

# L 09: OS Security

- Schutzmatrix
- Unix User, Files, Memory, Shared Segments
- access control, capabilities
- RACF, SE Linux, Solaris RBAC
- Secure Boot

# Fragen

- Deadlock vs. Virtualisierung der Ressourcen?
- Virtualisierung von Applikationen? MPI? Grid?
- Service Oriented Architecture
- Standards, wozu? POSIX? Wichtige APIs?
- OS Einbindung von Bandlaufwerken? Restore?
- Band Roboter?
- HSM? VTL?
- OS Boot Prozess?

# Zugriffsmatrix (access matrix)

Auch: Schutzmatrix, Rechtematrix

- Zeilen: Alle Subjekte
  - User
  - Programme (z.B. der Mailserver)
- Spalten: Alle Objekte
  - Dateien
  - Systemcalls (Mount, Lesen von Dateien, ...)
  - Module/Klassen ggf. differenziert nach Methoden
- Zellen: Rechte
  - read, write, execute, ...
  - Verzeichnis: add/mod/del Datei,...

# Zugriffsmatrix: Eigenschaften

- Kann sehr gross werden
  - $10^5$ - $10^6$  Dateien mit  $10^4$ - $10^5$  Usern
- Komplexe Bedingungen:
  - User `operator` startet Programm `netbackup`
    - dann Zugriff auf alle Dateien für Backup und Restore
  - Normaler User startet Programm `netbackup`
    - Nur restore der eigenen Dateien möglich

Wie kann so etwas modelliert werden?

# Capability Listen

Speicherung als Liste (Hash) beim Subjekt

- Welche Objekte?
- Welche Rechte

Bewertung:

- Aufwendig für Dateien
- Ok für Systemcalls

# Access Control Listen

Speicherung als Liste (Hash) beim Objekt

- Welche Subjekte haben Zugriff?
- Welche Rechte sind erlaubt

Bewertung:

- Aufwendig für Systemcalls
- Sehr Aufwändig für Dateien
- Gruppierung hilft (user, group, other, ...)

# Capability Liste: Segment Pointer

## Segment Tabellen-Eintrag

- Adresse
  - Länge
  - Rechte (r, rw, rx, ...)
  - Speicherschutz-Ring (Match mit PSW)
- 
- Erhöhte Sicherheit bei durchgängiger Nutzung

# Capability Liste: OS

- Beispiel: Kopieren einer Datei
  - Quell Datei zum Lesen öffnen: Cap-Pointer
  - Ziel-Datei zum Schreiben öffnen: Cap-Pointer
  - Programm copy aufrufen: nutzt Capabilities
- Forschungs-Betriebssysteme:
  - Hydra, Amoeba, Birlix, Monads, Eros
- Real World OS:
  - IBM AS/400

# Capability Liste: /etc/passwd

Inhalte von `/etc/passwd` bzw. `/etc/shadow`

- User-Name
- User-Nummer (Cap)
- Gruppen-Nummer (Cap)
- Text
- Shell (Cap)
- Home-Directory (Cap)
- Password-Verschlüsselung (Zugang)

# Access Control Liste: Unix Rechte

Unix Datei Rechte (POSIX) `rwXrwxrwx`

- Rechte

- read (Dir: “list”)
- write (Dir: “add/del file”)
- execute (Dir: “use”)

- Subjekte sind gruppiert:

- owner
- group (multiple Gruppen via `/etc/group`)
- other (User: nobody)

# Unix Rechte: Mail Server

Situation: Unix Mail nutzt `/var/mail/userid`

- Mail Server empfängt Mail und legt sie ab
- User liest, modifiziert, löscht Mail

Realisierung: Rechte sind `rw-rw----`

- User kann lesen und schreiben
- Mail-Server läuft mit `setgid` auf Gruppe Mail und kann daher auch lesen und schreiben
- verschiedene lock-Mechanismen

# Erweiterte ACLs (1)

- Vorschlag bei Solaris 2.6
- Modifiziert bei Windows NT (!)
- Standard bei SMB, NFSv4, ZFS, WAFL, ...

