da segurança.

6.2 Políticas, modelos e mecanismos

Uma *política de controle de acesso* é uma visão abstrata das permissões de acesso a recursos (objetos) pelos usuários (sujeitos) de um sistema. Essa política consiste

basicamente de um conjunto de regras definindo os acessos possíveis aos recursos do sistema e eventuais condições necessárias para permitir cada acesso. Por exemplo, as regras a seguir poderiam constituir parte da política de segurança de um sistema de informações médicas:

1. Médicos podem consultar os prontuários de seus pacientes;
2. Médicos podem modificar os prontuários de seus pacientes enquanto estes estiverem internados;
3. O supervisor geral pode consultar os prontuários de todos os pacientes;
4. Enfermeiros podem consultar apenas os prontuários dos pacientes de sua seção e somente durante seu período de turno;
5. Assistentes não podem consultar prontuários;
6. Prontuários de pacientes de planos de saúde privados podem ser consultados pelo responsável pelo respectivo plano de saúde no hospital;
7. Pacientes podem consultar seus próprios prontuários (aceitar no máximo 30 pacientes simultâneos).

As regras ou definições individuais de uma política são denominadas *autorizações*. Uma política de controle de acesso pode ter autorizações baseadas em *identidades* (como sujeitos e objetos) ou em outros *atributos* (como idade, sexo, tipo, preço, etc.); as autorizações podem ser *individuais* (a sujeitos) ou *coletivas* (a grupos); também podem existir autorizações *positivas* (permitindo o acesso) ou *negativas* (negando o acesso); por fim, uma política pode ter autorizações dependentes de *condições externas* (como o horário ou a carga do sistema). Além da política de acesso aos objetos, também deve ser definida uma *política administrativa*, que define quem pode modificar/gerenciar as políticas vigentes no sistema [Samarati and De Capitani di Vimercati, 2001].

O conjunto de autorizações de uma política deve ser ao mesmo tempo *completo*, cobrindo todas as possibilidades de acesso que vierem a ocorrer no sistema, e *consistente*, sem regras conflitantes entre si (por exemplo, uma regra que permita um acesso e outra que negue esse mesmo acesso). Além disso, toda política deve buscar respeitar o *princípio do privilégio mínimo* [Saltzer and Schroeder, 1975], segundo o qual um usuário nunca deve receber mais autorizações que aquelas que necessita para cumprir sua tarefa. A construção e validação de políticas de controle de acesso é um tema complexo, que está fora do escopo deste texto, sendo melhor descrito em [di Vimercati et al., 2005, 2007].

As políticas de controle de acesso definem de forma abstrata como os sujeitos podem acessar os objetos do sistema. Existem muitas formas de se definir uma política, que podem ser classificadas em quatro grandes classes: políticas *discricionárias*, políticas *obrigatórias*, políticas *baseadas em domínios* e políticas *baseadas em papéis* [Samarati and De Capitani di Vimercati, 2001]. As próximas seções apresentam com mais detalhe cada uma dessas classes de políticas.

Geralmente a descrição de uma política de controle de acesso é muito abstrata e informal. Para sua implementação em um sistema real, ela precisa ser descrita de uma forma precisa, através de um *modelo de controle de acesso*. Um modelo de controle de acesso é uma representação lógica ou matemática da política, de forma a facilitar

sua implementação e permitir a análise de eventuais erros. Em um modelo de controle de acesso, as autorizações de uma política são definidas como relações lógicas entre *atributos do sujeito* (como seus identificadores de usuário e grupo) *atributos do objeto* (como seu caminho de acesso ou seu proprietário) e eventuais condições externas (como o horário ou a carga do sistema). Nas próximas seções, para cada classe de políticas de controle de acesso apresentada serão discutidos alguns modelos aplicáveis à mesma.

Por fim, os *mecanismos de controle de acesso* são as estruturas necessárias à implementação de um determinado modelo em um sistema real. Como é bem sabido, é de fundamental importância a separação entre políticas e mecanismos, para permitir a substituição ou modificação de políticas de controle de acesso de um sistema sem incorrer em custos de modificação de sua implementação. Assim, um mecanismo de controle de acesso ideal deveria ser capaz de suportar qualquer política de controle de acesso.

6.3 Políticas discricionárias

As políticas discricionárias (DAC - *Discretionary Access Control*) se baseiam na atribuição de permissões de forma individualizada, ou seja, pode-se claramente conceder (ou negar) a um sujeito específico s a permissão de executar a ação a sobre um objeto específico o. Em sua forma mais simples, as regras de uma política discricionária têm a forma $\langle s, o, +a\rangle$ ou $\langle s, o, -a\rangle$, para respectivamente autorizar ou negar a ação a do sujeito s sobre o objeto o (também podem ser definidas regras para grupos de usuários e/ou de objetos devidamente identificados). Por exemplo:

- O usuário `Beto` pode ler e escrever arquivos em `/home/beto`
- Usuários do grupo `admin` podem ler os arquivos em `/suporte`

O responsável pela administração das permissões de acesso a um objeto pode ser o seu proprietário ou um administrador central. A definição de quem estabelece as regras da política de controle de acesso é inerente a uma política administrativa, independente da política de controle de acesso em si[1].

