

# 9 IT-Sicherheit

- 1 Einführung
- 2 Prozesse und Threads
- 3 Speicherverwaltung
- 4 Dateisysteme
- 5 Eingabe und Ausgabe
- 6 Deadlocks
- 7 Virtualisierung und die Cloud
- 8 Multiprozessorsysteme
- 9 *IT-Sicherheit***
- 10 Fallstudie 1: Linux
- 11 Fallstudie 2: Windows
- 12 Entwurf von Betriebssystemen

# 9 IT-Sicherheit

- 9.1 Die Sicherheitsumgebung
- 9.2 Betriebssystemsicherheit
- 9.3 Zugriff auf Ressourcen steuern
- 9.4 Formale Modelle von sicheren Systemen
- 9.5 Grundlagen der Kryptografie
- 9.6 Authentifizierung
- 9.7 Ausnutzen von Sicherheitslücken
- 9.8 Insider-Angriffe
- 9.9 Malware
- 9.10 Abwehrmechanismen

# 9.1 Die Sicherheitsumgebung

9.1.1 Bedrohungen

9.1.2 Angreifer

# Schutzziele und Bedrohungen

<b>Schutzziel</b>	<b>Bedrohung</b>
Vertraulichkeit	Enthüllung der Daten
Integrität	Manipulation der Daten
Verfügbarkeit	Dienstverweigerung (Denial of Service)

**Abbildung 9.1:** Schutzziele und Bedrohungen.

# Angreifer

- **Passive Angreifer:**
  - Wollen Daten nur lesen, für die sie keine Berechtigung haben
- **Aktive Angreifer:**
  - Möchten unautorisierte Änderungen an Daten vornehmen.

## Kategorien:

- Herumschnüffeln nichttechnische Benutzer
- Insider
- Gezielte Versuche einen wirtschaftlichen Nutzen zu erzielen
- Spionage (Militär, Wirtschaft usw.)

## 9.2 Betriebssystemsicherheit

9.2.1 Können wir sichere Systeme bauen?

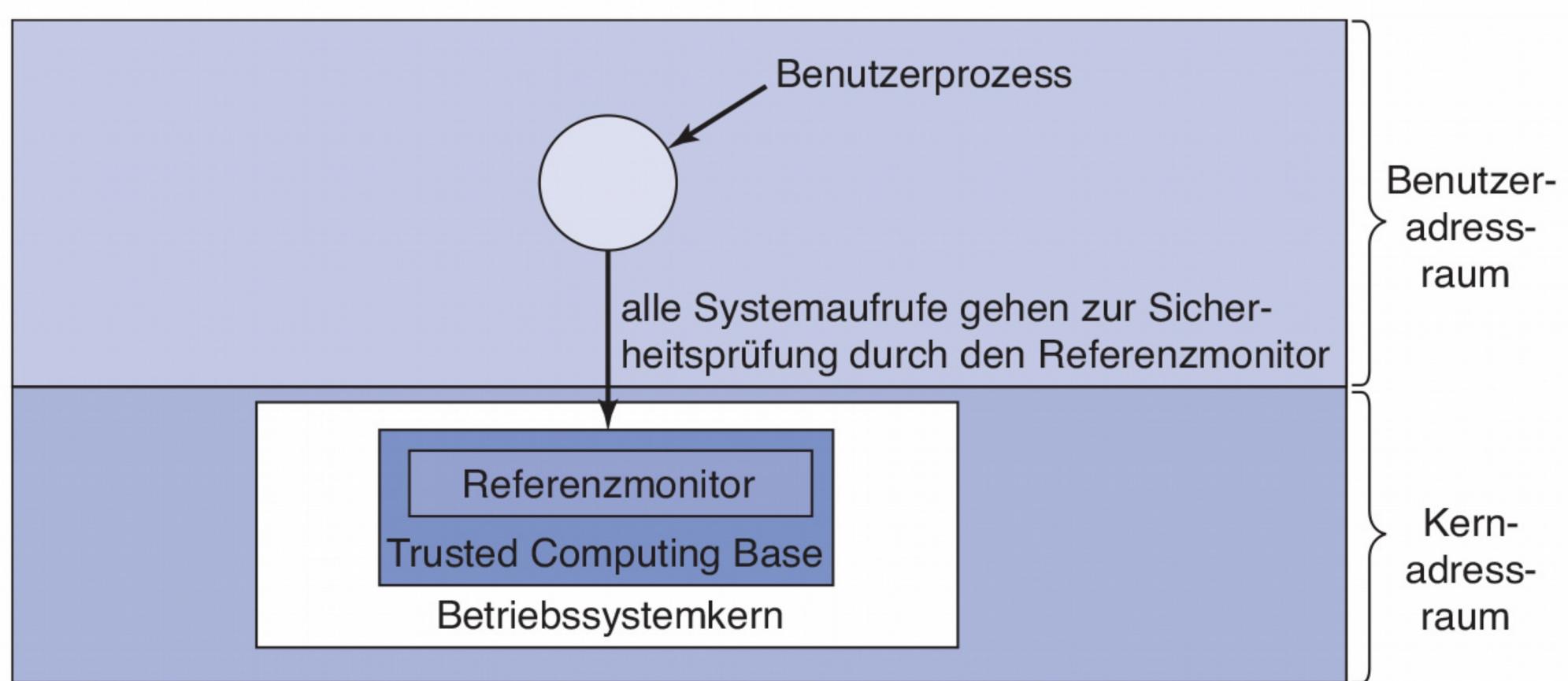
9.2.2 Trusted Computing Base

# Können wir sichere Systeme bauen?

**Zwei Fragen zur Sicherheit:**

1. Ist es möglich, ein sicheres Computersystem zu bauen?
2. Wenn ja, warum ist es nicht getan

# Trusted Computing Base



**Abbildung 9.2:** Referenzmonitor.

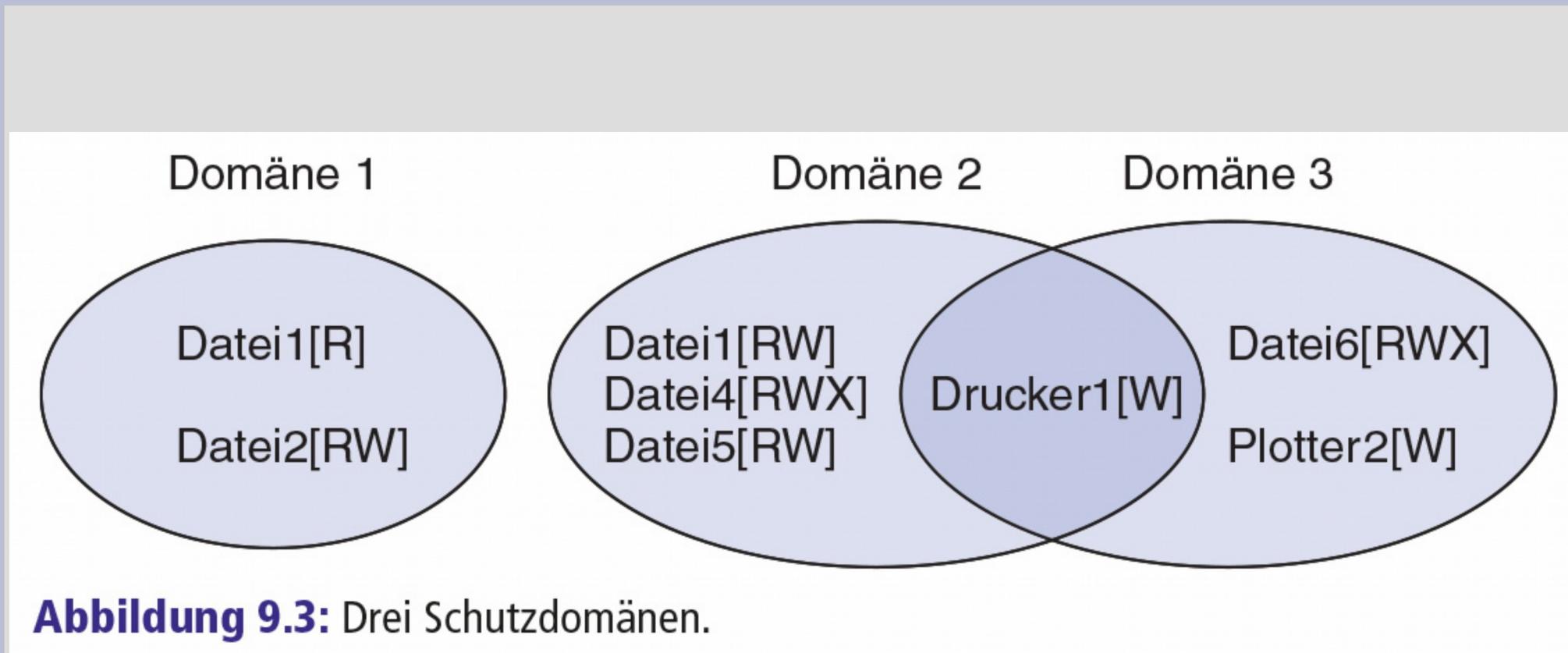
# **9.3 Zugriff auf Ressourcen steuern**

**9.3.1 Schutzdomänen**

**9.3.2 Zugriffskontrolllisten**

**9.3.3 Capabilities**

# Schutzdomänen (1)



# Schutzdomänen (2)

		Objekt							
		Datei1	Datei2	Datei3	Datei4	Datei5	Datei6	Drucker1	Plotter2
Domäne	1	Read	Read Write						
	2			Read	Read Write Execute	Read Write		Write	
3							Read Write Execute	Write	Write

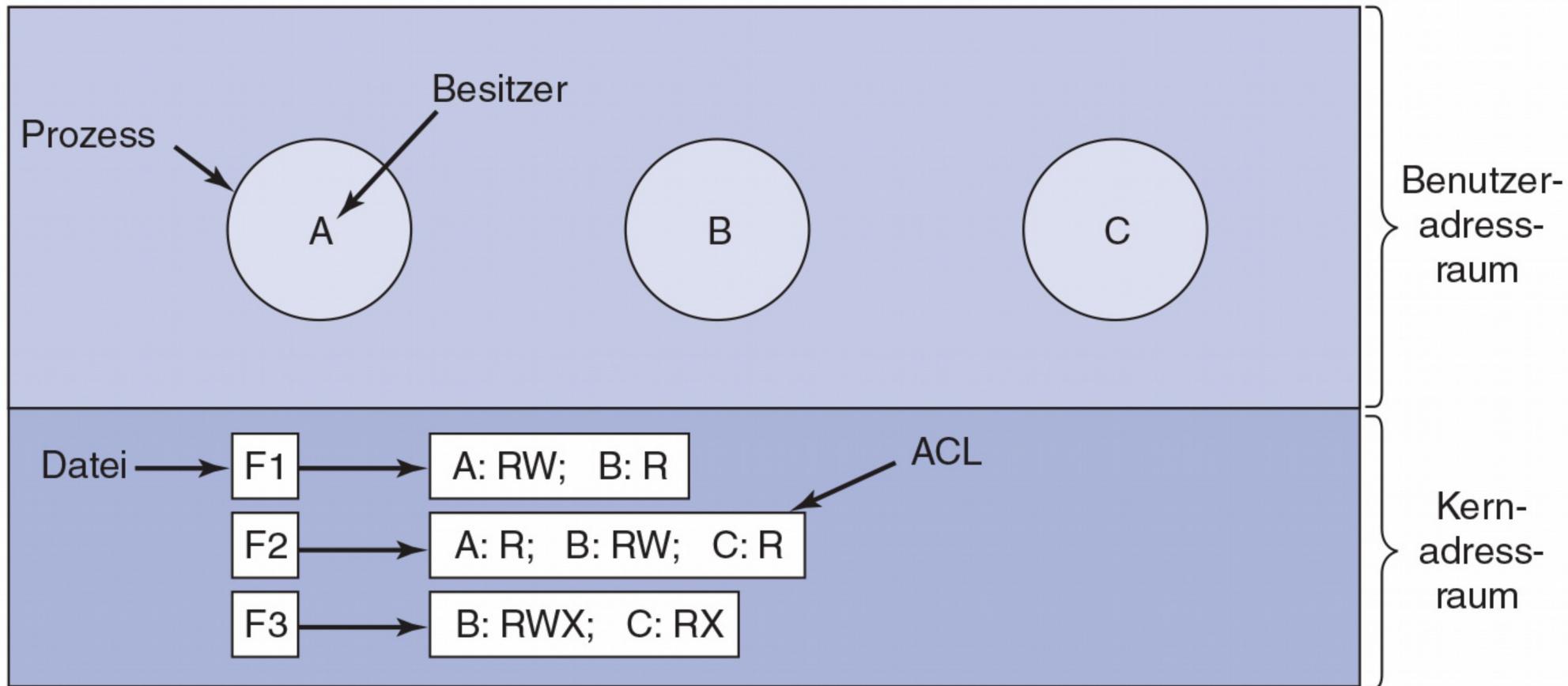
**Abbildung 9.4:** Schutzmatrix.

