

Secure Computing mit L⁴Linux

Umsetzung einer *Secure Computing Base* für Linux unter Ausnutzung der L4- μ Kern-Technologie

18. Oktober 2003

Oliver Stecklina

ost@stecklina-net.de

Was uns in der nächsten Stunde erwartet

eine einführende Motivation

die Probleme der Systemarchitektur von Linux

- monolithische Systemkerne
- ladbare Kernel-Module

Linux als Applikation des L4- μ Kern

- der Mikrokern-Ansatz
- die Basisarchitektur
- Performance der L⁴Linux-Systemarchitektur

Aufteilen des L⁴Linux-Servers

- Auslagerung von LKMs in separate Adressräume
- μ SINA - μ Kern-basierte Systemarchitektur für sichere Systemkomponenten

Präambel

Motivation

Sobig & Co

- eigentlich eine DoS-Attacke auf den Web-Server von Microsoft
 - ▷ wegen fehlerhaftem Programmcode fehlgeschlagen
- mehrere Millionen Windows-Rechner betroffen
 - ▷ Wirtschaft beklagt Verluste in Millionenhöhe
 - ▷ 30 Millionen US-Bürger hatten bis zu 2 Tage lang keinen Strom

Buffer-Overflow im Programmcode der openssh

- nahezu alle öffentlichen Zugänge basieren auf der openssh
- Patches und Exploits + RootKits findet man im Internet
 - Wer sich nicht informiert wird gehackt

”Bei steigender Abhängigkeit von der IT wird der Schutz der Menschen vor deren Ausfall nötig.”

- Zitat: Dr. Udo Helmbrecht, Präsident des BSI

Warum Linux?

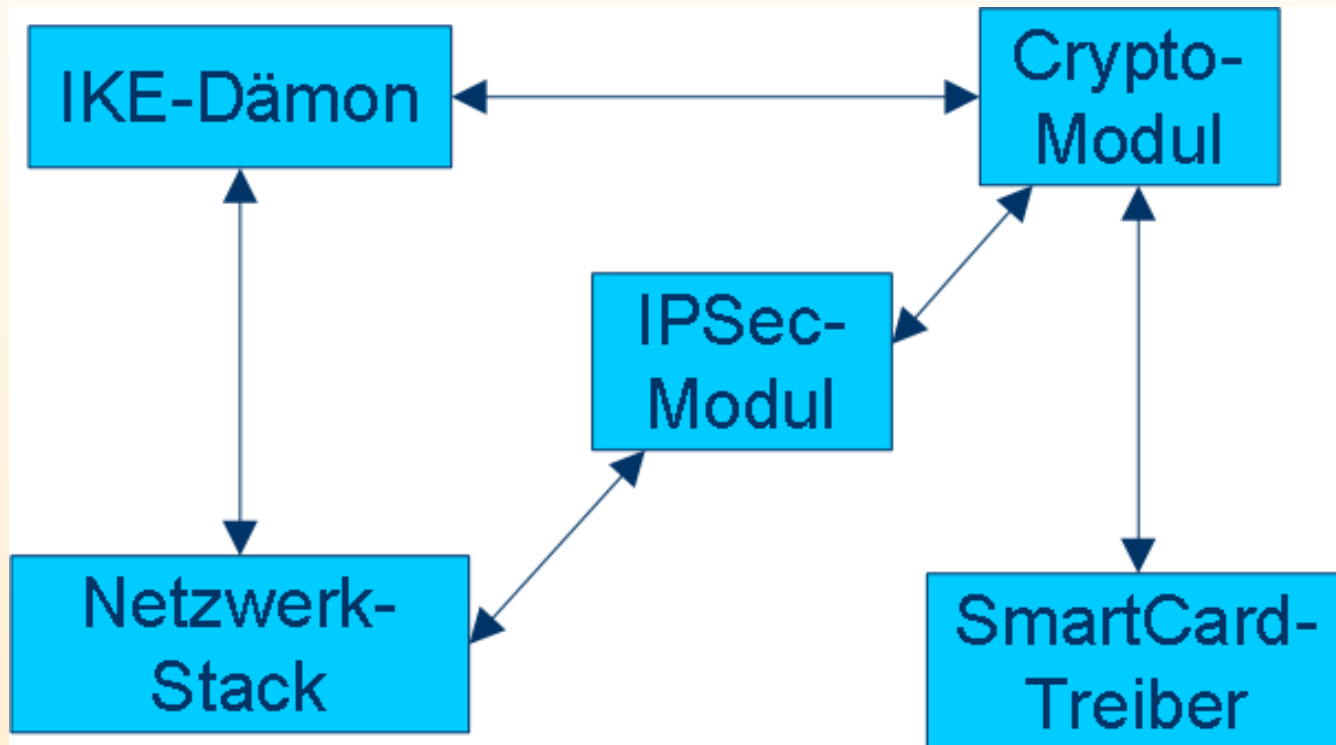
Linux ist frei

- jeder kann es kostenlos aus dem Internet herunterladen
- der Programmcode ist offen
 - ▷ Anpassung an die eigenen Bedürfnisse ist möglich
 - ▷ Fehlersuche erwünscht

für kein Unix-Derivat existieren mehr Exploits

- nahezu jeder kann Linux installieren
 - ▷ moderne Distributionen sind unkompliziert
 - ▷ testen der Exploits am heimischen Rechner
- über 20 Prozent der Server laufen mit Linux
 - ▷ das Einspielen von Patches kostet Zeit und zerstört die uptime
 - ▷ genügend Angriffsziele sind vorhanden

Anwendungsbeispiel



Die Probleme der Systemarchitektur von Linux

Die Systemarchitektur von Linux

monolithischer Systemkern

- alle Systemkomponenten sind Bestandteil des Kerns
 - ▷ Netzwerk-Stack, Dateisystem, Gerätetreiber, ...
- keine klare Trennung zwischen den Komponenten
 - ▷ Verwendung eines einzigen Adressraumes
 - ▷ keine Zugriffsbeschränkungen für Systemfunktionen

Allmacht der Systemkerns

- Ausführung der Systemfunktionen im SuperVisor-Modus der CPU
 - ▷ keine zusätzlichen Schutzmechanismen des Kerns vorhanden
 - ▷ globale Zugriffsrechte innerhalb des Systems
- Entziehen der Rechte ist nicht möglich