6.3.1 Matriz de controle de acesso

O modelo matemático mais simples e conveniente para representar políticas discricionárias é a *Matriz de Controle de Acesso*, proposta em [Lampson, 1971]. Nesse modelo, as autorizações são dispostas em uma matriz, cujas linhas correspondem aos sujeitos do sistema e cujas colunas correspondem aos objetos. Em termos formais, considerando um conjunto de sujeitos $\mathbb{S} = \{s_1, s_2, \ldots, s_m\}$, um conjunto de objetos $\mathbb{O} = \{o_1, o_2, \ldots, o_n\}$ e um conjunto de ações possíveis sobre os objetos $\mathbb{A} = \{a_1, a_2, \ldots, a_p\}$, cada elemento M_{ij} da matriz de controle de acesso é um subconjunto (que pode ser vazio) do conjunto de ações possíveis, que define as ações que $s_i \in \mathbb{S}$ pode efetuar sobre $o_j \in \mathbb{O}$:

[1] Muitas políticas de controle de acesso discricionárias são baseadas na noção de que cada recurso do sistema possui um proprietário, que decide quem pode acessar o recurso. Isso ocorre por exemplo nos sistemas de arquivos, onde as permissões de acesso a cada arquivo ou diretório são definidas pelo respectivo proprietário. Contudo, a noção de "proprietário" de um recurso não é essencial para a construção de políticas discricionárias [Shirey, 2000].

$$\forall s_i \in \mathbb{S}, \forall o_j \in \mathbb{O}, M_{ij} \subseteq \mathbb{A}$$

Por exemplo, considerando um conjunto de sujeitos $\mathbb{S} = \{Alice, Beto, Carol, Davi\}$, um conjunto de objetos $\mathbb{O} = \{file_1, file_2, program_1, socket_1\}$ e um conjunto de ações $\mathbb{A} = \{read, write, execute, remove\}$, podemos ter uma matriz de controle de acesso como a apresentada na Tabela 6.1.

	$file_1$	$file_2$	$program_1$	$socket_1$
Alice	*read* *write* *remove*	*read* *write*	*execute*	*write*
Beto	*read* *write*	*read* *write* *remove*	*read*	
Carol		*read*	*execute*	*read* *write*
Davi	*read*	*append*	*read*	*read* *append*

Tabela 6.1: Uma matriz de controle de acesso

Apesar de simples, o modelo de matriz de controle de acesso é suficientemente flexível para suportar políticas administrativas. Por exemplo, considerando uma política administrativa baseada na noção de proprietário do recurso, poder-se-ia considerar que cada objeto possui um ou mais proprietários (*owner*), e que os sujeitos podem modificar as entradas da matriz de acesso relativas aos objetos que possuem. Uma matriz de controle de acesso com essa política administrativa é apresentada na Tabela 6.2.

Embora seja um bom modelo conceitual, a matriz de acesso é inadequada para implementação. Em um sistema real, com milhares de sujeitos e milhões de objetos, essa matriz pode se tornar gigantesca e consumir muito espaço. Como em um sistema real cada sujeito tem seu acesso limitado a um pequeno grupo de objetos (e vice-versa), a matriz de acesso geralmente é esparsa, ou seja, contém muitas células vazias. Assim, algumas técnicas simples podem ser usadas para implementar esse modelo, como as tabelas de autorizações, as listas de controle de acesso e as listas de capacidades [Samarati and De Capitani di Vimercati, 2001], explicadas a seguir.

6.3.2 Tabela de autorizações

Na abordagem conhecida como **Tabela de Autorizações**, as entradas não vazias da matriz são relacionadas em uma tabela com três colunas: *sujeitos*, *objetos* e *ações*, onde cada tupla da tabela corresponde a uma autorização. Esta abordagem é muito utilizada em sistemas gerenciadores de bancos de dados (DBMS - *Database Management Systems*), devido à sua facilidade de implementação e consulta nesse tipo de ambiente. A Tabela

	$file_1$	$file_2$	$program_1$	$socket_1$
Alice	*read* *write* *remove* ***owner***	*read* *write*	*execute*	*write*
Beto	*read* *write*	*read* *write* *remove* ***owner***	*read* ***owner***	
Carol		*read*	*execute*	*read* *write*
Davi	*read*	*write*	*read*	*read* *write* ***owner***

Tabela 6.2: Uma matriz de controle de acesso com política administrativa

6.3 mostra como ficaria a matriz de controle de acesso da Tabela 6.2 sob a forma de uma tabela de autorizações.

6.3.3 Listas de controle de acesso

Outra abordagem usual é a **Lista de Controle de Acesso**. Nesta abordagem, para cada objeto é definida uma lista de controle de acesso (ACL - *Access Control List*), que contém a relação de sujeitos que podem acessá-lo, com suas respectivas permissões. Cada lista de controle de acesso corresponde a uma coluna da matriz de controle de acesso. Como exemplo, as listas de controle de acesso relativas à matriz de controle de acesso da Tabela 6.2 seriam:

Sujeito	Objeto	Ação	Sujeito	Objeto	Ação
Alice	$file_1$	*read*	Beto	$file_2$	*owner*
Alice	$file_1$	*write*	Beto	$program_1$	*read*
Alice	$file_1$	*remove*	Beto	$socket_1$	*owner*
Alice	$file_1$	*owner*	Carol	$file_2$	*read*
Alice	$file_2$	*read*	Carol	$program_1$	*execute*
Alice	$file_2$	*write*	Carol	$socket_1$	*read*
Alice	$program_1$	*execute*	Carol	$socket_1$	*write*
Alice	$socket_1$	*write*	Davi	$file_1$	*read*
Beto	$file_1$	*read*	Davi	$file_2$	*write*
Beto	$file_1$	*write*	Davi	$program_1$	*read*
Beto	$file_2$	*read*	Davi	$socket_1$	*read*
Beto	$file_2$	*write*	Davi	$socket_1$	*write*
Beto	$file_2$	*remove*	Davi	$socket_1$	*owner*

Tabela 6.3: Tabela de autorizações

$$
\begin{aligned}
ACL(file_1) &= \{\ Alice : (read, write, remove, owner),\\
&\quad\ \ Beto : (read, write),\\
&\quad\ \ Davi : (read)\ \}\\
ACL(file_2) &= \{\ Alice : (read, write),\\
&\quad\ \ Beto : (read, write, remove, owner),\\
&\quad\ \ Carol : (read),\\
&\quad\ \ Davi : (write)\ \}\\
ACL(program_1) &= \{\ Alice : (execute),\\
&\quad\ \ Beto : (read, owner),\\
&\quad\ \ Carol : (execute),\\
&\quad\ \ Davi : (read)\ \}\\
ACL(socket_1) &= \{\ Alice : (write),\\
&\quad\ \ Carol : (read, write),\\
&\quad\ \ Davi : (read, write, owner)\ \}
\end{aligned}
$$

Esta forma de implementação é a mais frequentemente usada em sistemas operacionais, por ser simples de implementar e bastante robusta. Por exemplo, o sistema de arquivos associa uma ACL a cada arquivo ou diretório, para indicar quem são os sujeitos autorizados a acessá-lo. Em geral, somente o proprietário do arquivo pode modificar sua ACL, para incluir ou remover permissões de acesso.

