

# Kapitel 2

## Grundlagen

Für das bessere Verständnis der folgenden Kapitel werden zunächst einige grundlegende Aspekte beim Design von Betriebssystemen besprochen. Ebenfalls wird kurz das Konzept der Zero-Knowledge-Beweise und ein Integritätsmodell vorgestellt. Kenntnisse über kryptographische Grundlagen wie symmetrische und asymmetrische Verschlüsselung, Hashfunktionen sowie über Verfahren zur digitalen Signatur und *Public-Key-Infrastrukturen* (PKI) werden vorausgesetzt. Einen guten Einstieg in diese Thematik bietet ein Grundlagenbuch<sup>1</sup> zum Thema Kryptographie.

### 2.1 CPU-Ringe

Die Sicherheit eines Computersystems hängt nicht nur von der Sicherheit der darauf laufenden Software, sondern auch von den von der Hardware bereitgestellten Schutzfunktionen ab. Ein wichtiges Beispiel einer solchen Schutzfunktion sind die Privilegierungsstufen einer CPU. Intel 80286-kompatible<sup>2</sup> Prozessoren unterscheiden vier Privilegierungsstufen, auch Ringe genannt: *Ring 0*, *1*, *2* und *3*. Dabei stellt *Ring 0*, genannt „*supervisor mode*“ oder auch „*kernel mode*“, die höchste Privilegierungsstufe dar, die bis zur *Stufe 3* immer weiter eingeschränkt wird. Die folgende Abb. 2.1 verdeutlicht das Konzept der Ringe.

Der volle Umfang des Befehlssatzes der CPU wird nur Prozessen zur Verfügung gestellt, welche im *Ring 0* ausgeführt werden. In allen anderen Ringen steht nur ein Subset der Funktionen zur Verfügung. So wird unter anderem sichergestellt, dass unprivilegierte Prozesse nicht direkt auf die Hardware zugreifen und sich auch nicht aus ihrer Privilegierungsstufe befreien können. Der Zugriff auf den Speicherbereich anderer Prozesse wird durch Speichervirtualisierung verhindert. Somit wird gewährleistet, dass Prozesse zum Beispiel im *Ring 3* in keinem Fall Prozesse im *Ring 0* oder auch andere Prozesse im *Ring 3* beeinflussen können. Übergänge von

---

<sup>1</sup> [Schmeh2001]

<sup>2</sup> Intel 80286 ist Mikroprozessor der Firma Intel aus dem Jahre 1982.

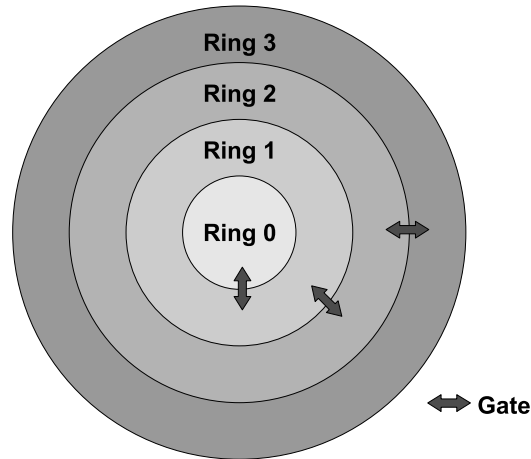


**Abb. 2.1** CPU-Ringe

einer unprivilegierten Stufe in eine Ebene mit mehr Umfang werden als *Gate* bezeichnet. Sie stellen die sichere Kommunikation zwischen Prozessen in unterschiedlichen Privilegierungsstufen zur Verfügung. Die dabei ausgelösten Interrupts führen in der Regel zu einem so genannten Kontextwechsel (*context switch*). Dabei wird der Kontext (im Wesentlichen die Prozessor-Register) des aktuellen Prozesses gesichert und der Kontext des neuen restauriert bzw. erstellt.

