Capítulo 29

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.

29.1 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.

29.2 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.

29.3 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].

29.3.1 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 $\mathbb{S} = \{s_1, s_2, \dots, s_m\}$, um conjunto de objetos $\mathbb{O} = \{o_1, o_2, \dots, o_n\}$ e um conjunto de ações possíveis sobre os objetos $\mathbb{A} = \{a_1, a_2, \dots, a_p\}$, 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 29.1.

	$file_1$	$file_2$	$program_1$	$socket_1$
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 29.1: 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 29.2.

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.

29.3.2 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 29.3 mostra como ficaria a matriz de controle de acesso da Tabela 29.2 sob a forma de uma tabela de autorizações.

	$file_1$	$file_2$	$program_1$	$socket_1$
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 29.2: Uma matriz de controle de acesso com política administrativa

29.3.3 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 29.2 seriam:

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

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 29.3: Tabela de autorizações

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.

29.3.4 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 29.2 seriam:

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

$$
\begin{aligned}
CL(Davi) = \{ \quad & file_1 : (read), \\
& file_2 : (write), \\
& program_1 : (read), \\
& socket_1 : (read, write, owner) \; \}
\end{aligned}
$$

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 29.7.4.

29.4 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].

29.4.1 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 29.1. 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.

29.4.2 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)

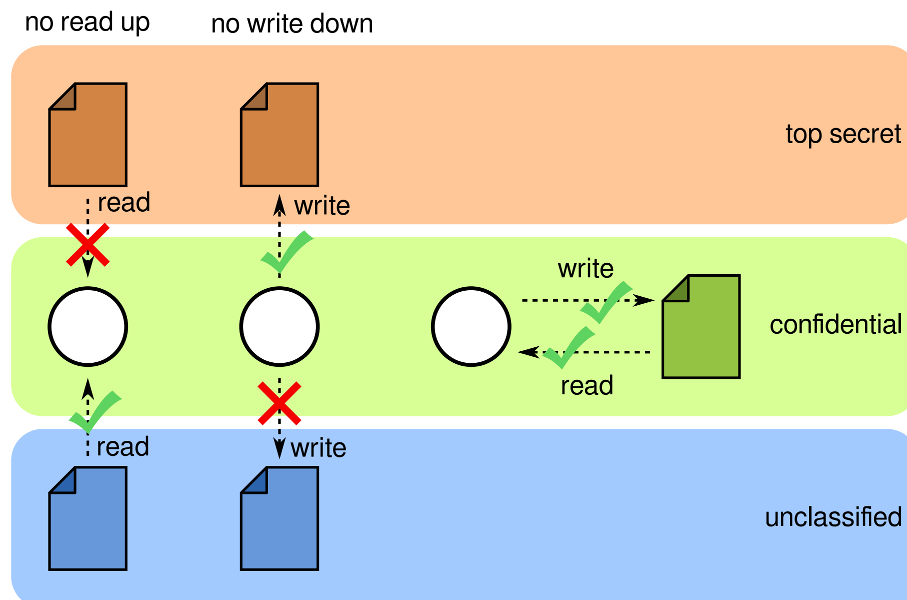


Figura 29.1: Política de Bell-LaPadula.

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 ⋆ 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 29.2. 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.

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.

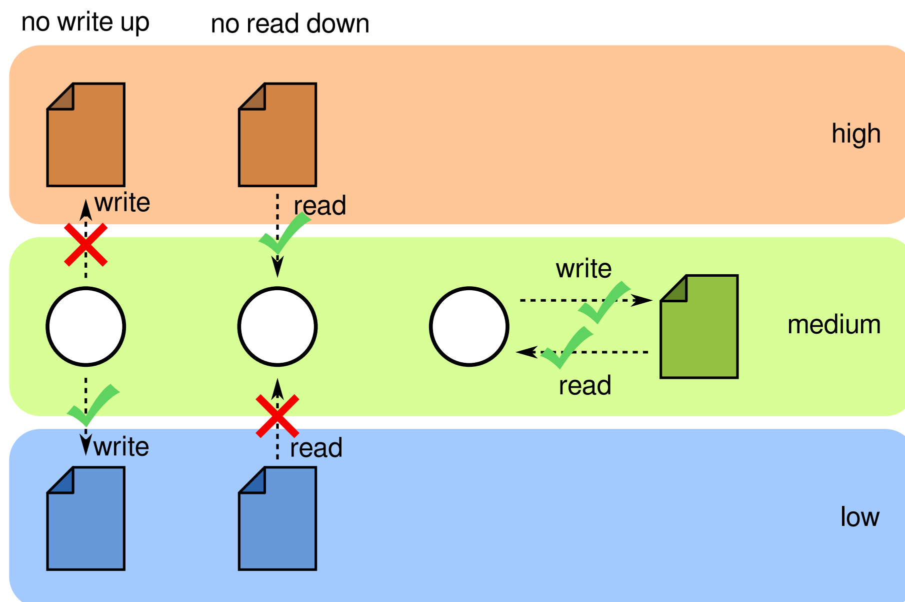


Figura 29.2: Política de Biba.

29.4.3 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].

29.5 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 29.3.

```
/* 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;
```

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.