6.3.4 Listas de capacidades

Uma terceira abordagem possível para a implementação da matriz de controle de acesso é a **Lista de Capacidades** (CL - *Capability List*), ou seja, uma lista de objetos que um dado sujeito pode acessar e suas respectivas permissões sobre os mesmos. Cada lista de capacidades corresponde a uma linha da matriz de acesso. Como exemplo, as listas de capacidades correspondentes à matriz de controle de acesso da Tabela 6.2 seriam:

$$
\begin{aligned}
CL(Alice) &= \{\ file_1 : (read, write, remove, owner),\\
&\quad\ file_2 : (read, write),\\
&\quad\ program_1 : (execute),\\
&\quad\ socket_1 : (write)\ \}\\
CL(Beto) &= \{\ file_1 : (read, write),\\
&\quad\ file_2 : (read, write, remove, owner),\\
&\quad\ program_1 : (read, owner)\ \}\\
CL(Carol) &= \{\ file_2 : (read),\\
&\quad\ program_1 : (execute),\\
&\quad\ socket_1 : (read, write)\ \}\\
CL(Davi) &= \{\ file_1 : (read),\\
&\quad\ file_2 : (write),\\
&\quad\ program_1 : (read),\\
&\quad\ socket_1 : (read, write, owner)\ \}
\end{aligned}
$$

Uma capacidade pode ser vista como uma ficha ou *token*: sua posse dá ao proprietário o direito de acesso ao objeto em questão. Capacidades são pouco usadas em sistemas operacionais, devido à sua dificuldade de implementação e possibilidade de fraude, pois uma capacidade mal implementada pode ser transferida deliberadamente a outros sujeitos, ou modificada pelo próprio proprietário para adicionar mais permissões a ela. Outra dificuldade inerente às listas de capacidades é a administração das autorizações: por exemplo, quem deve ter permissão para modificar uma lista de capacidades, e como retirar uma permissão concedida anteriormente a um sujeito? Alguns sistemas operacionais que implementam o modelo de capacidades são discutidos na Seção 7.4.

6.4 Políticas obrigatórias

Nas *políticas obrigatórias* (MAC - *Mandatory Access Control*) o controle de acesso é definido por regras globais incontornáveis, que não dependem das identidades dos sujeitos e objetos nem da vontade de seus proprietários ou mesmo do administrador do sistema [Samarati and De Capitani di Vimercati, 2001]. Essas regras são normalmente baseadas em atributos dos sujeitos e/ou dos objetos, como mostram estes exemplos bancários (fictícios):

- Cheques com valor acima de R$ 5.000,00 devem ser obrigatoriamente depositados e não podem ser descontados;
- Clientes com renda mensal acima de R$3.000,00 não têm acesso ao crédito consignado.

Uma das formas mais usuais de política obrigatória são as *políticas multinível* (MLS - *Multi-Level Security*), que se baseiam na classificação de sujeitos e objetos do sistema em *níveis de segurança* (*clearance levels*, $\mathbb{S}$) e na definição de regras usando esses níveis. Um exemplo bem conhecido de escala de níveis de segurança é aquela usada pelo governo britânico para definir a confidencialidade de um documento:

- *TS*: *Top Secret* (*Ultrassecreto*)
- *S*: *Secret* (*Secreto*)
- *C*: *Confidential* (*Confidencial*)
- *R*: *Restrict* (*Reservado*)
- *U*: *Unclassified* (*Público*)

Em uma política MLS, considera-se que os níveis de segurança estão ordenados entre si (por exemplo, $U < R < C < S < TS$) e são associados a todos os sujeitos e objetos do sistema, sob a forma de *habilitação* dos sujeitos ($h(s_i) \in \mathbb{S}$) e *classificação* dos objetos ($c(o_j) \in \mathbb{S}$). As regras da política são então estabelecidas usando essas habilitações e classificações, como mostram os modelos de Bell-LaPadula e de Biba, descritos a seguir.

Além das políticas multinível, existem também políticas denominadas *multilaterais*, nas quais o objetivo é evitar fluxos de informação indevidos entre departamentos ou áreas distintas em uma organização. *Chinese Wall* e *Clark-Wilson* são exemplos dessa família de políticas [Anderson, 2008].

6.4.1 Modelo de Bell-LaPadula

Um modelo de controle de acesso que permite formalizar políticas multinível é o de *Bell-LaPadula* [Bell and LaPadula, 1974], usado para garantir a confidencialidade das informações. Esse modelo consiste basicamente de duas regras:

No-Read-Up ("não ler acima", ou "propriedade simples"): impede que um sujeito leia objetos que se encontrem em níveis de segurança acima do seu. Por exemplo, um sujeito habilitado como confidencial (*C*) somente pode ler objetos cuja classificação seja confidencial (*C*), reservada (*R*) ou pública (*U*). Considerando um sujeito s e um objeto o, formalmente temos:

$$request(s, o, read) \iff h(s) \geq c(o)$$

No-Write-Down ("não escrever abaixo", ou "propriedade $\star$"): impede que um sujeito escreva em objetos abaixo de seu nível de segurança, para evitar o "vazamento" de informações dos níveis superiores para os inferiores. Por exemplo, um sujeito habilitado como confidencial somente pode escrever em objetos cuja classificação seja confidencial, secreta ou ultrassecreta. Formalmente, temos:

$$request(s, o, write) \iff h(s) \leq c(o)$$

As regras da política de Bell-LaPadula estão ilustradas na Figura 6.1. Pode-se perceber que a política obrigatória representada pelo modelo de Bell-LaPadula visa proteger a *confidencialidade* das informações do sistema, evitando que estas fluam dos níveis superiores para os inferiores. Todavia, nada impede um sujeito com baixa habilitação escrever sobre um objeto de alta classificação, destruindo seu conteúdo.