Schon beim Design der Privilegierungsstufen einer CPU wurde definiert, welche Klasse von Prozessen in welcher Stufe ausgeführt werden soll um die Integrität des Betriebssystems zu gewährleisten. Die nachfolgende Abbildung veranschaulicht diese Aufteilung:

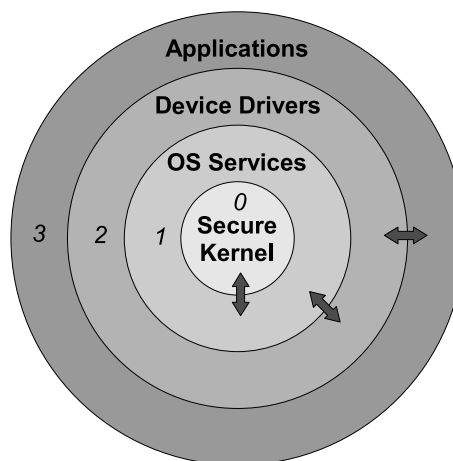


**Abb. 2.2** Verwendung der CPU-Ringe

So sollten im *Ring 0* nur die Grundfunktionen eines Betriebssystems (z. B. Speicher-verwaltung, Scheduling) sowie ein Zugriffskontrollsystem (*Access Control*) ausgeführt werden. Die rechtlichen Dienste des Betriebssystems sowie die Treiber für die Hardware des Systems sollen nur mit eingeschränkten Rechten in den *Ringen 1* und *2* laufen. Anwendungen, mit denen der Benutzer direkt interagiert, erhalten die wenigsten Rechte und kommen im *Ring 3* zur Ausführung. Eine Umsetzung dieser Aufteilung ist häufig ein Aspekt bei der Implementierung eines Betriebssystems in Form eines Mikrokernels<sup>3</sup>.

## 2.2 Verwendung der CPU-Ringe

Verbreitete Betriebssysteme wie Linux und Windows folgen jedoch nicht vollständig dem Konzept der Privilegierungsstufen, sondern verwenden nur zwei der vier Ringe. Sie nutzen *Ring 0* für den Kernel und die Gerätetreiber. *Ring 1* und *Ring 2* werden nicht verwendet. Die Anwendungen der Benutzer laufen wie vorgesehen im *Ring 3*. Für die Kommunikation zwischen Applikationen und dem Kernel stellt das Betriebssystem Systemaufrufe (*system calls*) zur Verfügung. Die folgende Abbildung beschreibt diese Aufteilung:

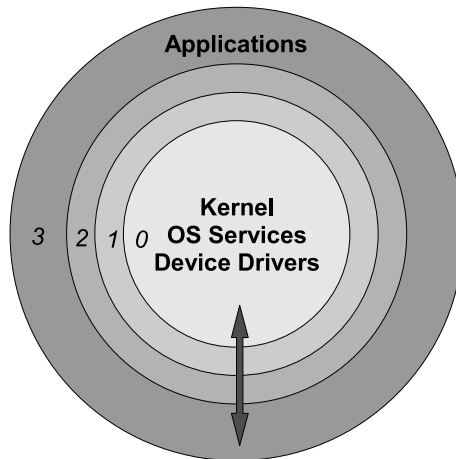


**Abb. 2.3** Windows und CPU-Ringe

Somit sind zwar Applikationen und das Betriebssystem voneinander getrennt, ein Zugriff auf den Speicherbereich des Kernels mittels einer unprivilegierten Applikation ist nicht möglich, jedoch werden sämtliche Treiber im selben Kontext wie der Kernel ausgeführt und erhalten somit vollständigen Zugriff auf dessen Ressourcen. Dies wirkt sich negativ auf die Sicherstellung der Integrität des Kernels aus,

<sup>3</sup> Ein Beispiel für einen Mikrokern ist beispielsweise der L4 Kernel.

da Treiber häufig aus nicht vertrauenswürdigen Quellen stammen und zusätzlich die Qualität des Codes nicht sichergestellt werden kann. So kann beispielsweise ein Druckertreiber unter Windows sämtliche Sicherheitsfunktionen des Betriebssystems untergraben.

