Capítulo 26

Conceitos básicos de segurança

A segurança de um sistema de computação diz respeito à garantia de algumas propriedades fundamentais associadas às informações e recursos presentes no sistema. Por "informação", compreende-se todos os recursos disponíveis no sistema, como registros de bancos de dados, arquivos, áreas de memória, dados de entrada/saída, tráfego de rede, configurações, etc.

Em Português, a palavra "segurança" abrange muitos significados distintos e por vezes conflitantes. Em Inglês, as palavras "security", "safety" e "reliability" permitem definir mais precisamente os diversos aspectos da segurança: a palavra "security" se relaciona a ameaças intencionais, como intrusões, ataques e roubo de informações; a palavra "safety" se relaciona a problemas que possam ser causados pelo sistema aos seus usuários ou ao ambiente, como acidentes provocados por erros de programação; por fim, o termo "reliability" é usado para indicar sistemas confiáveis, construídos para tolerar erros de software, de hardware ou dos usuários [Avizienis et al., 2004]. Neste capítulo serão considerados somente os aspectos de segurança relacionados à palavra inglesa "security", ou seja, a proteção do sistema contra ameaças intencionais.

Este capítulo trata dos principais conceitos de segurança, como as propriedades e princípios de segurança, ameaças, vulnerabilidades e ataques típicos em sistemas operacionais, concluindo com uma descrição da infraestrutura de segurança típica de um sistema operacional. Grande parte dos tópicos de segurança apresentados neste e nos próximos capítulos não são exclusivos de sistemas operacionais, mas se aplicam a sistemas de computação em geral.

26.1 Propriedades e princípios de segurança

A segurança de um sistema de computação pode ser expressa através de algumas propriedades fundamentais [Amoroso, 1994]:

Confidencialidade: os recursos presentes no sistema só podem ser consultados por usuários devidamente autorizados a isso;

Integridade: os recursos do sistema só podem ser modificados ou destruídos pelos usuários autorizados a efetuar tais operações;

Disponibilidade: os recursos devem estar disponíveis para os usuários que tiverem direito de usá-los, a qualquer momento.

Além destas, outras propriedades importantes estão geralmente associadas à segurança de um sistema:

Autenticidade: todas as entidades do sistema são autênticas ou genuínas; em outras palavras, os dados associados a essas entidades são verdadeiros e correspondem às informações do mundo real que elas representam, como as identidades dos usuários, a origem dos dados de um arquivo, etc.;

Irretratabilidade: Todas as ações realizadas no sistema são conhecidas e não podem ser escondidas ou negadas por seus autores; esta propriedade também é conhecida como *irrefutabilidade* ou *não-repúdio*.

É função do sistema operacional garantir a manutenção das propriedades de segurança para todos os recursos sob sua responsabilidade. Essas propriedades podem estar sujeitas a violações decorrentes de erros de software ou humanos, praticadas por indivíduos mal intencionados (maliciosos), internos ou externos ao sistema.

Além das técnicas usuais de engenharia de software para a produção de sistemas corretos, a construção de sistemas computacionais seguros é pautada por uma série de princípios específicos, relativos tanto à construção do sistema quanto ao comportamento dos usuários e dos atacantes. Alguns dos princípios mais relevantes, compilados a partir de [Saltzer and Schroeder, 1975; Lichtenstein, 1997; Pfleeger and Pfleeger, 2006], são indicados a seguir:

Privilégio mínimo: todos os usuários e programas devem operar com o mínimo possível de privilégios ou permissões de acesso necessários para poder funcionar. Dessa forma, os danos provocados por erros ou ações maliciosas intencionais serão minimizados.

Separação de privilégios: sistemas de proteção baseados em mais de um controle ou regra são mais robustos, pois se o atacante conseguir burlar um dos controles, mesmo assim não terá acesso ao recurso. Em um sistema bancário, por exemplo, uma operação de valor elevado pode requerer a autorização de dois gerentes.

Mediação completa: todos os acessos a recursos, tanto diretos quanto indiretos, devem ser verificados pelos mecanismos de segurança. Eles devem estar dispostos de forma a ser impossível contorná-los.

Default seguro: o mecanismo de segurança deve identificar claramente os acessos permitidos; caso um certo acesso não seja explicitamente permitido, ele deve ser negado. Este princípio impede que acessos inicialmente não previstos no projeto do sistema sejam inadvertidamente autorizados.

Economia de mecanismo: o projeto de um sistema de proteção deve ser pequeno e simples, para que possa ser facilmente e profundamente analisado, testado e validado.