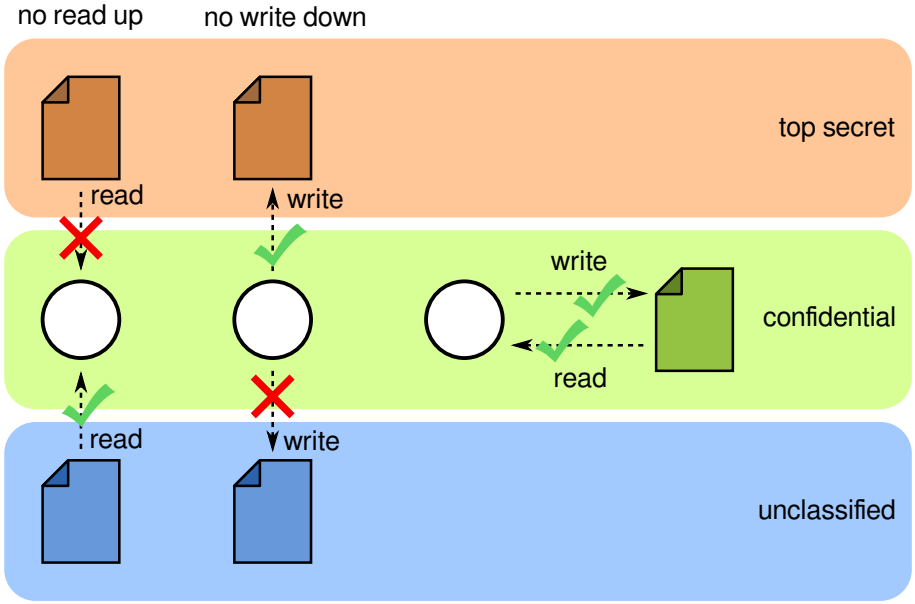

Figura 6.1: Política de Bell-LaPadula.

6.4.2 Modelo de Biba

Para garantir a *integridade* das informações, um modelo dual ao de Bell-LaPadula foi proposto por Kenneth Biba [Biba, 1977]. Esse modelo define níveis de integridade $i(x) \in \mathbb{I}$ para sujeitos e objetos (como *Baixa*, *Média*, *Alta* e *Sistema*, com $B < M < A < S$), e também possui duas regras básicas:

No-Write-Up ("não escrever acima", ou "propriedade simples de integridade"): impede que um sujeito escreva em objetos acima de seu nível de integridade, preservando-os íntegros. Por exemplo, um sujeito de integridade média (M) somente pode escrever em objetos de integridade baixa (B) ou média (M). Formalmente, temos:

$$request(s, o, write) \iff i(s) \geq i(o)$$

No-Read-Down ("não ler abaixo", ou "propriedade $\star$ de integridade"): impede que um sujeito leia objetos em níveis de integridade abaixo do seu, para não correr o risco de ler informação duvidosa. Por exemplo, um sujeito com integridade alta (A) somente pode ler objetos com integridade alta (A) ou de sistema (S). Formalmente, temos:

$$request(s, o, read) \iff i(s) \leq i(o)$$

As regras da política de Biba estão ilustradas na Figura 6.2. Essa política obrigatória evita violações de integridade, mas não garante a confidencialidade das informações. Para que as duas políticas (confidencialidade e integridade) possam funcionar em conjunto, é necessário portanto associar a cada sujeito e objeto do sistema um nível de confidencialidade e um nível de integridade, possivelmente distintos, ou seja, combinar as políticas de Bell-LaPadula e Biba.

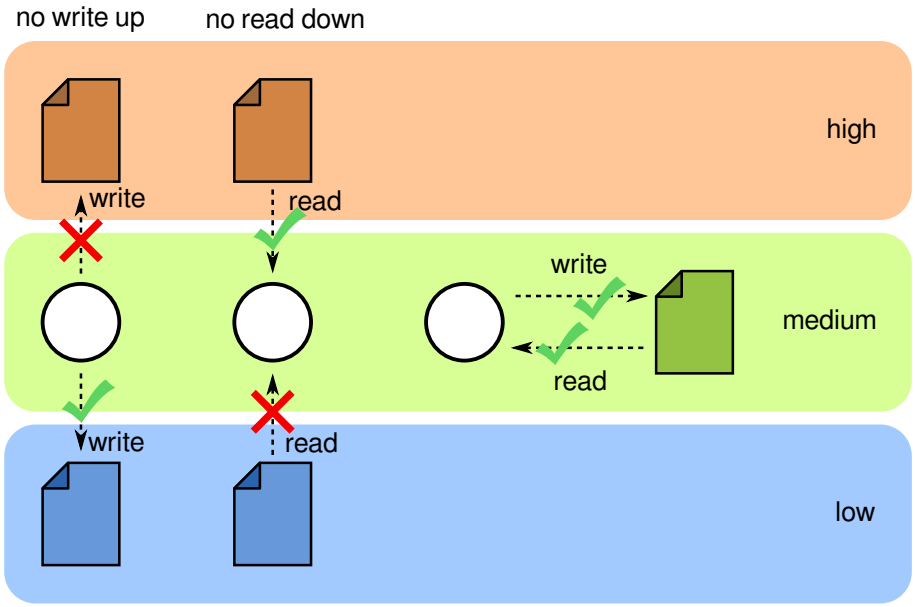

Figura 6.2: Política de Biba.