# Schutzdomänen (3)

		Objekt										
		Datei1	Datei2	Datei3	Datei4	Datei5	Datei6	Drucker1	Plotter2	Domäne1	Domäne2	Domäne3
Domäne	1	Read	Read Write								Enter	
	2			Read	Read Write Execute	Read Write		Write				
	3						Read Write Execute	Write	Write			

**Abbildung 9.5:** Schutzmatrixt mit Domänen als Objekten.

# Zugriffskontrolllisten (1)



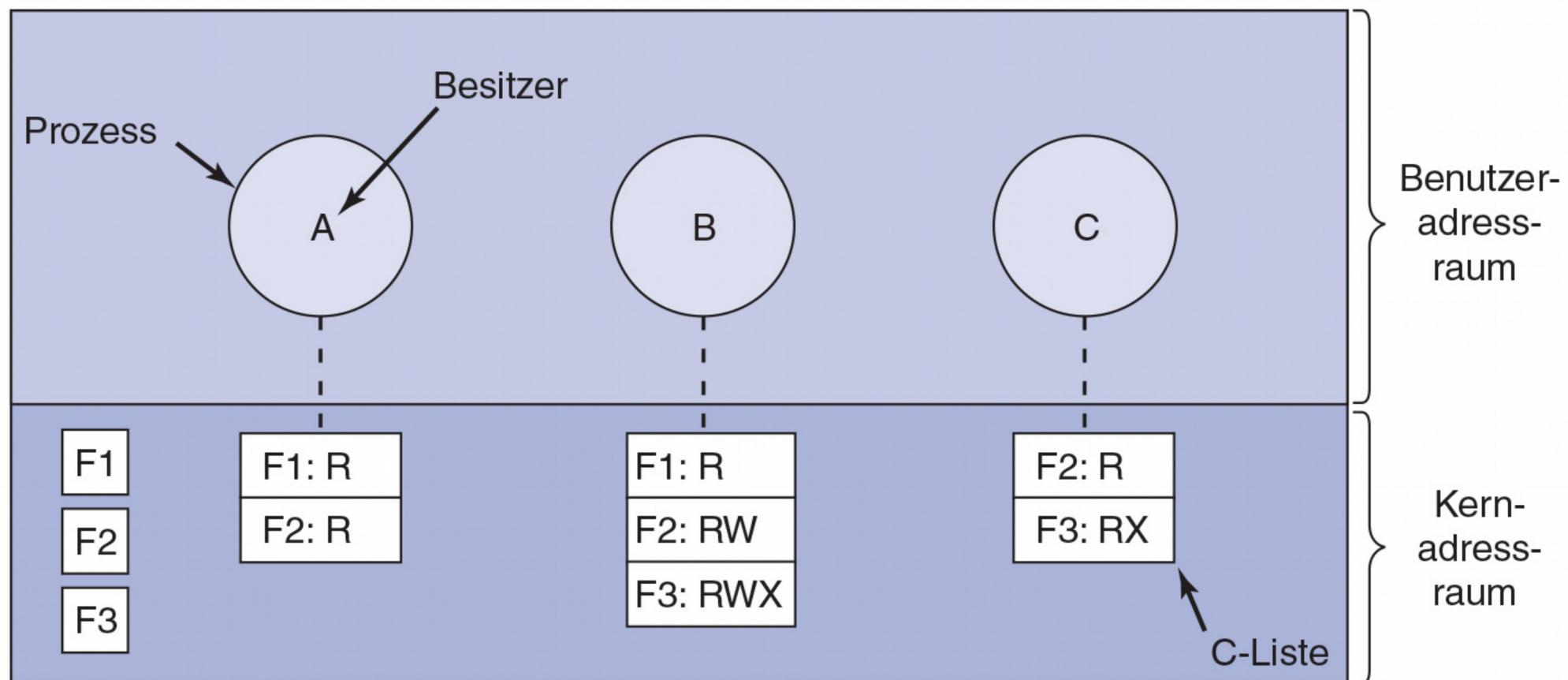
**Abbildung 9.6:** Verwendung von Zugriffskontrolllisten zur Verwaltung des Dateizugriffs.

# Zugriffskontrollisten (2)

Datei	Zugriffskontrollliste
Passwort	tana; sysadm: RW
tauben_daten	willi, taubenfan: RW; tana, taubenfan: RW; ...

**Abbildung 9.7:** Zwei Zugriffskontrollisten.

# Capabilities (1)



**Abbildung 9.8:** Wenn Capability-Listen genutzt werden, hat jeder Prozess eine eigene Capability-Liste.

# Capabilities (2)

Server	Objekt	Rechte	$f(\text{Objekte}, \text{Rechte}, \text{Prüfwert})$
--------	--------	--------	---

**Abbildung 9.9:** Kryptografisch geschützte Capability.

# Capabilities (3)

## Beispiele für generische Rechte:

1. Kopierfunktion: Erstellen von neuen Funktionen für dasselbe Objekt.
2. Objekt kopieren: Erzeuge doppeltes Objekt mit neuer Fähigkeit.
3. Fähigkeit entfernen: Eintrag aus der C-Liste löschen; Objekt nicht betroffen.
4. Objekt zerstören: Objekt und Fähigkeit dauerhaft entfernen.

# 9.4 Formale Modelle von sicheren Systemen

9.4.1 Multilevel-Sicherheit

9.4.2 Verdeckte Kanäle

# Autorisierter und nicht autorisierter Zustand

		Objekte		
		Compiler	Mailbox 7	Geheim
		Read Execute		
Erich				
Henry		Read Execute	Read Write	
Robert		Read Execute		Read Write

a

		Objekte		
		Compiler	Mailbox 7	Geheim
		Read Execute		
Erich				
Henry		Read Execute	Read Write	
Robert		Read Execute	Read	Read Write

b

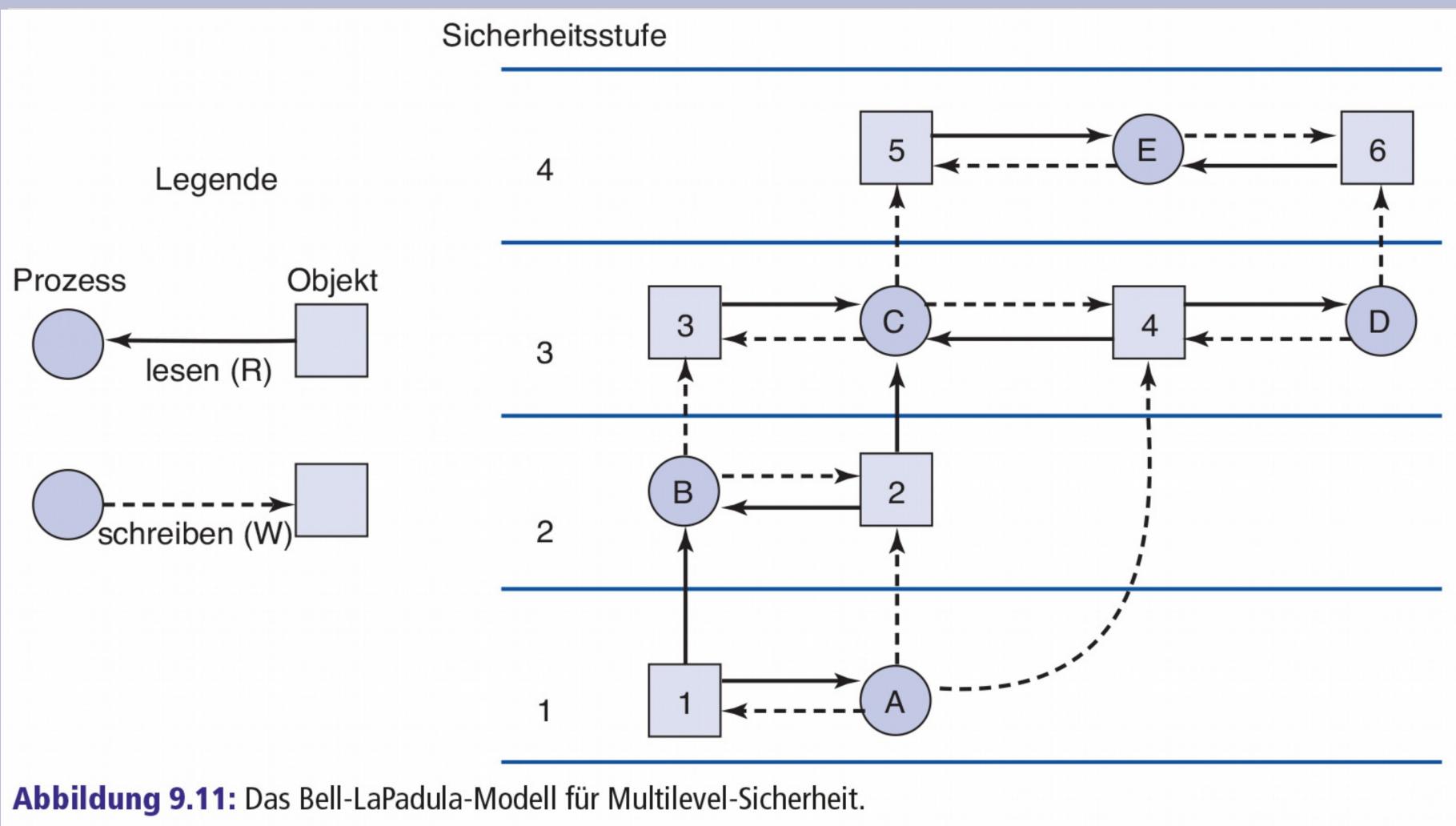
**Abbildung 9.10:** (a) Autorisierter Zustand. (b) Nicht autorisierter Zustand.

# Multilevel Sicherheit Bell-LaPadula Model

Die Bell-LaPadula Modellregeln für den Informationsfluss:

1. Die einfache Sicherheitseigenschaft:
  - Ein Prozess, der auf Sicherheitsstufe k läuft, kann nur Objekte auf seiner Ebene oder niedriger lesen
2. Die \* Eigenschaft:
  - Ein Prozess, der auf Sicherheitsstufe k läuft, kann nur Objekte auf seiner Ebene oder höher schreiben

# Bell-LaPadula Model



**Abbildung 9.11:** Das Bell-LaPadula-Modell für Multilevel-Sicherheit.

# Das Biba Model

Für die Gewährleistung der Integrität der Daten:

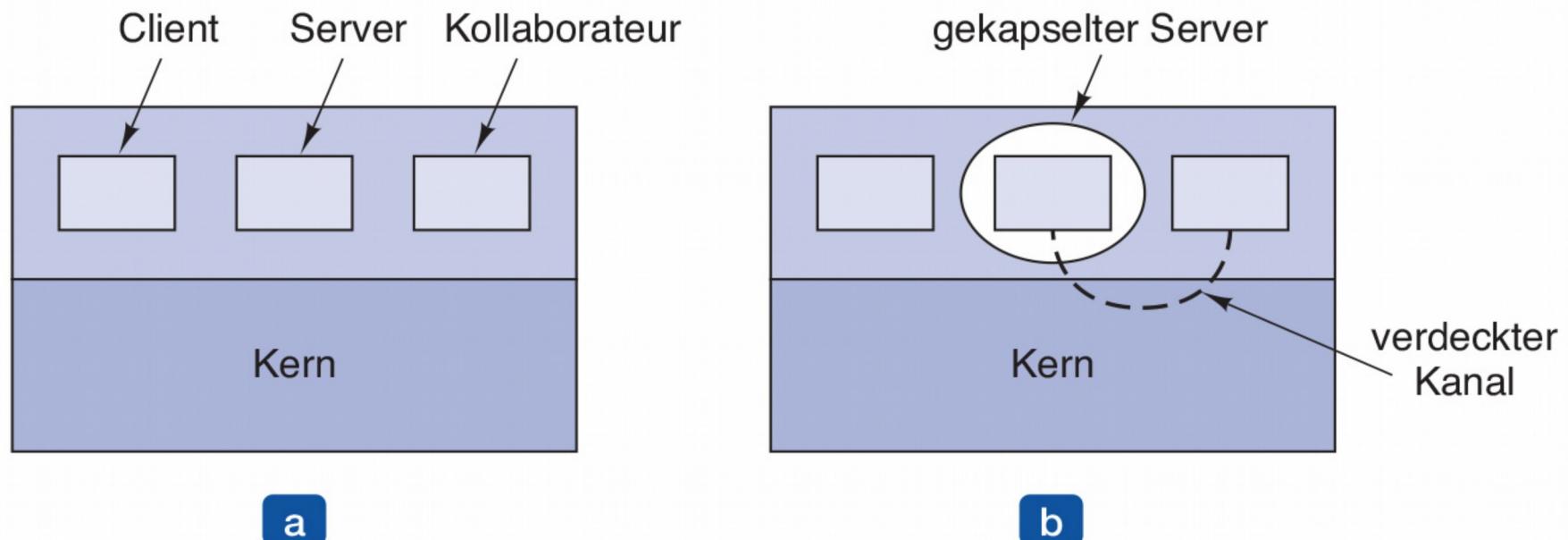
1. Das einfache Integritätsprinzip:

- Ein Prozess, der mit Sicherheitsstufe k ausgeführt wird, kann nur Objekte auf seiner Ebene oder niedriger schreiben (kein Schreiben nach oben).

2. Die Integrität \* -Eigenschaft:

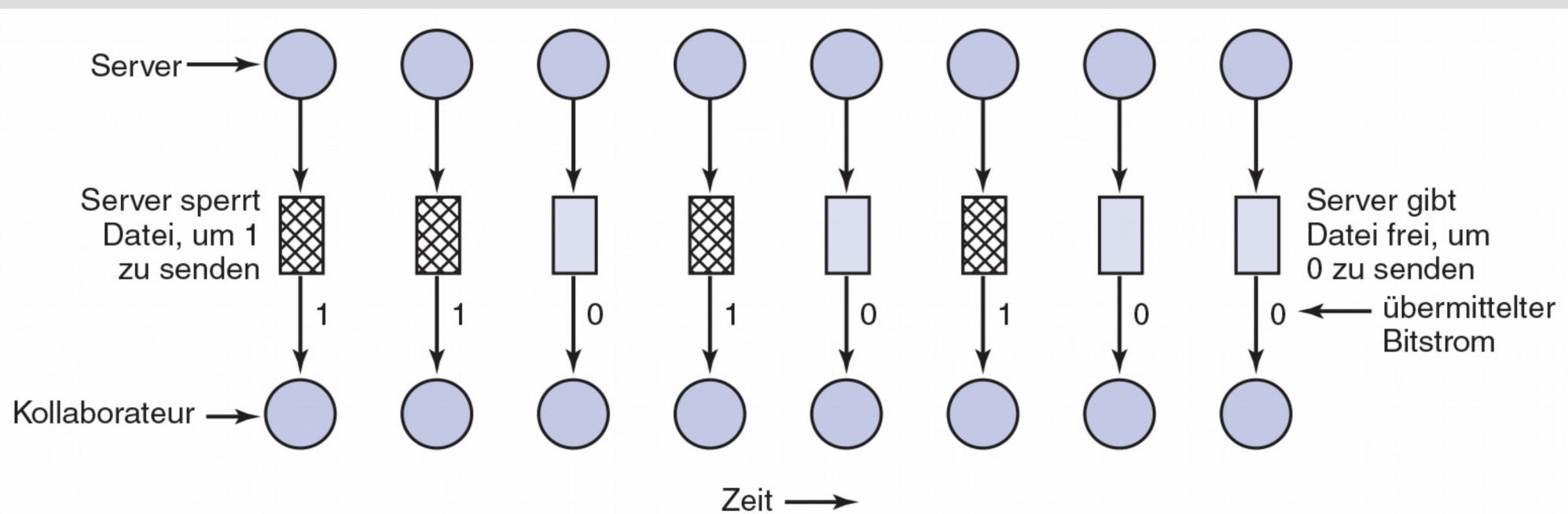
Ein Prozess, der auf Sicherheitsstufe k läuft, kann nur Objekte auf seiner Ebene oder höher lesen (kein Lesen nach unten).