Compartilhamento mínimo: mecanismos compartilhados entre usuários são fontes potenciais de problemas de segurança, devido à possibilidade de fluxos de informação imprevistos entre usuários. Por isso, o uso de mecanismos compartilhados deve ser minimizado, sobretudo se envolver áreas de memória

compartilhadas. Por exemplo, caso uma certa funcionalidade do sistema operacional possa ser implementada como chamada ao núcleo ou como função de biblioteca, deve-se preferir esta última forma, pois envolve menos compartilhamento.

Projeto aberto: a robustez do mecanismo de proteção não deve depender da ignorância dos atacantes; ao invés disso, o projeto deve ser público e aberto, dependendo somente do segredo de poucos itens, como listas de senhas ou chaves criptográficas. Um projeto aberto também torna possível a avaliação por terceiros independentes, provendo confirmação adicional da segurança do mecanismo.

Proteção adequada: cada recurso computacional deve ter um nível de proteção coerente com seu valor intrínseco. Por exemplo, o nível de proteção requerido em um servidor Web de serviços bancário é bem distinto daquele de um terminal público de acesso à Internet.

Facilidade de uso: o uso dos mecanismos de segurança deve ser fácil e intuitivo, caso contrário eles serão evitados pelos usuários.

Eficiência: os mecanismos de segurança devem ser eficientes no uso dos recursos computacionais, de forma a não afetar significativamente o desempenho do sistema ou as atividades de seus usuários.

Elo mais fraco: a segurança do sistema é limitada pela segurança de seu elemento mais vulnerável, seja ele o sistema operacional, as aplicações, a conexão de rede ou o próprio usuário.

Esses princípios devem pautar a construção, configuração e operação de qualquer sistema computacional com requisitos de segurança. A imensa maioria dos problemas de segurança dos sistemas atuais provém da não observação desses princípios.

26.2 Ameaças

Como ameaça, pode ser considerada qualquer ação que coloque em risco as propriedades de segurança do sistema descritas na seção anterior. Alguns exemplos de ameaças às propriedades básicas de segurança seriam:

- *Ameaças à confidencialidade*: um processo vasculhar as áreas de memória de outros processos, arquivos de outros usuários, tráfego de rede nas interfaces locais ou áreas do núcleo do sistema, buscando dados sensíveis como números de cartão de crédito, senhas, e-mails privados, etc.;
- *Ameaças à integridade*: um processo alterar as senhas de outros usuários, instalar programas, *drivers* ou módulos de núcleo maliciosos, visando obter o controle do sistema, roubar informações ou impedir o acesso de outros usuários;
- *Ameaças à disponibilidade*: um usuário alocar para si todos os recursos do sistema, como a memória, o processador ou o espaço em disco, para impedir que outros usuários possam utilizá-lo.

Obviamente, para cada ameaça possível, devem existir estruturas no sistema operacional que impeçam sua ocorrência, como controles de acesso às áreas de memória e arquivos, quotas de uso de memória e processador, verificação de autenticidade de *drivers* e outros softwares, etc.

As ameaças podem ou não se concretizar, dependendo da existência e da correção dos mecanismos construídos para evitá-las ou impedi-las. As ameaças podem se tornar realidade à medida em que existam vulnerabilidades que permitam sua ocorrência.

26.3 Vulnerabilidades

Uma vulnerabilidade é um defeito ou problema presente na especificação, implementação, configuração ou operação de um software ou sistema, que possa ser explorado para violar as propriedades de segurança do mesmo. Alguns exemplos de vulnerabilidades são descritos a seguir:

- um erro de programação no serviço de compartilhamento de arquivos, que permita a usuários externos o acesso a outros arquivos do computador local, além daqueles compartilhados;
- uma conta de usuário sem senha, ou com uma senha predefinida pelo fabricante, que permita a usuários não autorizados acessar o sistema;
- ausência de quotas de disco, permitindo a um único usuário alocar todo o espaço em disco para si e assim impedir os demais usuários de usar o sistema.

A grande maioria das vulnerabilidades ocorre devido a erros de programação, como, por exemplo, não verificar a conformidade dos dados recebidos de um usuário ou da rede. Em um exemplo clássico, o processo servidor de impressão `lpd`, usado em alguns UNIX, pode ser instruído a imprimir um arquivo e a seguir apagá-lo, o que é útil para imprimir arquivos temporários. Esse processo executa com permissões administrativas pois precisa acessar a porta de entrada/saída da impressora, o que lhe confere acesso a todos os arquivos do sistema. Por um erro de programação, uma versão antiga do processo `lpd` não verificava corretamente as permissões do usuário sobre o arquivo a imprimir; assim, um usuário malicioso podia pedir a impressão (e o apagamento) de arquivos do sistema. Em outro exemplo clássico, uma versão antiga do servidor HTTP Microsoft IIS não verificava adequadamente os pedidos dos clientes; por exemplo, um cliente que solicitasse a URL `http://www.servidor.com/../../../../windows/system.ini`, receberia como resultado o conteúdo do arquivo de sistema `system.ini`, ao invés de ter seu pedido recusado.