Der Hauptgrund für die Wahl einer solchen Architektur (Monolithischer Kernel) ist die im Vergleich zu einem Mikrokern meist bessere Performance. Diese liegt hauptsächlich im Wegfall von Kontextwechseln bei Operationen innerhalb des Betriebssystemkerns begründet. Ein weiterer Grund dürfte die zum Zeitpunkt der Entwicklung der ersten Generationen heutiger Betriebssysteme geforderte Kompatibilität zu Prozessoren mit nur zwei Privilegierungsstufen sein.

## 2.3 Virtualisierung und die CPU-Ringe

Vor allem im Bereich der Virtualisierung von Servern wird eine zusätzliche Schicht zwischen der Hardware und den virtualisierten Betriebssystemen eingefügt. Diese als *Virtual Machine Monitor* (VMM) oder auch *Hypervisor* bezeichnete Schicht kontrolliert die Hardware des Systems und sorgt für die notwendige Isolation der einzelnen Betriebssysteme durch Aufteilung der vorhandenen Ressourcen. Um hierfür die notwendigen Rechte zu haben muss der VMM innerhalb der höchsten Privilegierungsstufe, sprich dem *Ring 0*, ausgeführt werden. Die Betriebssysteme müssten daher in den *Ring 1* verschoben werden. Da diese jedoch aufgrund ihrer Architektur auf den vollen Funktionsumfang der CPU angewiesen sind, erweist sich diese Verlagerung als nicht unproblematisch.

Es existieren mehrere Lösungen für das Problem. Die populärsten Ansätze sind dabei die Emulation einer vollständigen CPU mittels Software und die Modifikation des zu virtualisierenden Betriebssystems. Da das Nachbilden einer CPU zu erheblichen Geschwindigkeitseinbußen führt, ist es nur in wenigen Fällen ein gangbarer Ansatz. Wird das Betriebssystem für die Ausführung oberhalb eines VMM angepasst, spricht man in der Regel von *Paravirtualisierung*. Der VMM *Xen* ist der aktuell wohl populärste Vertreter dieser Gattung der Virtualisierungstechnik. Die hierbei notwendigen Anpassungen sind jedoch nur bei Betriebssystemen ohne Weiteres möglich, deren Quellcode verfügbar ist. Die Verwendung von Windows als Gastbetriebssystem ist damit nicht möglich. Um diese Einschränkung zu umgehen verändern VMM wie beispielsweise VMware ESX die Logik der virtualisierten Betriebssysteme zur Laufzeit. Dabei werden Aufrufe des Betriebssystems auf Ring-0-Funktionen der CPU beim Start des Systems im Speicher manipuliert und auf eine entsprechende Schnittstelle des VMM umgeleitet. Der VMM führt nach Kontrolle der Zugriffsrechte der VM die aufgerufene Operation im Auftrag der VM aus. Anschließend übergibt der VMM das Ausführungsrecht wieder zurück an die VM.

Eine einfachere Möglichkeit zur Virtualisierung von unmodifizierten Betriebssystemen bieten aktuelle CPUs der Hersteller Intel und AMD. Intel nennt seine

Technik zur Virtualisierung der CPU *Intel VT-x*<sup>4</sup>, AMD hat mit *AMD SVM*<sup>5</sup> eine vergleichbare Lösung im Programm. Beide Hersteller erweitern hierbei den Befehlssatz ihrer CPUs um weitere Befehle zur Ausführung eines Betriebssystems innerhalb einer isolierten Umgebung. Ebenfalls erweitert wurde hierfür das Konzept der CPU-Ringe. Die folgende Abbildung visualisiert die neue Aufteilung:

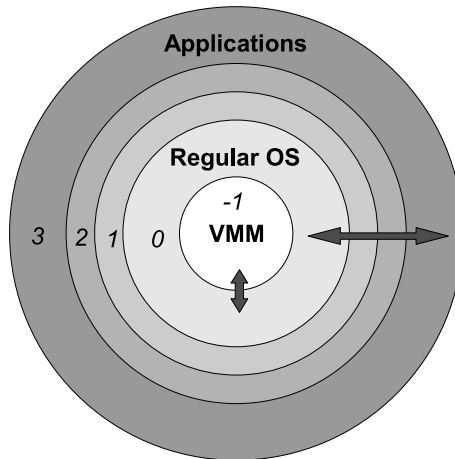


**Abb. 2.4** Zusätzlicher CPU-Ring

Der VMM wird dabei innerhalb des *Ringes -1* ausgeführt. Dieser Ring verfügt über sämtliche Rechte zur Ausführung von Operationen auf der CPU. Für jede VM erstellt die CPU auf Anfrage des VMM eine virtualisierte CPU mit den bekannten *Ringen 0–3*. Aus Sicht des Gastbetriebssystems befindet es sich innerhalb des *Ringes 0* und unterliegt somit scheinbar keinerlei Einschränkungen. Greift nun eines dieser Betriebssysteme auf eine Funktion des *Ringes 0* zu, erzeugt die CPU einen Interrupt und aktiviert den im *Ring -1* ausgeführten VMM. Dieser kann nun nach Prüfung der angeforderten Operation diese, transparent für das Gastbetriebssystem, in dessen Auftrag ausführen. Eine Anpassung des Betriebssystems ist daher nicht mehr notwendig.

Aus Sicht der VMs ist die Existenz eines VMM in der Regel nicht mehr zu erkennen. Obwohl durchaus gewünscht, kann dieser Umstand auch von Schadsoftware missbraucht werden. So verwenden beispielsweise so genannte Virtualisierungs-Rootkits diese Funktion um ein Betriebssystem zur Laufzeit in eine virtuelle Maschine zu verschieben<sup>6</sup>. Die Rolle des VMM übernimmt hierbei das Rootkit selbst. Daher wird empfohlen, den Zugriff auf diese Befehlssatzerweiterung bei Nichtverwendung durch das System im BIOS zu deaktivieren.

<sup>4</sup> Auch als Intel „Vanderpool“ bekannt

<sup>5</sup> AMD *Secure Virtual Machine*, auch als „Pacifica“ bekannt

<sup>6</sup> Demonstriert wurde diese Möglichkeit u. a. durch „BluePill“ [URL28].

## 2.4 Zero-Knowledge-Beweis<sup>7</sup>

Das Konzept der *Zero-Knowledge-Proofs* wurde 1985 von Goldwasser, Micali und Rackoff eingeführt. Zero-Knowledge-Beweissysteme sind spezielle interaktive Beweissysteme. Bei interaktiven Beweissystemen wird ein Protokoll zwischen Beweiser  $A$  und Verifizierer  $B$  durchgeführt.  $A$  kennt ein Geheimnis,  $B$  nicht. In dem Protokoll überzeugt der Beweiser den Verifizierer davon, dass er das Geheimnis kennt. Ein Betrüger kann das nicht. (Diese Eigenschaften haben auch die Public-Key-Verfahren zur Authentifizierung.)

Hinzu kommt bei Zero-Knowledge-Beweissystemen folgende Forderung:

- Der Verifizierer erfährt nur, dass der Beweiser das Geheimnis kennt, aber nichts weiter (gleichgültig, welche Strategie er anwendet).

Wir geben ein Beispiel, das so genannte Fiat-Shamir-Verfahren (1986):

$A$ , die Beweiserin, wählt  $p, q$ , große Primzahlen, bildet  $n = p \cdot q$ . Sie wählt  $s \in \{1, \dots, n-1\}$  zufällig mit  $\text{ggT}(s, n) = 1$ . Sie berechnet  $v = s^2 \bmod n$ . ( $v, n$ ) ist öffentlich,  $s$  geheim. (Beachte:  $s$  ist nur dann aus  $n$  zu bestimmen, wenn man die Faktorisierung von  $n$  kennt.)  $A$  beweist  $B$ , dass sie eine Quadratwurzel von  $v$  modulo  $n$  kennt.