É importante observar que, na maioria dos sistemas reais, **as políticas obrigatórias não substituem as políticas discricionárias**, mas as complementam. Em um sistema que usa políticas obrigatórias, cada acesso a recurso é verificado usando a política obrigatória e também uma política discricionária; o acesso é permitido somente se ambas as políticas o autorizarem. A ordem de avaliação das políticas MAC e DAC obviamente não afeta o resultado final, mas pode ter impacto sobre o desempenho do sistema. Por isso, deve-se primeiro avaliar a política mais restritiva, ou seja, aquela que tem mais probabilidades de negar o acesso.

6.4.3 Categorias

Uma extensão frequente às políticas multinível é a noção de *categorias* ou *compartimentos*. Uma categoria define uma área funcional dentro do sistema computacional, como "pessoal", "projetos", "financeiro", "suporte", etc. Normalmente o conjunto de categorias é estático e não há uma ordem hierárquica entre elas. Cada sujeito e cada objeto do sistema são "rotulados" com uma ou mais categorias; a política então consiste em restringir o acesso de um sujeito somente aos objetos pertencentes às mesmas categorias dele, ou a um subconjunto destas. Dessa forma, um sujeito com as categorias $\{suporte, financeiro\}$ só pode acessar objetos rotulados como $\{suporte, financeiro\}$, $\{suporte\}$, $\{financeiro\}$ ou $\{\phi\}$. Formalmente: sendo $\mathbb{C}(s)$ o conjunto de categorias associadas a um sujeito s e $\mathbb{C}(o)$ o conjunto de categorias associadas a um objeto o, s só pode acessar o se $\mathbb{C}(s) \supseteq \mathbb{C}(o)$ [Samarati and De Capitani di Vimercati, 2001].

6.5 Políticas baseadas em domínios e tipos

O *domínio de segurança* de um sujeito define o conjunto de objetos que ele pode acessar e como pode acessá-los. Muitas vezes esse domínio está definido implicitamente nas regras das políticas obrigatórias ou na matriz de controle de acesso de uma política discricionária. As *políticas baseadas em domínios e tipos* (DTE - *Domain/Type Enforcement policies*) [Boebert and Kain, 1985] tornam explícito esse conceito: cada sujeito s do sistema é rotulado com um atributo constante definindo seu domínio $domain(s)$ e cada objeto o é associado a um tipo $type(o)$, também constante.

No modelo de implementação de uma política DTE definido em [Badger et al., 1995], as permissões de acesso de sujeitos a objetos são definidas em uma tabela global chamada *Tabela de Definição de Domínios* (DDT - *Domain Definition Table*), na qual cada linha é associada a um domínio e cada coluna a um tipo; cada célula $DDT[x, y]$ contém as permissões de sujeitos do domínio x a objetos do tipo y:

$$request(s, o, action) \iff action \in DDT[domain(s), type(o)]$$

Por sua vez, as interações entre sujeitos (trocas de mensagens, sinais, etc.) são reguladas por uma *Tabela de Interação entre Domínios* (DIT - *Domain Interaction Table*). Nessa tabela, linhas e colunas correspondem a domínios e cada célula $DIT[x, y]$ contém as interações possíveis de um sujeito no domínio x sobre um sujeito no domínio y:

$$request(s_i, s_j, interaction) \iff interaction \in DIT[domain(s_i), domain(s_j)]$$

Eventuais mudanças de domínio podem ser associadas a programas executáveis rotulados como *pontos de entrada* (*entry points*). Quando um processo precisa mudar de domínio, ele executa o ponto de entrada correspondente ao domínio de destino, se tiver permissão para tal.

O código a seguir define uma política de controle de acesso DTE, usada como exemplo em [Badger et al., 1995]. Essa política está representada graficamente (de forma simplificada) na Figura 6.3.

```
/* type definitions */
type unix_t,      /* normal UNIX files, programs, etc. */
     specs_t,     /* engineering specifications */
     budget_t,    /* budget projections */
     rates_t;     /* labor rates */

#define DEFAULT (/bin/sh), (/bin/csh), (rxd->unix_t) /* macro */

/* domain definitions */
domain engineer_d = DEFAULT, (rwd->specs_t);
domain project_d = DEFAULT, (rwd->budget_t), (rd->rates_t);
domain accounting_d = DEFAULT, (rd->budget_t), (rwd->rates_t);
domain system_d  = (/etc/init), (rwxd->unix_t), (auto->login_d);
domain login_d   = (/bin/login), (rwxd->unix_t),
                   (exec-> engineer_d, project_d, accounting_d);

initial_domain system_d; /* system starts in this domain */

/* assign resources (files and directories) to types */
assign -r unix_t   /;       /* default for all files */
assign -r specs_t  /projects/specs;
assign -r budget_t /projects/budget;
assign -r rates_t  /projects/rates;
```