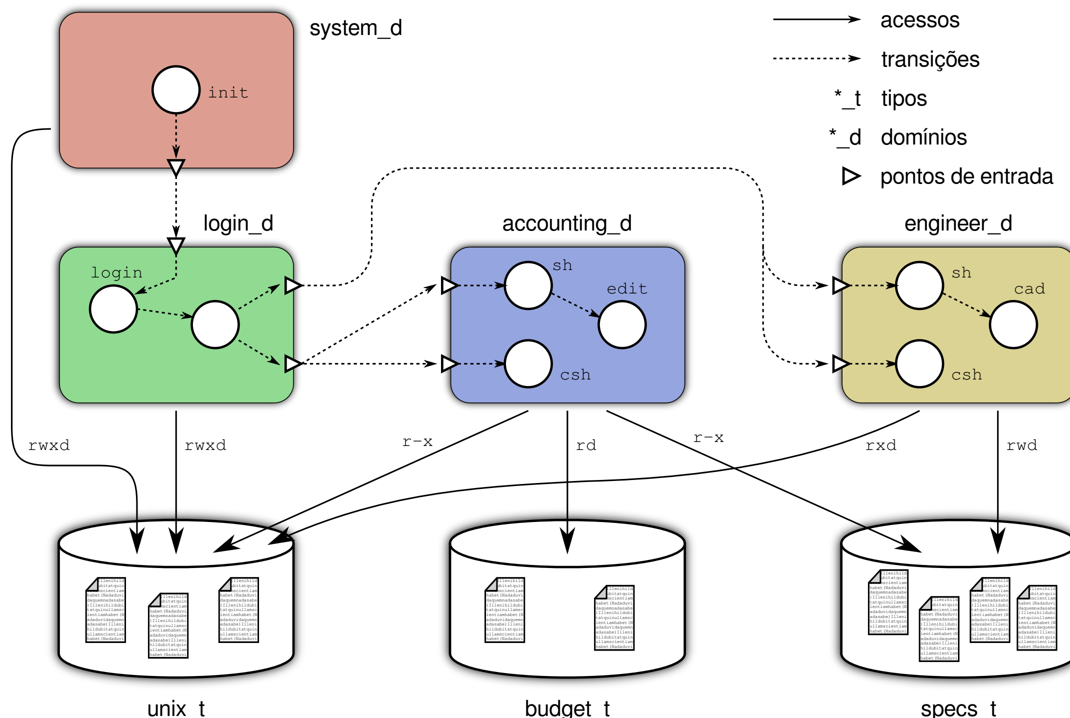


Figura 29.3: Exemplo de política baseada em domínios e tipos.

29.6 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 29.4 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

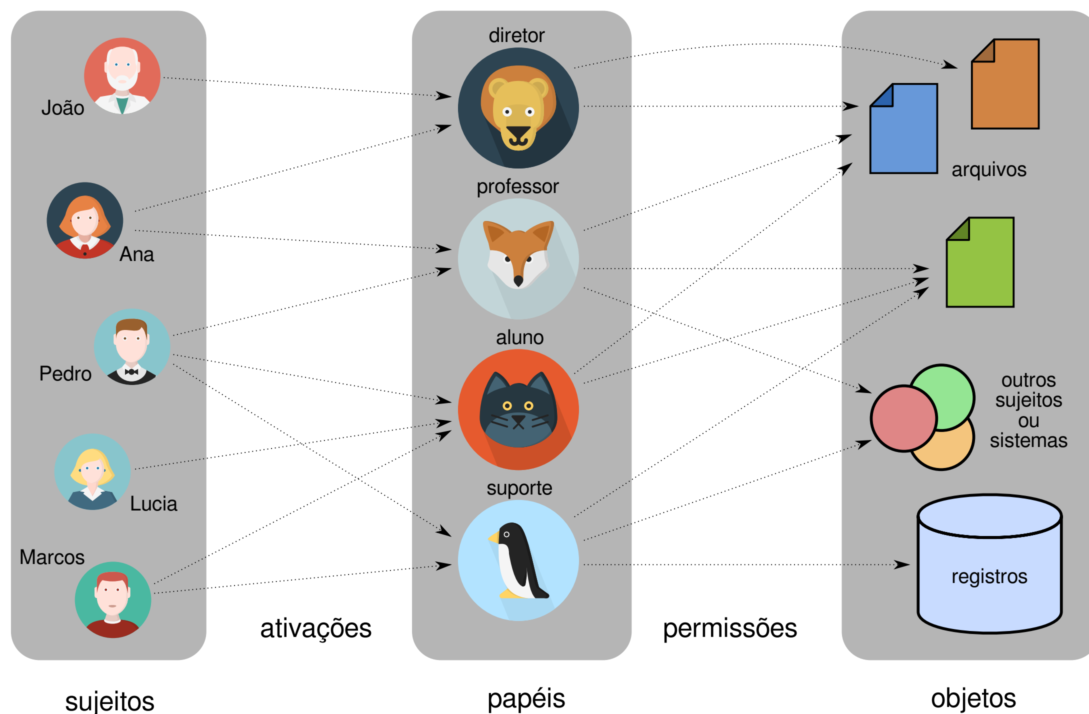


Figura 29.4: Políticas baseadas em papéis.

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.

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.

29.7 Mecanismos de controle de acesso

A implementação do controle de acesso em um sistema computacional deve ser independente das políticas de controle de acesso adotadas. Como nas demais áreas de um sistema operacional, a separação entre mecanismo e política é importante, por possibilitar trocar a política de controle de acesso sem ter de modificar a implementação do sistema. A infraestrutura de controle de acesso deve ser ao mesmo tempo *inviolável* (impossível de adulterar ou enganar) e *incontornável* (todos os acessos aos recursos do sistema devem passar por ela).

29.7.1 Infraestrutura básica

A arquitetura básica de uma infraestrutura de controle de acesso típica é composta pelos seguintes elementos (Figura 29.5):

Bases de sujeitos e objetos (*User/Object Bases*): relação dos sujeitos e objetos que compõem o sistema, com seus respectivos atributos;

Base de políticas (*Policy Base*): base de dados contendo as regras que definem como e quando os objetos podem ser acessados pelos sujeitos, ou como/quando os sujeitos podem interagir entre si;

Monitor de referências (*Reference monitor*): elemento que julga a pertinência de cada pedido de acesso. Com base em atributos do sujeito e do objeto (como suas respectivas identidades), nas regras da base de políticas e possivelmente em informações externas (como horário, carga do sistema, etc.), o monitor decide se um acesso deve ser permitido ou negado;

Mediador (impositor ou *Enforcer*): elemento que medeia a interação entre sujeitos e objetos; a cada pedido de acesso a um objeto, o mediador consulta o monitor de referências e permite/nega o acesso, conforme a decisão deste último.