Uma classe especial de vulnerabilidades decorrentes de erros de programação são os chamados "estouros" de *buffer* e de pilha (*buffer/stack overflows*). Nesse erro, o programa escreve em áreas de memória indevidamente, com resultados imprevisíveis, como mostra o exemplo a seguir e o resultado de sua execução:

```c
#include <stdio.h>
#include <stdlib.h>

int i, j, buffer[20], k;  // declara buffer[0] a buffer[19]

int main()
{
  i = j = k = 0 ;

  for (i = 0; i<= 20; i++) // usa buffer[0] a buffer[20] <-- erro!
    buffer[i] = random() ;

  printf ("i: %d\nj: %d\nk: %d\n", i, j, k) ;

  return(0);
}
```

A execução desse código gera o seguinte resultado:

```
host:~> cc buffer-overflow.c -o buffer-overflow
host:~> buffer-overflow
i: 21
j: 35005211
k: 0
```

Pode-se observar que os valores $i = 21$ e $k = 0$ são os previstos, mas o valor da variável j mudou "misteriosamente" de 0 para 35005211. Isso ocorreu porque, ao acessar a posição `buffer[20]`, o programa extrapolou o tamanho do vetor e escreveu na área de memória sucessiva[1], que pertence à variável j. Esse tipo de erro é muito frequente em linguagens como C e C++, que não verificam os limites de alocação das variáveis durante a execução. O erro de estouro de pilha é similar a este, mas envolve variáveis alocadas na pilha usada para o controle de execução de funções.

Se a área de memória invadida pelo estouro de *buffer* contiver código executável, o processo pode ter erros de execução e ser abortado. A pior situação ocorre quando os dados a escrever no *buffer* são lidos do terminal ou recebidos através da rede: caso o atacante conheça a organização da memória do processo, ele pode escrever inserir instruções executáveis na área de memória invadida, mudando o comportamento do processo ou abortando-o. Caso o *buffer* esteja dentro do núcleo, o que ocorre em *drivers* e no suporte a protocolos de rede como o TCP/IP, um estouro de *buffer* pode travar o sistema ou permitir acessos indevidos a recursos. Um bom exemplo é o famoso *Ping of Death* [Pfleeger and Pfleeger, 2006], no qual um pacote de rede no protocolo ICMP, com um conteúdo específico, podia paralisar computadores na rede local.

Além dos estouros de *buffer* e pilha, há uma série de outros erros de programação e de configuração que podem constituir vulnerabilidades, como o uso descuidado das strings de formatação de operações de entrada/saída em linguagens como C e C++ e condições de disputa na manipulação de arquivos compartilhados. Uma explicação mais detalhada desses erros e de suas implicações pode ser encontrada em [Pfleeger and Pfleeger, 2006].

[1] As variáveis não são alocadas na memória necessariamente na ordem em que são declaradas no código fonte. A ordem de alocação das variáveis varia com o compilador usado e depende de vários fatores, como a arquitetura do processador, estratégias de otimização de código, etc.

26.4 Ataques

Um ataque é o ato de utilizar (ou explorar) uma vulnerabilidade para violar uma propriedade de segurança do sistema. De acordo com [Pfleeger and Pfleeger, 2006], existem basicamente quatro tipos de ataques, representados na Figura 26.1:

Interrupção: consiste em impedir o fluxo normal das informações ou acessos; é um ataque à disponibilidade do sistema;

Interceptação: consiste em obter acesso indevido a um fluxo de informações, sem necessariamente modificá-las; é um ataque à confidencialidade;

Modificação: consiste em modificar de forma indevida informações ou partes do sistema, violando sua integridade;

Fabricação: consiste em produzir informações falsas ou introduzir módulos ou componentes maliciosos no sistema; é um ataque à autenticidade.