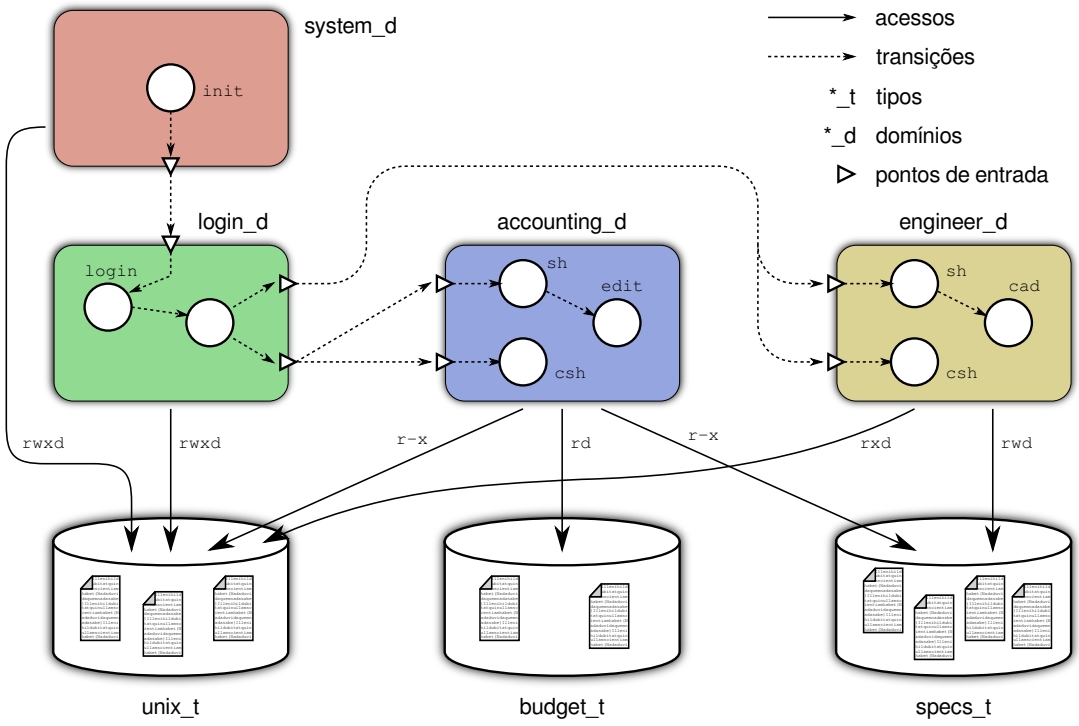

Figura 6.3: Exemplo de política baseada em domínios e tipos.

A implementação direta desse modelo sobre um sistema real pode ser inviável, pois exige a classificação de todos os sujeitos e objetos do mesmo em domínios e tipos. Para atenuar esse problema, [Badger et al., 1995; Cowan et al., 2000] propõem o uso de *tipagem implícita*: todos os objetos que satisfazem um certo critério (como por exemplo ter como caminho `/usr/local/*`) são automaticamente classificados em um dado tipo. Da

mesma forma, os domínios podem ser definidos pelos nomes dos programas executáveis que os sujeitos executam (como `/usr/bin/httpd` e `/usr/lib/httpd/plugin/*` para o domínio do servidor Web). Além disso, ambos os autores propõem linguagens para a definição dos domínios e tipos e para a descrição das políticas de controle de acesso.

6.6 Políticas baseadas em papéis

Um dos principais problemas de segurança em um sistema computacional é a administração correta das políticas de controle de acesso. As políticas MAC são geralmente consideradas pouco flexíveis e por isso as políticas DAC acabam sendo muito mais usadas. Todavia, gerenciar as autorizações à medida em que usuários mudam de cargo e assumem novas responsabilidades, novos usuários entram na empresa e outros saem pode ser uma tarefa muito complexa e sujeita a erros.

Esse problema pode ser reduzido através do *controle de acesso baseado em papéis* (RBAC - *Role-Based Access Control*) [Sandhu et al., 1996]. Uma política RBAC define um conjunto de *papéis* no sistema, como "diretor", "gerente", "suporte", "programador", etc. e atribui a cada papel um conjunto de autorizações. Essas autorizações podem ser atribuídas aos papéis de forma discricionária ou obrigatória.

Para cada usuário do sistema é definido um conjunto de papéis que este pode assumir. Durante sua sessão no sistema (geralmente no início), o usuário escolhe os papéis que deseja ativar e recebe as autorizações correspondentes, válidas até este desativar os papéis correspondentes ou encerrar sua sessão. Assim, um usuário autorizado pode ativar os papéis de "professor" ou de "aluno" dependendo do que deseja fazer no sistema.

Os papéis permitem desacoplar os usuários das permissões. Por isso, um conjunto de papéis definido adequadamente é bastante estável, restando à gerência apenas atribuir a cada usuário os papéis a que este tem direito. A Figura 6.4 apresenta os principais componentes de uma política RBAC.

Existem vários modelos para a implementação de políticas baseadas em papéis, como os apresentados em [Sandhu et al., 1996]. Por exemplo, no modelo *RBAC hierárquico* os papéis são classificados em uma hierarquia, na qual os papéis superiores herdam as permissões dos papéis inferiores. No modelo *RBAC com restrições* é possível definir restrições à ativação de papéis, como o número máximo de usuários que podem ativar um determinado papel simultaneamente ou especificar que dois papéis são conflitantes e não podem ser ativados pelo mesmo usuário simultaneamente.