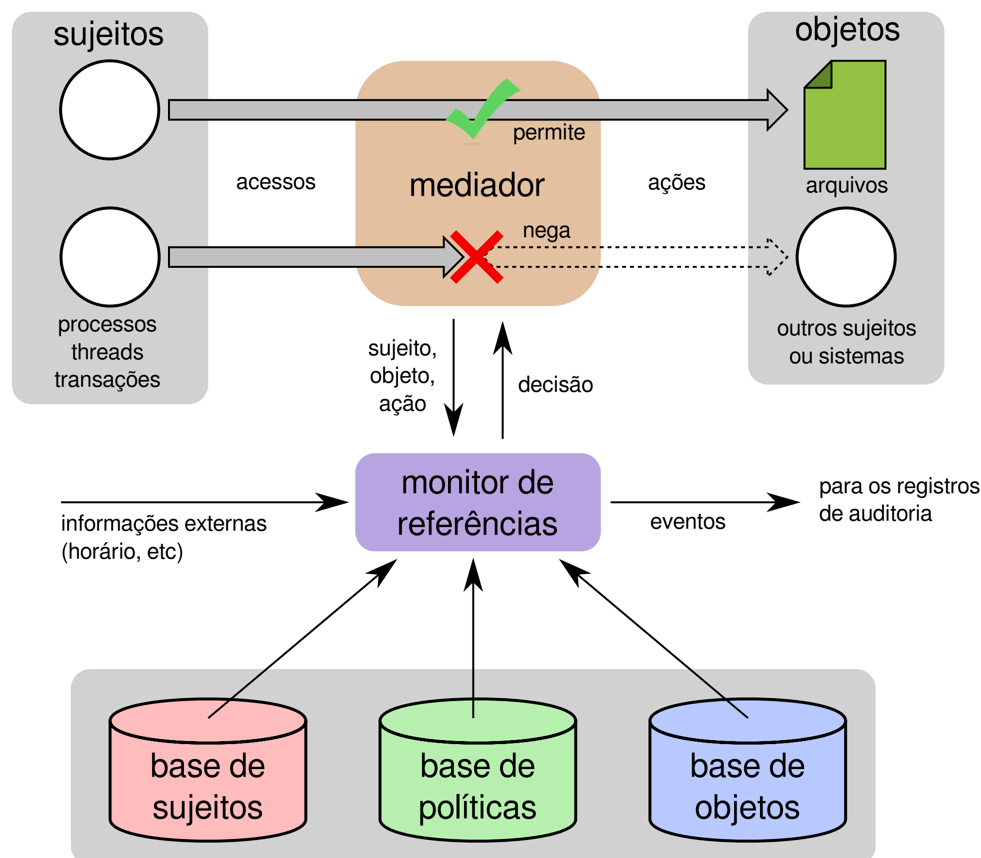


Figura 29.5: Estrutura genérica de uma infraestrutura de controle de acesso.

É importante observar que os elementos dessa estrutura são componentes lógicos, que não impõem uma forma de implementação rígida. Por exemplo, em um

sistema operacional convencional, o sistema de arquivos possui sua própria estrutura de controle de acesso, com permissões de acesso armazenadas nos próprios arquivos, e um pequeno monitor/mediador associado a algumas chamadas de sistema, como `open` e `mmap`. Outros recursos (como áreas de memória ou semáforos) possuem suas próprias regras e estruturas de controle de acesso, organizadas de forma diversa.

29.7.2 Controle de acesso em UNIX

Os sistemas operacionais do mundo UNIX implementam um sistema de ACLs básico bastante rudimentar, no qual existem apenas três sujeitos: *user* (o dono do recurso), *group* (um grupo de usuários ao qual o recurso está associado) e *others* (todos os demais usuários do sistema). Para cada objeto existem três possibilidades de acesso: *read*, *write* e *execute*, cuja semântica depende do tipo de objeto (arquivo, diretório, *socket* de rede, área de memória compartilhada, etc.). Dessa forma, são necessários apenas 9 bits por arquivo para definir suas permissões básicas de acesso.

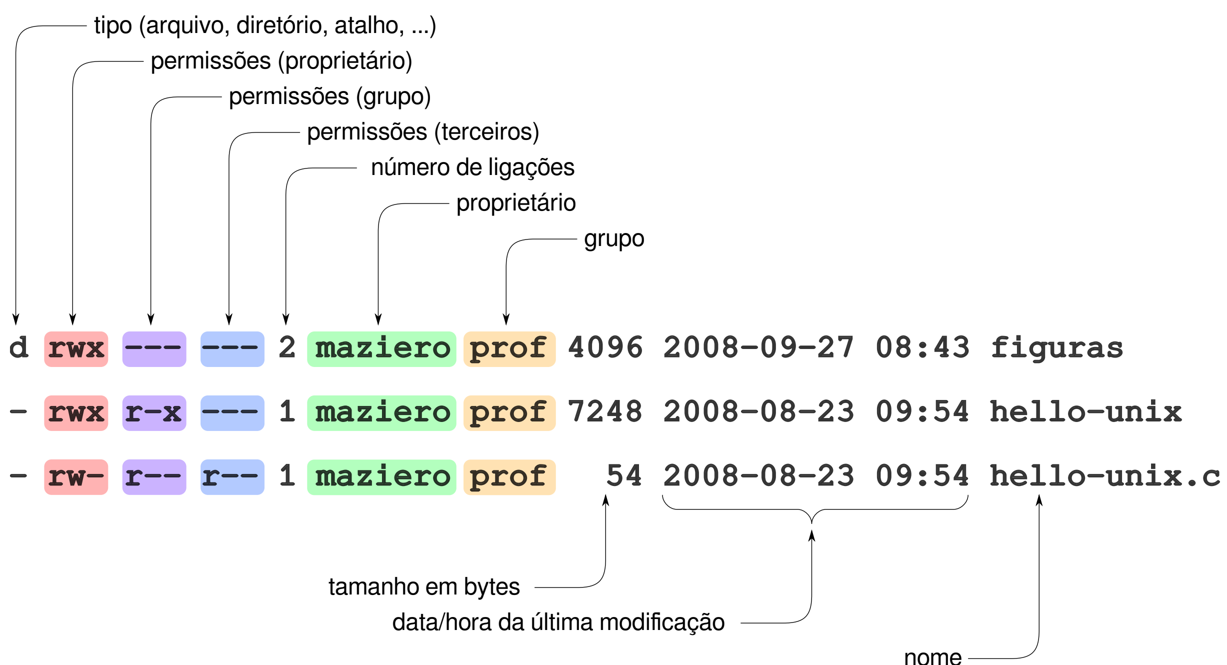


Figura 29.6: Listas de controle de acesso em UNIX.

