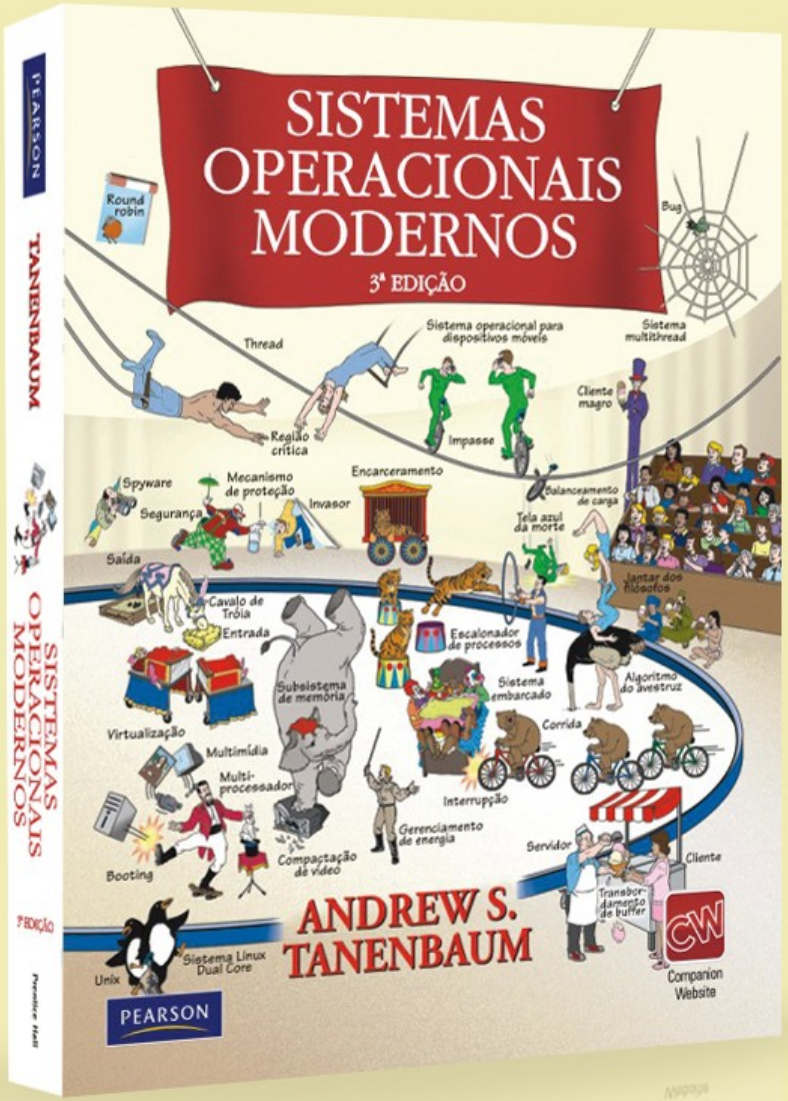

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO
ANDREW S. TANENBAUM
PEARSON
Round robin
Bug
Thread
Sistema operacional para dispositivos móveis
Sistema multithread
Região crítica
Impasse
Cliente magro
Spyware
Segurança
Mecanismo de proteção
Invasor
Encarceramento
Balanceamento de carga
Tela azul da morte
Saída
Jantar dos filósofos
Cavalo de Tróia
Entrada
Escalonador de processos
Subsistema de memória
Sistema embarcado
Algoritmo do avestruz
Virtualização
Multimídia
Multi-processador
Corrida
Interrupção
Gerenciamento de energia
Servidor
Cliente
Booting
Compactação de vídeo
Transbordamento de buffer
Unix
Sistema Linux Dual Core
CW
Companion Website

Sistemas operacionais modernos
Terceira edição

ANDREW S. TANENBAUM

Capítulo 9
Segurança

Ameaças

Meta	Ameaça
Confidencialidade de dados	Exposição de dados
Integridade de dados	Manipulação de dados
Disponibilidade do sistema	Recusa de serviços
Exclusão de invasores	Controle do sistema por vírus

Tabela 9.1 Segurança: metas e ameaças.

Intrusos

Categorias comuns

- Curiosidade casual de usuários leigos.
- Espionagem por pessoal interno.
- Tentativas determinadas para se ganhar dinheiro.
- Espionagem comercial ou militar.

Perda de dados acidental

Causas comuns para perda de dados acidental:

- Fenômenos naturais: incêndios, enchentes, terremotos, guerras, motins, ratos roendo fitas ou discos flexíveis.
- Erros de hardware ou software: defeitos na CPU, discos ou fitas com problemas de leitura, erros de telecomunicação ou de programas.
- Erros humanos: entrada incorreta de dados, montagem incorreta de disco ou fita, execução de programa errado, perda de disco ou fita ou outros erros.

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Criptografia básica

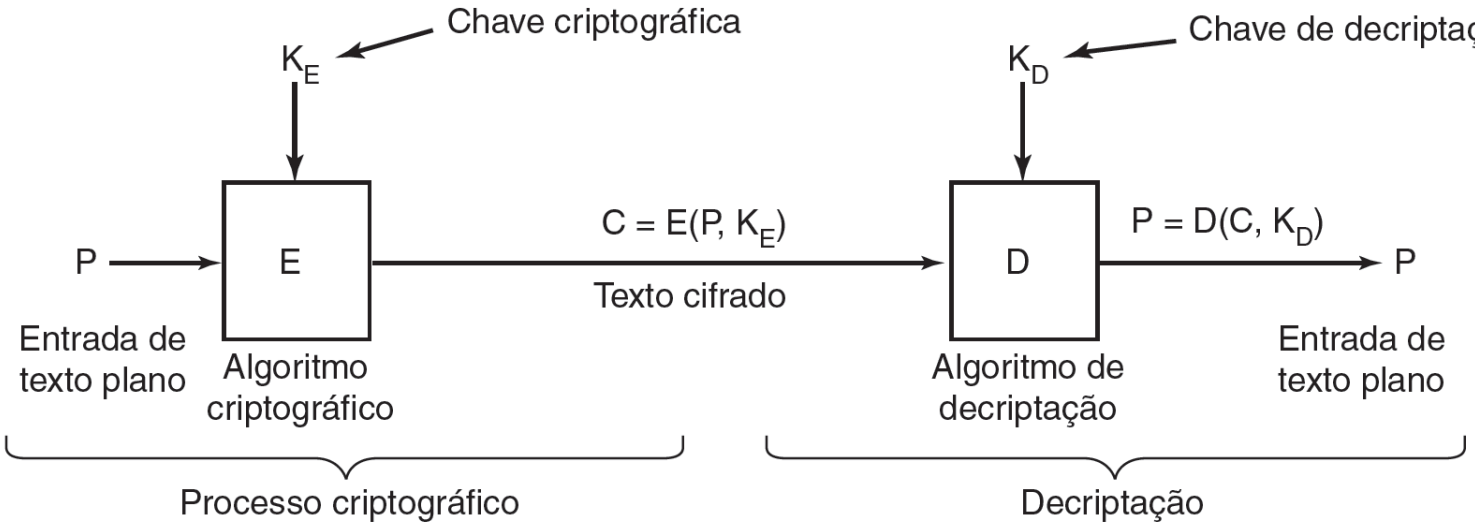


Figura 9.1 Relacionamento entre o texto puro e o texto cifrado.

Criptografia por chave secreta

Substituição monoalfabética:

Texto puro: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

Texto cifrado: `QWERTYUIOPASDFGHJKLZXCVBNM`

Criptografia por chave pública

- A criptografia usa uma operação “fácil”, tal como quanto é 314159265358979 × 314159265358979?

- A decriptação sem a chave requer uma operação difícil, tal como fazer a raiz quadrada de 391257150641938709059482850824 1.

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Assinaturas digitais

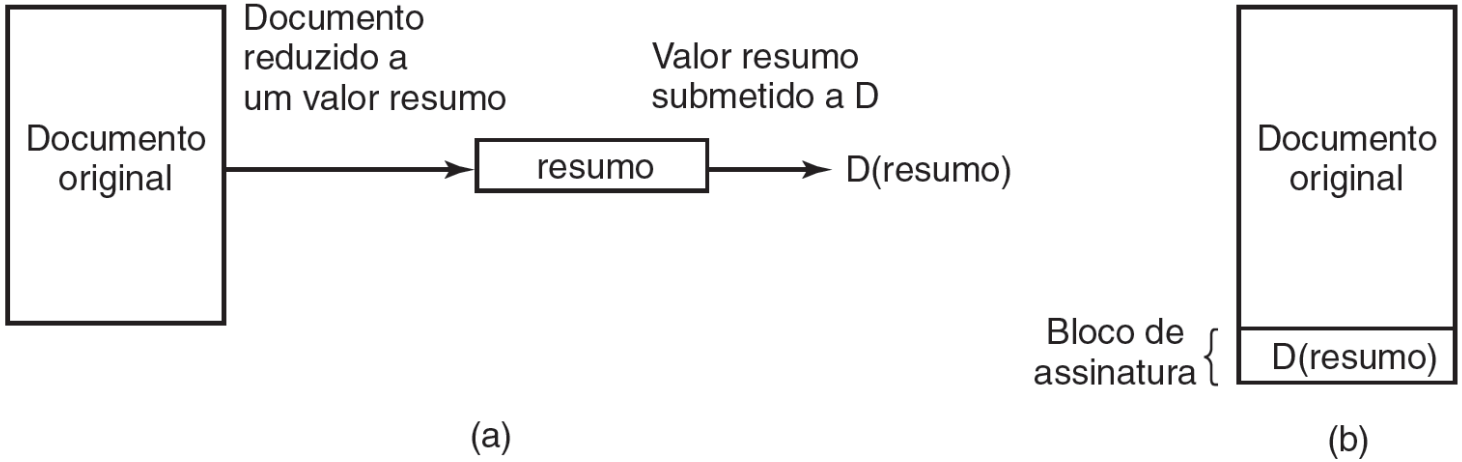


Figura 9.2 (a) Calculando um bloco de assinatura. (b) O que o receptor recebe.

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Domínios de proteção: um domínio é um conjunto de pares (objetos, direitos).