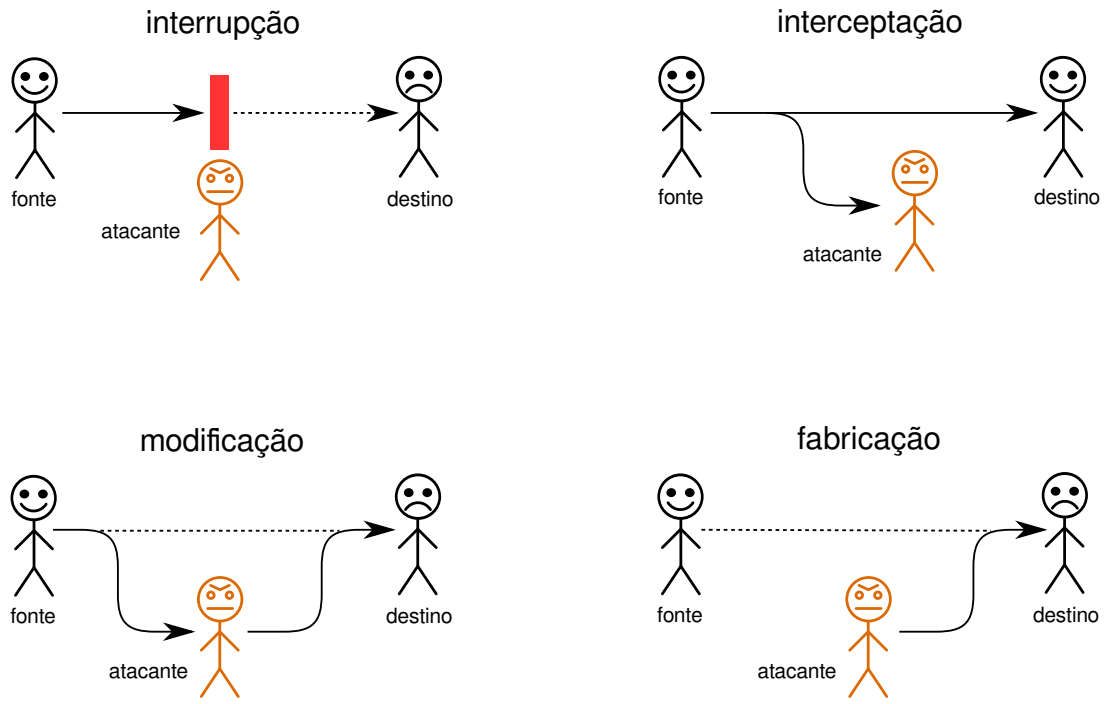

Figura 26.1: Tipos básicos de ataques (inspirado em [Pfleeger and Pfleeger, 2006]).

Existem ataques **passivos**, que visam capturar informações confidenciais, e ataques **ativos**, que visam introduzir modificações no sistema para beneficiar o atacante ou impedir seu uso pelos usuários válidos. Além disso, os ataques a um sistema operacional podem ser **locais**, quando executados por usuários válidos do sistema, ou **remotos**, quando são realizados através da rede, sem fazer uso de uma conta de usuário local. Um programa especialmente construído para explorar uma determinada vulnerabilidade de sistema e realizar um ataque é denominado *exploit*.

Uma intrusão ou invasão é um ataque bem sucedido, que dá ao atacante acesso indevido a um sistema. Uma vez dentro do sistema, o intruso pode usá-lo para fins escusos, como enviar *spam*, atacar outros sistemas ou minerar moedas

digitais. Frequentemente o intruso efetua novos ataques para aumentar seu nível de acesso no sistema, o que é denominado *elevação de privilégio* (*privilege escalation*). Esses ataques exploram vulnerabilidades em programas do sistema (que executam com mais privilégios), ou do próprio núcleo, através de chamadas de sistema, para alcançar os privilégios do administrador.

Por outro lado, os ataques de negação de serviços (DoS – *Denial of Service*) visam prejudicar a disponibilidade do sistema, impedindo que os usuários válidos do sistema possam utilizá-lo, ou seja, que o sistema execute suas funções. Esse tipo de ataque é muito comum em ambientes de rede, com a intenção de impedir o acesso a servidores Web, DNS e de e-mail. Em um sistema operacional, ataques de negação de serviço podem ser feitos com o objetivo de consumir todos os recursos locais, como processador, memória, arquivos abertos, *sockets* de rede ou semáforos, dificultando ou mesmo impedindo o uso desses recursos pelos demais usuários.