A Figura 29.6 apresenta uma listagem de diretório típica em UNIX. Nessa listagem, o arquivo `hello-unix.c` pode ser acessado em leitura e escrita por seu proprietário (o usuário `maziero`, com permissões `rw-`), em leitura pelos usuários do grupo `prof` (permissões `r--`) e em leitura pelos demais usuários do sistema (permissões `r--`). Já o arquivo `hello-unix` pode ser acessado em leitura, escrita e execução por seu proprietário (permissões `rwx`), em leitura e execução pelos usuários do grupo `prof` (permissões `r-x`) e não pode ser acessado pelos demais usuários (permissões `---`). No caso de diretórios, a permissão de leitura autoriza a listagem do diretório, a permissão de escrita autoriza sua modificação (criação, remoção ou renomeação de arquivos ou sub-diretórios) e a permissão de execução autoriza usar aquele diretório como diretório de trabalho ou parte de um caminho.

É importante destacar que o controle de acesso é normalmente realizado apenas durante a abertura do arquivo, para a criação de seu descritor em memória. Isso significa

que, uma vez aberto um arquivo por um processo, este terá acesso ao arquivo enquanto o mantiver aberto, mesmo que as permissões do arquivo sejam modificadas para impedir esse acesso. O controle contínuo de acesso a arquivos é pouco frequentemente implementado em sistemas operacionais, porque verificar as permissões de acesso a cada operação de leitura ou escrita teria um forte impacto negativo sobre o desempenho do sistema.

Dessa forma, um descritor de arquivo aberto pode ser visto como uma capacidade (vide Seção 29.3.4), pois a posse do descritor permite ao processo acessar o arquivo referenciado por ele. O processo recebe esse descritor ao abrir o arquivo e deve apresentá-lo a cada acesso subsequente; o descritor pode ser transferido aos processos filhos ou até mesmo a outros processos, outorgando a eles o acesso ao arquivo aberto. A mesma estratégia é usada em *sockets* de rede, semáforos e outros mecanismos de IPC.

O padrão POSIX 1003.1e definiu ACLs mais detalhadas para o sistema de arquivos, que permitem definir permissões para usuários e grupos específicos além do proprietário do arquivo. Esse padrão é parcialmente implementado em vários sistemas operacionais, como o Linux e o FreeBSD. No Linux, os comandos `getfacl` e `setfacl` permitem manipular essas ACLs, como mostra o exemplo a seguir:

```
1  host:~> ll
2  -rw-r--r-- 1 maziero prof 2450791 2009-06-18 10:47 main.pdf
3  
4  host:~> getfacl main.pdf
5  # file: main.pdf
6  # owner: maziero
7  # group: maziero
8  user::rw-
9  group::r--
10 other::r--
11 
12 host:~> setfacl -m diogo:rw,rafael:rw main.pdf
13 
14 host:~> getfacl main.pdf
15 # file: main.pdf
16 # owner: maziero
17 # group: maziero
18 user::rw-
19 user:diogo:rw-
20 user:rafael:rw-
21 group::r--
22 mask::rw-
23 other::r--
```

No exemplo, o comando da linha 12 define permissões de leitura e escrita específicas para os usuários `diogo` e `rafael` sobre o arquivo `main.pdf`. Essas permissões estendidas são visíveis na linha 19 e 20, junto com as permissões UNIX básicas (nas linhas 18, 21 e 23).

29.7.3 Controle de acesso em Windows

Os sistemas Windows baseados no núcleo NT (NT, 2000, XP, Vista e sucessores) implementam mecanismos de controle de acesso bastante sofisticados [Brown, 2000; Russinovich et al., 2008]. Em um sistema Windows, cada sujeito (computador, usuário, grupo ou domínio) é unicamente identificado por um *identificador de segurança* (SID - *Security IDentifier*). Cada sujeito do sistema está associado a um *token de acesso*, criado no momento em que o respectivo usuário ou sistema externo se autentica no sistema. A autenticação e o início da sessão do usuário são gerenciados pelo LSASS (*Local Security Authority Subsystem*), que cria os processos iniciais e os associa ao *token* de acesso criado para aquele usuário. Esse *token* normalmente é herdado pelos processos filhos, até o encerramento da sessão do usuário. Ele contém o identificador do usuário (SID), dos grupos aos quais ele pertence, privilégios a ele associados e outras informações. Privilégios são permissões para realizar operações genéricas, que não dependem de um recurso específico, como reiniciar o computador, carregar um *driver* ou depurar um processo.

Por outro lado, cada objeto do sistema está associado a um *descritor de segurança* (SD - *Security Descriptor*). Como objetos, são considerados arquivos e diretórios, processos, impressoras, serviços e chaves de registros, por exemplo. Um descritor de segurança indica o proprietário e o grupo primário do objeto, uma lista de controle de acesso de sistema (SACL - *System ACL*), uma lista de controle de acesso discricionária (DACL - *Discretionary ACL*) e algumas informações de controle.

A DACL contém uma lista de regras de controle de acesso ao objeto, na forma de ACEs (*Access Control Entries*). Cada ACE contém um identificador de usuário ou grupo, um modo de autorização (positiva ou negativa), um conjunto de permissões (ler, escrever, executar, remover, etc.), sob a forma de um mapa de bits. Quando um sujeito solicita acesso a um recurso, o SRM (*Security Reference Monitor*) compara o *token* de acesso do sujeito com as entradas da DACL do objeto, para permitir ou negar o acesso. Como sujeitos podem pertencer a mais de um grupo e as ACEs podem ser positivas ou negativas, podem ocorrer conflitos entre as ACEs. Por isso, um mecanismo de resolução de conflitos é acionado a cada acesso solicitado ao objeto.

A SACL define que tipo de operações sobre o objeto devem ser registradas pelo sistema, sendo usada basicamente para fins de auditoria (Seção 30). A estrutura das ACEs de auditoria é similar à das ACEs da DACL, embora defina quais ações sobre o objeto em questão devem ser registradas para quais sujeitos. A Figura 29.7 ilustra alguns dos componentes da estrutura de controle de acesso dos sistemas Windows.