# Verdeckte Kanäle (1)



**Abbildung 9.12:** (a) Die Client-, Server- und Kollaborateur-Prozesse. (b) Der gekapselte Server kann über verdeckte Kanäle immer noch Informationen zum Kollaborateur übermitteln.

# Verdeckte Kanäle (2)



**Abbildung 9.13:** Verdeckter Kanal, der das Sperren von Dateien benutzt.

# 9.5 Grundlagen der Kryptografie

9.5.1 Symmetrische Kryptografie

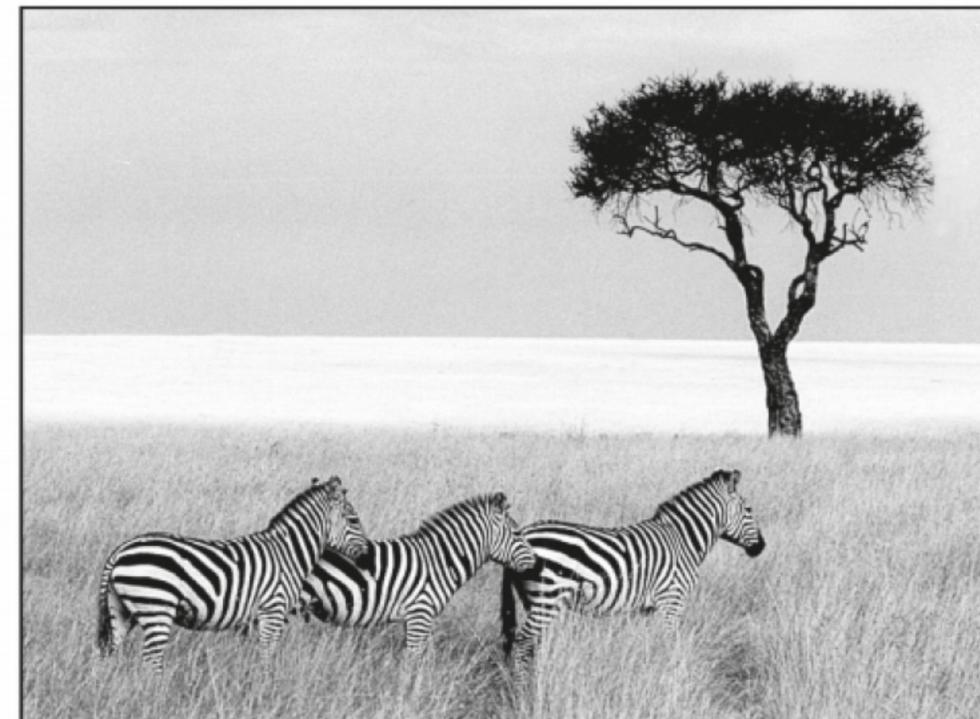
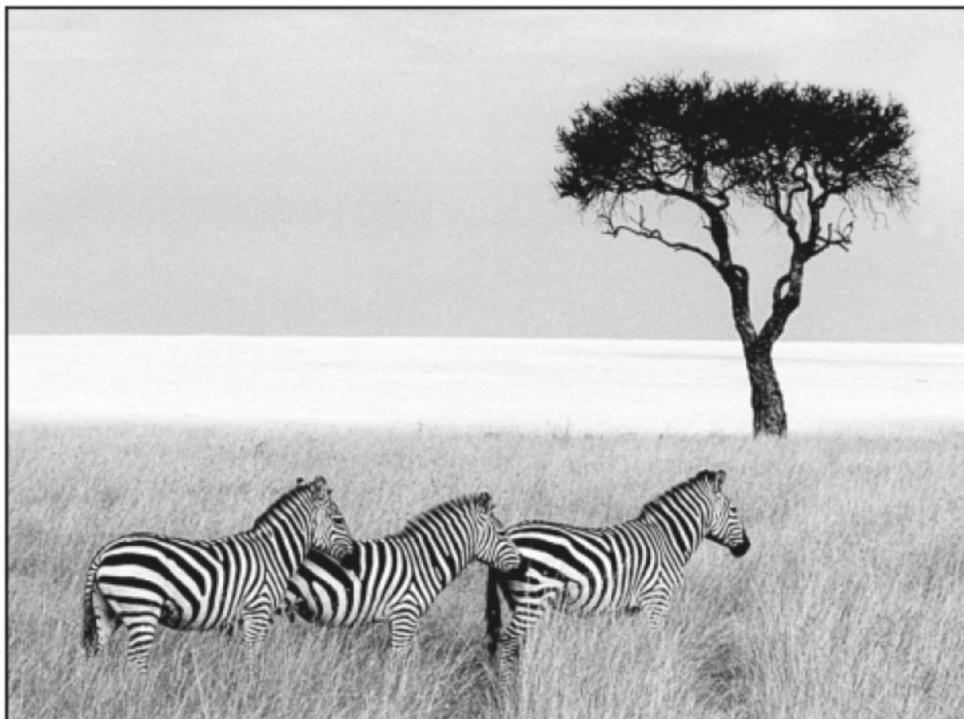
9.5.2 Public-Key-Kryptografie

9.5.3 Einwegfunktionen

9.5.4 Digitale Signaturen

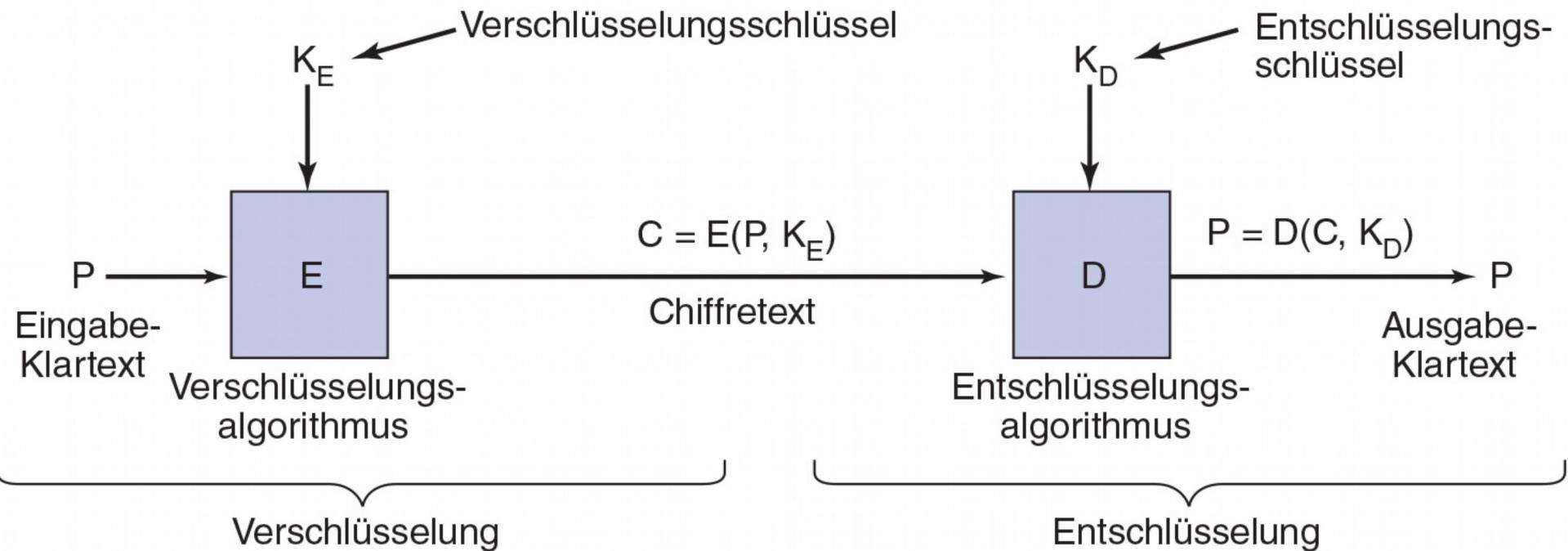
9.5.5 Trusted Platform Module (TPM)

# Steganografie



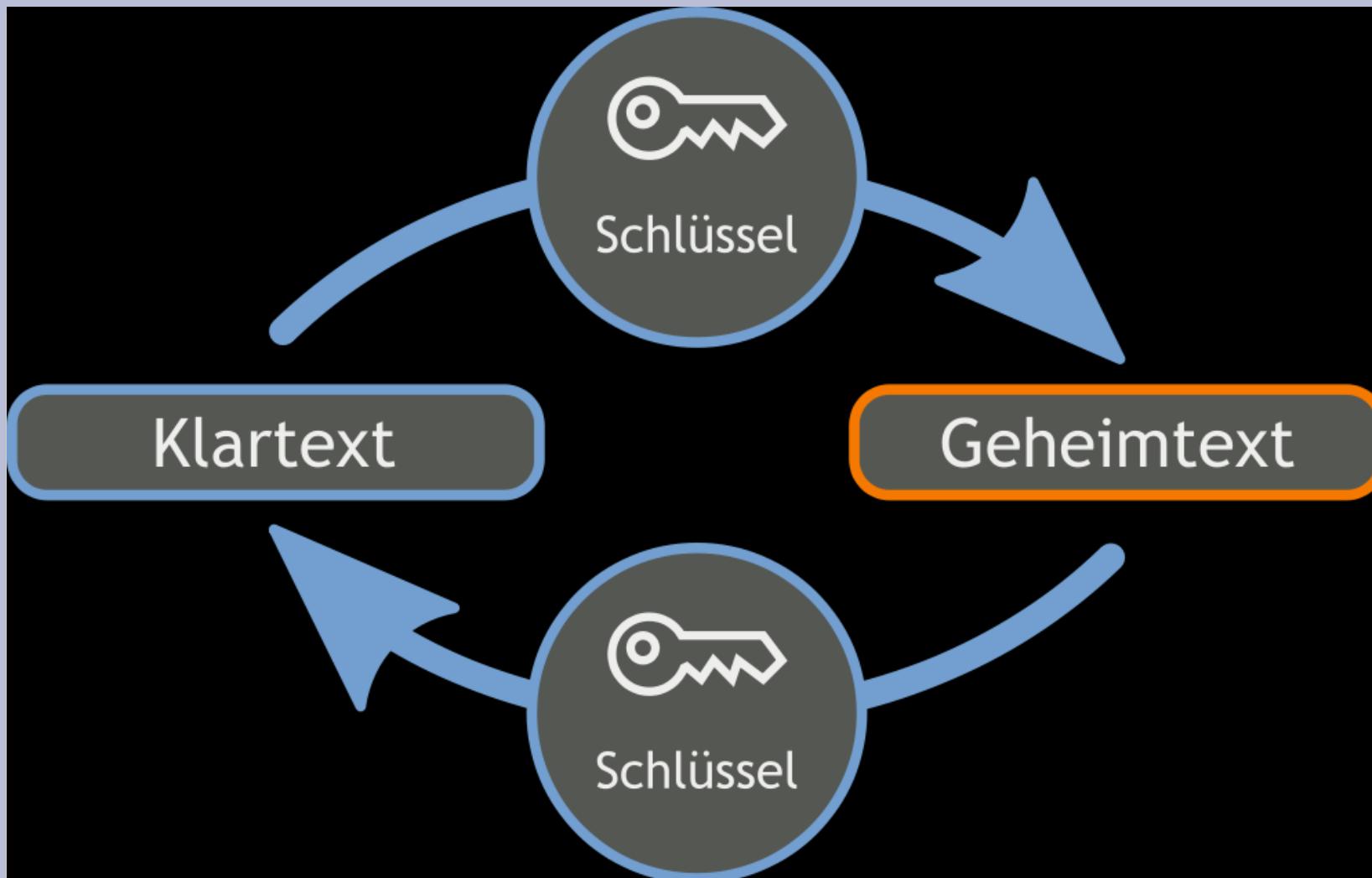
**Abbildung 9.14:** (a) Drei Zebras und ein Baum. (b) Drei Zebras, ein Baum und der vollständige Text von fünf Theaterstücken von William Shakespeare.

# Grundlagen der Kryptografie



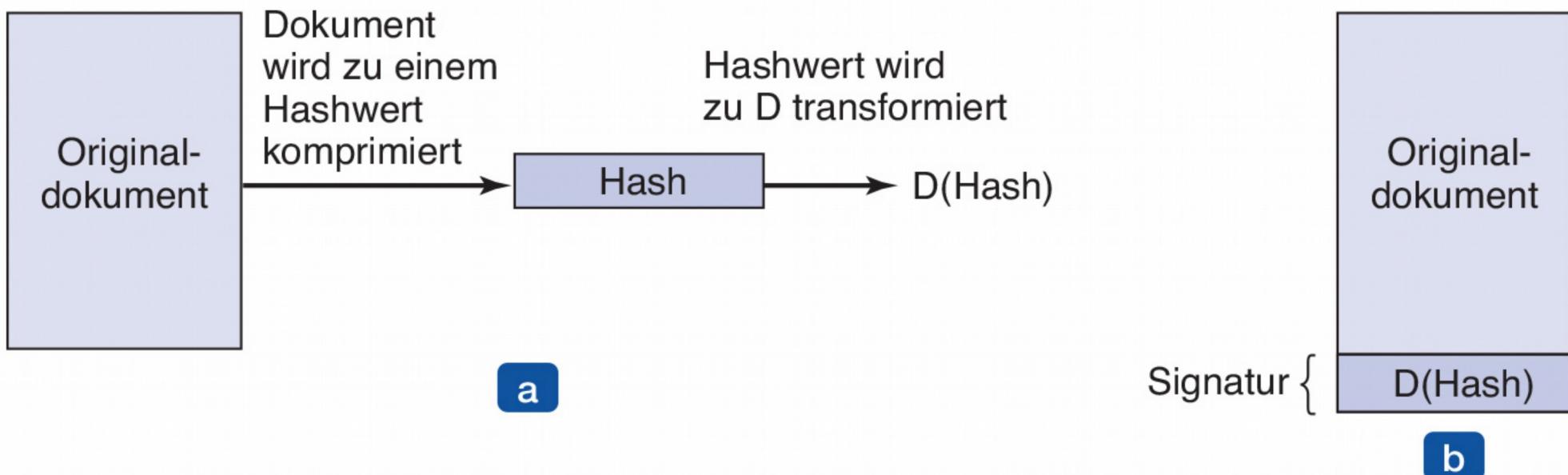
**Abbildung 9.15:** Beziehungen zwischen Klartext und Chiffertext.