Konzept der ladbaren Kernel-Module

Konzept zur dynamischen Erweiterung des Systemkerns

- Auslagerung von Systemkomponenten
 - ▷ selten genutzte Gerätetreiber oder Dateisysteme
 - ▷ Minimalisierung des statischen Kernels
- Laden und Entfernen je nach Bedarf
- keine sicherheitsrelevanten Kontrollen

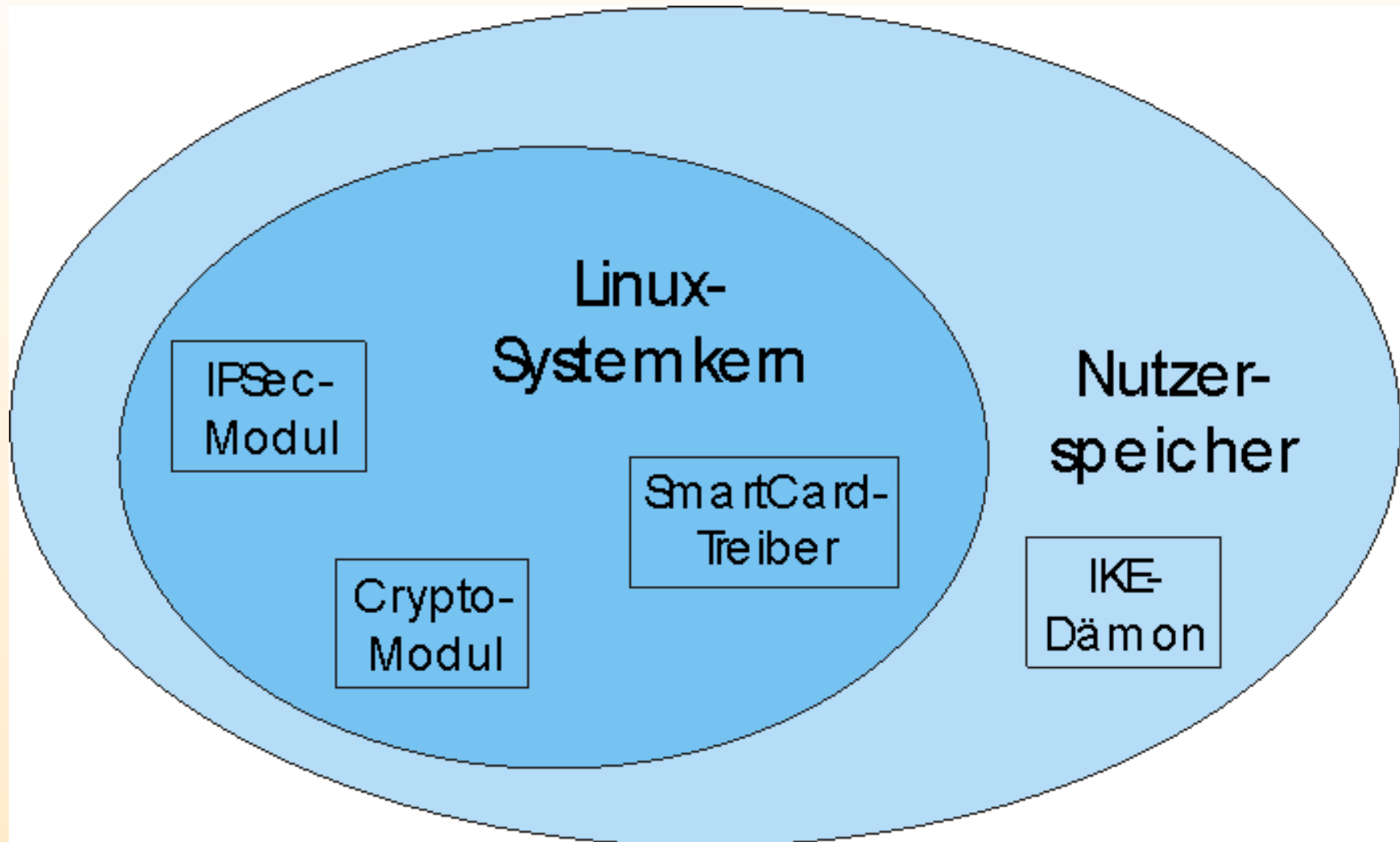
gleichberechtigte Integration in den Kern

- keine Unterscheidung zwischen den Funktionen
- keine Beschränkung bzgl. des Zugriffs auf Systemkomponenten

Grundlage moderner Rootkits

- direkte Manipulation des Systemspeichers
- Installation von Hintertüren

Beispiel im monolithischen System



Mehr Sicherheit durch minimalisierte Systeme

oftmals auch als Härten des Systems bezeichnet

- Deaktivieren und Deinstallieren von unnötigen Diensten
- Verkleinern des Systemkerns
 - ▷ Verzicht auf unwichtige Komponenten
 - ▷ Bereitstellen einer Basisfunktionalität
- Minimierung der Zugänge

Realisierung einer möglichst statischen Konfiguration

- LKMs werden gar nicht unterstützt
- dynamisches Einbinden von externen Komponenten unterbinden

Minimieren der Release-Wechsel

- neue Kernel-Releases sind nur selten notwendig
- verifizieren neuer Software-Releases

Linux als Applikation des L4- μ Kerns

Die Systemarchitektur von L⁴Linux

Unterteilung des Gesamtsystems

- der μ Kern
 - ▷ minimalisierte Systemkomponente
 - ▷ wird im SuperVisor-Modus der CPU ausgeführt
- Tasks des μ -Kerns
 - ▷ Dienste des μ Kerns
 - ▷ werden im Nutzermodus der CPU ausgeführt

Linux als Task des μ Kerns

- Ausführung im Nutzermodus
- kann keine privilegierten Befehle des Prozessors ausführen

μ Kern-Ansatz

Minimalisierung des Systemkerns

- beschränkt auf fundamentale Komponenten
 - ▷ Verwaltung von Adressräumen
 - ▷ Thread-Management
 - ▷ Interprozesskommunikation
- einzige privilegierte Komponente des Systems

übrige Funktionalität obliegt den μ Kern-Servern

- laufen in einem eigenen Adressraum
- von Server angebotene Dienste
 - ▷ Interruptbehandlung (Gerätetreiber)
 - ▷ Netzwerk-Protokoll-Stack
 - ▷ Dateisystem

μ Kerne der ersten Generation

Entwicklung im TopDown-Verfahren

- verringern der Komplexität vorhandener monolithischer Systemkerne
- Ergebnis dementsprechend unflexibel und ineffizient
 - ▷ oftmals gebunden an die ursprüngliche Architektur
 - ▷ kein angepasstes Design des neuen Kerns

zum Beispiel MACH:

- Ausgangspunkt BSD-Unix
- Kernaufgaben in Nutzermodus ausgelagert
 - ▷ Schnittstellen-Design ungeeignet für häufige Adressraumwechsel
 - ▷ Re-Integration von Systemdiensten zur Leistungssteigerung
- heutige Einsatzgebiete
 - ▷ MacOS X, GNU/Hurd

μ Kerne der zweiten Generation

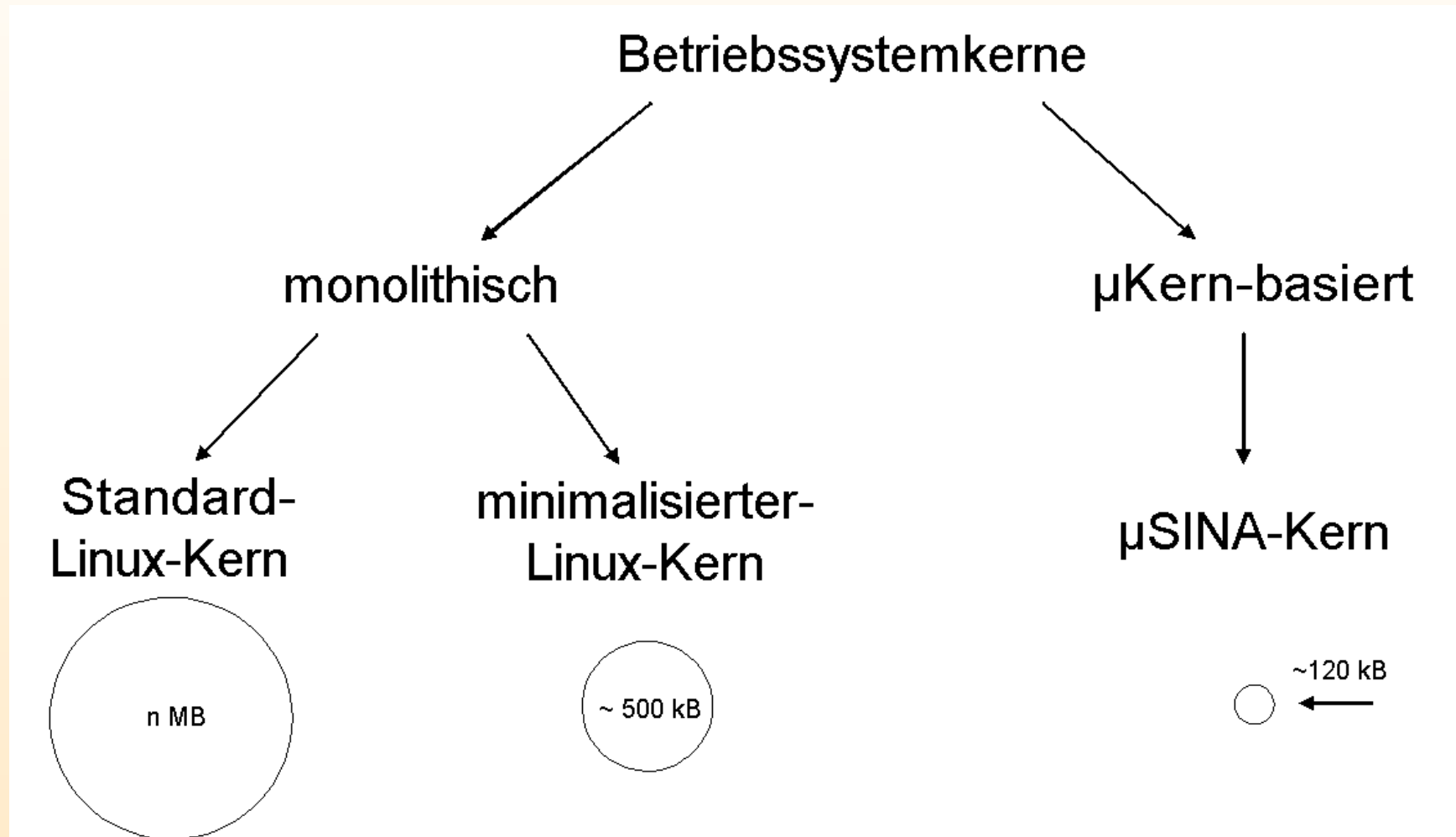
Entwicklung im BottomUp-Verfahren

- Ziel ist ein kleiner und effizienter μ Kern
- Beschränkung auf wesentliche Komponenten

L4-Familie (Jochen Liedtke)

- enthält nur unbedingt erforderliche Bestandteile
 - ▷ lediglich sieben Systemrufe
 - ▷ leistungsfähige IPC-Schnittstelle
- Fiasco
 - ▷ μ Kern-Implementierung der TU-Dresden
 - ▷ vollständig in C++ implementiert

Teile der Trusted Computing Base



L⁴Linux ein Server des μ Kerns

Portierung des Linux-Systemkerns

- erste Portierung für Kernel der Version 2.0.x
- aktuell Kernel 2.2.25 und 2.4.21

Operationen des Kerns werden im Nutzermodus ausgeführt

- Zustellung von Interrupts erfolgt mittels Nachrichten
 - ▷ μ Kern nimmt sie entgegen
 - ▷ Behandlung erfolgt immer durch eine Server-Task
- I/O-Bereiche werden in den Adressraum des L⁴Linux-Servers eingeblendet

Nutzerprozesse sind eigene L4-Tasks

- Scheduling durch den μ Kern
- Speicherverwaltung obliegt dem L⁴Linux-Server
 - ▷ Server enthält Pager-Thread für Nutzerprozesse