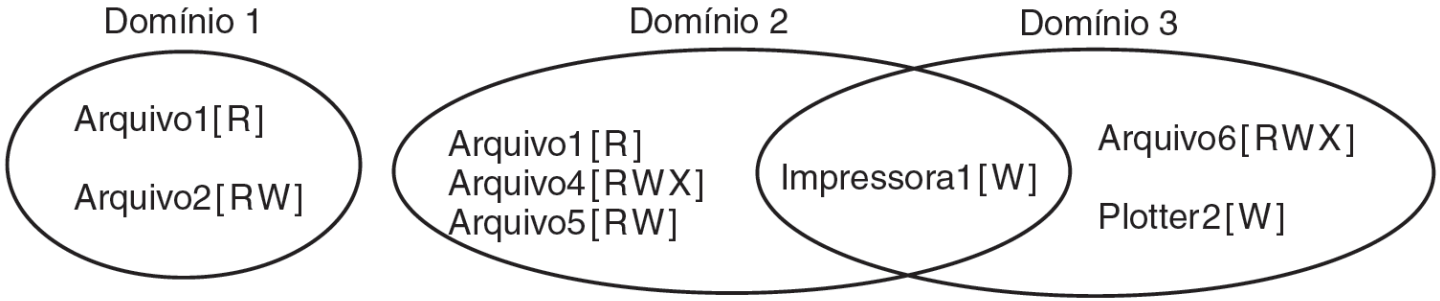


Figura 9.3 Três domínios de proteção.

Para cada domínio, permite identificar quais objetos (e operações sobre estes) que estão acessíveis.

	Objeto							
	Arquivo1	Arquivo2	Arquivo3	Arquivo4	Arquivo5	Arquivo6	Impressora1	Plotter2
Domínio 1	Leitura	Leitura Escrita						
2			Leitura	Leitura Escrita Execução	Leitura Escrita		Escrita	
3						Leitura Escrita Execução	Escrita	Escrita

Figura 9.4 Uma matriz de proteção.

Os próprios **domínios inclusos como objetos**: nesse exemplo, mostra-se que estando no domínio 1 pode-se entrar no domínio 2 mas, uma vez lá, não se pode retornar.

	Objeto										
Domínio	Arquivo1	Arquivo2	Arquivo3	Arquivo4	Arquivo5	Arquivo6	Impressora1	Plotter2	Domínio1	Domínio2	Domínio3
1	Leitura	Leitura Escrita								Entra	
2			Leitura	Leitura Escrita Execução	Leitura Escrita		Escrita				
3						Leitura Escrita Execução	Escrita	Escrita			

Figura 9.5 Uma matriz de proteção com domínios como objetos.

Listas de controle de acesso (*access control list, ACL*): no exemplo (para simplificar, usuário = domínio), **cada arquivo possui sua ACL**; o ACL do arquivo F1 indica que qualquer processo do usuário A pode ler e escrever no arquivo F1, enquanto o usuário B pode apenas ler o arquivo.

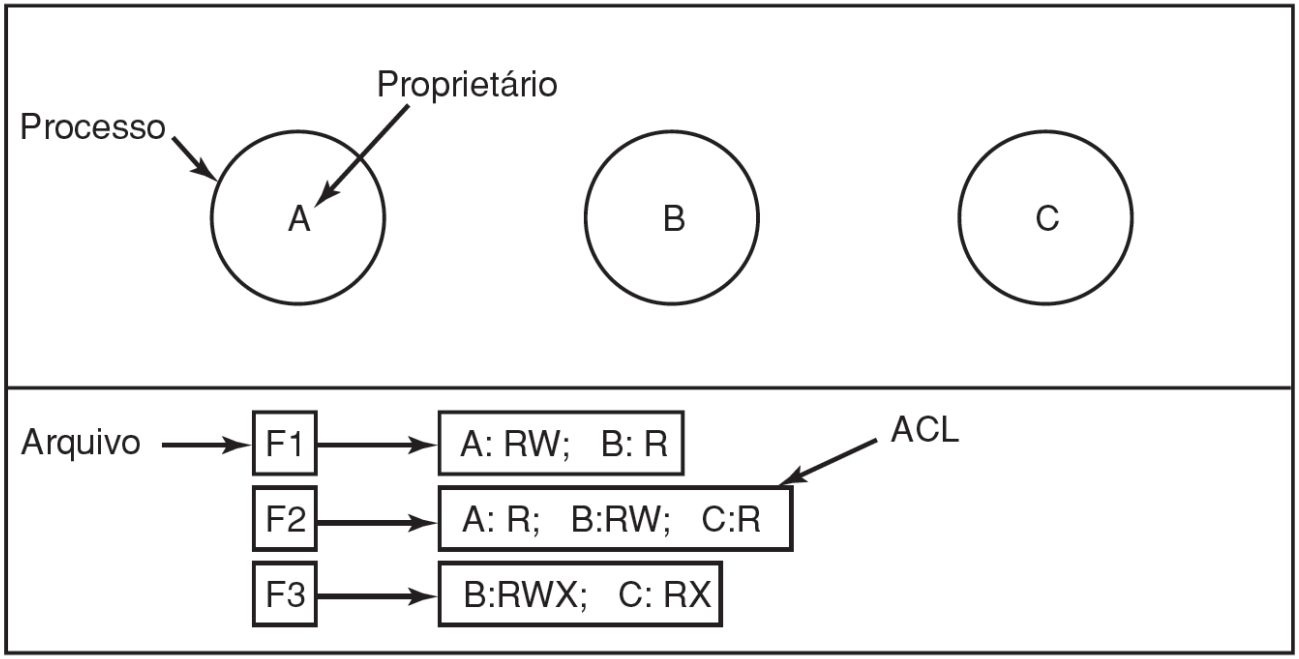


Figura 9.6 Uso de listas de controle de acesso no gerenciamento de acesso aos arquivos.

Controle de acesso pode ser realizado por grupo. Em alguns sistemas, o usuário deve escolher qual dos grupos associados (que ele esteja cadastrado) ele quer utilizar. Ex.: poderia ser que enquanto **ana** estivesse autenticada no grupo **crdpmb** ela não conseguisse alterar o arquivo de Senha, a não ser que se autenticasse naquele grupo.

Arquivo	Lista de controle de acesso
Senha	ana, sysadm: RW
Dados_pombos	bill, crdpmb: RW; ana, crdpmb: RW; ...

Tabela 9.2 Duas listas de controle de acesso.

Capacidades: a matriz de proteção (por domínio) pode ser implementada com listas; nesse caso, **as operações que podem ser executadas pelo usuário em cada domínio são denominadas capacidades**, e as listas denominadas de lista de capacidades.

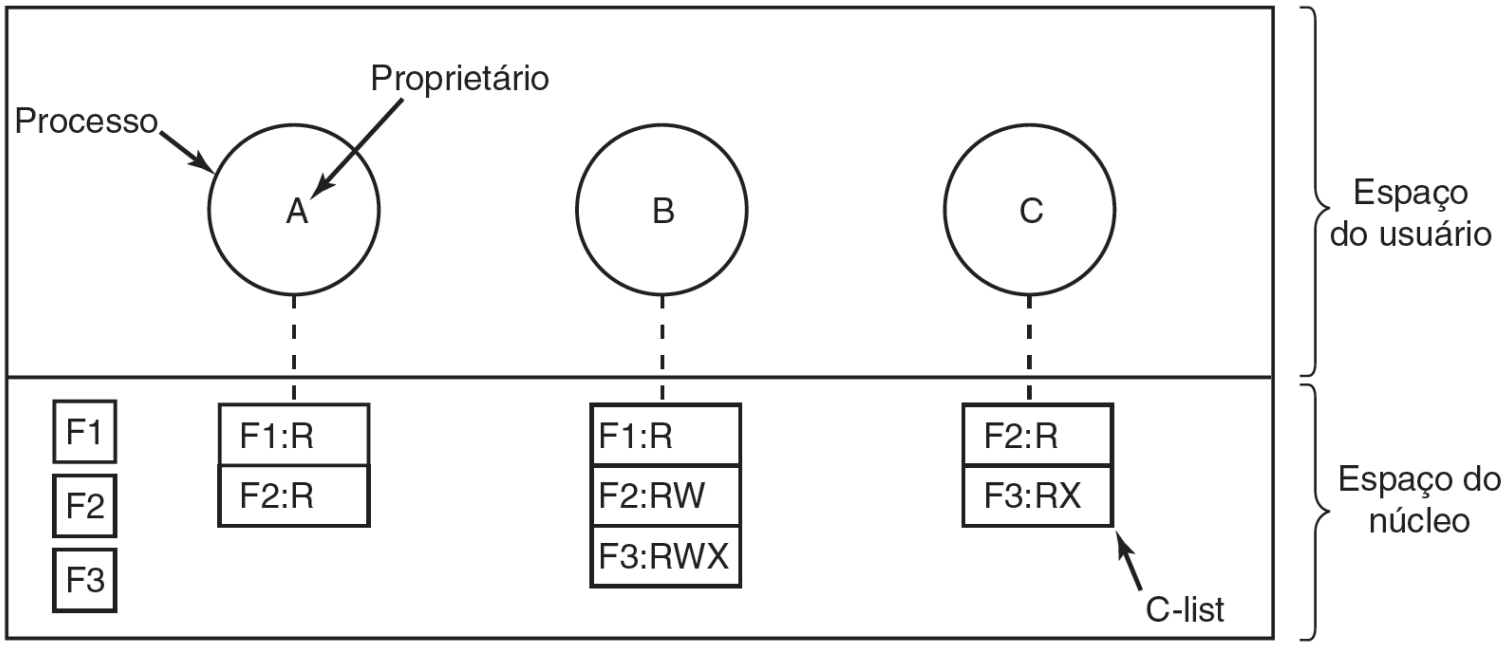


Figura 9.7 Quando as capacidades são utilizadas, cada processo possui uma lista de capacidades.

Sistemas confiáveis

- Considere reportagens sobre vírus, vermes, etc.

- Duas questões ingênuas (mas lógicas):
  - É possível construir um sistema computacional seguro?
  - Se sim, por que isso não é feito?

Base computacional confiável (*trusted computing base*)*: hardware* e *software* necessário para garantir todas as regras de segurança.