O antigo ataque *fork bomb* dos sistemas UNIX é um exemplo trivial de ataque DoS local: ao executar, o processo atacante se reproduz rapidamente, usando a chamada de sistema `fork` (vide código a seguir). Cada processo filho continua executando o mesmo código do processo pai, criando novos processos filhos, e assim sucessivamente. Em consequência, a tabela de processos do sistema é rapidamente preenchida, impedindo a criação de processos pelos demais usuários. Além disso, o grande número de processos solicitando chamadas de sistema mantém o núcleo ocupado, impedindo os a execução dos demais processos.

```
1 #include <unistd.h>
2
3 int main()
4 {
5   while (1)    // laço infinito
6     fork() ;   // reproduz o processo
7 }
```

Ataques similares ao *fork bomb* podem ser construídos para outros recursos do sistema operacional, como memória, descritores de arquivos abertos, *sockets* de rede e espaço em disco. Cabe ao sistema operacional impor limites máximos (quotas) de uso de recursos para cada usuário e definir mecanismos para detectar e conter processos excessivamente "gulosos".

Recentemente têm ganho atenção os ataques à confidencialidade, que visam roubar informações sigilosas dos usuários. Com o aumento do uso da Internet para operações financeiras, como acesso a sistemas bancários e serviços de compras *online*, o sistema operacional e os navegadores manipulam informações sensíveis, como números de cartões de crédito, senhas de acesso a contas bancárias e outras informações pessoais. Programas construídos com a finalidade específica de realizar esse tipo de ataque são denominados *spyware*.

Deve ficar clara a distinção entre *ataques* e *incidentes de segurança*. Um incidente de segurança é qualquer fato intencional ou acidental que comprometa uma das propriedades de segurança do sistema. A intrusão de um sistema ou um ataque de negação de serviços são considerados incidentes de segurança, assim como o vazamento acidental de informações confidenciais.

26.5 Malwares

Denomina-se genericamente *malware* todo programa cuja intenção é realizar atividades ilícitas, como realizar ataques, roubar informações ou dissimular a presença de intrusos em um sistema. Existe uma grande diversidade de *malwares*, destinados às mais diversas finalidades [Shirey, 2000; Pfleeger and Pfleeger, 2006]. As funcionalidades mais comuns dos *malwares* são:

Vírus: um vírus de computador é um trecho de código que se infiltra em programas executáveis existentes no sistema operacional, usando-os como suporte para sua execução e replicação[2]. Quando um programa "infectado" é executado, o vírus também se executa, infectando outros executáveis e eventualmente executando outras ações danosas. Alguns tipos de vírus são programados usando macros de aplicações complexas, como editores de texto, e usam os arquivos de dados dessas aplicações como suporte. Outros tipos de vírus usam o código de inicialização dos discos e outras mídias como suporte de execução.

Worm: ao contrário de um vírus, um "verme" é um programa autônomo, que se propaga sem infectar outros programas. A maioria dos vermes se propaga explorando vulnerabilidades nos serviços de rede, que os permitam invadir e instalar-se em sistemas remotos. Alguns vermes usam o sistema de e-mail como vetor de propagação, enquanto outros usam mecanismos de autoexecução de mídias removíveis (como *pendrives*) como mecanismo de propagação. Uma vez instalado em um sistema, o verme pode instalar *spywares* ou outros programas nocivos.

Trojan horse: de forma análoga ao personagem da mitologia grega, um "cavalo de Tróia" computacional é um programa com duas funcionalidades: uma funcionalidade lícita conhecida de seu usuário e outra ilícita, executada sem que o usuário a perceba. Muitos cavalos de Tróia são usados como vetores para a instalação de outros *malwares*. Um exemplo clássico é o famoso *Happy New Year 99*, distribuído através de e-mails, que usava uma animação de fogos de artifício como fachada para a propagação de um verme. Para convencer o usuário a executar o cavalo de Tróia podem ser usadas técnicas de *engenharia social* [Mitnick and Simon, 2002].

Exploit: é um programa escrito para explorar vulnerabilidades conhecidas, como prova de conceito ou como parte de um ataque. Os *exploits* podem estar incorporados a outros *malwares* (como vermes e *trojans*) ou constituírem ferramentas autônomas, usadas em ataques manuais.

Packet sniffer: um "farejador de pacotes" captura pacotes de rede do próprio computador ou da rede local, analisando-os em busca de informações sensíveis como senhas e dados bancários. A cifragem do conteúdo da rede resolve parcialmente esse problema, embora um *sniffer* na máquina local possa capturar os dados antes que sejam cifrados, ou depois de decifrados.

[2] De forma análoga, um vírus biológico precisa de uma célula hospedeira, pois usa o material celular como suporte para sua existência e replicação.