Adressraumverwaltung mittels Pager

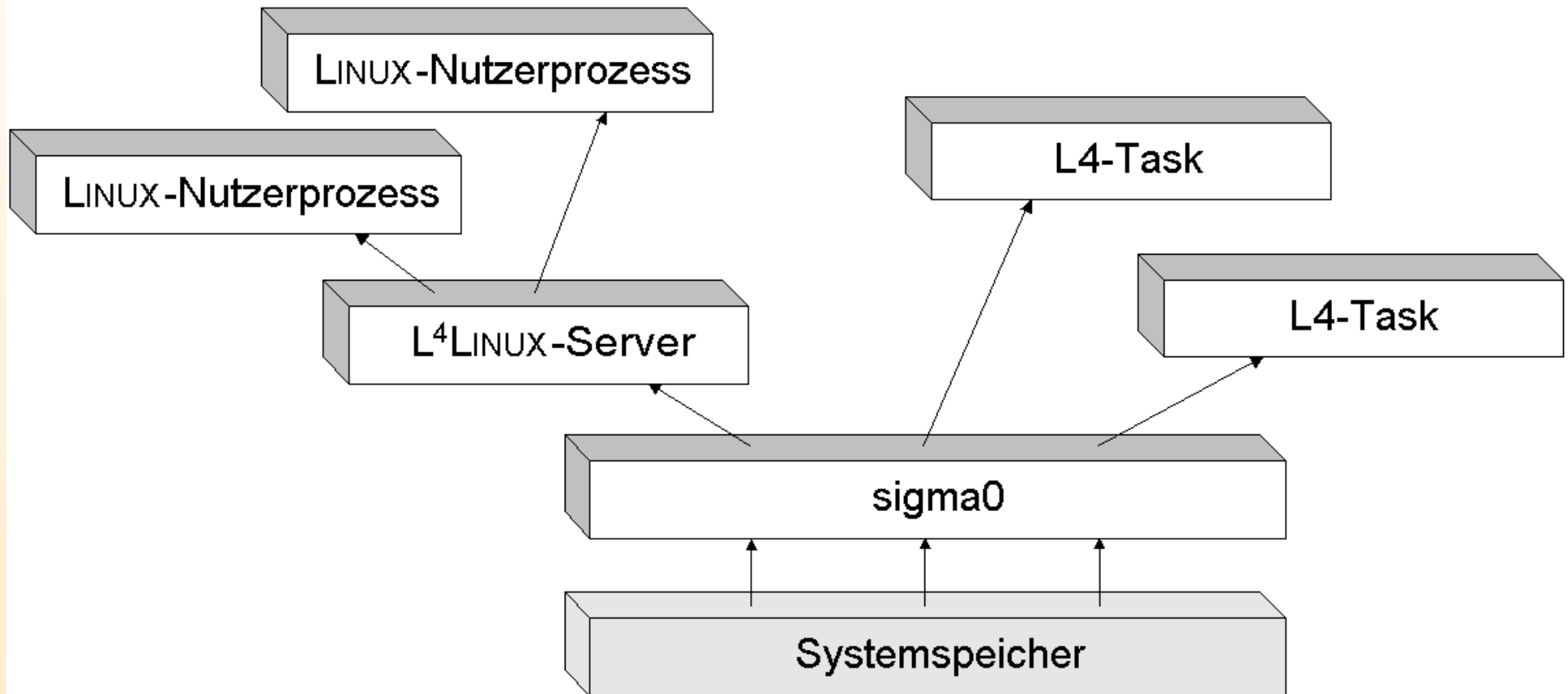
Adressraum ist der virtuelle Speicher einer Task

- physische Seiten werden in den Adressraum eingeblendet
- Behandlung von Zugriffsverletzungen erfolgt mittels Pagen
 - ▷ kann innerhalb und außerhalb des Adressraumes laufen
 - ▷ versucht die Seite einzublenden

Adressräume können hierarchisch organisiert werden

- Adressraum eines Pagers kann von einem anderen Pager verwaltet werden
 - ▷ gemeinsame Nutzung mittels map-Operation
 - ▷ Delegation von Seiten mittels grant-Operation
- initialer Adressraum ist σ_0
 - ▷ enthält den gesamten physischen Speicher des Systems
 - ▷ wird von einer speziellen Task namens `sigma0` verwaltet

Speicherhierarchien in L⁴Linux



Performance eines L⁴Linux-Systems

Vergleich des getpid(...)-Systemrufs

System	Time	Cycles
Linux	1,68 μs	223
L ⁴ Linux	3,95 μs	526
MkLinux in-kernel	15,41 μs	2050

Testsystem: Pentium 133 MHz, L⁴Linux 2.0.21

Übersetzen des Linux-Systemkerns

Release	System	mm:ss
linux-2.2	nativ	5:34
	l4 (fiasco)	6:22
linux-2.4	nativ	5:37
	l4 (fiasco)	6:27

Testsystem: AMD Duron 1,3 GHz, 256 MB, ATA-Disk

Verlagerung des Problems

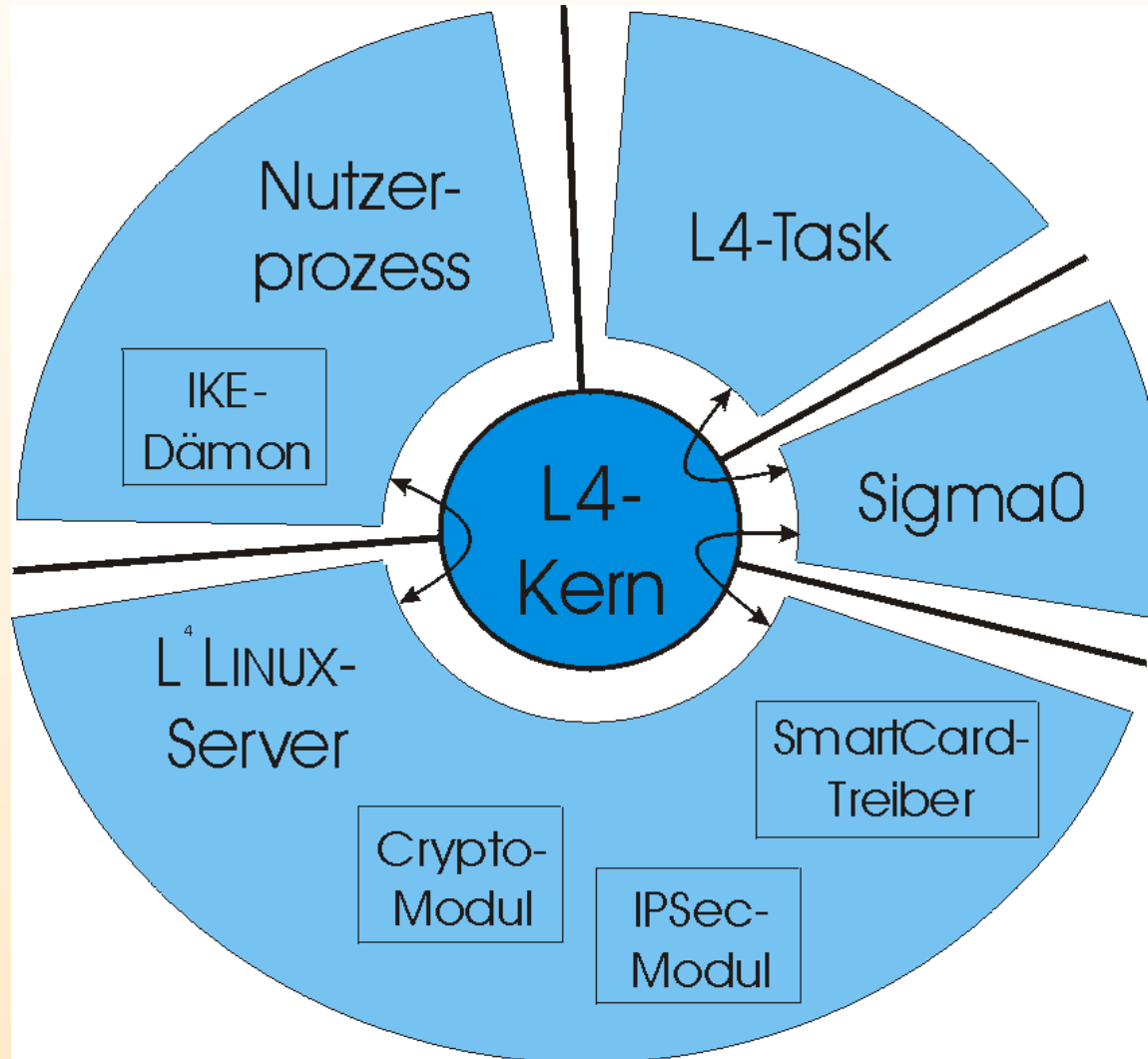
gesamter Funktionsumfang ist im L⁴Linux-Server implementiert