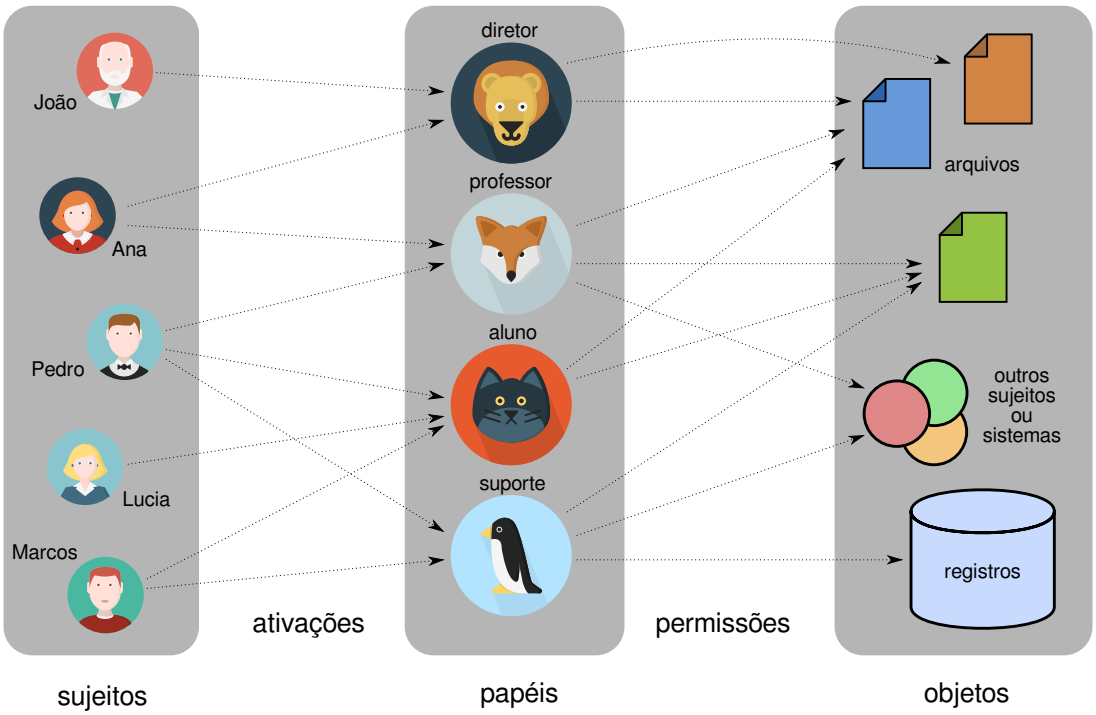

Figura 6.4: Políticas baseadas em papéis.

6.7 Outras políticas

Existem muitas outras políticas de controle de acesso além das apresentadas neste texto. Uma política que está ganhando popularidade é a ABAC – *Attribute-Based Access Control*, na qual o controle de acesso é feito usando regras baseadas em atributos dos sujeitos e objetos, não necessariamente suas identidades. Essas regras também podem levar em conta informações externas aos sujeitos e objetos, como horário, carga computacional do servidor, etc. Políticas baseadas em atributos são úteis em sistemas dinâmicos e de larga escala, como a Internet, onde a identidade de cada usuário específico é menos relevante que sua região geográfica, seu tipo de subscrição ao serviço desejado, ou outros atributos. O padrão ABAC definido pelo NIST [Hu et al., 2014] pode ser visto como uma estrutura formal genérica que permite construir políticas baseadas em atributos, além de permitir modelar políticas clássicas (discricionárias, obrigatórias), baseadas em papéis ou em domínios e tipos.

Exercícios

1. Sobre as afirmações a seguir, relativas aos modelos de controle de acesso, indique quais são **incorretas**, justificando sua resposta:

   (a) Nos modelos de controle de acesso obrigatórios, o controle é definido por regras globais incontornáveis, que não dependem das identidades dos sujeitos e objetos nem da vontade de seus proprietários ou mesmo do administrador do sistema.

   (b) Os modelos de controle de acesso discricionários se baseiam na atribuição de permissões de forma individualizada, ou seja, pode-se conceder ou negar

a um sujeito específico a permissão de executar uma ação sobre um dado objeto.

(c) O Modelo da matriz de controle de acesso é uma forma de representação lógica de políticas discricionárias.

(d) O modelo de Bell-LaPadula é uma forma de representar políticas de controle de acesso obrigatórias que tenham foco em confidencialidade de dados.

(e) O modelo de Biba é uma forma de representar políticas de controle de acesso obrigatórias que tenham foco em integridade de dados.

(f) Os modelos de controle de acesso baseados em papéis permitem desvincular os usuários das permissões sobre os objetos, através da definição e atribuição de papéis.

2. Analise a seguinte matriz de controle de acesso:

	$file_1$	$file_2$	$program_1$	$socket_1$
Alice	*read* *write* *remove* ***owner***	*read* *write*	*execute*	*write*
Beto	*read* *write*	*read* *write* *remove* ***owner***	*read* ***owner***	
Carol		*read*	*execute*	*read* *write*
Davi	*read*	*write*	*read*	*read* *write* ***owner***

Assinale a alternativa correta:

(a) O usuário *Beto* pode alterar as permissões dos recursos $file_1$ e $program_1$

(b) O usuário *Alice* tem a seguinte lista de capacidades: $\{file_1 : (read, write, remove, owner), file_2 : (read, write), program_1 : (read, execute), socket_1 : (write)\}$

(c) A lista de controle de acesso de $file_2$ é: $\{Alice : (read, write), Beto : (read, write, remove), Carol : (read), Davi : (write)\}$

(d) A lista de capacidades de *Beto* é: $\{file_1 : (read, write), file_2 : (read, write, remove, owner), program_1 : (read, owner)\}$

(e) Nenhuma das anteriores

3. Escreva as listas de controle de acesso (*ACLs*) equivalentes às listas de capacidades a seguir:

$$
\begin{aligned}
CL(Alice) &= \{\ file_1 : (read, write, remove, owner),\\
&\quad\ file_2 : (read),\\
&\quad\ program_1 : (execute),\\
&\quad\ socket_1 : (read, write)\ \}\\
CL(Beto) &= \{\ file_1 : (read),\\
&\quad\ file_2 : (read, write, remove, owner),\\
&\quad\ program_1 : (read, execute, owner)\ \}\\
CL(Carol) &= \{\ file_2 : (read, write),\\
&\quad\ program_1 : (execute),\\
&\quad\ socket_1 : (read, write)\ \}\\
CL(Davi) &= \{\ file_1 : (read),\\
&\quad\ file_2 : (write),\\
&\quad\ program_1 : (read, execute),\\
&\quad\ socket_1 : (read, write, owner)\ \}
\end{aligned}
$$

4. Relacione as expressões a seguir aos modelos de controle de acesso de Bell (L)aPadula, (B)iba ou da (M)atriz de controle de acesso. Considere s um sujeito, o um objeto, $h(s)$ o nível de habilitação ou de integridade do sujeito e $c(o)$ a classificação do objeto.