# Symmetrische Kryptografie



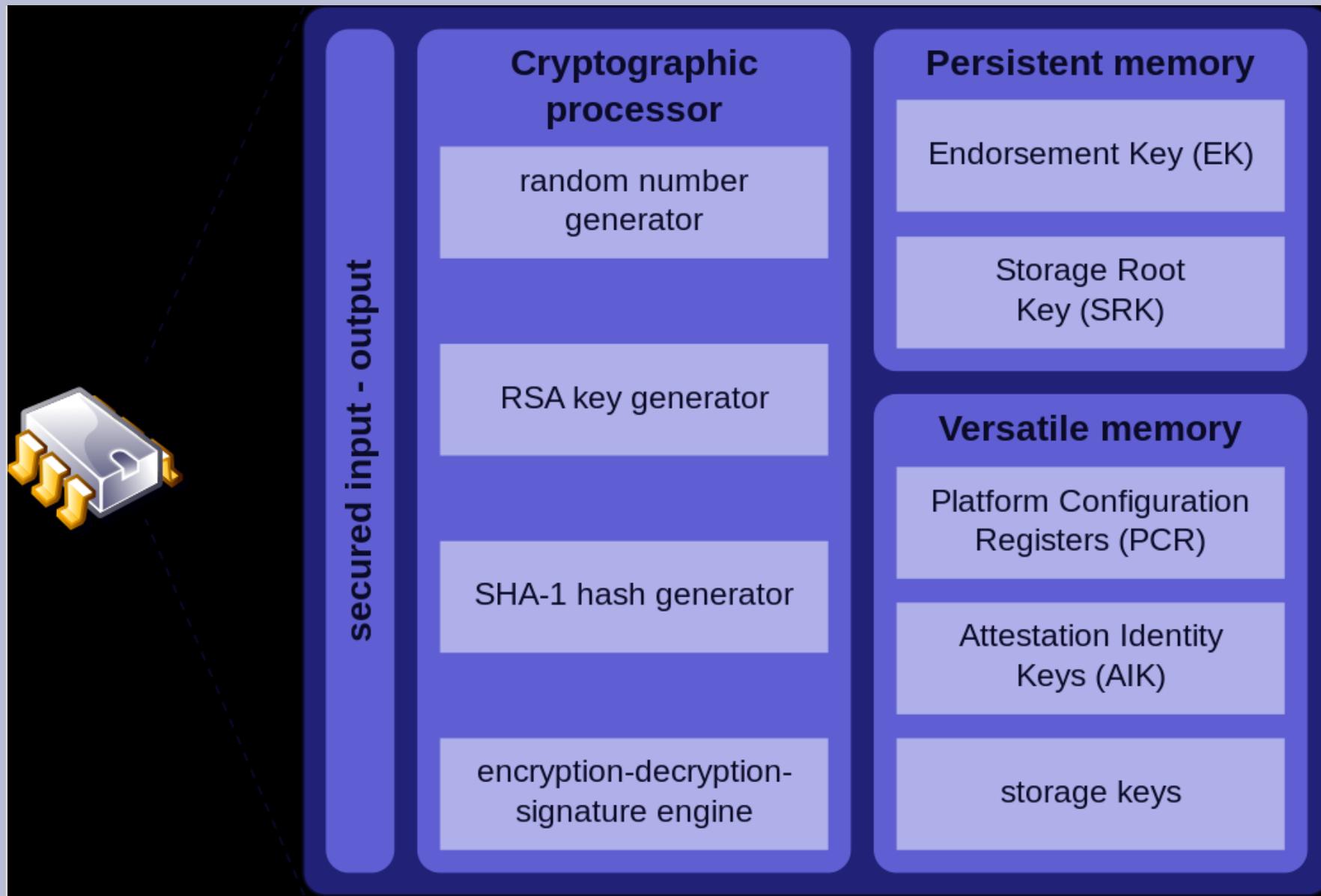
By Bananenfalter (Own work) [CC0], via Wikimedia Commons

# Digitale Signatur



**Abbildung 9.16:** (a) Berechnung einer Signatur. (b) Was der Empfänger erhält.

# Trusted Platform Module (TPM)



Eusebius (Guillaume Piole), Own work, LGPL via  
Wikimedia Commons

# 9.6 Authentifizierung

9.6.1 Authentifizierung durch Besitz

9.6.2 Biometrische Authentifizierung

# Authentifizierung (1)

Die Methoden zur Authentifizierung von Benutzern, wenn sie versuchen Sie sich anzumelden, basieren auf einem von drei allgemeinen Prinzipien:

1. Etwas, das der Benutzer kennt.
2. Etwas, das der Benutzer hat.
3. Etwas, was der Benutzer ist.

# Authentifizierung (2)

LOGIN: mitch  
PASSWORD: FooBar!-7  
SUCCESSFUL LOGIN

(a)

LOGIN: carol  
INVALID LOGIN NAME  
LOGIN:

(b)

LOGIN: carol  
PASSWORD: Idunno  
INVALID LOGIN  
LOGIN:

(c)

**Abbildung 9.17:** (a) Erfolgreiches Login. (b) Login abgewiesen, nachdem Name eingegeben wurde. (c) Login abgewiesen, nachdem Name und Passwort eingetippt wurden.

# UNIX Passort Sicherheit

Bobbie, 4238, e(Dog, 4238)

Tony, 2918, e(6 %%TaeFF, 2918)

Lauro, 6902, e(Shakespeare, 6902)

Mark, 1694, e(XaB#Bwcz, 1694)

Deborah, 1092, e(LordByron, 1092)

---

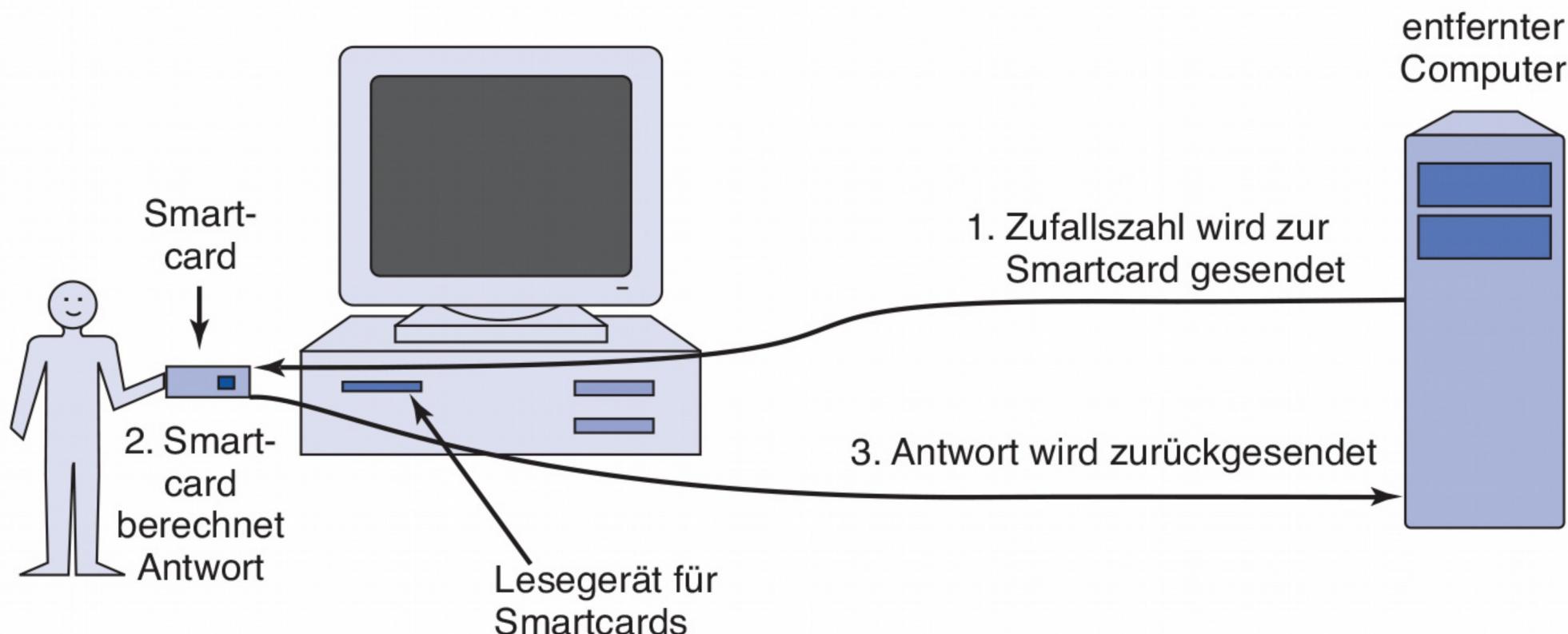
**Abbildung 9.18:** Die Nutzung von Salt zur Verhinderung der Vorausberechnung von verschlüsselten Passwörtern.

# Challenge-Response Authentifizierung

**Die Fragen sollten so gewählt werden, dass der Benutzer sie nicht aufschreiben muss. Beispiele:**

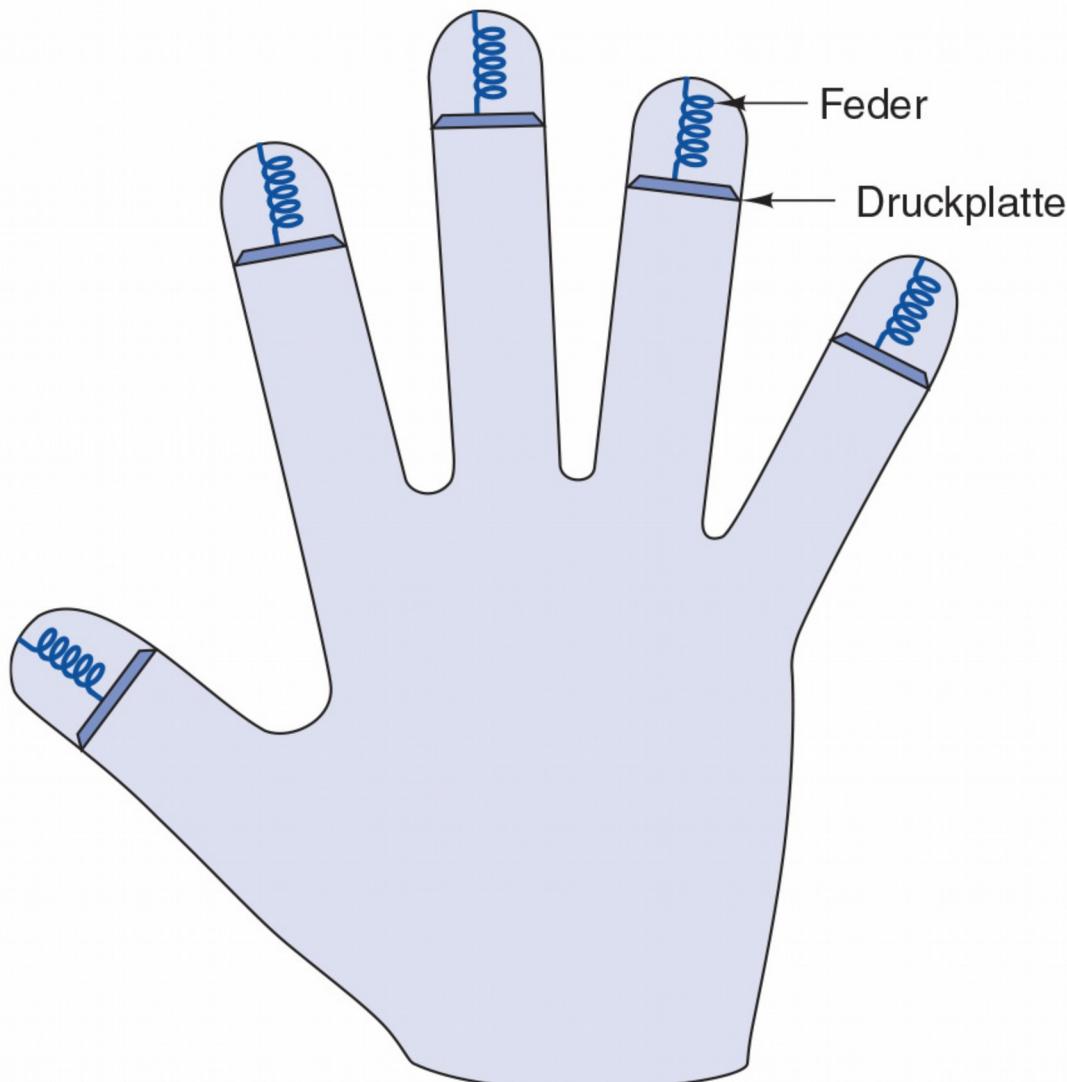
1. Wer ist Marjoleins Schwester?
2. Auf welcher Straße war deine Grundschule?
3. Was hat Frau Ellis gelehrt?