29.7.4 Outros mecanismos

As políticas de segurança básicas utilizadas na maioria dos sistemas operacionais são discricionárias, baseadas nas identidades dos usuários e em listas de controle de acesso. Entretanto, políticas de segurança mais sofisticadas vêm sendo gradualmente agregadas aos sistemas operacionais mais complexos, visando aumentar sua segurança. Algumas iniciativas dignas de nota são apresentadas a seguir:

- O SELinux é um mecanismo de controle de acesso multipolíticas, desenvolvido pela NSA (*National Security Agency*, USA) [Loscocco and Smalley, 2001] a partir da arquitetura flexível de segurança *Flask* (*Flux Advanced Security Kernel*) [Spencer et al., 1999]. Ele constitui uma infraestrutura complexa de segurança

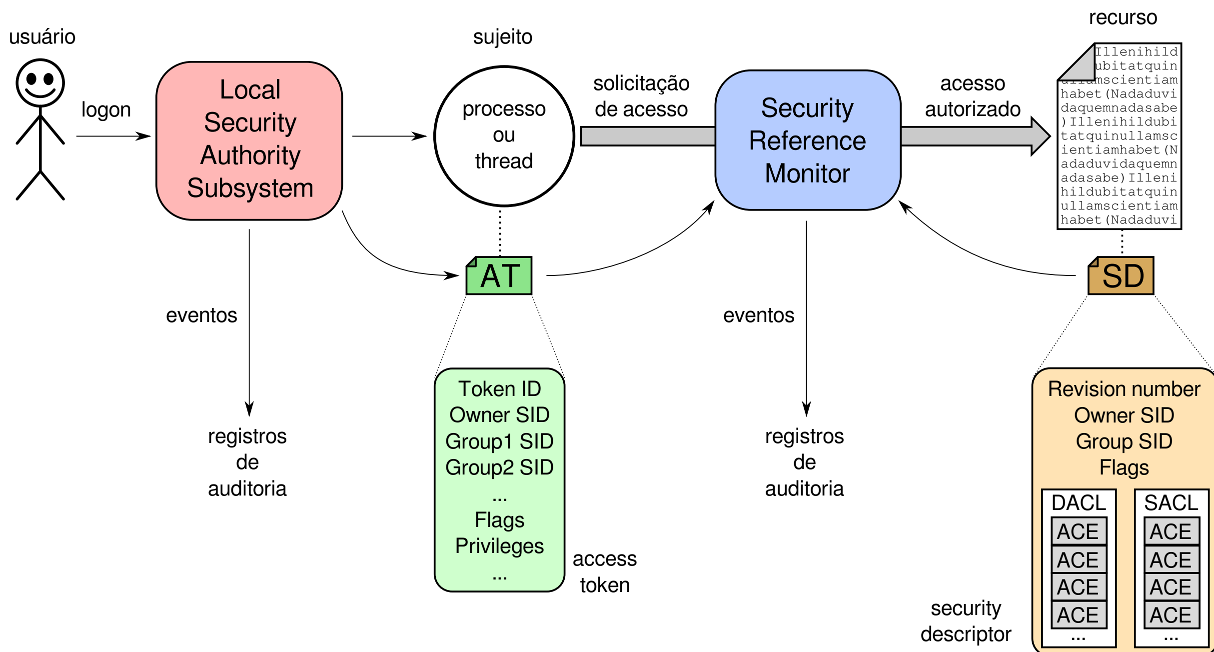


Figura 29.7: Listas de controle de acesso no Windows.

para o núcleo Linux, capaz de aplicar diversos tipos de políticas obrigatórias aos recursos do sistema operacional. A política default do SELinux é baseada em RBAC e DTE, mas ele também é capaz de implementar políticas de segurança multinível. O SELinux tem sido criticado devido à sua complexidade, que torna difícil sua compreensão e configuração. Em consequência, outros projetos visando adicionar políticas MAC mais simples e fáceis de usar ao núcleo Linux têm sido propostos, como *LIDS*, *SMACK* e *AppArmor*.

- O sistema operacional Windows Vista incorpora uma política denominada *Mandatory Integrity Control* (MIC) que associa aos processos e recursos os níveis de integridade *Low*, *Medium*, *High* e *System* [Microsoft], de forma similar ao modelo de Biba (Seção 29.4.2). Os processos normais dos usuários são classificados como de integridade média, enquanto o navegador Web e executáveis provindos da Internet são classificados como de integridade baixa. Além disso, o Vista conta com o UAC (*User Account Control*) que aplica uma política baseada em RBAC: um usuário com direitos administrativos inicia sua sessão como usuário normal, e somente ativa seu papel administrativo quando necessita efetuar uma ação administrativa.

- O projeto TrustedBSD [Watson, 2001] implementa ACLs no padrão POSIX, capacidades POSIX e o suporte a políticas obrigatórias como Bell LaPadula, Biba, categorias e TE/DTE. Uma versão deste projeto foi portada para o MacOS X, sendo denominada *MacOS X MAC Framework*.

- Desenvolvido nos anos 90, o sistema operacional experimental *EROS* (*Extremely Reliable Operating System*) [Shapiro and Hardy, 2002] implementou um modelo de controle de acesso totalmente baseado em capacidades. Nesse modelo, todas as interfaces dos componentes do sistema só são acessíveis através de capacidades, que são usadas para nomear as interfaces e para controlar seu

acesso. O sistema *EROS* deriva de desenvolvimentos anteriores feitos no sistema operacional KeyKOS para a plataforma S/370 [Bomberger et al., 1992].

- Em 2009, o sistema operacional experimental *SeL4*, que estende o sistema micronúcleo L4 [Liedtke, 1996] com um modelo de controle de acesso baseado em capacidades similar ao utilizado no sistema EROS, tornou-se o primeiro sistema operacional cuja segurança foi formalmente verificada [Klein et al., 2009]. A verificação formal é uma técnica de engenharia de software que permite demonstrar matematicamente que a implementação do sistema corresponde à sua especificação, e que a especificação está completa e sem erros.