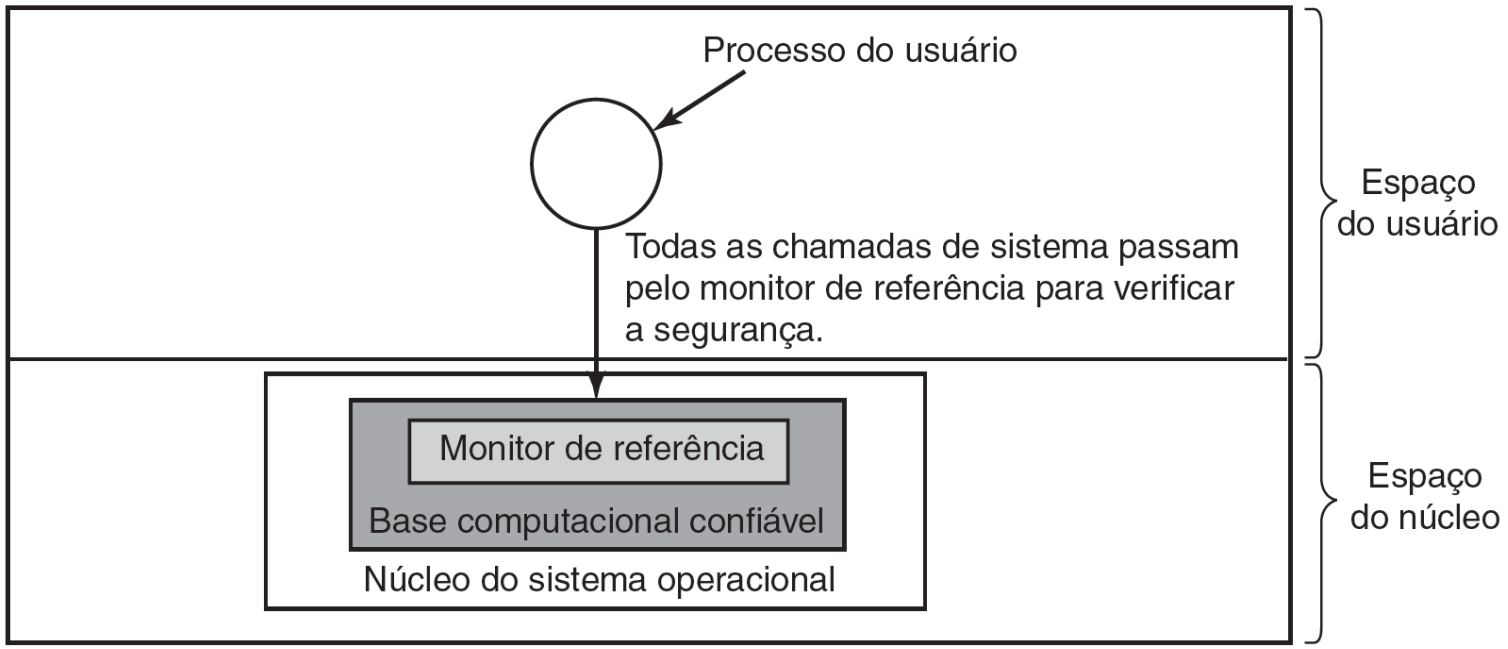


Figura 9.9 Um monitor de referência.

Modelos formais de sistemas seguros: **como garantir que o sistema nunca possa alcançar um estado não autorizado** (no exemplo abaixo, em (b), o usuário Roberto conseguiu alterar, indevidamente, a lista de capacidades, conferindo a ele acesso de leitura na Caixa postal 7).

(a)

	Compilador	Caixa postal 7	Secreto
Érico	Lê Executa		
Henrique	Lê Executa	Lê Escreve	
Roberto	Lê Executa		Lê Escreve

(b)

	Compilador	Caixa postal 7	Secreto
Érico	Lê Executa		
Henrique	Lê Executa	Lê Escreve	
Roberto	Lê Executa	Lê	Lê Escreve

Figura 9.10 (a) Um estado autorizado. (b) Um estado não autorizado.

Segurança multiníveis: O modelo Bell-La Padula

Regras sobre como a informação pode fluir, segundo o modelo Bell-La Padula:

- **A propriedade de segurança simples**: Um processo executado no nível de segurança k somente pode ler objetos no seu nível ou num nível inferior (e.g., um general pode ler os documentos de um tenente, mas este não pode ler os documentos do general).

O modelo Bell-La Padula

- **A propriedade *:** Um processo executado no nível de segurança k pode somente escrever em objetos no seu nível ou superior. Por exemplo, um tenente pode colocar uma mensagem na caixa postal de um general contando tudo o que ele sabe, já o general não pode colocar uma mensagem na caixa postal do tenente contando tudo o que ele sabe, pois o general pode ter visto documentos secretos que não podem ser revelados a um tenente.
  OBS.: o nome * foi utilizado temporariamente até obter outro; como não aconteceu, ficou * mesmo :)

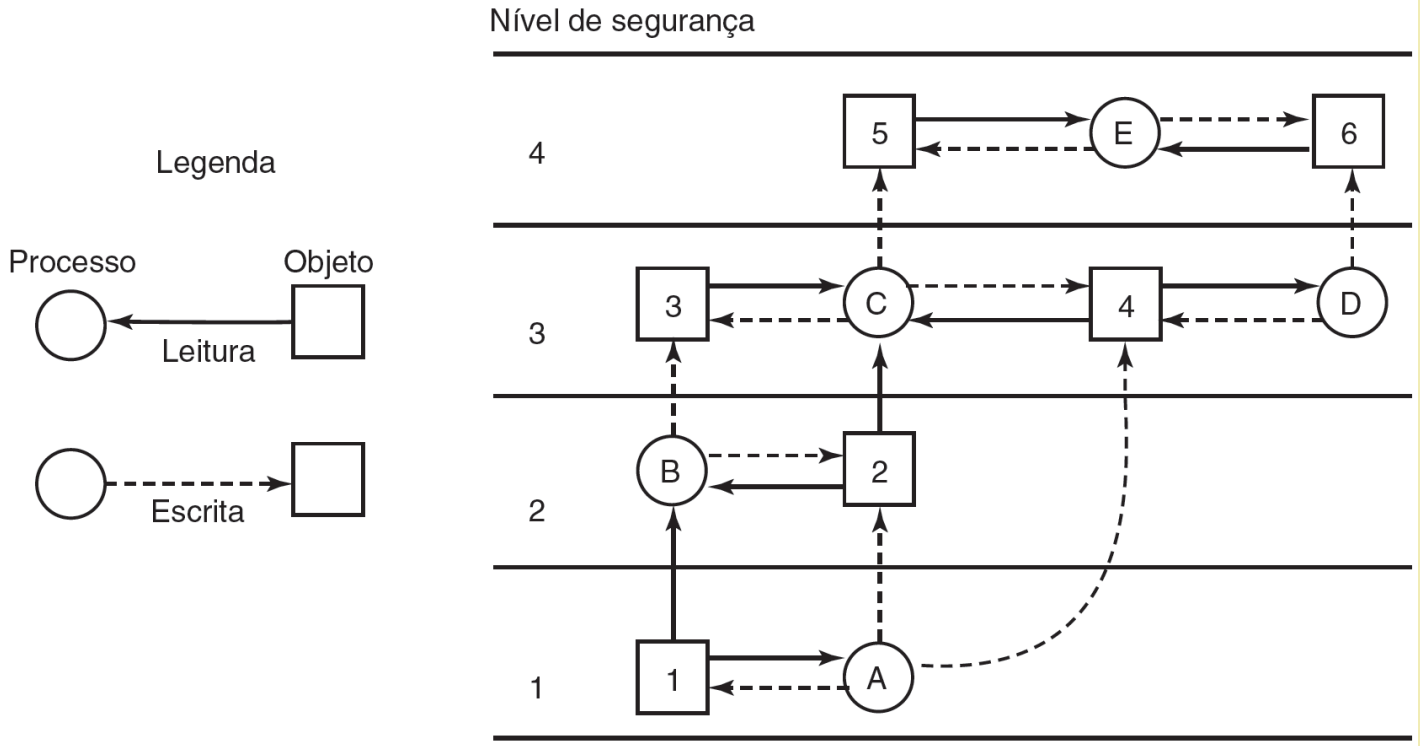


Figura 9.11 O modelo de segurança multiníveis Bell–La Padula.

O modelo Biba

O modelo Bell-La Padula foi projetado para manter segredos, sem garantir a integridade dos dados.

Regras para o modelo Biba (garante integridade, mas não garante segredo):

- **O príncipio da integridade simples**: Um processo executando no nível k somente pode escrever em objetos em seu nível ou inferior (não nos níveis superiores).
- **A propriedade de integridade ***: Um processo executando no nível de segurança k somente pode ler objetos no seu nível ou superior (não nos níveis inferiores).

O modelo Biba *versus* Bell-La Padula

Os dois modelos estão em conflito direto, tornando-se uma tarefa difícil implementá-los simultaneamente!

Canais ocultos: Caso os mecanismos de comunicação convencionais estejam bloqueados, o servidor "envia" ao colaborador (**comparsa**) um bit 1 quando fica ativo por um período definido e um bit 0 quando fica inativo por um outro determinado período. Aqui "enviar" significa que o colaborador fica constantemente monitorando o comportamento do servidor.

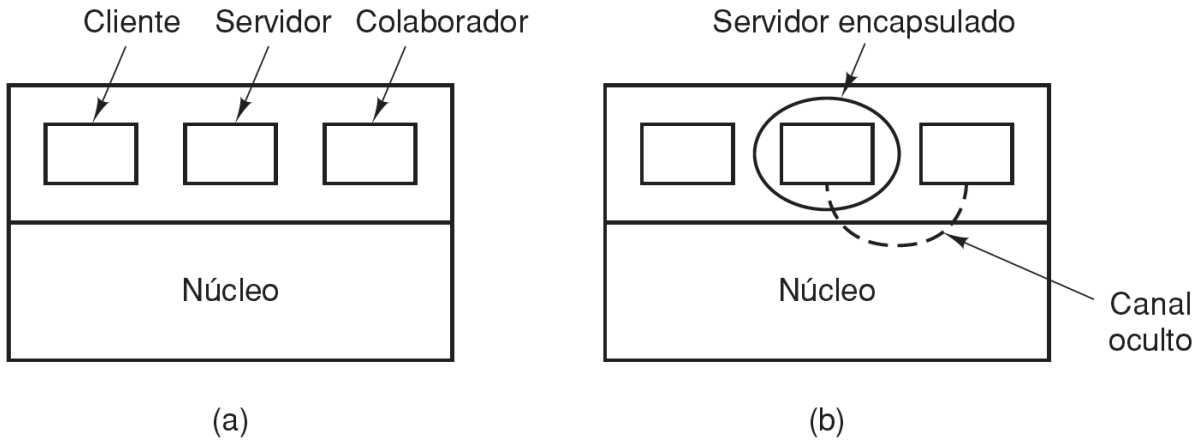


Figura 9.12 (a) Os processos cliente, servidor e colaborador. (b) O servidor encapsulado pode, ainda, passar informações ao colaborador por canais ocultos.