# Authentifizierung mit einem physischen Objekt



**Abbildung 9.19:** Verwendung einer Smartcard zur Authentifizierung.

# Biometrische Authentifizierung



**Abbildung 9.20:** Eine Vorrichtung zur Fingerlängenmessung.

# 9.7 Ausnutzen von Sicherheitslücken

9.7.1 Pufferüberlaufangriffe

9.7.2 Formatstring-Angriffe

9.7.3 Hängende Zeiger

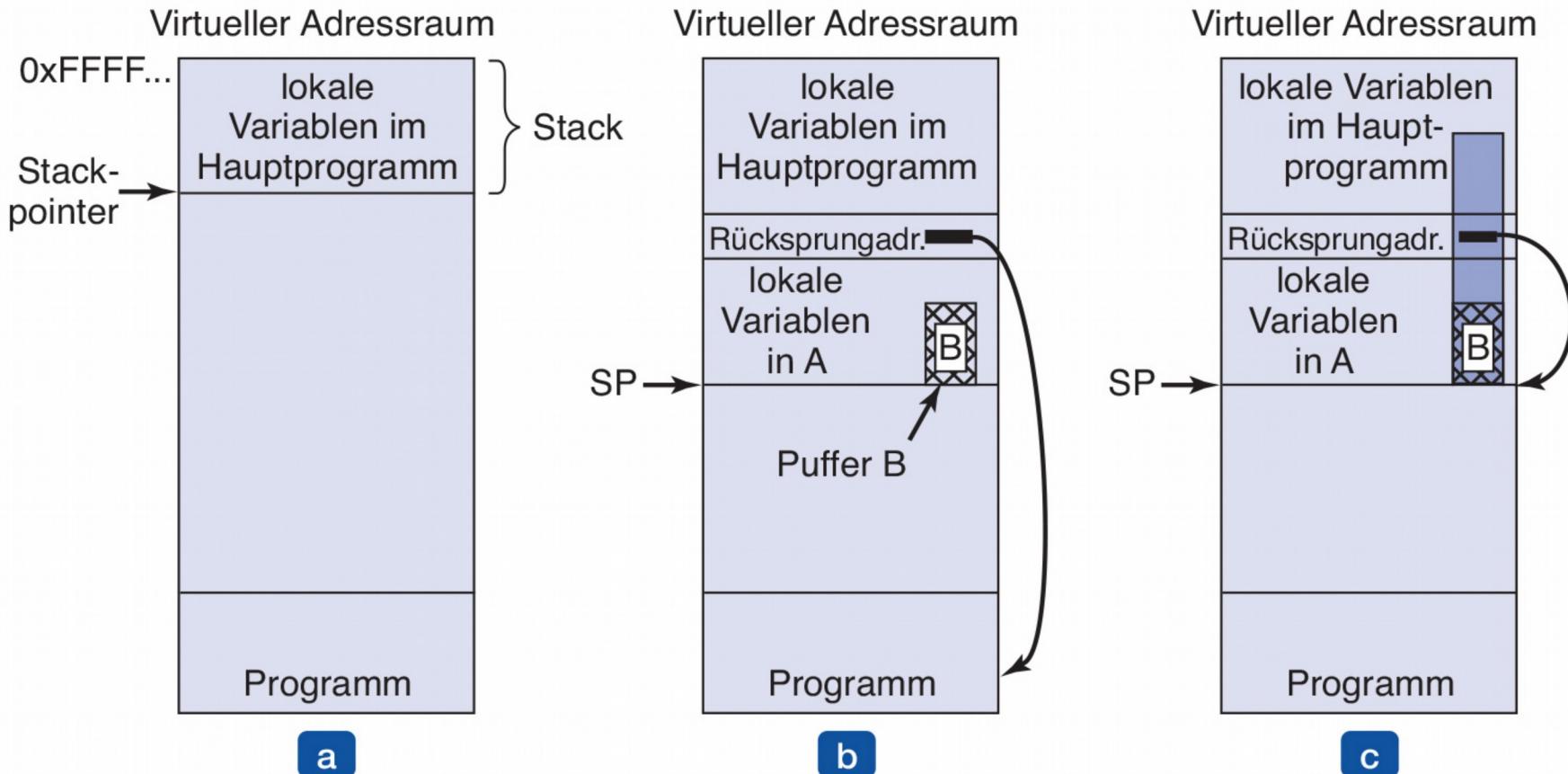
9.7.4 NULL-Zeiger-Dereferenzierungsangriff

9.7.5 Angriffe durch Ganzzahlüberlauf

9.7.6 Angriffe durch Kommando-Injektion

9.7.7 Time-of-Check-to-Time-of-Use-Angriff

# Pufferüberlaufangriffe



**Abbildung 9.21:** (a) Zustand bei laufendem Hauptprogramm. (b) Nachdem die Prozedur A aufgerufen wurde. (c) Pufferüberlauf wird dunkler dargestellt.

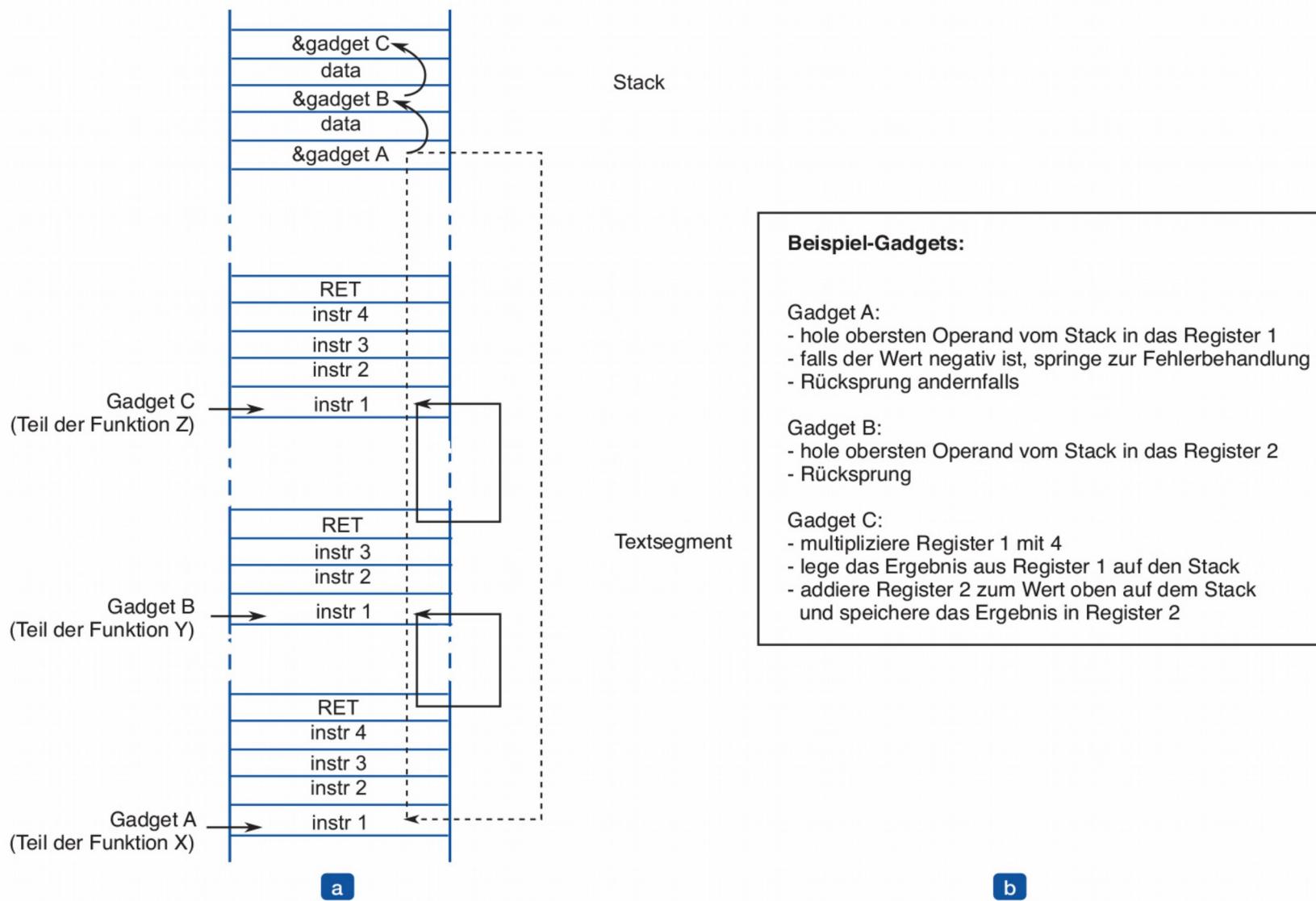
# Stack-Canary überspringen

```
01. void A (char *date) {  
02.     int len;  
03.     char B [128];  
04.     char logMsg [256];  
05.  
06.     strcpy (logMsg, date);      /* kopiere zuerst die Zeichenkette */  
07.                                         /* mit dem Datum in den Protokolleintrag */  
08.     len = strlen (date);       /* stelle fest, wie viele Zeichen */  
09.                                         /* in der Datumszeichenkette sind */  
10.     gets (B);                /* nun lies die Eingabe */  
11.     strcpy (logMsg+len, B);    /* und kopiere diese hinter das */  
12.                                         /* Datum in den Protokolleintrag */  
13.     writeLog (logMsg);        /* und schreibe zuletzt den */  
14.                                         /* Protokolleintrag auf die Platte */  
15. }
```

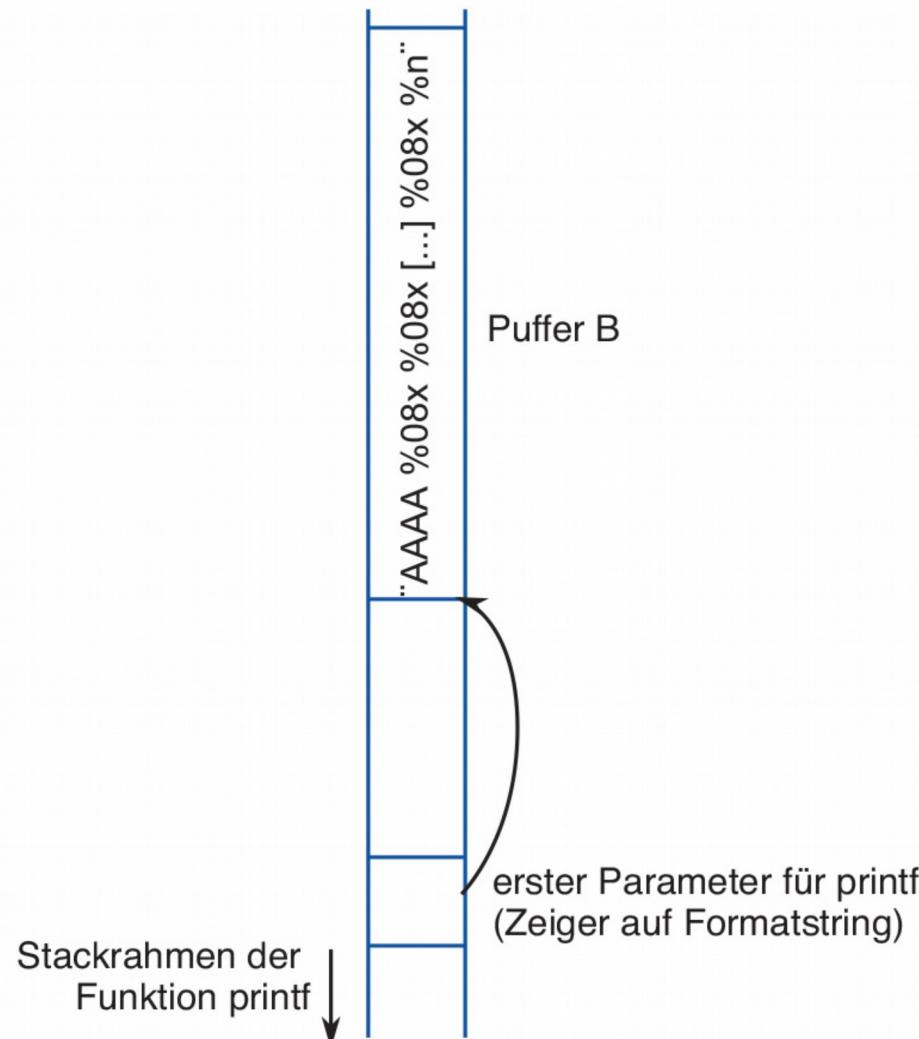
**Abbildung 9.22:** Überspringen des Stack-Canary: Indem zuerst *len* verändert wird, kann das Canary umgangen und die Rücksprungadresse direkt verändert werden.

# Code Reuse Attacks

## Return-Oriented Programming (ROP)



# Formatstring-Angriffe



**Abbildung 9.24:** Ein Formatstring-Angriff. Indem genau die richtige Anzahl von `%0x` verwendet wird, kann der Angreifer die ersten vier Zeichen des Formatstrings als Adresse benutzen.