- die monolithische Struktur bleibt erhalten
- Module werden gleichberechtigt integriert
 - ▷ kein zusätzlicher Schutz vor LKM-Rootkits

eine Unterteilung des Speichers existiert nur bedingt

- L⁴Linux-Server verwaltet den gesamten Speicher
 - ▷ Speicher wird beim Systemstart angefordert und initialisiert
 - ▷ Nutzerspeicher obliegt seiner Kontrolle
- für Funktionen des Servers existieren keine Zugriffsbeschränkungen
 - ▷ befinden sich im gleichen Adressraum
 - ▷ haben Zugriff auf den gesamten Speicher des L⁴Linux-Servers

Beispiel in der L⁴Linux-Umgebung



Aufteilen des L⁴Linux-Servers

Interne Schnittstellen des Linux-Systemkerns

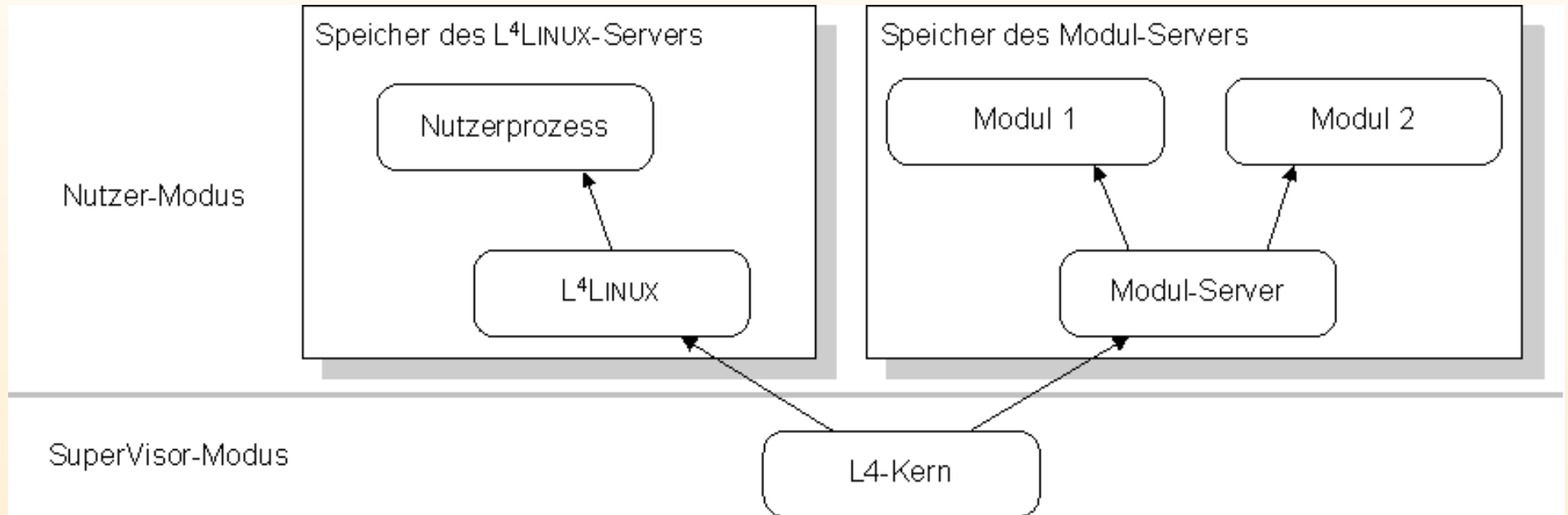
monolithischer Kern

- Schnittstellen zwischen integralen Bestandteilen fehlen
 - ▷ Netzwerk-Stack
 - ▷ Speichermanagement
 - ▷ ...
- Auslagerung ist nicht möglich