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Canais ocultos: nesse exemplo, o servidor comunica-se com o colaborador (**comparsa**) via bloqueio e desbloqueio de um determinado arquivo.

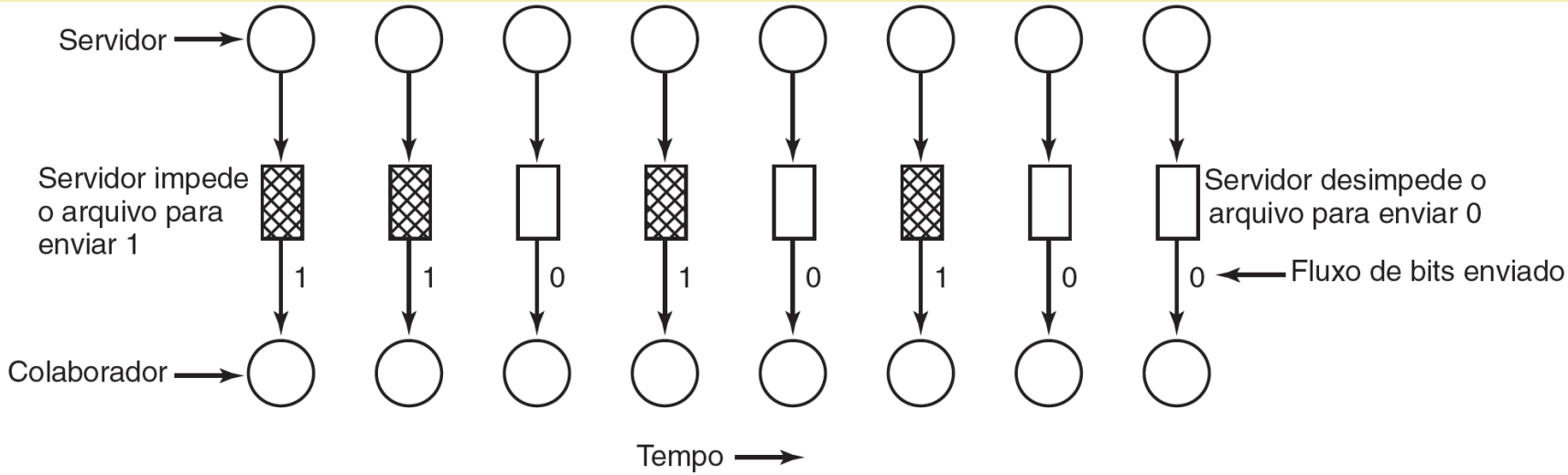
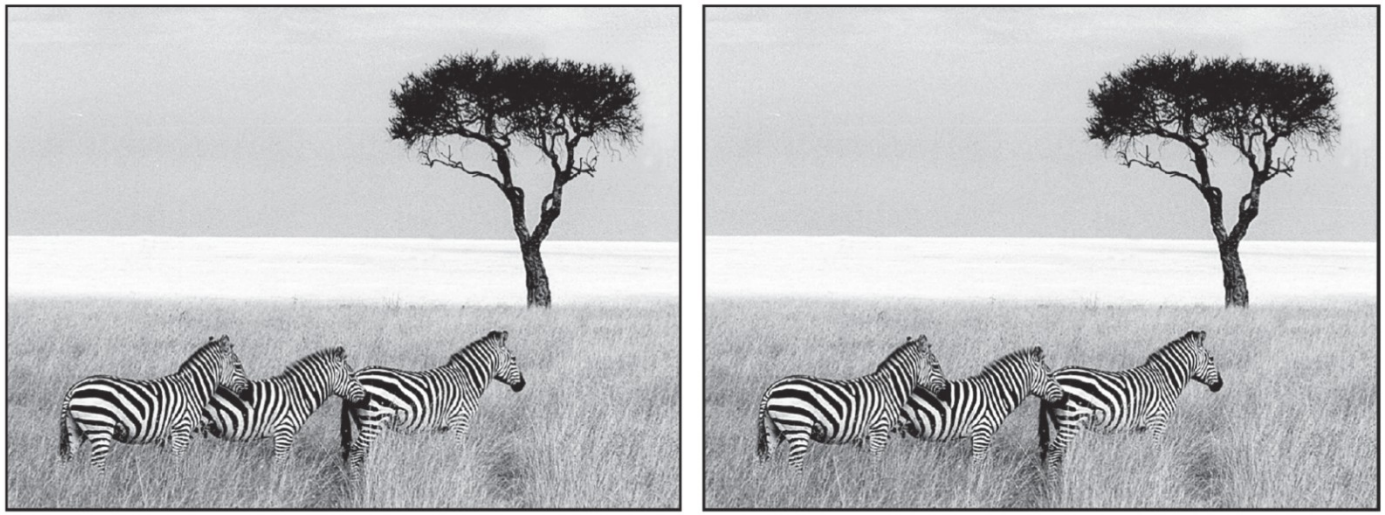


Figura 9.13 Um canal subliminar bloqueando um arquivo.

Canais ocultos utilizando **esteganografia** (do grego "escrita oculta"): aproveita-se ***bits*** que podem ser reutilizados sem nenhum (ou quase nenhum) impacto na qualidade do objeto (imagem, texto, etc).



(a) (b)

Figura 9.14 (a) Três zebras e uma árvore. (b) Três zebras, uma árvore e o texto completo de cinco peças de William Shakespeare.

Autenticação

Princípios gerais da autenticação de usuários:

- Algo que o usuário sabe;
- Algo que o usuário tem;
- Algo que o usuário é.

Autenticação usando senhas

(a)	(b)	(c)
LOGIN: mauro	LOGIN: carolina	LOGIN: carolina
SENHA: qualquer	NOME INVÁLIDO	SENHA: umdois
LOGIN COM SUCESSO	LOGIN:	LOGIN INVÁLIDO
		LOGIN:

Figura 9.15 (a) Uma autenticação de usuário bem-sucedida. (b) Autenticação de usuário rejeitada após o nome ser informado. (c) Autenticação de usuário rejeitada após o nome e a senha serem informados.

Como os *crackers* invadem: no exemplo abaixo, o *cracker* explorou uma conta com privilégios de *root* (conta *uucp*) que mantinha a senha *default* de instalação.

```
LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

Figura 9.16 Como um cracker invadiu o computador do Departamento de Energia dos Estados Unidos no LBL.

Autenticação desafio-resposta

As perguntas devem ser escolhidas de modo que o usuário não precise escrevê-las. Dentre um conjunto de perguntas/respostas (também criptografadas) cadastradas pelo usuário, o sistema escolhe uma aleatóriamente.

Exemplos:

- Quem é a irmã de João?
- Em que rua ficava sua escola primária?
- Que matéria a Dona Maria leciona?

Autenticação usando um objeto físico

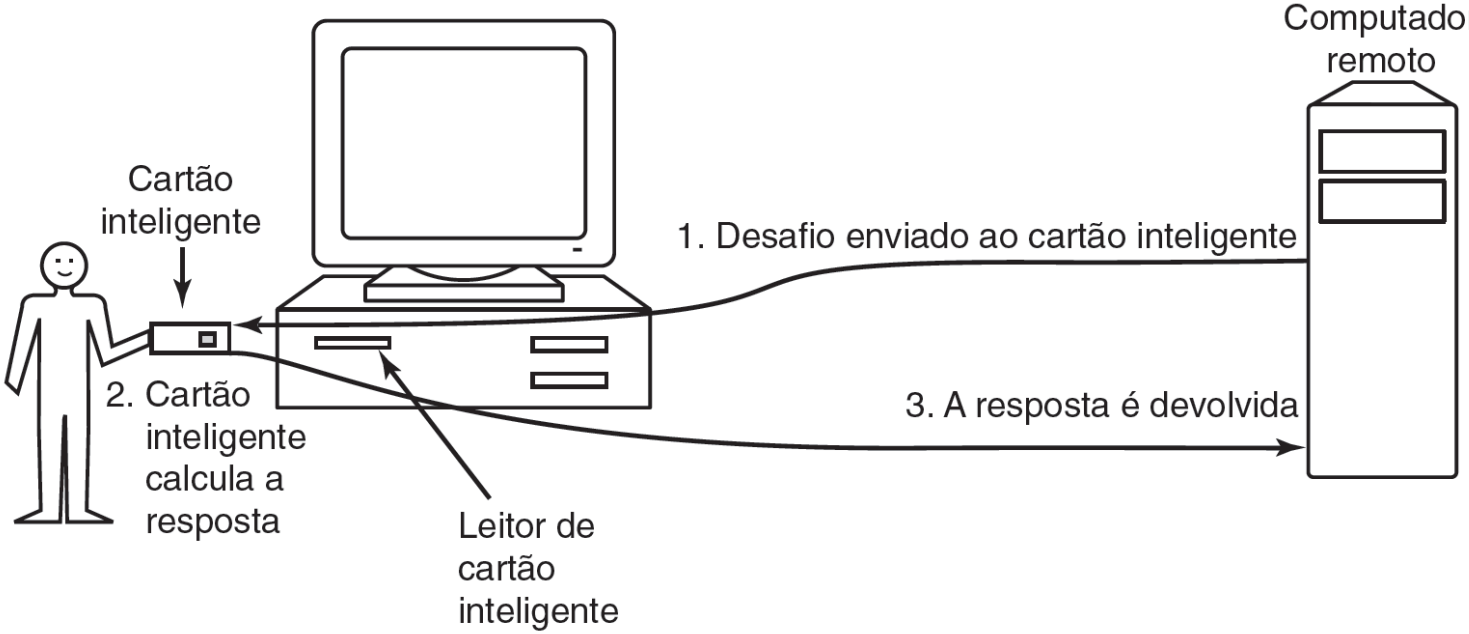


Figura 9.18 Uso de um cartão inteligente para autenticação.

Autenticação usando **biometria**: como exemplo, pode-se citar o reconhecimento pela íris e o comprimento dos dedos (nesse caso, um atacante poderia utilizar um molde em gesso da mão da pessoa).

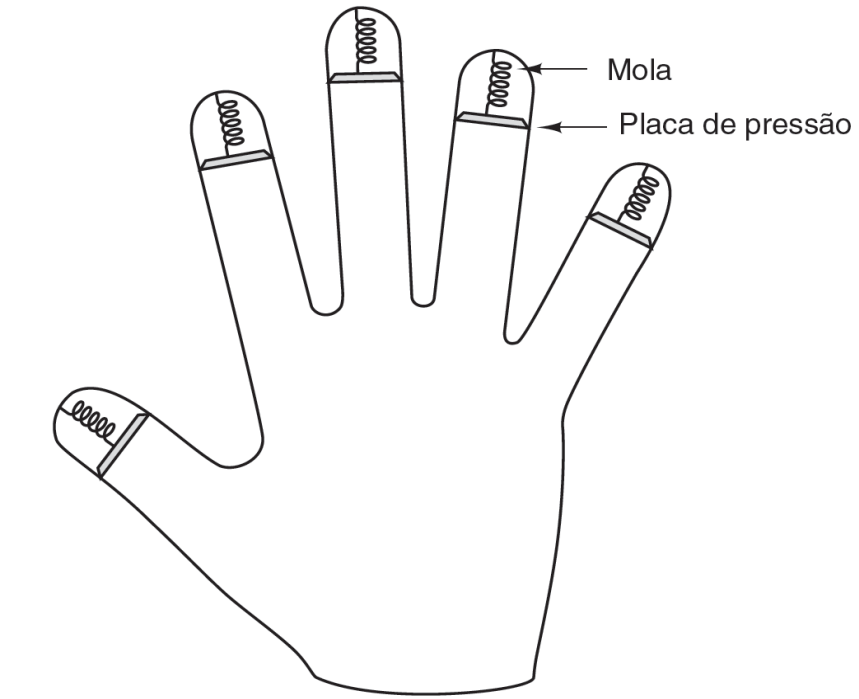


Figura 9.19 Um dispositivo para medir o tamanho dos dedos.

Ataques de dentro do sistema: Alçapão (código inserido para permitir acesso irrestrito a um usuário criado pelo atacante; realizar revisões de código periodicamente diminui as chances de um desenvolvedor inserir alçapões)