Keylogger: software dedicado a capturar e analisar as informações digitadas pelo usuário na máquina local, sem seu conhecimento. Essas informações podem ser transferidas a um computador remoto periodicamente ou em tempo real, através da rede.

Rootkit: é um conjunto de programas destinado a ocultar a presença de um intruso no sistema operacional. Como princípio de funcionamento, o *rootkit* modifica os mecanismos do sistema operacional que mostram os processos em execução, arquivos nos discos, portas e conexões de rede, etc., para ocultar o intruso. Os *rootkits* mais simples substituem utilitários do sistema, como `ps` (lista de processos), `ls` (arquivos), `netstat` (conexões de rede) e outros, por versões adulteradas que não mostrem os arquivos, processos e conexões de rede do intruso. Versões mais elaboradas de *rootkits* substituem bibliotecas do sistema operacional ou modificam partes do próprio núcleo, o que torna complexa sua detecção e remoção.

Backdoor: uma "porta dos fundos" é um programa que facilita a entrada posterior do atacante em um sistema já invadido. Geralmente a porta dos fundos é criada através um processo servidor de conexões remotas (usando SSH, telnet ou um protocolo ad-hoc). Muitos *backdoors* são instalados a partir de *trojans*, vermes ou *rootkits*.

Ransomware: categoria recente de malware, que visa sequestrar os dados do usuário. O sequestro é realizado cifrando os arquivos do usuário com uma chave secreta, que só será fornecida pelo atacante ao usuário se este pagar um valor de resgate.

Deve-se ter em mente que existe muita confusão na mídia e na literatura em relação à nomenclatura de *malwares*; além disso, a maioria dos *malwares* atuais são complexos, apresentando várias funcionalidades complementares. Por exemplo, um mesmo *malware* pode se propagar como *worm*, dissimular-se no sistema como *rootkit*, capturar informações locais usando *keylogger* e manter uma *backdoor* para acesso remoto. Por isso, esta seção procurou dar uma definição tecnicamente precisa de cada funcionalidade, sem a preocupação de aspresentar exemplos reais de *malwares* em cada uma dessas categorias.

26.6 Infraestrutura de segurança

De forma genérica, o conjunto de todos os elementos de hardware e software considerados críticos para a segurança de um sistema são denominados **Base Computacional Confiável** (TCB – *Trusted Computing Base*) ou **núcleo de segurança** (*security kernel*). Fazem parte da TCB todos os elementos do sistema cuja falha possa representar um risco à sua segurança. Os elementos típicos de uma base de computação confiável incluem os mecanismos de proteção do hardware (tabelas de páginas/segmentos, modo usuário/núcleo do processador, instruções privilegiadas, etc.) e os diversos subsistemas do sistema operacional que visam garantir as propriedades básicas de segurança, como o controle de acesso aos arquivos, acesso às portas de rede, etc.

O sistema operacional emprega várias técnicas complementares para garantir a segurança de um sistema operacional. Essas técnicas estão classificadas nas seguintes grandes áreas:

Autenticação: conjunto de técnicas usadas para identificar inequivocamente usuários e recursos em um sistema; podem ir de simples pares *login/senha* até esquemas sofisticados de biometria ou certificados criptográficos. No processo básico de autenticação, um usuário externo se identifica para o sistema através de um procedimento de autenticação; no caso da autenticação ser bem-sucedida, é aberta uma *sessão*, na qual são criados uma ou mais entidades (processos, *threads*, transações, etc.) para representar aquele usuário dentro do sistema.

Controle de acesso: técnicas usadas para definir quais ações são permitidas e quais são negadas no sistema; para cada usuário do sistema, devem ser definidas regras descrevendo as ações que este pode realizar no sistema, ou seja, que recursos este pode acessar e sob que condições. Normalmente, essas regras são definidas através de uma *política de controle de acesso*, que é imposta a todos os acessos que os usuários efetuam sobre os recursos do sistema.

Auditoria: técnicas usadas para manter um registro das atividades efetuadas no sistema, visando a contabilização de uso dos recursos, a análise posterior de situações de uso indevido ou a identificação de comportamentos suspeitos.

A Figura 26.2 ilustra alguns dos conceitos vistos até agora. Nessa figura, as partes indicadas em cinza e os mecanismos utilizados para implementá-las constituem a base de computação confiável do sistema.