Protokoll:

1.  $A$  wählt zufällig und gleichverteilt  $r \in \{1, 2, \dots, n-1\}$ . Sie berechnet  $x = r^2 \bmod n$ .
2.  $A$  sendet  $x$  an Verifizierer  $B$ .
3.  $B$  wählt zufällig mit Wahrscheinlichkeit  $1/2$  ein  $e \in \{0, 1\}$  und sendet  $e$  an  $A$  (Challenge).
4. Erhält  $A$  den Wert  $e = 0$ , so schickt sie  $r$  (die Wurzel von  $x$ ) zurück.  
Erhält  $A$  den Wert  $e = 1$ , so schickt sie  $y = rs \bmod n$  (also Wurzel von  $xv$ ) an  $B$  (Response).
5.  $B$  verifiziert, dass  $r^2 \equiv x \bmod n$  (im Fall  $e = 0$ ) bzw.  $y^2 \equiv xv \bmod n$  (im Fall  $e = 1$ ).

Analyse des Verfahrens:

- a) Kennt  $A$  das Geheimnis, so kann sie jede der Fragen von  $B$  beantworten.
- b) Ein Betrüger kann aus  $(n, v)$  das Geheimnis nicht entschlüsseln (Ziehen von Quadratwurzeln modulo  $n$  genauso schwierig wie Faktorisierung von  $n$ ).
- c) Was passiert, wenn ein Betrüger trotzdem das Protokoll mit  $B$  durchführt, um sich als  $A$  auszuweisen? Er kann nicht beide möglichen Fragen beantworten, denn kennt er  $r$  und  $rs$ , so auch  $s$ . Er hat also nur zwei Möglichkeiten:

---

<sup>7</sup> Aus [Haug2002]

- Er wählt  $r$  wie vorgesehen und sendet  $x = r^2 \bmod n$ .  
Sendet  $B \ e = 0$ , so antwortet er  $r$ , korrekt.  
Sendet  $B \ e = 1$ , so kann er nicht antworten.
- Er wählt  $y$  vor und sendet  $x = y^2 v^{-1} \bmod n$ .  
Bei  $e = 1$  kann er jetzt korrekt antworten.  
Bei  $e = 0$  müsste er die Wurzel von  $v$  bestimmen. Das kann er nicht. Da er  $x$  angeben muss, bevor er  $e$  erhält, kann er mit einer Wahrscheinlichkeit von  $1/2$  richtig antworten. Nach  $k$  Wiederholungen ist die Wahrscheinlichkeit  $1/2^k$ , dass er immer richtig antwortet. Bei immer richtiger Antwort ist  $B$  mit einer Wahrscheinlichkeit von  $1 - 1/2^k$  überzeugt, dass  $A$  tatsächlich das Geheimnis  $s$  kennt.

- d) Man kann zeigen, dass  $B$  aus diesem Beweissystem nichts über  $s$  erfährt, außer dass  $A$   $s$  kennt. (Gleichgültig, ob er sich dem Protokoll entsprechend verhält ( $e$  zufällig wählen) oder eine andere Strategie verfolgt.) (Zero-Knowledge-Eigenschaft)

Man formalisiert dies dadurch, dass man fordert:

Ist die Behauptung des Beweises wahr, so kann die Interaktion zwischen  $A$  und  $B$  in polynomialer Zeit (von  $B$ ) simuliert werden, ohne dass eine Interaktion mit dem Beweiser stattfindet.