```
while (TRUE) {
        printf("login:    ");
        get_string(name);
        disable_echoing(    );
        printf("password:    ");
        get_string(password);
        enable_echoing(    );
        v = check_validity(name, password) ;
        if (v ) b reak;
}
execute_shell(name);
```

(a)

```
while (TRUE) {
        printf("login:    ");
        get_string(name)    ;
        disable_echoing(    );
        printf("password:    ");
        get_string(password)    ;
        enable_echoing(    );
        v = check_validity(name, password) ;
        if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name)    ;
```

(b)

Figura 9.20 (a) Código normal. (b) Código com um alçapão inserido.

Logro na autenticação do usuário: aplicativo de *login* falso captura pares de usuário e senha, enviando-os para o atacante via rede e, posteriormente, após encerrar o aplicativo falso de *login*, reinicia o aplicativo de *login* legítimo.

Figura 9.21 (a) Tela correta de autenticação. (b) Tela de autenticação adulterada.

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Explorando erros de códigos: Ataque por transbordamento de *buffer* (caso o programa atacado tenha SETUID de *root* no UNIX, o retorno pode ser para um fragmento de código inserido que execute um *shell* que estará a disposição do usuário (atacante) com direitos de root!!!

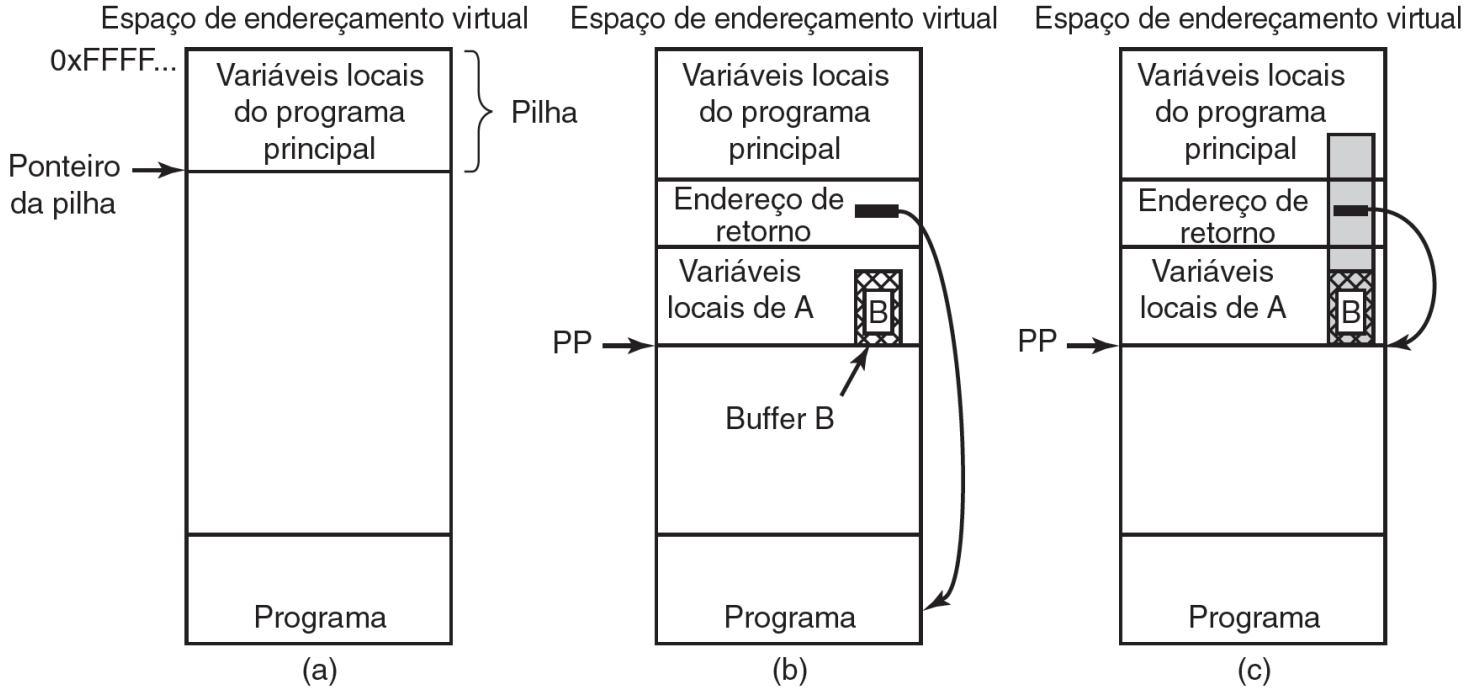
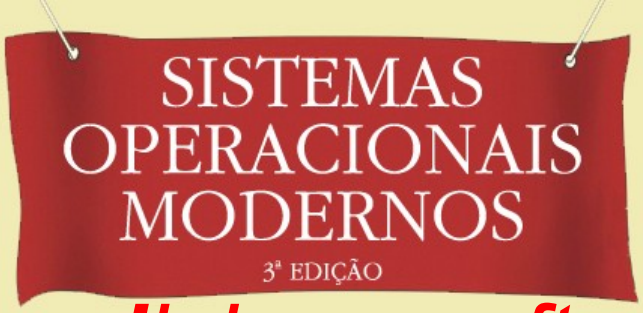


Figura 9.22 (a) Situação na qual o programa principal está sendo executado. (b) Depois da chamada de procedimento *A*. (c) Transbordamento do buffer mostrado em cinza.

Ataque por injeção de código: um programador preguiçoso, ao implementar um programa que duplica um arquivo, pode copiar arquivos via chamadas *system* (execução de linhas de comando via *shell*); uma entrada construída com segundas inteções (exemplo: 'abc' e 'xyz; rm -rf /') pode causar resultados desastrosos (nesse exemplo seria: **cp abc xyz; rm -rf /**).

```
int main(int argc, char *argv[])
{
    char src[100], dst [100], cmd[205]= "cp ";          /* declara três cadeias de caracteres */
    printf("Por favor, digite o nome do arquivo-fonte: ");   /* solicita o nome do arquivo fonte */
    gets(src);                                          /* recebe entrada via teclado */
    strcat(cmd, src);                                   /* concatena src depois de cp */
    strcat(cmd, " ");                                   /* acrescenta um espaço no final de cmd */
    printf("Por favor, informe o nome do arquivo destino: ");  /* solicita o nome do arquivo destino */
    gets(dst);                                          /* recebe entrada via teclado */
    strcat(cmd, dst);                                   /* completa a string de comandos */
    system (cmd);                                       /* executa o comando cp */
}
```

Figura 9.24 Código que pode levar a um ataque por injeção de código.



*Malware (**mal**icious soft**ware**)*: vírus, cavalos de tróia, vermes, etc.

Pode ser usado como uma forma de chantagem.

Exemplo: Encriptar arquivos no disco da vítima, então mostrar uma mensagem …

Saudações da General Encryption

Para comprar uma chave de decriptação para seu disco rigído, por favor envie
$100 em notas de baixo valor
não marcadas para caixa postal 2154, Cidade do Panamá, Panamá.
Obrigado. Foi bom fazer negócios com você.

Tipos de vírus

- Vírus companheiro (e.g., no DOS: prog.com e prog. exe)
- Vírus de programas executáveis
- Vírus parasitas
- Vírus residentes de memória
- Vírus de setor de inicialização
- Vírus de *drivers* de dispositivo
- Vírus de macro
- Vírus de código fonte

Vírus de sobreposição: vírus simplesmente sobrepõe o arquivo alvo (o código abaixo procura, recursivamente, por aquivos executáveis e realiza a sobreposição).