## Subjekte:

- owner + weitere User
- gruppe + weitere Gruppen
- other

# Erweiterte ACLs (2)

## Objekte

- Dateien, Verzeichnisse

## Rechte

- file: read/write, execute, append
- dir: list/add directory, add/delete file, read/write/change attributes
- write owner, synchronize
- file\_inherit, dir\_inherit, inherit\_only, no\_propagate
- allow/deny, mehrere Regeln in Kombination

# IBM RACF / SecureWay Security Server

## Kontrolle:

- Login, Technische User (Jobs  $\approx$  Unix daemons)
- Administrative Rechte
- Systemcalls
- Dateien
- Transaktionen (Teilhaber System)

## Implementierung:

- Aktiv beim Objekt (z.B. Transaktion fragt bei RACF)
- Gruppierung bei Subjekten

# SE Linux

## Security Context

- User
- Rolle
- Typ
- Level (Label)

## Alle Systemcalls prüfen

- Typ des Calls und Security Context vereinbar
- ggf. Wandlung für Zugriff auf andere Objekte
- komplex

# Solaris Privilegien (1)

## Sicherheit bei Unix:

- Privilegierte Aktionen darf nur root (userid == 0)
  - Das ist die einzige Prüfung!
- Privilegiertes durch Programme
  - setuid r-s--x--x
  - setgid r-x--s--x

# Solaris Privilegien (2)

Solaris (und andere trusted Systeme):

- Privilegien sind fein granular zuteilbar, z.B.:
  - `sys_mount`
  - `file_dac_read`
  - `net_privaddr`
- root hat erstmal alle Privilegien
- In `/etc/security/user_attr` konfigurierbar
- Online zuteilbar (`proc_setid`)

# Solaris Privilegien (3)

- Realisierung
  - Systemcall checkt Privilegien Bit
  - Child-Prozess erbt Privilegien kontrolliert
- Privilegien-Sets
  - E effective      Aktuell effektiv
  - I inheritable    Können weitergegeben werden
  - L limit            Limit für Sub-Prozess
  - P permitted        Maximal erlaubte Privilegien

# Solaris Autorisierungen

## Authorizations

- Benannte Tags für Prozesse
- In `user_attr` einem User zuteilbar
- Wird vom Programm geprüft
- Selbst definierbar
  - Privilegien relativ statisch
  - Autorisierungen sind schnell einrichtbar

# Role Based Access Control (1)

## Idee:

- Rollen (role) als Gruppierung der User
  - unabhängig von Gruppen (genutzt für Filezugriff)
  - User-Seite: Zuteilung von Rollen
  
- Profile als Gruppierung der Sonder-Rechte
  - Programme mit setuid/setgid
  - Programme mit Privilegien
  - Programme mit Autorisierungen
  - Programm-Seite: Zuteilung von definierten Programmen

# Role Based Access Control (2)

- `exec_attr`
  - Profil
  - Programm
  - Userid / Groupid Setzungen
  - Autorisierung
  - Privilegien
- `prof_attr`
  - Profil Name
  - Beschreibung
  - Hilfe
  - Autorisierungen

# Role Based Access Control (3)

- `user_attr`
  - Userid / Rollenid
  - Zugeteilte Rollen
  - Zugeteilte Autorisierungen
  - Zugeteilte Profile
  - Zugeteilte Privilegien
  - ..., Projekte,
- Nutzung
  - `pfexec` zum Aufruf eines Programms
  - `su` zum Annehmen der Rolle

# Labelled Security

- Objekte erhalten zusätzlich Security Label
- Subjekte werden in Security Labels eingruppiert
- System erzwingt Compliance
  - Unterstützung auf Kommando Ebene
  - Unterstützung Graphischer Oberflächen

# Sicheres Booten

## Abgesicherter Boot-Vorgang

- Rom-Code verifiziert Boot Sektor
  - Prüfsumme aus RAM/PROM
  - Public/Private Key aus RAM/PROM
- Erst dann Start von Boot Sektor
- Boot Sektor verifiziert Boot Loader
- Boot Loader verifiziert Betriebssystem
- Optionen
  - HW Support für Verifikation
  - Verifikation der PCI-Slot ROMs

# Trusted Computing

## Trusted Computing Platform Alliance

- umstrittene Ziele
- bisher nicht durchgesetzt