## 2.5 Clark-Wilson-Integritätsmodell<sup>8</sup>

Das *Clark-Wilson-Modell* ist ein Zugriffsmodell, das vor allem vor Manipulationen schützt und somit die Integrität adressiert. Innerhalb des *Clark-Wilson-Modelles* können Akteure (Benutzer, Prozesse) nie direkt auf ein Objekt (Datei, Gerät) zugreifen, sondern müssen zwingend mittels einer Applikation zugreifen. Das Konzept ist nicht analog zu dem des Referenzmonitors. Der Referenzmonitor fällt die Entscheidung, welche Applikation für den Akteur ausführbar ist. Der Zugriff über die Applikation erzeugt eine zusätzliche Schicht, in welcher applikatorische Einschränkungen implementiert werden können.

Das Modell beschreibt mittels Einhaltungs- (enforcement) und Zertifizierungsregeln (certification) die informationstechnischen Daten und Prozesse. Diese Regeln bilden die Basis zur Sicherstellung der Integrität eines Systems. Das Modell basiert immer auf einer in sich geschlossenen Transaktion.

- Eine gültige Transaktion ist eine Abfolge von Operationen, welche das System von einem Zustand in den nächsten Zustand bringt. Die Transaktion muss immer atomar sein. Dies bedeutet, dass die Zustandsänderung nur erfolgt, wenn die Transaktion keine Fehler aufweist.
- Im Clark-Wilson-Modell wird die Integrität über die Transaktionskontrolle sichergestellt.

---

<sup>8</sup> Aus [Clark1987]

- Das Prinzip der Gewaltentrennung (seperation of duty) erfordert, dass der Prozess, der eine Transaktion kontrolliert, unabhängig von dem transaktionsausführenden Prozess ist.

Dazu werden folgende Konstrukte verwendet:

- Constrained Data Item (CDI)  
Daten, die einen sicheren Zustand im System repräsentieren.
- Unconstrained Data Item (UDI)  
Daten, die noch nicht gesichert sind, z. B. Benutzer-Eingaben.
- Integrity Verification Procedure (IVP)  
Stellt sicher, dass alle CDI in einem System einen gültigen Zustand besitzen.
- Transformational Procedures (Tps)  
Die Transaktion nimmt ein CDI oder UDI entgegen und nimmt die Transaktion auf den Daten vor. Die Transaktion muss sicherstellen, dass immer wenn sie ein UDI entgegennimmt, ein sicheres CDI produziert wird; dies wird über eine Zertifizierung gelöst.

Die Regeln des Integritätsmodells sind dabei wie folgt:

- C1: Wenn ein IVP ausgeführt wird, muss es sich immer in einem gültigen Zustand befinden.
- C2: Alle Daten (CDIs) müssen von einem TP, von einem gültigen Zustand in den nächsten übergeführt werden.
- E1: Das System muss eine Liste von zertifizierten Verknüpfungen führen, welche sicherstellen, dass für ein CDI nur zertifizierte TP erlaubt werden.
- E2: Das System muss jedem TP und CDI einen Benutzer zuweisen. Der Tupel {Benutzer,TP,CDI} wird auch als allowed relations bezeichnet.
- E3: Jeder Benutzer, welcher auf ein TP zugreift, muss auf Anfragebasis (request) authentifiziert werden.
- C4: Jeder TP muss ein Transaktionsjournal schreiben, welches erlaubt, alle Operationen zu rekonstruieren.
- C5: Jeder TP, welcher ein UDI als Eingangsdaten verarbeitet, darf nur Operationen auf diesem UDI ausführen. Kann er das UDI nicht zu einem CDI konvertieren, muss er die Verarbeitung abbrechen.
- E4: Nur der Zertifizierer (Hersteller oder Besitzer) des Systems darf Einträge in den Verknüpfungen {TP,CDI} vornehmen.



<http://www.springer.com/978-3-540-76409-0>

Trusted Computing Systeme  
Konzepte und Anforderungen

Müller, Th.

2008, XV, 160 S., Hardcover

ISBN: 978-3-540-76409-0