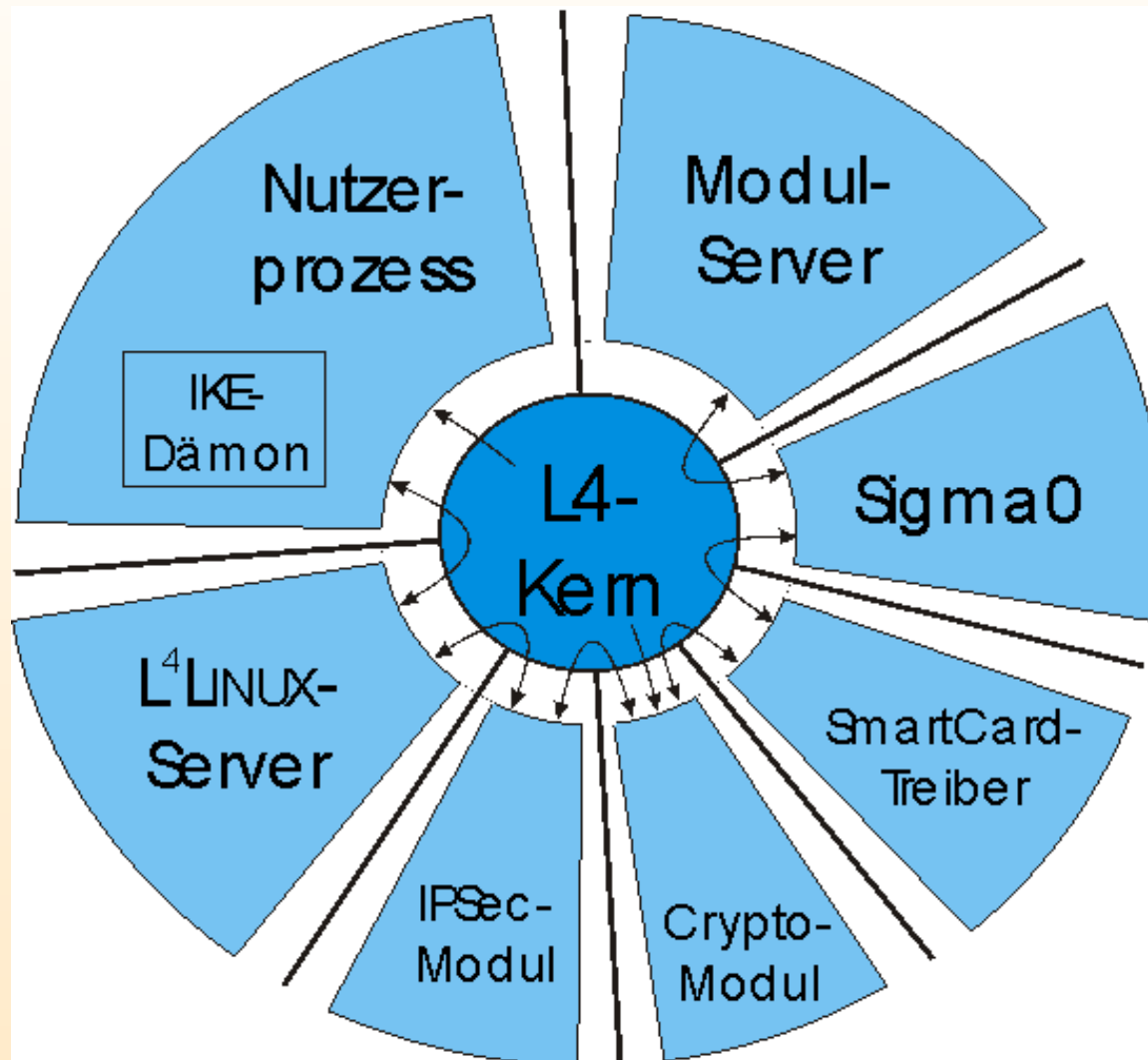
LKMs unterteilen den Kern in quasi unabhängige Komponenten

- Interaktion untereinander über eine wohldefinierte Schnittstelle
 - ▷ Zugriff auf exportierte (öffentliche) Symbole begrenzt
 - ▷ Modul-eigene Symbole müssen beim Kern angemeldet werden
- der Systemkern ist ohne LKMs lauffähig

Die Modul-Server-Architektur



Beispiel in der Modul-Server-Architektur



Zugriff auf öffentliche Symbole

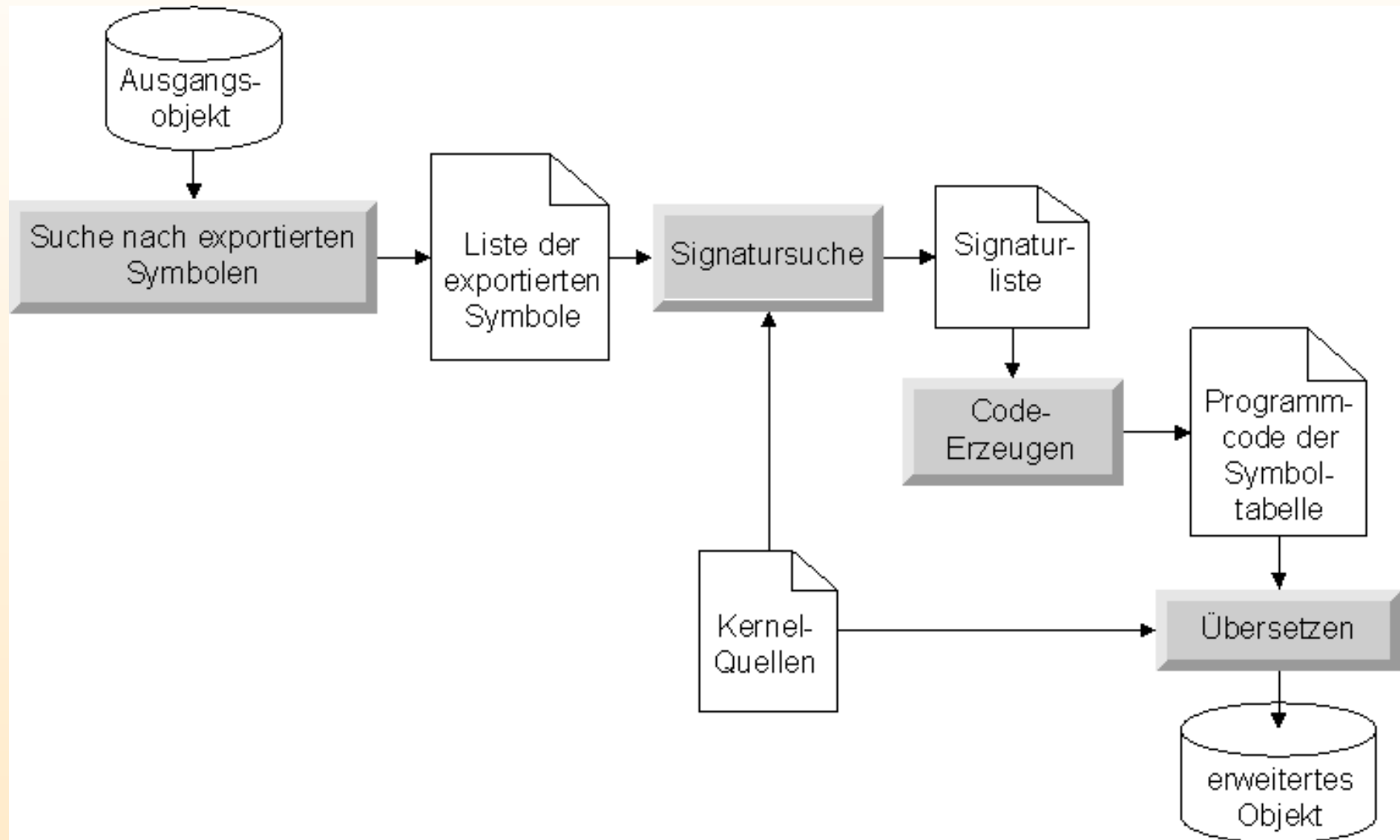
Zugriff auf exportierte Funktionen

- kein direkter Zugriff möglich
 - ▷ Funktionen sind in einem fremden Adressraum implementiert
 - ▷ Schutz der Implementierung
- Aufruf mittels Remote Procedure Calls (RPCs)
 - ▷ Adressierung über Funktions- und Modul-Identifizier
 - ▷ Module enthalten lediglich RPC-Stubs