```
#include <sys/types.h>                  /* cabeçalhos-padrão POSIX */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                       /* para a chamada lstat veja se o arquivo é uma ligação simb. */

search(char *dir_name)
{                                       /* busca recursivamente por executáveis */
    DIR *dirp;                          /* ponteiro para um fluxo aberto de diretório */
    struct dirent *dp;                  /* ponteiro para uma entrada de diretório */

    dirp = opendir(dir_name);           /* abrir este diretório */
    if (dirp == NULL) return;           /* se dir não puder ser aberto, esqueça-o */
    while (TRUE)  {
        dp = readdir(dirp);             /* leia a próxima entrada de diretório */
        if (dp == NULL) {               /* NULL significa que terminamos */
        chdir ("..");                   /* volte ao diretório-pai */
        break;                          /* sai do laço */
        }
        if (dp->d_name[0] == '.') continue;     /* salte os diretórios . e .. */
        lstat(dp->d_name, &sbuf);               /* a entrada é uma ligação simbólica? */
        if (S_ISLNK(sbuf.st_mode)) continue;    /* salte as ligações simbólicas */
        if(chdir(dp->d_name) == 0) {            /* se chdir tiver sucesso, deve ser um diretório */
            search(".");                        /* sim, entre e busque-o */
        } else {                                /* não (arquivo), infecte-o */
            if (access(dp->d_name, X_OK) == 0)  /* se for executável, infecte-o */
                infect(dp->d_name);
        }
        closedir(dirp);                 /* diretório processado; feche e retorne */
    }
```

Figura 9.25 Um procedimento recursivo que encontra arquivos executáveis em um sistema UNIX.

Vírus parasitas: vírus de sobreposição são facilmente detectáveis; virus parasitas ficam acoplados ao arquivo executável original, dificultando sua detecção.

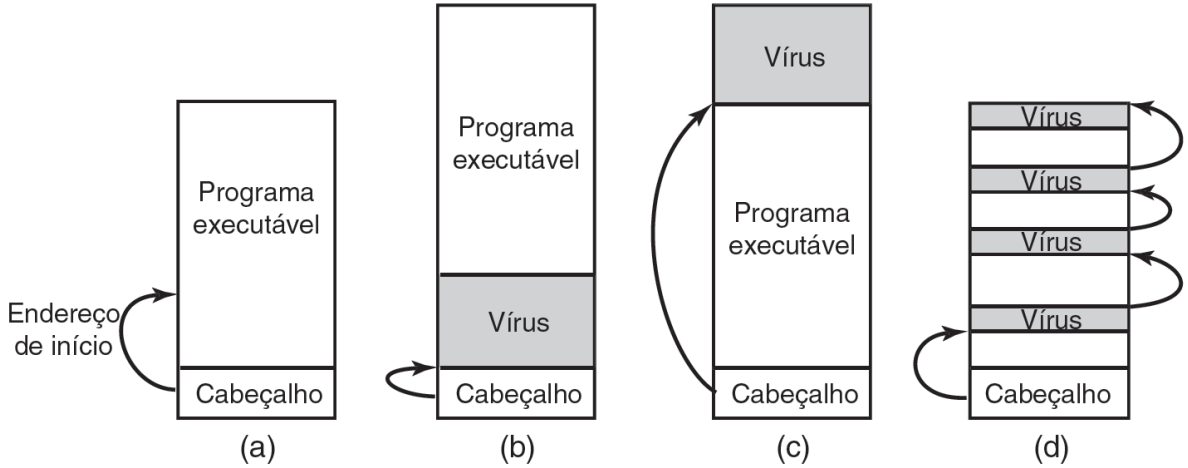


Figura 9.26 (a) Um programa executável. (b) Com um vírus na frente. (c) Com um vírus no final. (d) Com um vírus espalhado pelos espaços livres ao longo do programa.

Vírus do setor de inicialização (MBR): permite que o vírus se instale na memória (i.e., fica residente em memória), pois pode controlar todo o processo de inicialização (quando o sistema está em modo *kernel* e com a MMU desligada), desviando as interrupções e chamadas de sistema para si mesmo. Mesmo que o vetor de interrupções seja reescrito várias vezes pelo SO ele pode ir, gradativamente, monitorando e restaurando o desvio para si mesmo.

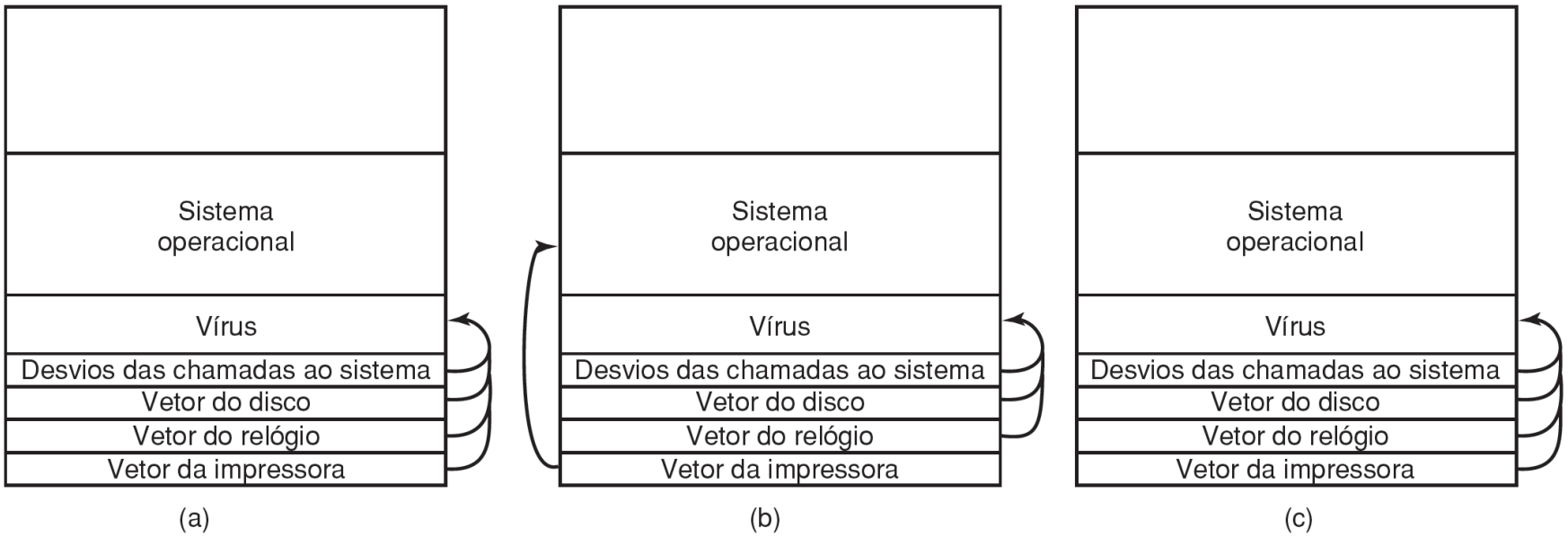


Figura 9.27 (a) Depois de o vírus capturar todos os vetores de interrupções. (b) Depois de o sistema operacional ter retomado o vetor de interrupções da impressora. (c) Após o vírus perceber a perda do vetor de interrupções da impressora e recuperá-la.

Spyware

Descrição:

- É carregado sorrateiramente no PC sem conhecimento do proprietário.
- Funciona nos bastidores fazendo coisas que o proprietário não tem conhecimento.
- ***Sistemas operacionais (e.g., Windows) fazem atualizações de segurança sem conhecimento do usuário, assim como os próprios programas antivírus. Isso também pode ser considerado spyware?***

Spyware

Características:

- Se esconde, a vítima não o encontra facilmente.
- Coleta dados sobre o usuário.
- Comunica os dados coletados a seu mestre.
- Tenta sobreviver a determinadas tentativas de remoção.

Como o *Spyware* se espalha

Maneiras possíveis:

- Do mesmo modo que o *malware*, por um cavalo de Tróia.
- Contágio por contato, visitando um *web site* infectado:
  - A *web page* tenta executar um arquivo *.exe*;
  - Um usuário não suspeito instala uma barra de ferramentas infectada;
  - Um controle *activeX* malicioso é instalado.

Ações executadas pelo Spyware

- Muda a página inicial do navegador.
- Modifica a lista de favoritos do navegador.
- Adiciona novas barras de ferramentas ao navegador.
- Altera o tocador de mídia padrão do usuário.
- Muda a ferramenta de busca padrão do usuário.
- Adiciona novos ícones à área de trabalho do Windows.
- Substitui faixas de anúncios nas páginas da web por outras escolhidas pelo Spyware.
- Coloca anúncios nas caixas de diálogo padrão do Windows.
- Gera uma cadeia contínua de anúncios pop-up.

Tipos de *rootkits* (programa ou conjunto de programas e arquivos que tenta esconder sua existência). Por exemplo, interceptando uma chamada de sistema de arquivo, deixando passar apenas arquivos não infectados, evitando que um antivírus vasculhe um arquivo já infectado. Originou inicialmente como *kits* na plataforma Linux/Unix. Caso executado como root, o software tem acesso completo ao sistema (daí o nome *rootkit* que, na atualidade, atinge um maior número de plataformas).

- Rootkits de *firmware*
- Rootkits de *hipervisor*
- Rootkits de núcleo
- Rootkits de biblioteca
- Rootkits de aplicação

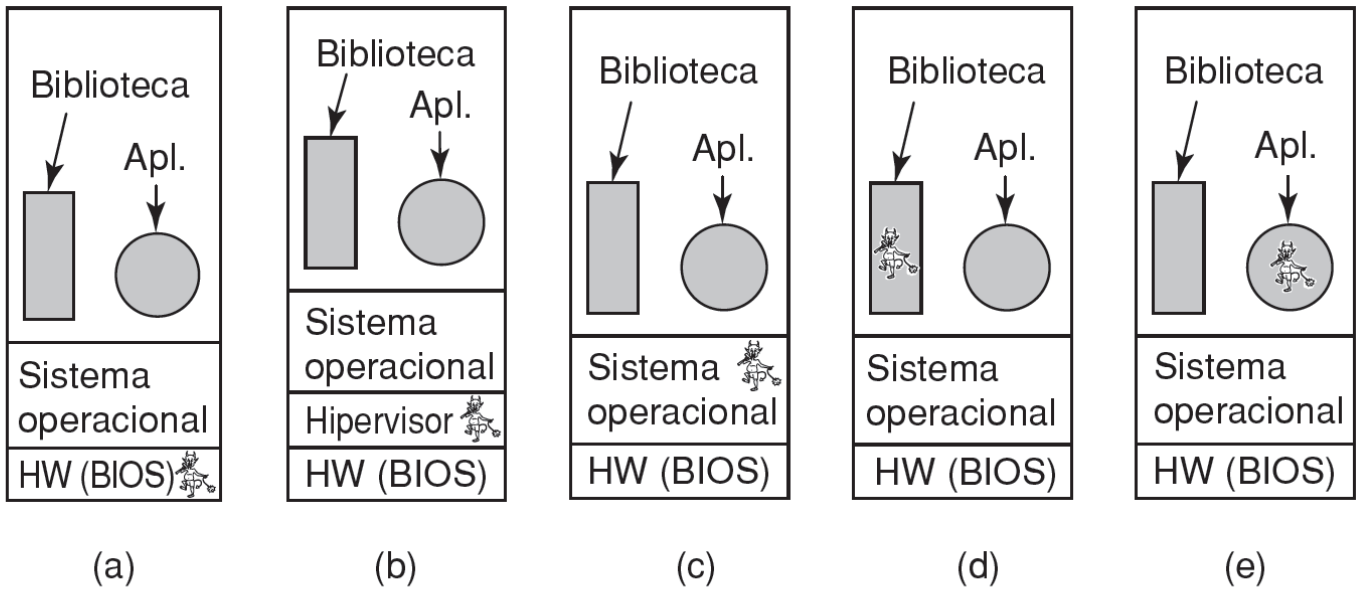


Figura 9.28 Cinco locais onde um rootkit pode se esconder.

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Firewalls

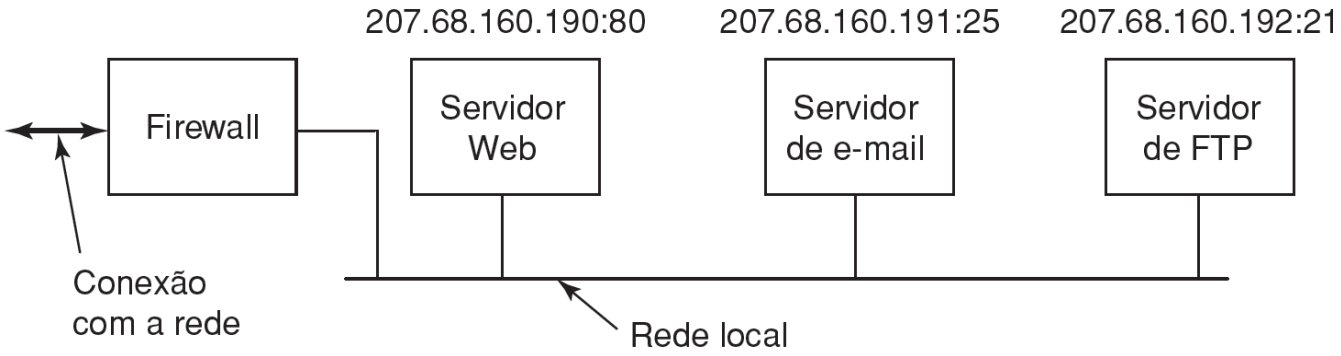


Figura 9.29 Uma visão simplificada de um firewall de hardware protegendo uma rede local com três computadores.

Técnicas de antivírus e antiantivírus

- Verificadores de vírus
- Verificadores de integridade
- Verificadores de comportamento
- Proteção contra vírus

SISTEMAS OPERACIONAIS MODERNOS
3ª EDIÇÃO

Verificadores de vírus

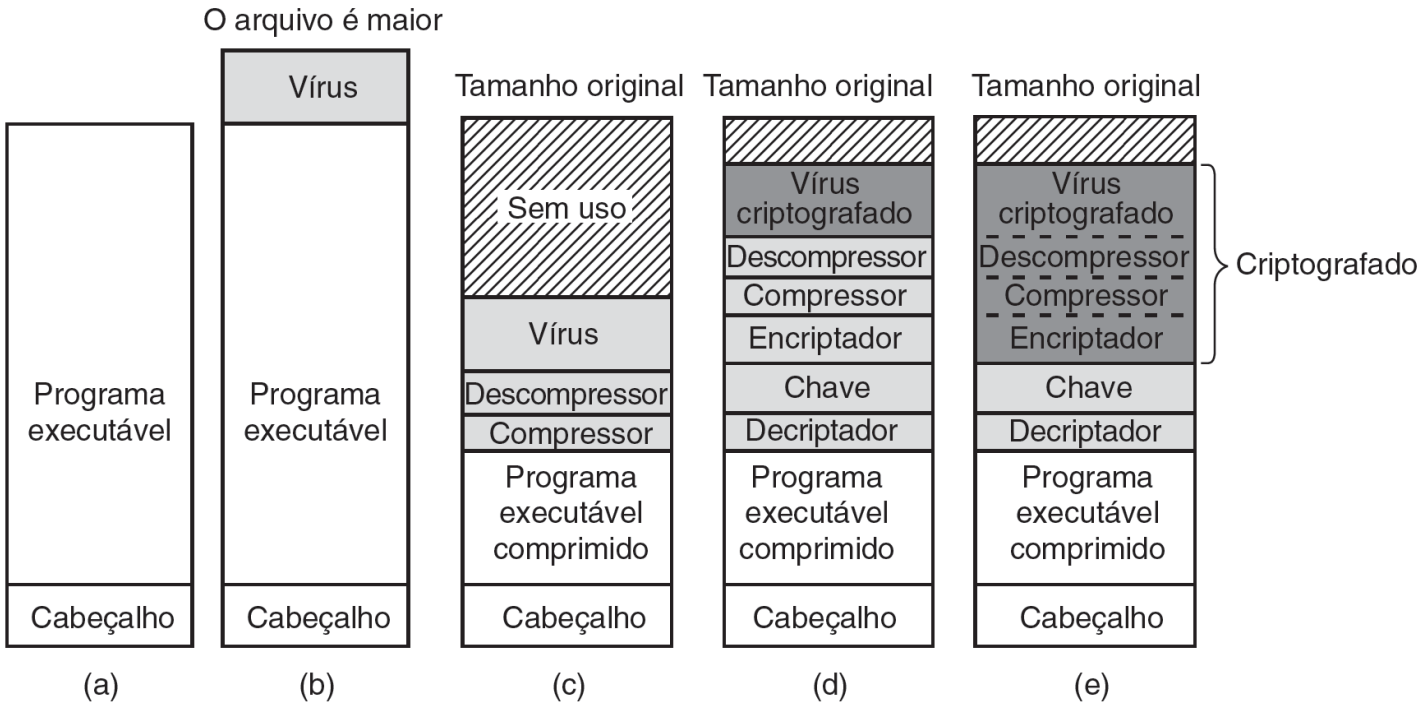
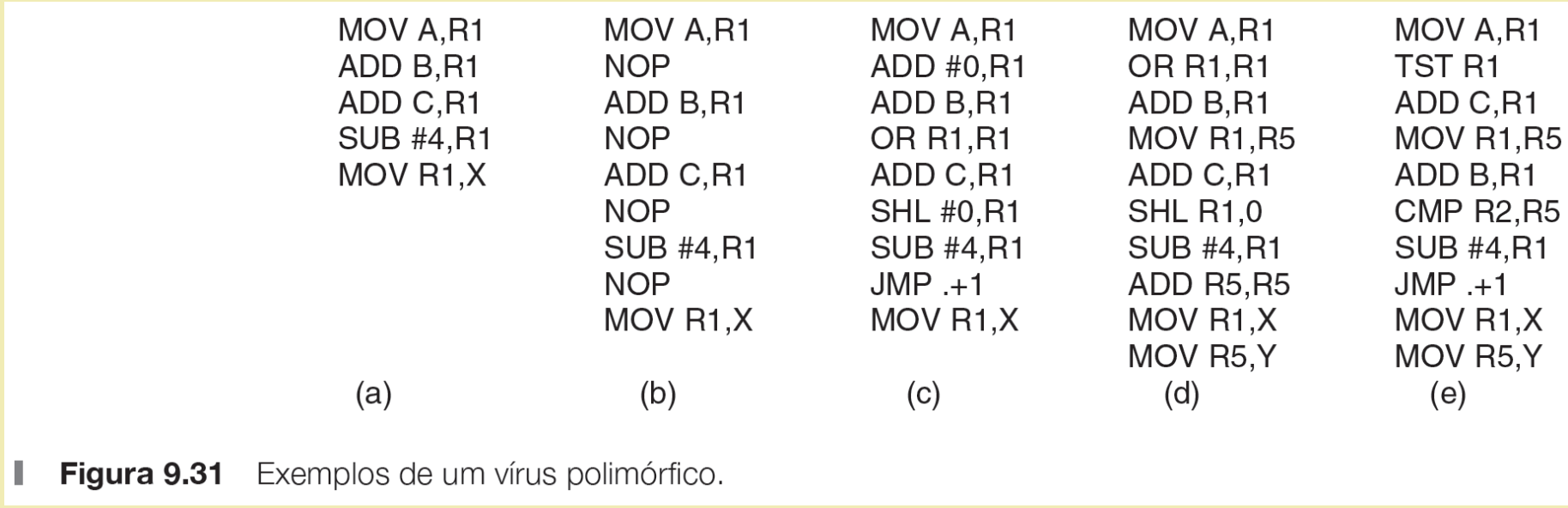


Figura 9.30 (a) Um programa. (b) Um programa infectado. (c) Um programa compactado infectado. (d) Um vírus criptografado. (e) Um vírus compactado com código de cifragem compactado.

Vírus polimórfico é um vírus que sofre mutação a cada cópia. Os códigos apresentados de (a) a (e) são funcionalmente equivalentes. Muito difíceis de serem detectados!