- O sistema *Trusted Solaris* [Sun Microsystems] implementa várias políticas de segurança: em MLS (*Multi-Level Security*), níveis de segurança são associados aos recursos do sistema e aos usuários. Além disso, a noção de domínios é implementada através de "compartimentos": um recurso associado a um determinado compartimento só pode ser acessado por sujeitos no mesmo compartimento. Para limitar o poder do super-usuário, é usada uma política de tipo RBAC, que divide a administração do sistema em vários papéis que podem ser atribuídos a usuários distintos.

29.8 Mudança de privilégios

Normalmente, os processos em um sistema operacional são sujeitos que representam o usuário que os lançou. Quando um novo processo é criado, ele herda as credenciais de seu processo-pai, ou seja, seus identificadores de usuário e de grupo. Na maioria dos mecanismos de controle de acesso usados em sistemas operacionais, as permissões são atribuídas aos processos em função de suas credenciais. Com isso, normalmente cada novo processo herda as mesmas permissões de seu processo-pai, pois possui as mesmas credenciais dele.

O uso de privilégios fixos é adequado para o uso normal do sistema, pois os processos de cada usuário só devem ter acesso aos recursos autorizados para esse usuário. Entretanto, em algumas situações esse mecanismo se mostra inadequado. Por exemplo, caso um usuário precise executar uma tarefa administrativa, como instalar um novo programa, modificar uma configuração de rede ou atualizar sua senha, alguns de seus processos devem possuir permissões para as ações necessárias, como editar arquivos de configuração do sistema. Os sistemas operacionais atuais oferecem diversas abordagens para resolver esse problema:

Usuários administrativos: são associadas permissões administrativas às sessões de trabalho de alguns usuários específicos, permitindo que seus processos possam efetuar tarefas administrativas, como instalar softwares ou mudar configurações. Esta é a abordagem utilizada em alguns sistemas operacionais de amplo uso. Algumas implementações definem vários tipos de usuários administrativos, com diferentes tipos de privilégios, como acessar dispositivos externos, lançar máquinas virtuais, reiniciar o sistema, etc. Embora simples, essa solução é falha, pois se algum programa com conteúdo malicioso for executado por um usuário administrativo, terá acesso a todas as suas permissões.

Permissões temporárias: conceder sob demanda a certos processos do usuário as permissões de que necessitam para realizar ações administrativas; essas permissões podem ser descartadas pelo processo assim que concluir as ações. Essas permissões podem estar associadas a papéis administrativos (Seção 29.6), ativados quando o usuário tiver necessidade deles. Esta é a abordagem usada pela infraestrutura UAC (*User Access Control*) [Microsoft], na qual um usuário administrativo inicia sua sessão de trabalho como usuário normal, e somente ativa seu papel administrativo quando necessita efetuar uma ação administrativa, desativando-o imediatamente após a conclusão da ação. A ativação do papel administrativo pode impor um procedimento de reautenticação.

Mudança de credenciais: permitir que certos processos do usuário mudem de identidade, assumindo a identidade de algum usuário com permissões suficientes para realizar a ação desejada; pode ser considerada uma variante da atribuição de permissões temporárias. O exemplo mais conhecido de implementação desta abordagem são os flags `setuid` e `setgid` do UNIX, explicados a seguir.

Monitores: definir processos privilegiados, chamados *monitores* ou *supervisores*, recebem pedidos de ações administrativas dos processos não privilegiados, através de uma API predefinida; os pedidos dos processos normais são validados e atendidos. Esta é a abordagem definida como *separação de privilégios* em [Provos et al., 2003], e também é usada na infra-estrutura *PolicyKit*, usada para autorizar tarefas administrativas em ambientes *desktop* Linux.

Um mecanismo amplamente usado para mudança de credenciais consiste dos flags `setuid` e `setgid` dos sistemas UNIX. Se um arquivo executável tiver o flag `setuid` habilitado (indicado pelo caractere "s" em suas permissões de usuário), seus processos assumirão as credenciais do proprietário do arquivo. Portanto, se o proprietário de um arquivo executável for o usuário *root*, os processos lançados a partir dele terão todos os privilégios do usuário *root*, independente de quem o tiver lançado. De forma similar, processos lançados a partir de um arquivo executável com o flag `setgid` habilitado terão as credenciais do grupo associado ao arquivo. A Figura 29.8 ilustra esse mecanismo: o primeiro caso representa um executável normal (sem esses flags habilitados); um processo filho lançado a partir do executável possui as mesmas credenciais de seu pai. No segundo caso, o executável pertence ao usuário *root* e tem o flag `setuid` habilitado; assim, o processo filho assume a identidade do usuário *root* e, em consequência, suas permissões de acesso. No último caso, o executável pertence ao usuário *root* e tem o flag `setgid` habilitado; assim, o processo filho pertencerá ao grupo *mail*.

Os flags `setuid` e `setgid` são muito utilizados em programas administrativos no UNIX, como troca de senha e agendamento de tarefas, sempre que for necessário efetuar uma operação inacessível a usuários normais, como modificar o arquivo de senhas. Todavia, esse mecanismo pode ser perigoso, pois o processo filho recebe todos os privilégios do proprietário do arquivo, o que contraria o princípio do privilégio mínimo. Por exemplo, o programa `passwd` deveria somente receber a autorização para modificar o arquivo de senhas (`/etc/passwd`) e nada mais, pois o superusuário (*root user*) tem acesso a todos os recursos do sistema e pode efetuar todas as operações que desejar. Se o programa `passwd` contiver erros de programação, ele pode ser induzido pelo seu usuário a efetuar ações não previstas, visando comprometer a segurança do sistema (vide Seção 26.3).