Zugriff auf öffentliche Variablen

- Verwendung eines RPCs zu aufwendig
- Realisierung mittels eines gemeinsamen Speicherbereiches
 - ▷ Speicherseiten werden in die Adressräume der Module eingeblendet
 - ▷ Verwaltung obliegt dem Modul-Server

Erzeugen eines auslagerbaren Moduls



Zugriffskontrolle mittels einer Access Control List

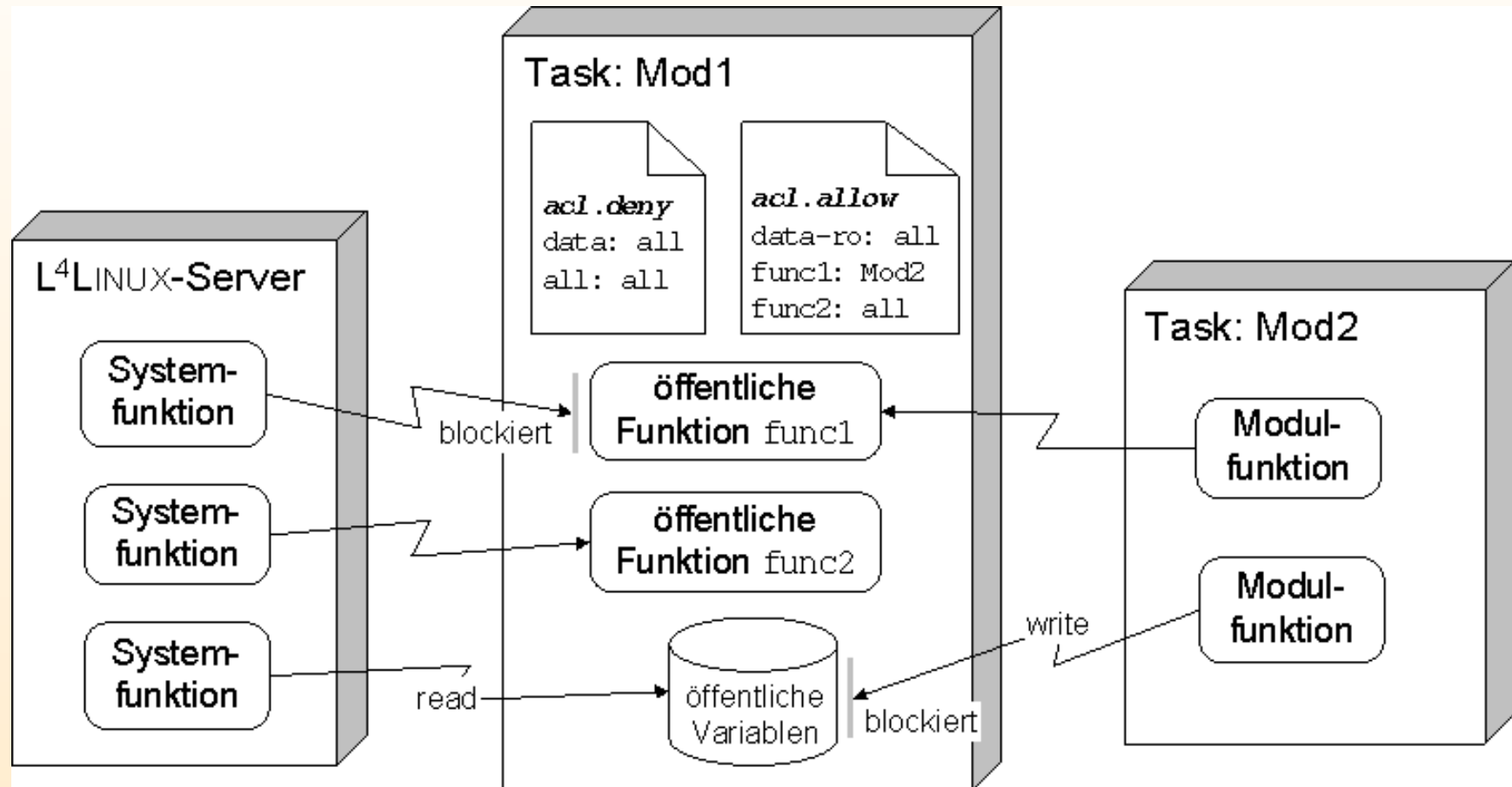
Einführung einer Access Control List (ACL)

- Funktionen können auf Task-Ebene unterschieden werden
 - ▷ Zugriffsregeln für einzelne Funktionen möglich
- Kontrolle der Zugriffe auf den öffentlichen Datenbereich
 - ▷ keine Unterscheidung zwischen einzelnen Variablen eines Moduls
 - ▷ öffentliche Variablen können lesend oder komplett freigeben werden

Einschränkung der Interaktion

- jedes Modul definiert seine eigene ACL
- Durchsetzung der Zugriffskontrolle erfolgt im Modul-Server
 - ▷ zur Laufzeit einzige statische Komponente
 - ▷ exportiert keine Datenbereiche

Zugriffskontrolle



Einsatzgebiete

Module mit hohem Sicherheitsbedarf

- Module zur Verschlüsselung von Daten
 - ▷ Sicherung des Schlüsselmaterials
 - ▷ Verbergen des Programmtextes
- Gerätetreiber
 - ▷ Geräte mit sicherheitskritischen Daten
 - ▷ SmartCard-Leser, Tastatur, ...