   [ ] $request(s_i, o_j, write) \iff h(s_i) \geq c(o_j)$

   [ ] $request(s_i, o_j, write) \iff write \in M_{ij}$

   [ ] $request(s_i, o_j, read) \iff h(s_i) \geq c(o_j)$

   [ ] $request(s_i, o_j, read) \iff read \in M_{ij}$

   [ ] $request(s_i, o_j, write) \iff h(s_i) \leq c(o_j)$

   [ ] $request(s_i, o_j, read) \iff h(s_i) \leq c(o_j)$

5. Construa a matriz de controle de acesso que corresponde à seguinte listagem de arquivo em um ambiente UNIX:

```
-rwxr-x---  2 maziero   prof      14321   2010-07-01 16:44   script.sh
-rw-------  2 lucas     aluno    123228   2008-12-27 08:53   relat.pdf
-rwxr-x--x  2 daniel    suporte    3767   2010-11-14 21:50   backup.py
-rw-rw-r--  2 sheila    prof      76231   2009-18-27 11:06   cmmi.xml
-rw-r-----  2 mariana   aluno      4089   2010-11-09 02:14   teste1.c
```

   Observações:

   - Composição do grupo `prof`: {`maziero`, `sheila`}
   - Composição do grupo `suporte`: {`maziero`, `daniel`}
   - Composição do grupo `aluno`: {`lucas`, `daniel`, `mariana`}

- Preencha os campos da matriz com os caracteres "`r`", "`w`", "`x`" e "`-`".

6. Em um sistema de documentação militar estão definidos os seguintes usuários e suas respectivas habilitações:

Usuário	Habilitação
Marechal Floriano	Ultrassecreto
General Motors	Secreto
Major Nelson	Confidencial
Sargento Tainha	Restrito
Recruta Zero	Público

Considerando operações sobre documentos classificados, indique quais das operações a seguir seriam permitidas pelo modelo de controle de acesso de Bell-LaPadula:

[ ] Sargento Tainha cria o documento secreto `comunicado.txt`
[ ] Recruta Zero lê o documento ultrassecreto `salarios-dos-generais.xls`
[ ] General Motors escreve um memorando público `aviso-sobre-ferias.doc`.
[ ] Major Nelson escreve um documento confidencial `avarias-no-submarino.doc`.
[ ] Marechal Floriano lê o documento restrito `comunicado.txt`.
[ ] General Motors lê o documento secreto `vendas-de-carros-2010.doc`.
[ ] Sargento Tainha lê o documento restrito `plano-de-ataque.pdf`.
[ ] Major Nelson lê o documento confidencial `processos-navais.html`.
[ ] Marechal Floriano escreve o documento secreto `novas-avenidas.doc`.
[ ] Recruta Zero escreve o documento ultrassecreto `meu-diario.txt`.

7. As listas de controle de acesso (*ACLs*) e as listas de capacidades (*CLs*) a seguir são complementares, mas estão incompletas. Complete-as com as regras faltantes.

$$
\begin{array}{lcl}
ACL(o_1) & = & \{\ (u_1 : rwx) \qquad\qquad\ \} \\
ACL(o_2) & = & \{\ \ (u_2 : r) \qquad\qquad\ \} \\
ACL(o_3) & = & \{\ \ (u_1 : r) \quad (u_4 : rw)\ \} \\
ACL(o_4) & = & \{\ (u_2 : rw) \quad (u_3 : r)\ \ \}
\end{array}
$$

$$
\begin{array}{lcl}
CL(u_1) & = & \{\ (o_2 : rw) \quad (o_4 : r)\ \ \} \\
CL(u_2) & = & \{\ \ (o_1 : rx) \qquad\qquad\ \} \\
CL(u_3) & = & \{\ \ (o_1 : rx) \qquad\qquad\ \} \\
CL(u_4) & = & \{\ (o_4 : rwx) \qquad\qquad\ \}
\end{array}
$$

8. Considerando o modelo de controle de acesso de Bell & LaPadula, indique que tipo de acesso (R, W, RW ou –) um usuário u pode ter sobre os documentos abaixo identificados. Considere que $h(u) = secreto$ e que $\mathbb{C}(u) = \{vendas, rh\}$.

[ ] d_1: $c(d_1) = ultrassecreto$ e $\mathbb{C}(d_1) = \{vendas\}$
[ ] d_2: $c(d_2) = publico$ e $\mathbb{C}(d_2) = \{rh, financeiro\}$
[ ] d_3: $c(d_3) = secreto$ e $\mathbb{C}(d_3) = \{rh\}$
[ ] d_4: $c(d_4) = reservado$ e $\mathbb{C}(d_4) = \{rh, vendas\}$
[ ] d_5: $c(d_5) = confidencial$ e $\mathbb{C}(d_5) = \{\ \}$

Referências

R. Anderson. *Security engineering*. John Wiley & Sons, 2008.

L. Badger, D. Sterne, D. Sherman, K. Walker, and S. Haghighat. Practical domain and type enforcement for UNIX. In *IEEE Symposium on Security and Privacy*, pages 66–77, 1995.

D. E. Bell and L. J. LaPadula. Secure computer systems. mathematical foundations and model. Technical Report M74-244, MITRE Corporation, 1974.

K. Biba. Integrity considerations for secure computing systems. Technical Report MTR-3153, MITRE Corporation, 1977.

W. Boebert and R. Kain. A practical alternative to hierarchical integrity policies. In *8th National Conference on Computer Security*, pages 18–27, 1985.

C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor. SubDomain: Parsimonious server security. In *14th USENIX Systems Administration Conference*, 2000.

S. di Vimercati, P. Samarati, and S. Jajodia. Policies, models, and languages for access control. In *Workshop on Databases in Networked Information Systems*, volume LNCS 3433, pages 225–237. Springer-Verlag, 2005.

S. di Vimercati, S. Foresti, S. Jajodia, and P. Samarati. Access control policies and languages in open environments. In T. Yu and S. Jajodia, editors, *Secure Data Management in Decentralized Systems*, volume 33 of *Advances in Information Security*, pages 21–58. Springer, 2007.

V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (ABAC) definition and considerations. Technical Report 800-162, NIST - National Institute of Standards and Technology, 2014.

B. Lampson. Protection. In *5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM Operating Systems Rev. 8, 1 (Jan. 1974), pp 18-24.

J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278 – 1308, September 1975.

P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*. Springer-Verlag, 2001.

R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, Feb. 1996.

R. Shirey. RFC 2828: Internet security glossary, May 2000.