# Angriffe durch Kommando-Injektion

```
int main(int argc, char *argv[])
{
    char src[100], dst[100], cmd[205] = "cp ";
    /* deklariere 3 */
    /* Zeichenketten */
    /* frage nach der */
    /* Quelldatei */
    /* hole Eingabe von */
    /* der Tastatur */
    /* hänge src hinter */
    /* cp an */
    /* füge ein Leer- */
    /* zeichen an das */
    /* Ende von cmd an */
    /* frage nach */
    /* Ausgabedateinamen */
    /* hole Eingabe von */
    /* der Tastatur */
    /* vervollständige */
    /* Kommandozeile */
    /* führe cp-Kommando */
    /* aus */

    printf("Please enter name of source file: ");
    gets(src);
    strcat(cmd, src);
    strcat(cmd, " ");
    printf("Please enter name of destination file: ");
    gets(dst);
    strcat(cmd, dst);
    system(cmd);
}
```

**Abbildung 9.25:** Code, der zu einem Kommando-Injektionsangriff führen könnte.

## 9.8 Insider-Angriffe

9.8.1 Logische Bomben

9.8.2 Hintertüren

9.8.3 Login-Spoofing

# Logische Bomben

**Eine Logische Bombe besteht aus zwei Komponenten:**

- 1. Eine hinreichend spezifische Bedingung**  
z. B. Erreichen eines bestimmten Datums, Fehlen einer bestimmten Datei oder Ausführen durch einen bestimmten Benutzer
- 2. Eine Explosion, d.h., eine schädliche Aktion**  
Löschen von Daten oder das Infizieren eines Systems mittels eines Virus

# Hintertür

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

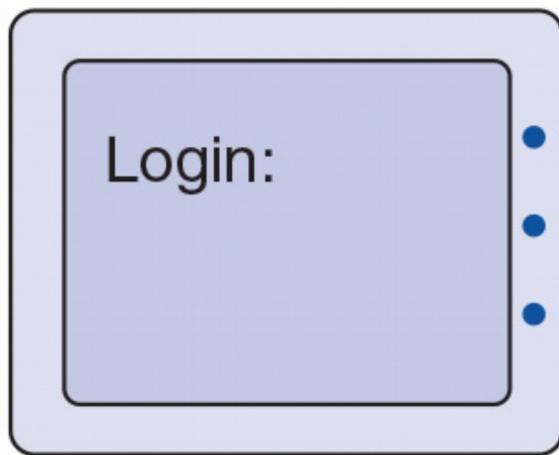
a

```
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);
```

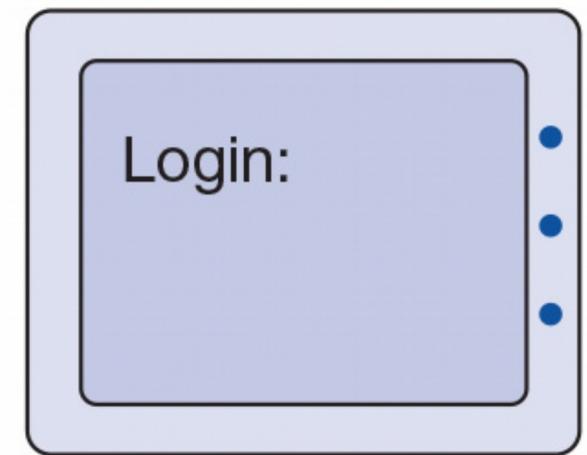
b

**Abbildung 9.26:** (a) Normaler Code. (b) Code mit eingebauter Hintertür.

# Login-Spoofing



a



b

**Abbildung 9.27:** (a) Korrekter Login-Bildschirm. (b) Falscher Login-Bildschirm.

## 9.9 Malware

9.9.1 Trojanische Pferde

9.9.2 Viren

9.9.3 Würmer

9.9.4 Spyware

9.9.5 Rootkits

# Viren für ausführbare Programme (1)

```
#include <sys/types.h>          /* Standard-POSIX-Header */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                /* für lstat-Aufruf: Ist Datei ein symb. Link? */

search(char *dir_name)
{
    DIR *dirp;                  /* rekursive Suche nach */
                                /* ausführbaren Dateien */
                                /* Zeiger auf offenen Verzeichnisstrom */
```

## Programme (2)

### Viren für ausführbare

Tanenbaum, A. S.; Bos, H.: Moderne  
Betriebssysteme. Pearson Studium 2016

```
struct dirent *dp;          /* Zeiger auf Verzeichniseintrag */

dirp = opendir(dir_name);   /* Öffnen dieses Verzeichnisses */
if (dirp == NULL) return;  /* Verzeichnis kann nicht geöffnet werden */

while (TRUE) {
    dp = readdir(dirp);    /* Lies nächsten Verzeichniseintrag */
    if (dp == NULL) {       /* NULL bedeutet: wir sind fertig */
        chdir ("..");      /* zurück zum Eltern-Verzeichnis */
        break;               /* Verlassen der Schleife */
    }
}

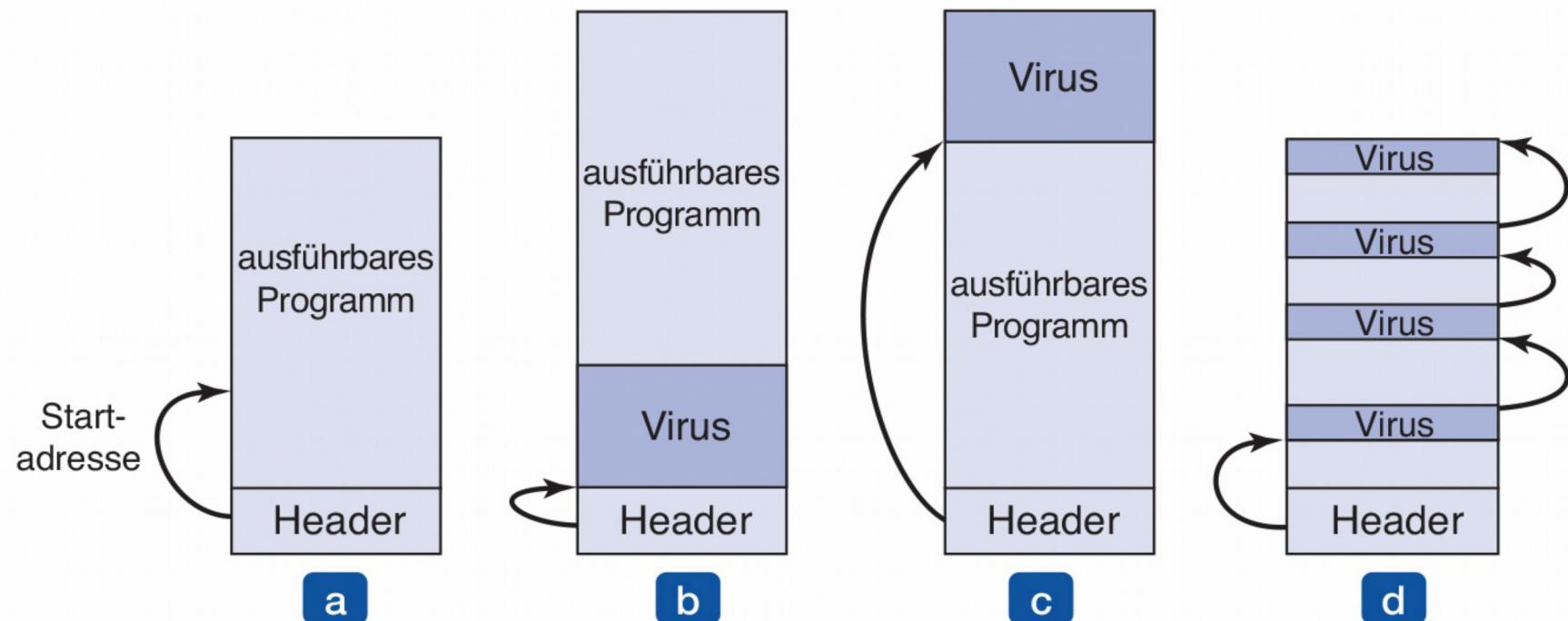
if (dp->d_name[0] == '.') continue; /* überspringe die . und .. */
                                    /* Verzeichnisse */
lstat(dp->d_name, &sbuf);         /* Ist Eintrag ein symb. Link? */
if (S_ISLNK(sbuf.st_mode)) continue; /* überspringe symb. Links */
if (chdir(dp->d_name) == 0) {       /* Ist chdir erfolgreich, */
    search(".");
}

} else {
    if (access(dp->d_name,X_OK) == 0)
        infect(dp->d_name);
}

closedir(dirp);                  /* Verzeichnis bearbeitet; */
                                /* schließen und Rücksprung */
```

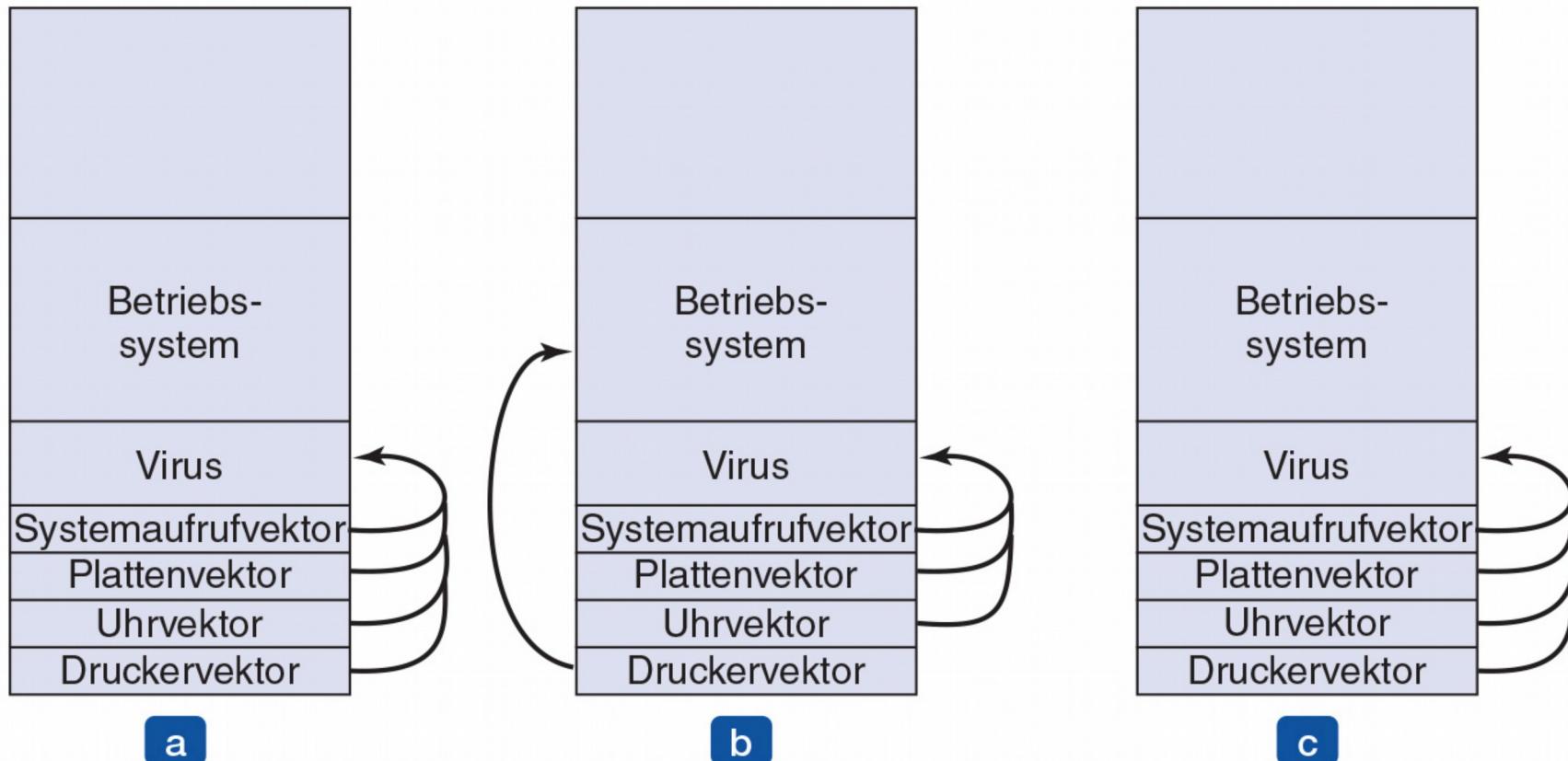
**Abbildung 9.28:** Eine rekursive Prozedur, die in einem UNIX-System ausführbare Dateien findet.

# Viren für ausführbare Programme (3)



**Abbildung 9.29:** (a) Ausführbares Programm. (b) Virus am Anfang. (c) Virus am Ende. (d) Virus, das über freien Platz innerhalb des Programms verteilt ist.

# Boot-Sektor Viren



**Abbildung 9.30:** (a) Nachdem das Virus alle Interruptvektoren belegt hat. (b) Nachdem das Betriebssystem den Drucker-Interruptvektor wiederbekommen hat. (c) Nachdem das Virus den Verlust des Drucker-Interruptvektors bemerkt und diesen wiedererobert hat.

# Spyware – Browser mit Toolbars



Dumont, A.: Mittel gegen den Toolbar-Terror. <https://www.com-magazin.de/praxis/internet/mittel-toolbar-terror-7433.html>

# Aktionen von Spyware (1)

1. Die Homepage des Browsers ändern.
2. Die Liste der bevorzugten (mit Lesezeichen versehenen) Seiten des Browsers ändern.
3. Dem Browser neue Symbolleisten hinzufügen.
4. Den Standard-Media Player des Benutzers ändern.
5. Die Standardsuchmaschine des Benutzers ändern.