```
MOV A,R1          MOV A,R1          MOV A,R1          MOV A,R1          MOV A,R1
ADD B,R1          NOP               ADD #0,R1         OR R1,R1          TST R1
ADD C,R1          ADD B,R1          ADD B,R1          ADD B,R1          ADD C,R1
SUB #4,R1         NOP               OR R1,R1          MOV R1,R5         MOV R1,R5
MOV R1,X          ADD C,R1          ADD C,R1          ADD C,R1          ADD B,R1
                  NOP               SHL #0,R1         SHL R1,0          CMP R2,R5
                  SUB #4,R1         SUB #4,R1         SUB #4,R1         SUB #4,R1
                  NOP               JMP .+1           ADD R5,R5         JMP .+1
                  MOV R1,X          MOV R1,X          MOV R1,X          MOV R1,X
                                                      MOV R5,Y          MOV R5,Y
      (a)               (b)               (c)               (d)               (e)
```

Figura 9.31 Exemplos de um vírus polimórfico.

Assinatura de código

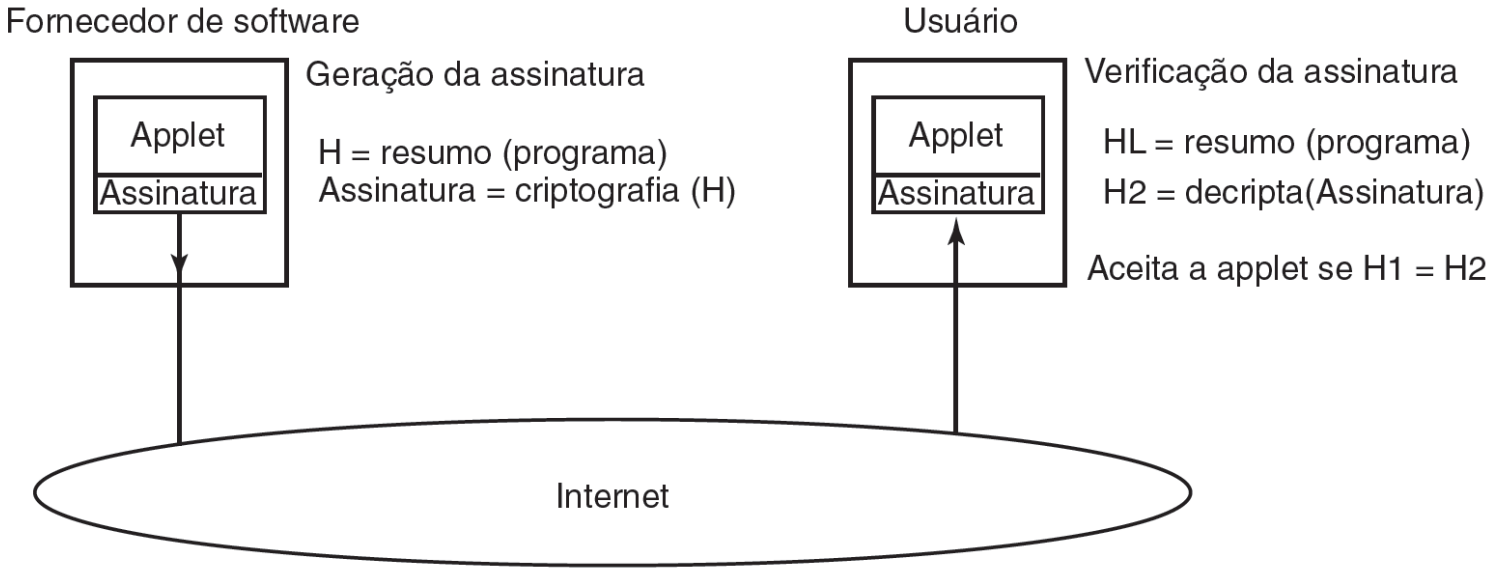


Figura 9.32 Como funciona a assinatura de código.

Encarceramento: semelhante ao processo de depuração de programas; o *kernel* desvia para o encarcerador toda a chamada de sistema antes de executá-la, assim pode-se verificar se a chamada está dentro do esperado (caso contrário, elimina o “prisioneiro!”).

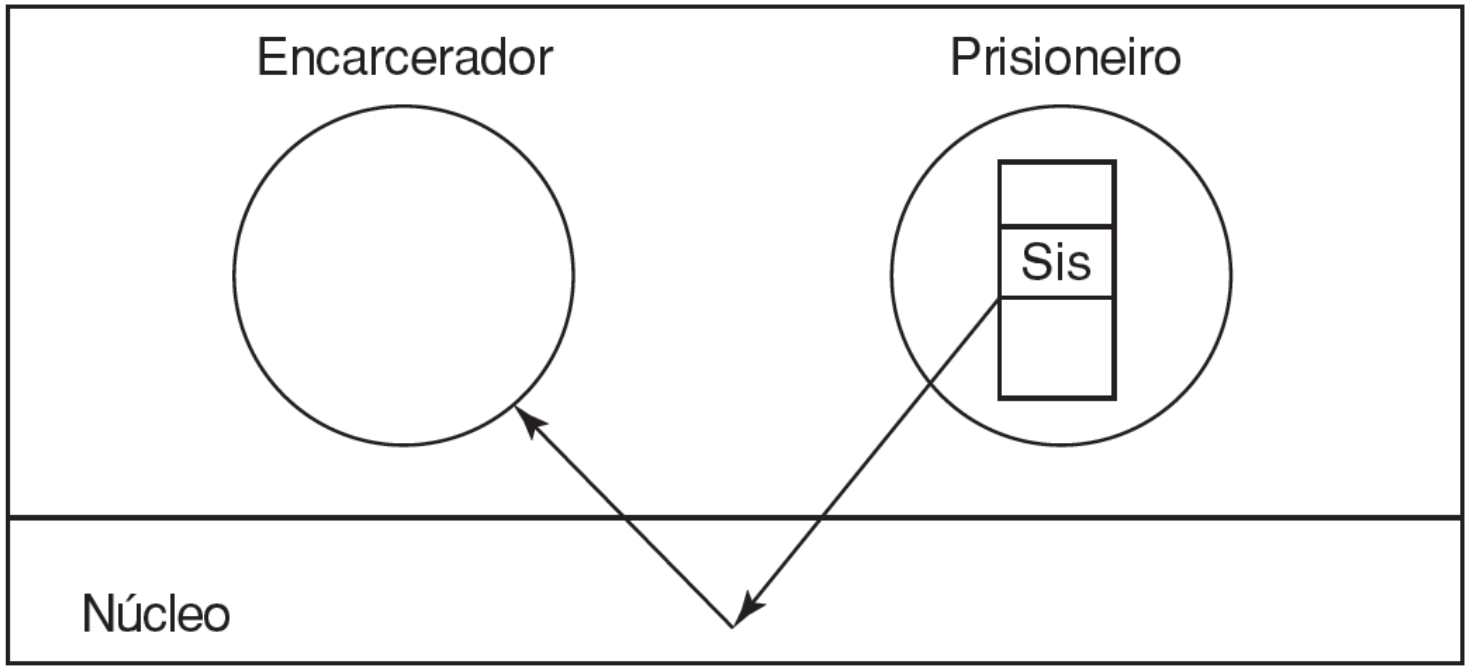


Figura 9.33 A operação de encarceramento.

Encapsulamento de código móvel (applet) com caixa de areia (*sandbox*) sem máquina virtual: garante que uma applet não seja capaz de saltar para um código externo à sua caixa de areia ou que não referencie dados fora de sua caixa de areia de dados. Antes de realizar um desvio (b), *jump* (JMP), verifica se endereço está dentro dos limites (SHR=*shift right*; no exemplo, desloca 24 bits à direita e testa os bits mais significativos (valor de comparação préviamente armazenado em S2; ou seja, S2 contém o prefixo do endereço que deve ser igual para todos os endereços dentro da mesma caixa de areia), os quais devem ser sempre iguais para endereços dentro do mesmo bloco (sandbox); caso não seja (i.e., fora da sandbox, TRAPNE desvia para o monitor que então encerra a applet).

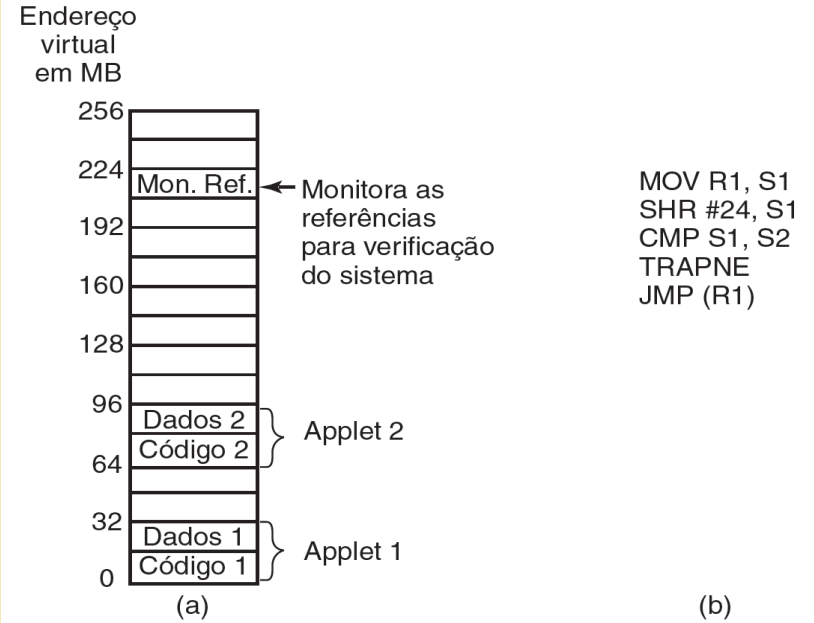


Figura 9.35 (a) Memória dividida em caixas de areia de 16 MB. (b) Uma maneira de verificar a validade de uma instrução.

Encapsulamento de código móvel com máquina virtual (e.g., JVM) executado via interpretação.

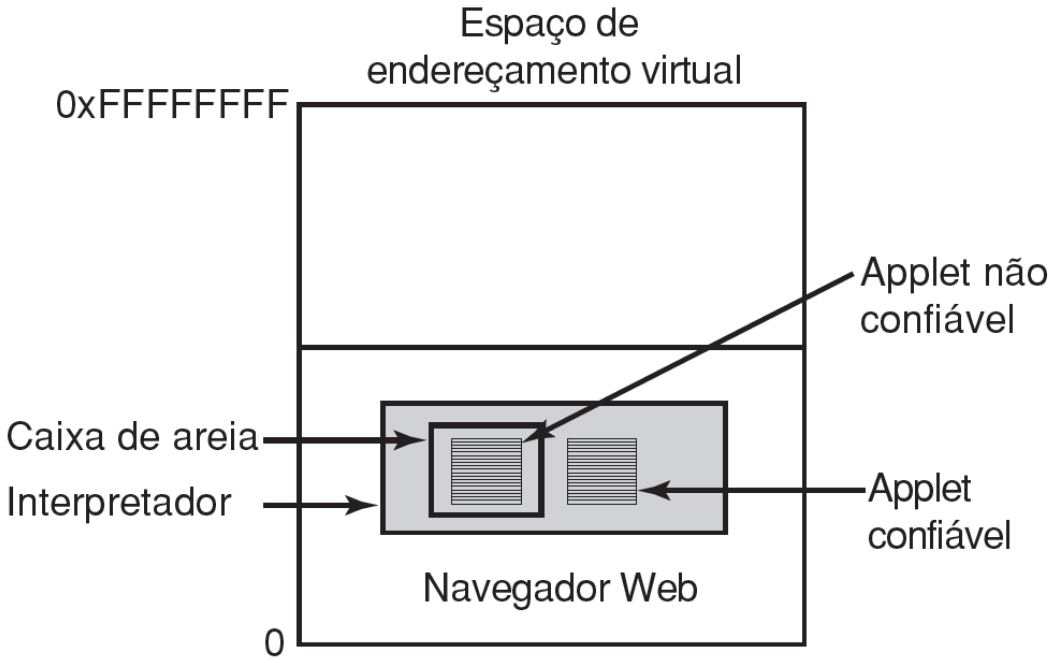


Figura 9.36 As applets podem ser interpretadas por um navegador da Web.

Segurança em Java

O verificador de *byte code* JVM checa se a *applet* obedece certas regras:

- A *applet* tenta forjar ponteiros?
- Viola as restrições de acesso em membros de classes privadas?
- Tenta usar uma variável de um tipo como se fosse de outro?
- Gera um transbordamento na pilha ? E um esvaziamento?
- Converte ilegalmente variáveis de um tipo para outro?