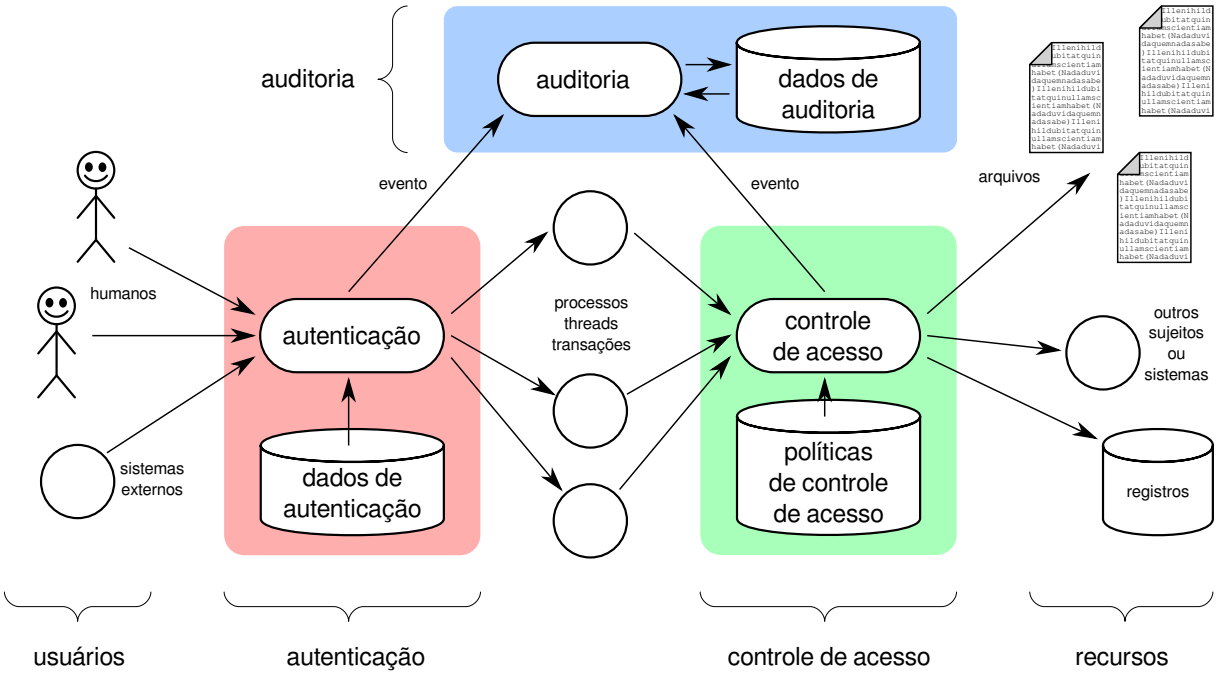

Figura 26.2: Base de computação confiável de um sistema operacional.

Sob uma ótica mais ampla, a base de computação confiável de um sistema informático compreende muitos fatores além do sistema operacional em si. A manutenção das propriedades de segurança depende do funcionamento correto de todos os elementos do sistema, do hardware ao usuário final.

O hardware fornece várias funcionalidades essenciais para a proteção do sistema: os mecanismos de memória virtual (MMU) permitem isolar o núcleo e os processos entre

si; o mecanismo de interrupção de software provê uma interface controlada de acesso ao núcleo; os níveis de execução do processador permitem restringir as instruções e as portas de entrada saída acessíveis aos diversos softwares que compõem o sistema; além disso, muitos tipos de hardware permitem impedir operações de escrita ou execução de código em certas áreas de memória.

No nível do sistema operacional surgem os processos isolados entre si, as contas de usuários, os mecanismos de autenticação e controle de acesso e os registros de auditoria. Em paralelo com o sistema operacional estão os utilitários de segurança, como antivírus, verificadores de integridade, detectores de intrusão, entre outros.

As linguagens de programação também desempenham um papel importante nesse contexto, pois muitos problemas de segurança têm origem em erros de programação. O controle estrito de índices em vetores, a restrição do uso de ponteiros e a limitação de escopo de nomes para variáveis e funções são exemplos de aspectos importantes para a segurança de um programa. Por fim, as aplicações também têm responsabilidade em relação à segurança, no sentido de ter implementações corretas e validar todos os dados manipulados. Isso é particularmente importante em aplicações multi-usuários (como sistemas corporativos e sistemas Web) e processos privilegiados que recebam requisições de usuários ou da rede (servidores de impressão, de DNS, etc.).

Exercícios

1. Sobre as afirmações a seguir, relativas às propriedades de segurança, indique quais são **incorretas**, justificando sua resposta:

   (a) A *Confidencialidade* consiste em garantir que as informações do sistema estarão criptografadas.

   (b) A *Integridade* consiste em garantir que as informações do sistema só poderão ser modificadas por usuários autorizados.