## Aktionen von Spyware (2)

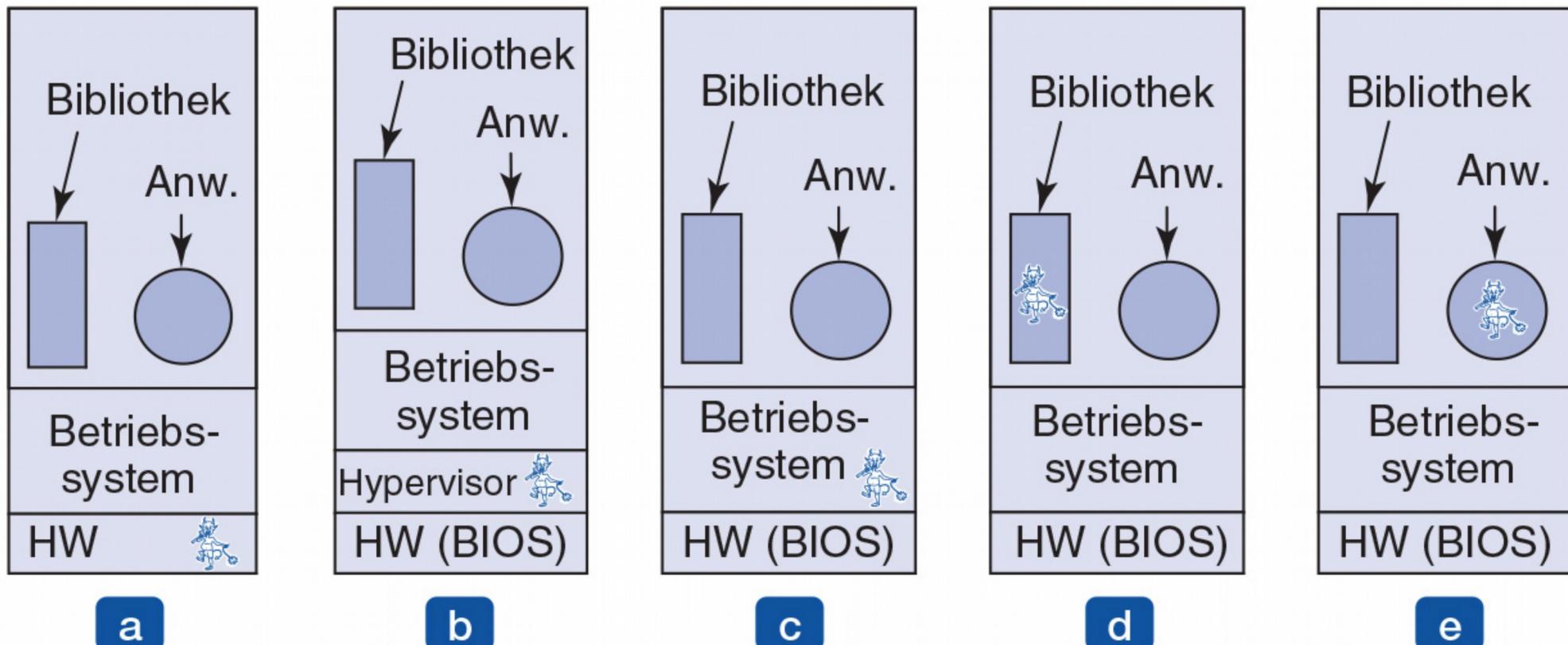
6. Dem Windows-Desktop neue Symbole hinzufügen.
7. Ersetzen von Werbebannern auf Webseiten durch von der Spyware auswählte.
8. Anzeigen in die Standard-Windows-Dialogfelder einsetzen.
9. Einen kontinuierlichen und unaufhaltsamen Popup-Popup-Stream generieren.

# Arten von Rootkits (1)

**Fünf Arten von Rootkits – die Frage ist, wo verstecken sie sich?**

1. Firmware-Rootkit
2. Hypervisor-Rootkit
3. Kernel-Rootkit
4. Bibliotheks-Rootkit
5. Anwendungs-Rootkit

# Arten von Rootkits (2)



**Abbildung 9.31:** Fünf Orte, an denen sich ein Rootkit verstecken kann.

# 9.10 Abwehrmechanismen

9.10.1 Firewalls

9.10.2 Antivieren- und Anti-Antivierentechniken

9.10.3 Codesignierung

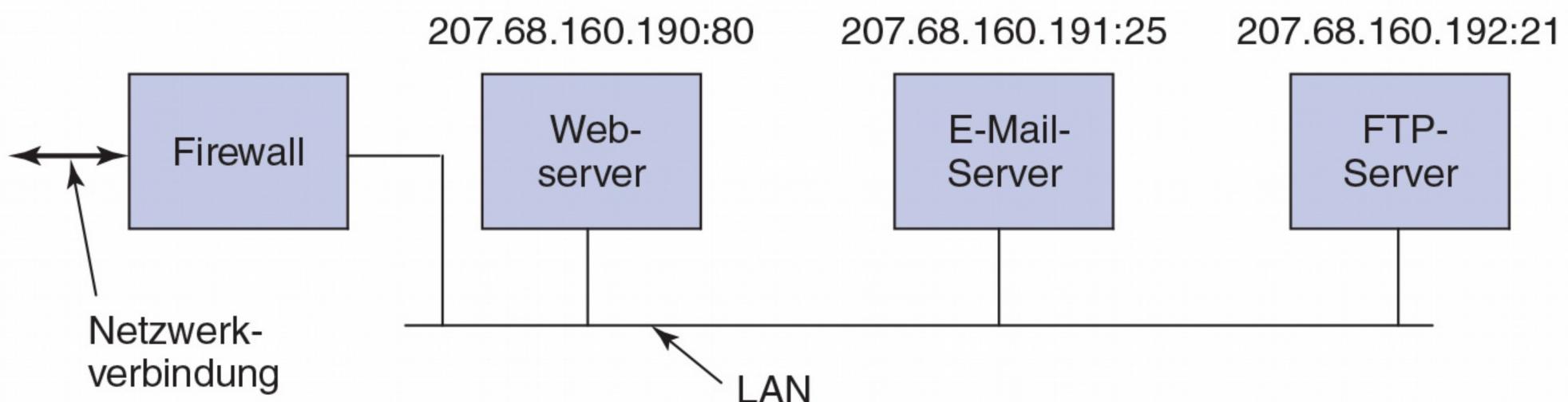
9.10.4 Jailing

9.10.5 Modellbasierte Angriffserkennung

9.10.6 Kapselung von mobilem Code

9.10.7 Java-Sicherheit

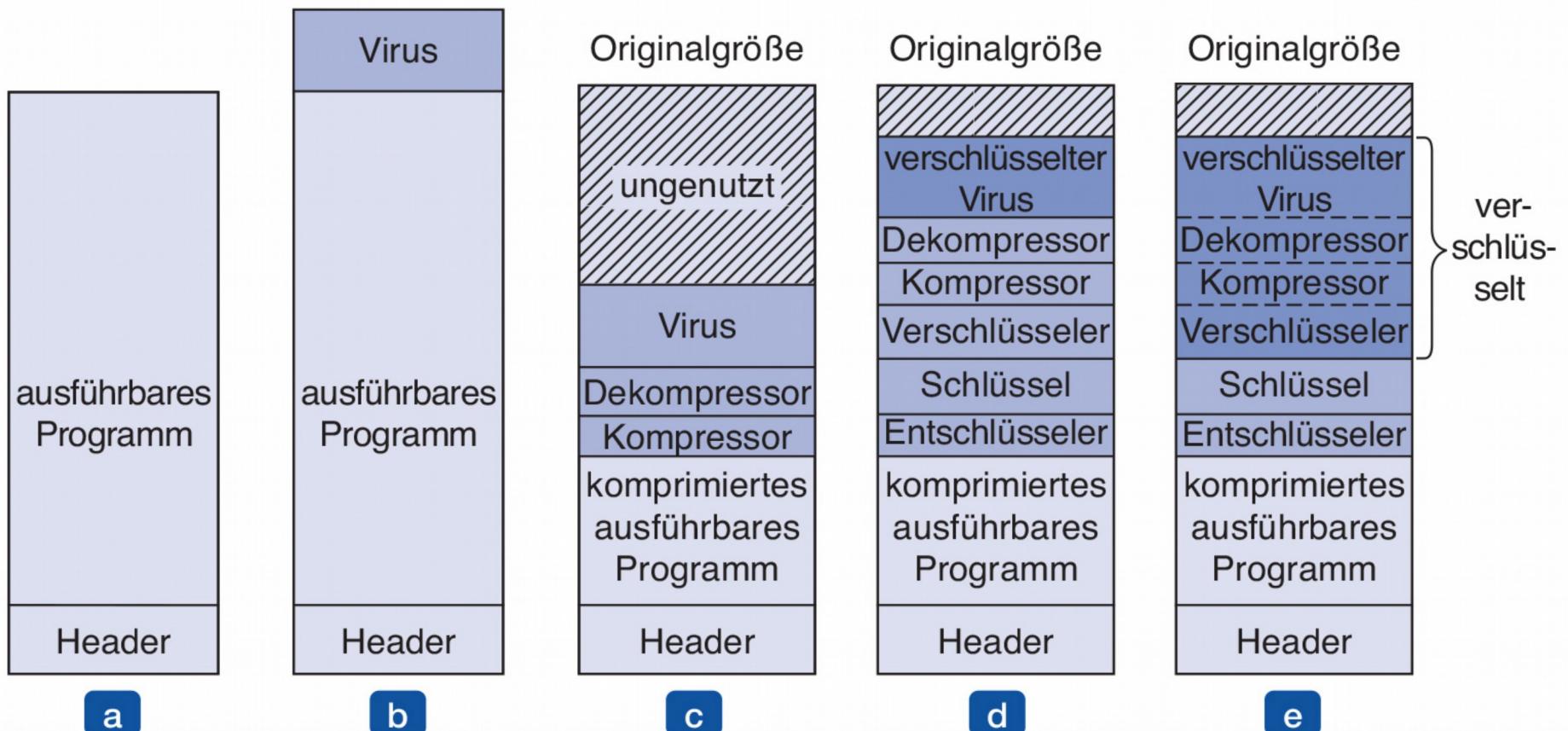
# Firewalls



**Abbildung 9.32:** Vereinfachte Sicht einer Hardware-Firewall, die ein LAN mit drei Computern schützt.

# Virenscanner (1)

Datei ist länger



**Abbildung 9.33:** (a) Programm. (b) Infiziertes Programm. (c) Komprimiertes infiziertes Programm. (d) Verschlüsselter Virus. (e) Komprimiertes Virus mit verschlüsseltem Kompressionscode.

# Virenscanner (2)

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

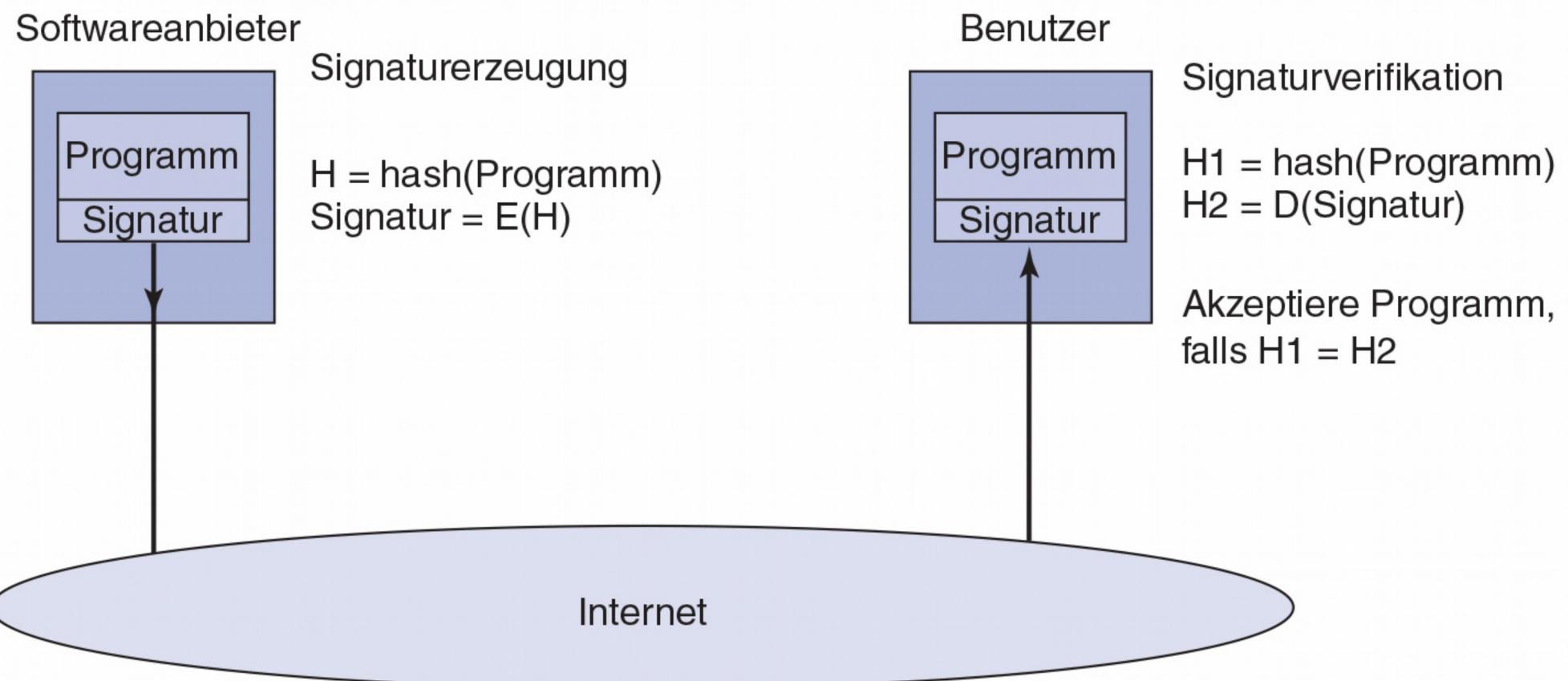
c

d

e

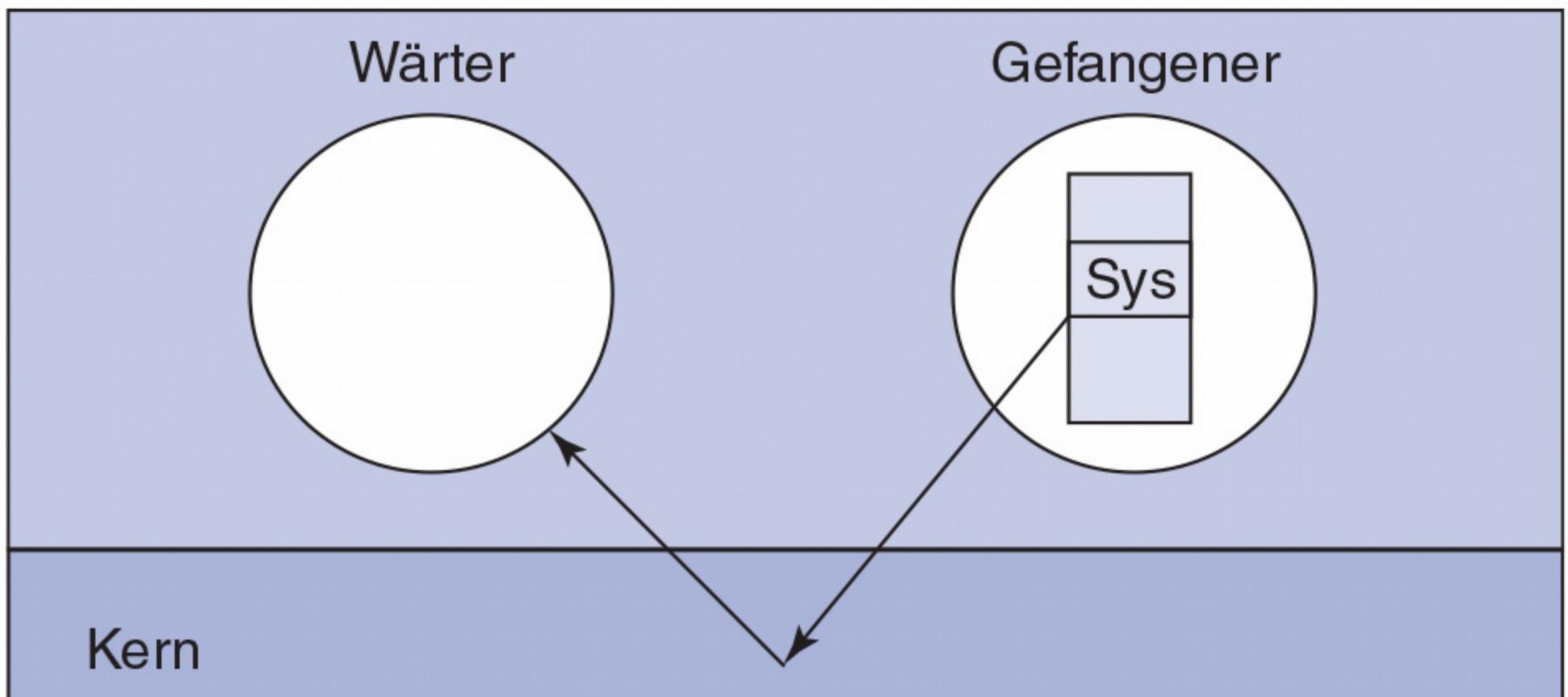
**Abbildung 9.34:** Beispiele für ein polymorphes Virus.