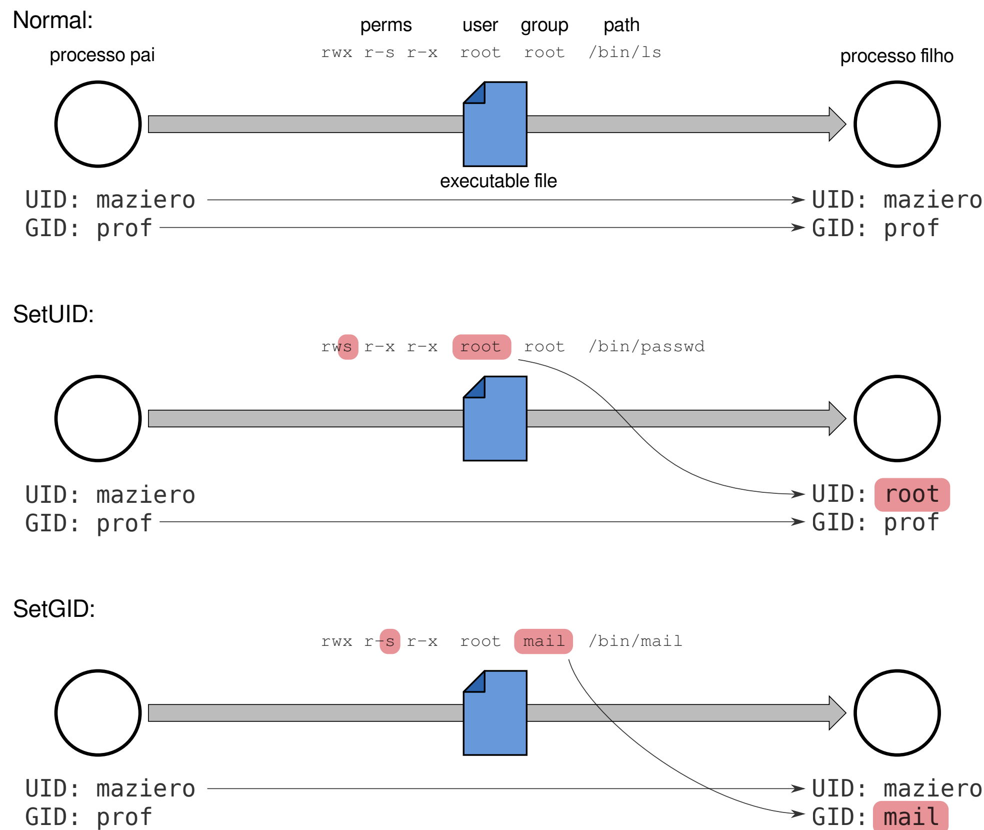


Figura 29.8: Funcionamento dos flags `setuid` e `setgid` do UNIX.

Uma alternativa mais segura aos flags `setuid` e `setgid` são os *privilégios POSIX* (*POSIX Capabilities*[2]), definidos no padrão POSIX 1003.1e [Gallmeister, 1994]. Nessa abordagem, o "poder absoluto" do super usuário é dividido em um grande número de pequenos privilégios específicos, que podem ser atribuídos a certos processos do sistema. Como medida adicional de proteção, cada processo pode ativar/desativar os privilégios que possui em função de sua necessidade. Vários sistemas UNIX implementam privilégios POSIX, como é o caso do Linux, que implementa:

- `CAP_CHOWN`: alterar o proprietário de um arquivo qualquer;
- `CAP_USER_DEV`: abrir dispositivos;
- `CAP_USER_FIFO`: usar *pipes* (comunicação);
- `CAP_USER_SOCK`: abrir *sockets* de rede;
- `CAP_NET_BIND_SERVICE`: abrir portas de rede com número abaixo de 1024;
- `CAP_NET_RAW`: abrir *sockets* de baixo nível (*raw sockets*);
- `CAP_KILL`: enviar sinais para processos de outros usuários.
- ... (outros +30 privilégios)

[2]O padrão POSIX usou indevidamente o termo "capacidade" para definir o que na verdade são privilégios associados aos processos. O uso indevido do termo *POSIX Capabilities* perdura até hoje em vários sistemas, como é o caso do Linux.

Para cada processo são definidos três conjuntos de privilégios: *Permitidos* (P), *Efetivos* (E) e *Herdáveis* (H). Os privilégios permitidos são aqueles que o processo pode ativar quando desejar, enquanto os efetivos são aqueles ativados no momento (respeitando-se $E \subseteq P$). O conjunto de privilégios herdáveis H é usado no cálculo dos privilégios transmitidos aos processos filhos. Os privilégios POSIX também podem ser atribuídos a programas executáveis em disco, substituindo os tradicionais (e perigosos) flags `setuid` e `setgid`. Assim, quando um executável for lançado, o novo processo recebe um conjunto de privilégios calculado a partir dos privilégios atribuídos ao arquivo executável e aqueles herdados do processo-pai que o criou [Bovet and Cesati, 2005].

Um caso especial de mudança de credenciais ocorre em algumas circunstâncias, quando é necessário **reduzir** as permissões de um processo. Por exemplo, o processo responsável pela autenticação de usuários em um sistema operacional deve criar novos processos para iniciar a sessão de trabalho de cada usuário. O processo autenticador geralmente executa com privilégios elevados, para poder acessar a bases de dados de autenticação dos usuários, enquanto os novos processos devem receber as credenciais do usuário autenticado, que normalmente tem menos privilégios. Em UNIX, um processo pode solicitar a mudança de suas credenciais através da chamada de sistema `setuid()`, entre outras. Em Windows, o mecanismo conhecido como *impersonation* permite a um processo ou *thread* abandonar temporariamente seu *token* de acesso e assumir outro, para realizar uma tarefa em nome do sujeito correspondente [Russinovich et al., 2008].

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:

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

$file_2 : (write),$
$program_1 : (read, execute),$
$socket_1 : (read, write, owner) \}$

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.

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

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 $\mathbb{C}(d_5) = \{\ \}$

9. Muitas vezes, usuários precisam executar ações que exigem privilégios administrativos, como instalar programas, reconfigurar serviços, etc. Neste contexto, indique quais das seguintes afirmações são incorretas; justifique suas respostas.

(a) No mecanismo UAC – *User Access Control* – dos sistemas Windows, um usuário administrativo inicia sua seção de trabalho com seus privilégios de usuário normal e recebe mais privilégios somente quando precisa efetuar ações que os requeiram.

(b) Alguns sistemas operacionais implementam mecanismos de *mudança de credenciais*, através dos quais um processo pode mudar de proprietário.