Auslagern von unsicheren Modulen

- neue bzw. wenig getestete Gerätetreiber
- Wrapper für closed Source Treiber

Testen von unbekannten und selbst entwickelten Modulen

- Überprüfung der Speicherzugriffe

μ Kern-basierte Systemarchitektur für sichere Systemkomponenten

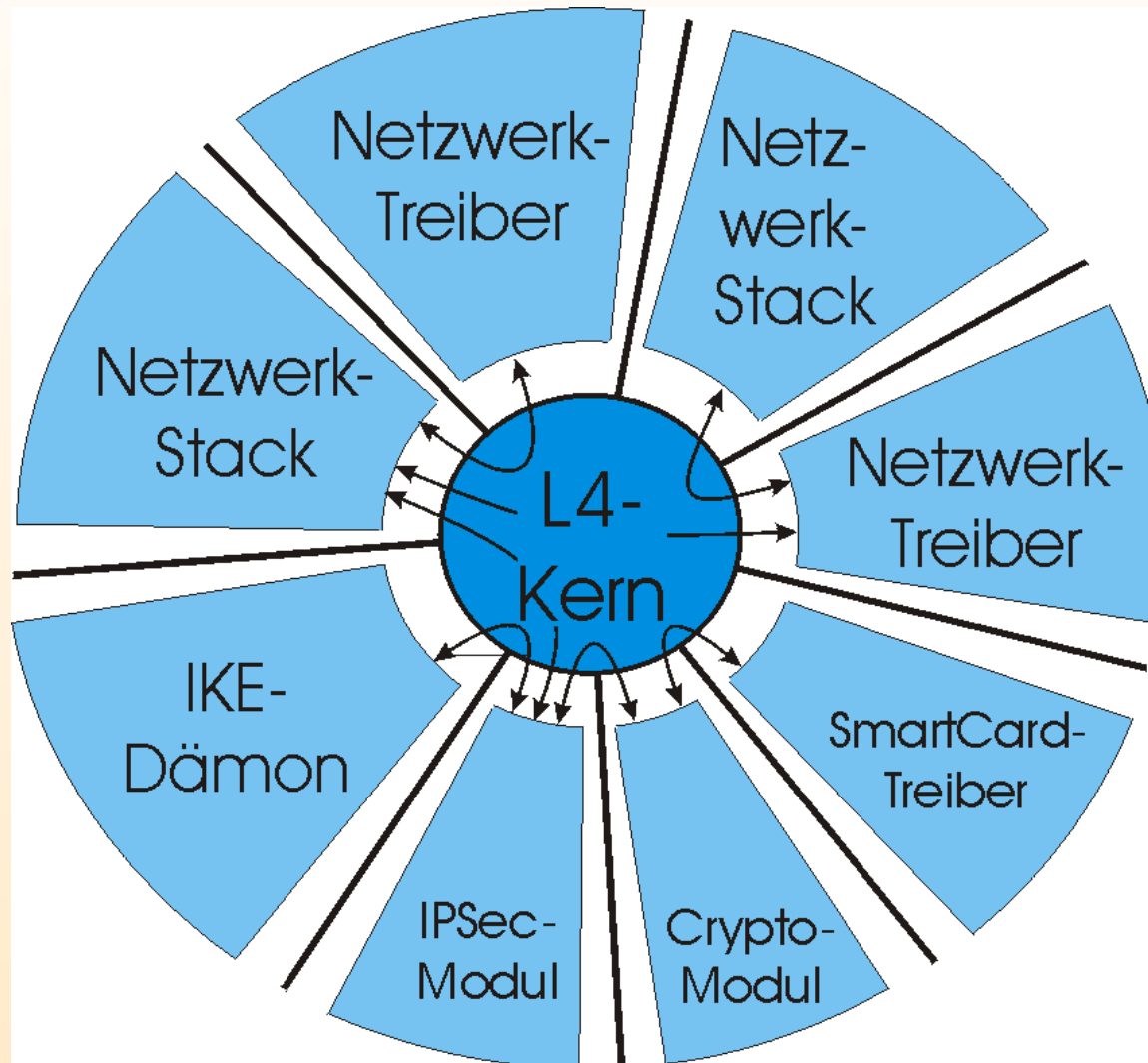
Umsetzung einer Sicherheitsanwendung auf einem μ Kern-basierten System

- Entwicklung eines μ Kern-basierten VPN-Gateways
- gemeinsame Entwicklung
 - ▷ des Instituts für Systemarchitektur der TU-Dresden und
 - ▷ der secunet Security Networks AG

Leistungsmerkmale

- Designansatz: *Divide et impera*
 - ▷ Identifizieren und Separierung von Systembestandteilen
 - ▷ Installation einer Kommunikationsmatrix
- Verkleinerung des Evaluationsgegenstandes
 - ▷ stricte Trennung von vertrauenswürdigen und offenen Komponenten
 - ▷ Evaluation der vertrauenswürdigen Bestandteile
- lokal begrenzte Auswirkung von Fehlern

Exemplarische Umsetzung eines μ SINA VPN-Gateways



Weitere Systemapplikationen für μ SINA

Absicherung von öffentlichen oder zentralen Systemen

- über das Internet erreichbare Server
 - ▷ WWW-, FTP-, DNS-Server und v.m.
- firmeninterne Server
 - ▷ NFS- oder Samba-Server

Härten von Systemen an neuralgischen Netzknoten

- Router, VPN-Gateways, Firewalls, Proxies, ...

Intrusion Detection und Response Systeme (IDS, IDRS)

- Host- und Netz-basierte Audit-Server
- IDS- und IDRS-Server

mobile Endgeräte (PDAs)

Weiterführende Informationen

Fiasco und L⁴Linux

- l4-linux@os.inf.tu-dresden.de
- L⁴Linux: <http://os.inf.tu-dresden.de/L4/LinuxOnL4/>
- fiasco-core@os.inf.tu-dresden.de/fiasco
- Fiasco: <http://os.inf.tu-dresden.de/fiasco>

SINA und μ SINA

- mikrosina@inf.tu-dresden.de
- SINA: <http://www.bsi.bund.de/fachthemen/sina/index.htm>
- μ SINA: <http://os.inf.tu-dresden.de/mikrosina>
- secunet: <http://www.secunet.com>

Modul-Server

- modld@stecklina-net.de
- <http://www.stecklina-net.de/~ost/arbeiten/modld/>

Referent

Oliver Stecklina

secunet Security Networks AG
(Niederlassung Dresden)
Ammonstraße 74
01067 Dresden

Telefon: (0351) 43 95 9 - 35
e-Mail: stecklina@secunet.de