# Codesignierung



**Abbildung 9.35:** Funktionsweise der Codesignierung.

# Jailing



**Abbildung 9.36:** Die Jailing-Operation.

# Modellbasierte Angriffserkennung

```
int main(int argc *char argv[])
{
    int fd, n = 0;
    char buf[1];

    fd = open("data", 0);
    if (fd < 0) {
        printf("Bad data file\n");
        exit(1);
    } else {
        while (1) {
            read(fd, buf, 1);
            if (buf[0] == 0) {
                close(fd);
                printf("n = %d\n", n);
                exit(0);
            }
            n = n + 1;
        }
    }
}
```

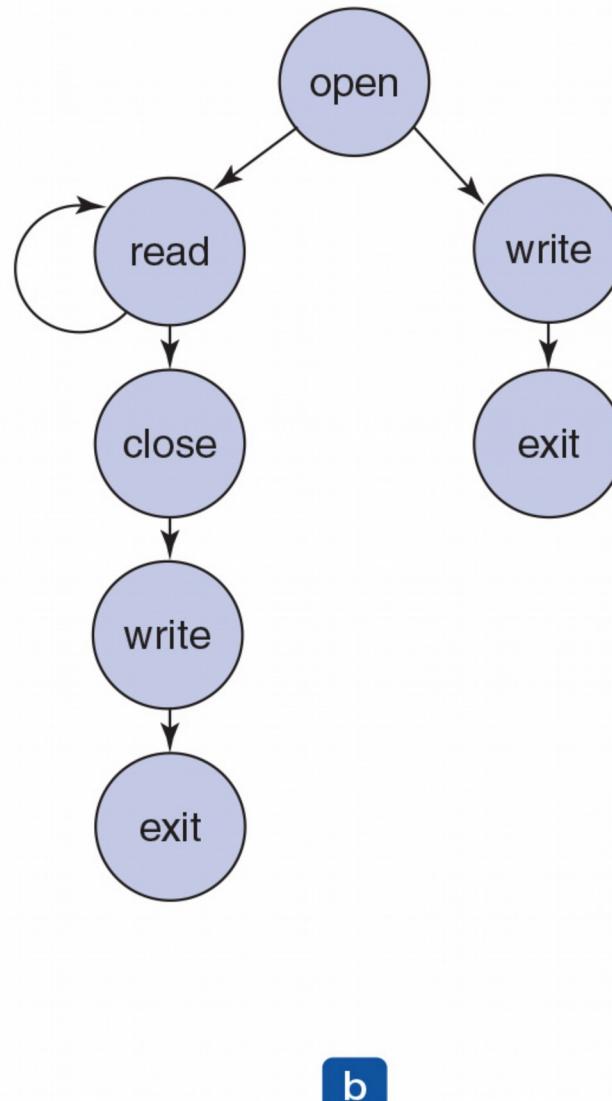
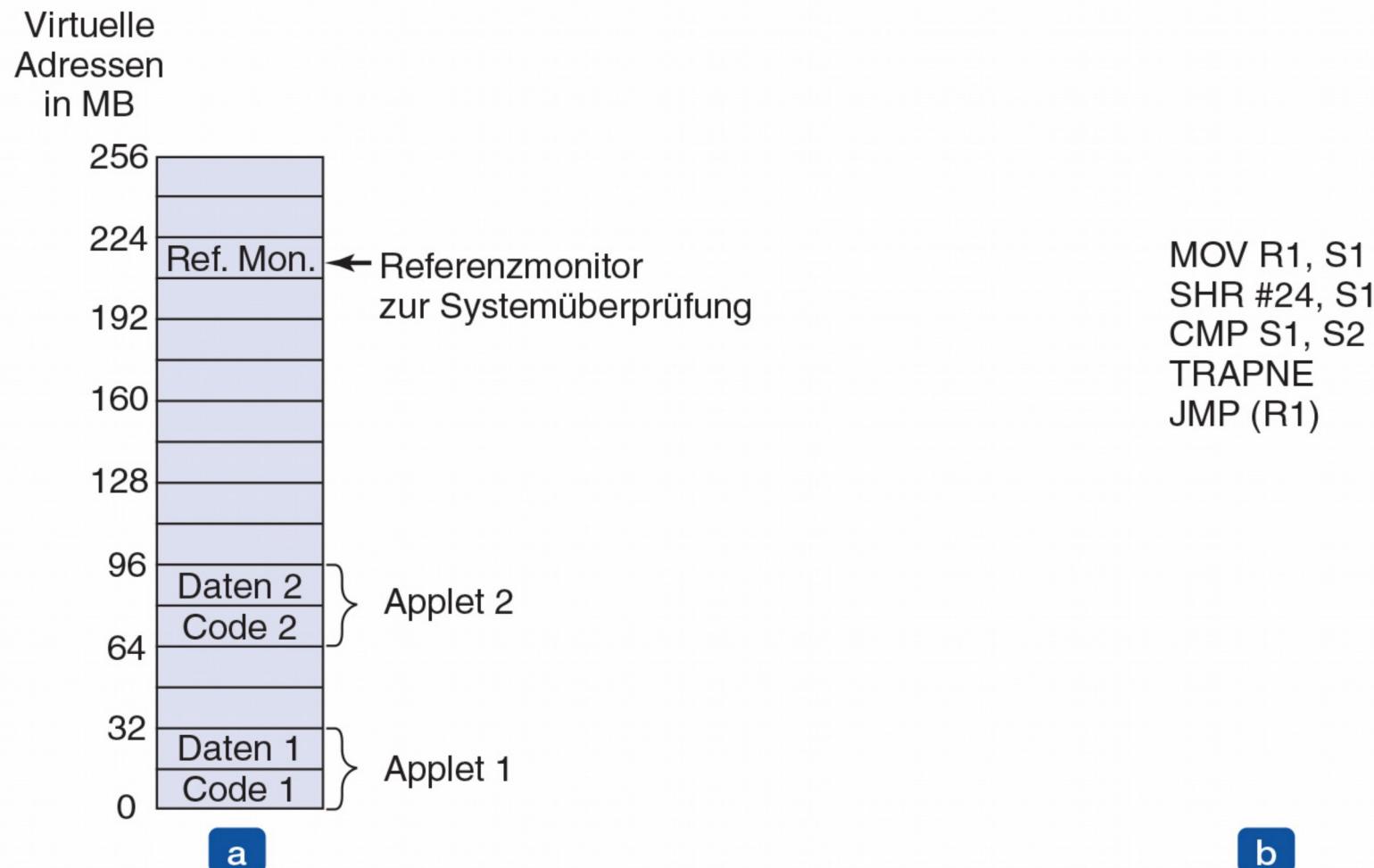


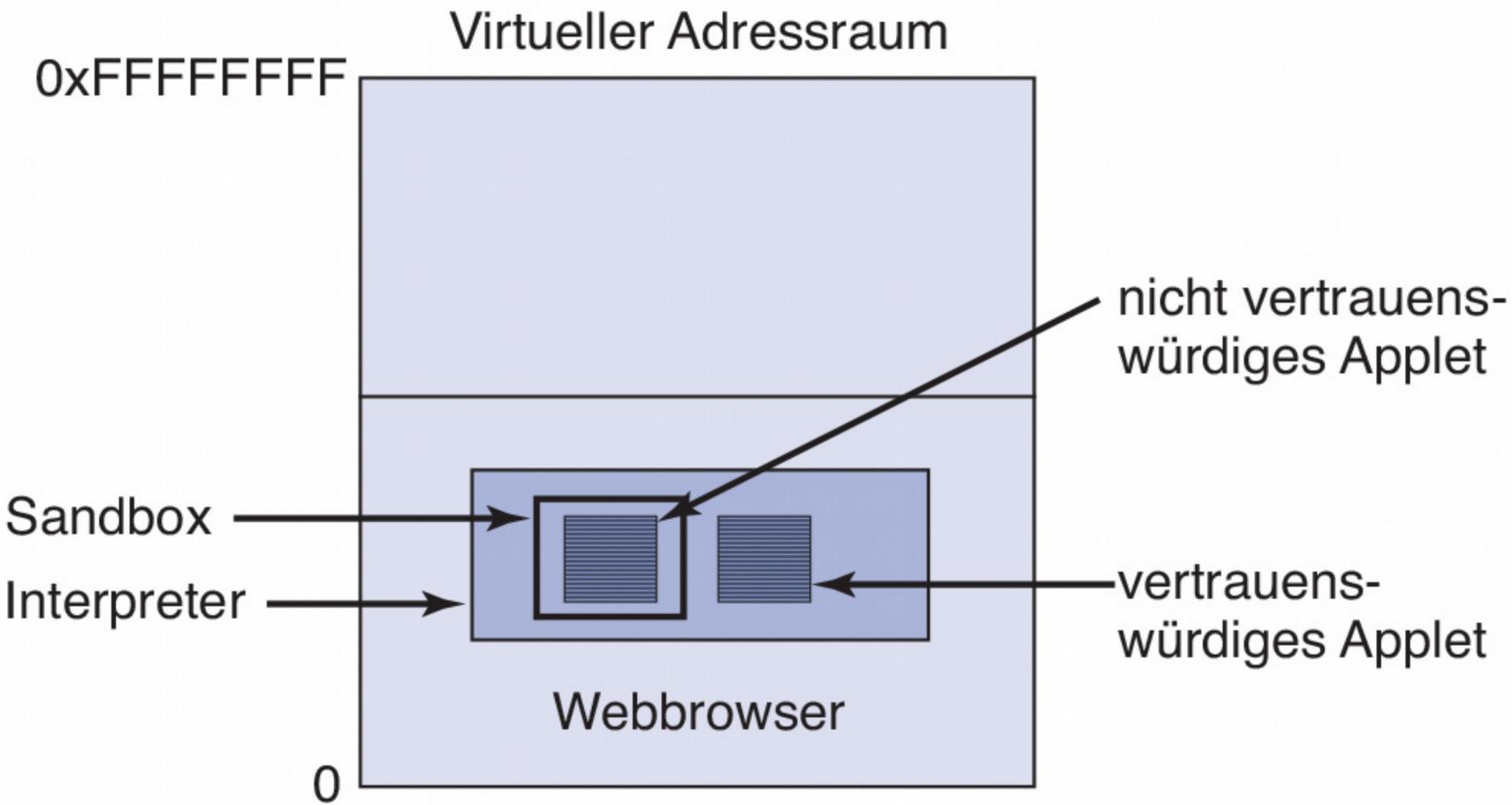
Abbildung 9.37: (a) Ein Programm. (b) Systemaufrufgraph für (a).

# Kapselung von mobilem Code



**Abbildung 9.38:** (a) Speicher, der in 16-MB-Sandboxen aufgeteilt ist. (b) Eine Möglichkeit, um die Zulässigkeit einer Anweisung zu testen.

# Interpretation



**Abbildung 9.39:** Applets können von einem Webbrowser interpretiert werden.

# Java Sicherheit (1)

**Überprüfungen von Applets schließen ein:**

1. Versuchen Applet Zeiger zu fälschen?
2. Verstößt es gegen Zugriffsbeschränkungen für Mitglieder der privaten Klasse?
3. Wird versucht, die Variable eines Typs als eine andere zu verwenden?
4. Führt es Stapelüberläufe oder -unterläufe aus?
5. Konvertiert es illegal Variablen eines Typs in einen anderen?

# Java Sicherheit (1)

**Überprüfungen von Applets schließen ein:**

1. Versuchen Applet Zeiger zu fälschen?
2. Verstößt es gegen Zugriffsbeschränkungen für Mitglieder der privaten Klasse?
3. Wird versucht, die Variable eines Typs als eine andere zu verwenden?
4. Führt es Stapelüberläufe oder -unterläufe aus?
5. Konvertiert es illegal Variablen eines Typs in einen anderen?

# Java Sicherheit (2)

URL	Signierer	Objekt	Aktion
www.taxprep.com	TaxPrep	/usr/susanne/1040.xls	Lesen
*		/usr/tmp/*	Lesen, Schreiben
www.microsoft.com	Microsoft	/usr/susanne/Office/-	Lesen, Schreiben, Löschen

**Abbildung 9.40:** Einige Beispiele dafür, welcher Schutz im JDK 1.2 spezifiziert werden kann.