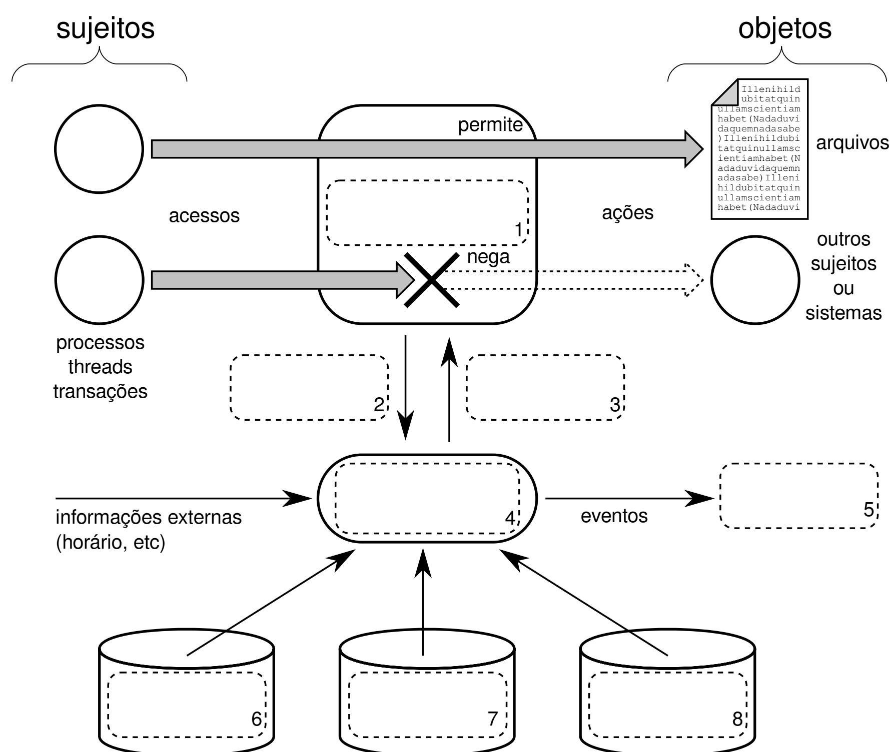
(c) As "POSIX Capabilities" são uma implementação do mecanismo de *capabilities* para sistemas operacionais que seguem o padrão POSIX.

(d) Alguns sistemas operacionais separam os usuários em *usuários normais* ou *administrativos*, atribuindo aos últimos permissões para efetuar tarefas administrativas, como instalar programas.

(e) Alguns sistemas operacionais implementam processos *monitores* que recebem pedidos de ações administrativas vindos de processos com baixo privilégio, que são avaliados e possivelmente atendidos.

(f) Os flags `setuid` e `setgid` do UNIX implementam um mecanismo de permissões temporárias.

10. O diagrama a seguir representa os principais componentes da infraestrutura de controle de acesso de um sistema operacional típico. Identifique e explique elementos representados pelas caixas tracejadas.



11. A listagem a seguir apresenta alguns programas executáveis e arquivos de dados em um mesmo diretório de um sistema UNIX, com suas respectivas permissões de acesso:

```
- rwx r-s ---  2 marge    family      indent
- rwx r-x --x  2 homer    family      more
- rws r-x --x  2 bart     men         nano
- rwx r-x ---  2 lisa     women       sha1sum

- rw- r-- ---  2 lisa     women       codigo.c
- rw- rw- ---  2 marge    family      dados.csv
- rw- r-- ---  2 bart     men         prova.pdf
- rw- rw- ---  2 homer    family      relatorio.txt
- rw- --- ---  2 bart     men         script.sh
```

Os programas executáveis precisam das seguintes permissões de acesso sobre os arquivos aos quais são aplicados para poder executar:

- `more`, `sha1sum`: leitura
- `nano`, `indent`: leitura e escrita

Considerando os grupos de usuários $men = \{bart, homer, moe\}$, $women = \{marge, lisa, maggie\}$ e $family = \{homer, marge, bart, lisa, maggie\}$, indique quais dos comandos a seguir serão permitidos e quais serão negados. O *prompt* indica qual usuário/grupo está executando o comando:

[ ] `lisa:women> nano codigo.c`

[ ] `lisa:women> more relatorio.txt`

[ ] `bart:men> nano relatorio.txt`

[ ] `bart:men> sha1sum prova.pdf`

[ ] `marge:women> more relatorio.txt`

[ ] `marge:women> indent codigo.c`

[ ] `homer:men> sha1sum prova.pdf`

[ ] `homer:men> nano dados.csv`

[ ] `moe:men> sha1sum relatorio.txt`

[ ] `moe:men> nano script.sh`

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.

A. Bomberger, A. Frantz, W. Frantz, A. Hardy, N. Hardy, C. Landau, and J. Shapiro. The KeyKOS nanokernel architecture. In *USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pages 95–112, 1992.

D. Bovet and M. Cesati. *Understanding the Linux Kernel, 3rd edition*. O'Reilly Media, Inc, 2005.

K. Brown. *Programming Windows Security*. Addison-Wesley Professional, 2000.

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.

B. Gallmeister. *POSIX.4: Programming for the Real World*. O'Reilly Media, Inc, 1994.

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.

G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. SeL4: Formal verification of an OS kernel. In *22nd ACM Symposium on Operating Systems Principles*, Big Sky, MT, USA, Oct 2009.

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. Liedtke. Toward real microkernels. *Communications of the ACM*, 39(9):70–77, 1996.

P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. In *USENIX Annual Technical Conference*, pages 29–42, 2001.

Microsoft. *Security Enhancements in Windows Vista*. Microsoft Corporation, May 2007.

N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. In *12th USENIX Security Symposium*, 2003.

M. Russinovich, D. Solomon, and A. Ionescu. *Microsoft Windows Internals, Fifth Edition*. Microsoft Press, 2008.

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.

J. Shapiro and N. Hardy. Eros: a principle-driven operating system from the ground up. *Software, IEEE*, 19(1):26–33, Jan/Feb 2002. ISSN 0740-7459. doi: 10.1109/52.976938.

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

R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask security architecture: System support for diverse security policies. In *8th USENIX Security Symposium*, pages 123–139, 1999.

Sun Microsystems. *Trusted Solaris User's Guide*. Sun Microsystems, Inc, June 2000.

R. Watson. TrustedBSD: Adding trusted operating system features to FreeBSD. In *USENIX Technical Conference*, 2001.