   (c) A *Disponibilidade* implica em assegurar que os recursos do sistema estarão disponíveis para consulta por qualquer usuário.

   (d) A *Autenticidade* implica em assegurar que os dados das entidades atuantes no sistema sejam verdadeiros e correspondam às informações do mundo real que elas representam.

   (e) A *Irretratabilidade* implica em garantir que nenhuma ação possa ser desfeita no sistema.

2. Sobre as afirmações a seguir, relativas aos princípios de segurança, indique quais são **incorretas**, justificando sua resposta:

   (a) Princípio do *Privilégio Mínimo*: os processos devem receber o mínimo possível de privilégios, para minimizar os riscos em caso de bugs ou erros.

   (b) Princípio do *Default Seguro*: os acessos permitidos devem ser explicitados; caso um acesso não seja explicitamente permitido, ele deve ser negado.

   (c) Princípio da *Separação de Privilégios*: os privilégios dos usuários comuns devem ser separados dos privilégios do administrador do sistema.

(d) Princípio do *Projeto Aberto*: a robustez do mecanismo de proteção não deve depender de segredos de programação.

(e) Princípio da *Facilidade de Uso*: o uso dos mecanismos de segurança deve ser fácil e intuitivo para os usuários.

3. Relacione as situações abaixo a ataques diretos à (C)onfidencialidade, (I)ntegridade, (D)isponibilidade ou (A)utenticidade, justificando suas escolhas.

[ ] Um programa que permite injetar pacotes falsos na rede.

[ ] Um ataque de negação de serviços através da rede.

[ ] Um processo *spyware* que vasculha os arquivos do sistema em busca de senhas.

[ ] `int main { while (1) fork(); }`

[ ] Um site malicioso que imita um site bancário.

[ ] Um programa quebrador de senhas.

[ ] Um processo que modifica o arquivo de sistema `/etc/hosts` para redirecionar acessos de rede.

[ ] Um programa baixado da Internet que instala um *malware* oculto no sistema operacional.

[ ] Uma página Web cheia de arquivos *Flash* para sobrecarregar o processador.

[ ] Um programa de captura de pacotes de rede.

4. O código a seguir apresenta uma vulnerabilidade de segurança. Indique qual é essa vulnerabilidade e explique como ela pode ser usada por um atacante.

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <ctype.h>
4
5  int confirma (char * pergunta)
6  {
7    char resp[20] ;
8
9    printf ("%s (sim/nao): ", pergunta) ;
10   scanf ("%s", &resp[0]) ;
11   if (! strcmp (resp, "sim")) return (1) ;
12   return (0) ;
13 }
14
15 int main ()
16 {
17   ...
18
19   if (confirma ("Devo apagar os valores?"))
20   {
21     ...
22   }
23   ...
24 }
```

5. Relacione os tipos de *malwares* às suas respectivas descrições:

   (V)írus, (W)orm, (T)rojan, (R)ootkit, (B)ackdoor, (E)xploit

   [ ] Pode operar no nível dos comandos, das bibliotecas, do núcleo do sistema operacional ou mesmo abaixo dele.

   [ ] Técnicas de engenharia social geralmente são empregadas para induzir o usuário a executar esse tipo de programa.

   [ ] É um programa que se propaga entre sistemas usando vulnerabilidades em seus serviços.

   [ ] É um trecho de código que se infiltra em programas executáveis, usando-os como suporte para sua execução e propagação.

   [ ] É um programa construído para demonstrar ou explorar uma vulnerabilidade de um sistema.

   [ ] Pode usar sistemas de e-mail ou de mensagens instantâneas para sua propagação.

   [ ] É um programa que facilita a entrada do intruso em um sistema já invadido, ou que permite seu comando remotamente.

   [ ] Programa usado para esconder a presença de um intruso no sistema.

   [ ] Sua execução depende da execução do programa hospedeiro.

   [ ] É um programa usado para enganar o usuário e fazê-lo instalar outros *malwares*.

   [ ] Pode usar suportes de execução internos (macros) de editores de texto para sua propagação.

   [ ] Costuma infectar *pendrives* plugados em portas USB.

Referências

E. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall PTR, 1994.

A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1), Mar. 2004.

S. Lichtenstein. A review of information security principles. *Computer Audit Update*, 1997(12):9–24, December 1997.

K. D. Mitnick and W. L. Simon. *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons, Inc., New York, NY, USA, 2002. ISBN 0471237124.

C. Pfleeger and S. L. Pfleeger. *Security in Computing, 4th Edition*. Prentice Hall PTR, 2006.

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

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