
Betriebssysteme (BS)

12. Systemsicherheit

<https://sys.cs.tu-dortmund.de/DE/Teaching/SS2021/BS/>

07.07.2021

Peter Ulbrich

`peter.ulbrich@tu-dortmund.de`

Basierend auf *Betriebssysteme* von Olaf Spinczyk, Universität Osnabrück

Inhalt

- Überblick über Sicherheitsprobleme
- Rechteverwaltung
- Systemsoftware und Sicherheit
- Softwarefehler
- Zusammenfassung

Tanenbaum
9: IT-Sicherheit

Silberschatz
14: Protection
15: Security

Sicherheitsprobleme

- **Begriffsdefinition: *Sicherheit***
 - **Safety**
 - Schutz vor Risiken durch (Software-)Fehler, Störungen oder Ausfällen
 - **Funktionale Sicherheit**
 - **Security**
 - Schutz von Menschen und Rechnern vor intendierten Fehlern (Angriffen)
 - **Datensicherheit**

- **Beide Themenbereiche sind für Systemsoftware relevant!**
 - Im Folgenden: Fokus auf Security

- **Ausnutzung von **Sicherheitslücken****
 - Schadsoftware („Malware“)
 - Social Engineering

Sicherheit in Betriebssystemen

Jemanden ...

- Unterscheidung von Personen und Gruppen von Personen

davon **abhalten** ...

- durch technische und organisatorische Maßnahmen

einige ...

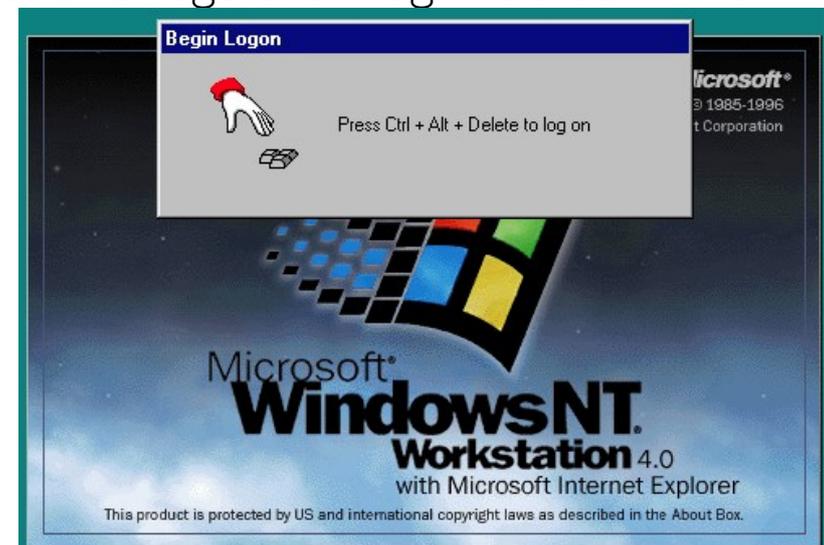
- Begrenzung durch unser Vorstellungsvermögen

unerwünschte Dinge zu tun.

- 1) nicht autorisiert Daten lesen (**Geheimhaltung, Vertraulichkeit**),
 - 2) nicht autorisiert Daten schreiben (**Integrität**),
 - 3) unter „falscher Flagge“ arbeiten (**Authentizität**),
 - 4) nicht autorisiert Ressourcen verbrauchen (**Verfügbarkeit**),
- ...

Beispiel: Login-Attrappe

- Angreifer startet Benutzerprogramm, das am Bildschirm einen Login-Screen simuliert
- Der ahnungslose Benutzer tippt **Benutzername** und sein privates **Passwort** ein.
 - Angreiferprogramm speichert Benutzername und Passwort in Datei,
 - terminiert aktuelles Shell-Programm
 - Login-Sitzung des Angreifers wird beendet und regulärer Login-Screen wird angezeigt
- **Abhilfe:**
Login-Sequenz durch Tastensequenz starten, die von Benutzerprogramm **nicht abgefangen** werden kann
 - z.B. Strg-Alt-Entf bei Windows NT und folgende.



Beispiel: Virus

- Programm, dessen Code an ein anderes Programm angefügt ist und sich auf diese Weise **reproduziert**
 - Virus schläft, bis infiziertes Programm ausgeführt wird
 - Start des infizierten Programms führt zur Virusreproduktion
 - Ausführung der Virusfunktion ist u.U. zeitgesteuert
- **Virus-Arten**
 - Bootsektor-Viren: beim Systemstart
 - Linkviren: Ausführbares Programm als Virus
 - Makroviren: in skriptbaren Programmen wie Word, Excel
 - Durch Dokumente verbreitet!
- Verbreitung durch Austausch von Datenträgern, E-Mails, Webseiten

Beispiel: Social Engineering

- Zugang zu Systemen durch **Ausnutzung menschlicher Fehler**
 - Kein Problem der Systemsoftware, aber immens wichtig
- **Phishing**
 - über gefälschte WWW-Adressen an Daten eines Internet-Benutzers gelangen
 - z.B. durch gefälschte E-Mails von Banken
- **Pharming**
 - Manipulation der DNS-Anfragen von Webbrowsern
 - Umleitung von Zugriffen, z.B. auf gefälschte Bank-Webseiten
 - Verdächtige Zertifikate werden von Nutzern meist ignoriert.

Arten von „Schädlingen“ (1)

■ Viren

- unabsichtliche Verbreitung „infizierter“ Programme/Dokumente
- Reproduktion: Einschleusen in andere Programme/Dokumente

■ Würmer

- keine Anwender-„Unterstützung“ bei der Verbreitung in neue Systeme
- versuchen, **aktiv** in neue Systeme einzudringen
- Ausnutzung von Sicherheitslücken auf Zielsystemen

■ Trojanische Pferde („Trojaner“)

- Programm, das **als nützliche Anwendung getarnt** ist
- ohne Wissen des Anwenders: Ausführen einer Schadfunktion, z.B. Netzwerkzugang für den Angreifer

Arten von „Schädlingen“ (2)

■ Rootkit

- Sammlung von Softwarewerkzeugen, um ...
 - zukünftige Logins des Eindringlings verbergen
 - Prozesse und Dateien zu verstecken
- wird nach Einbruch in ein Computersystem auf dem kompromittierten System installiert
- **Versteckt sich selbst** und seine Aktivitäten vor dem Benutzer
 - z.B. durch Manipulation der Werkzeuge zum Anzeigen von Prozessen (ps), Verzeichnisinhalten (ls), Netzwerkverbindungen (netstat) ...
 - ... oder durch Manipulation von systemweiten *shared libraries* (libc)
 - ... oder direkt durch Manipulation des Systemkerns

■ Oft treten Schädlinge als **Kombination** der vorgestellten Kategorien auf.

Inhalt

- Überblick über Sicherheitsprobleme
- Rechteverwaltung
- Systemsoftware und Sicherheit
- Softwarefehler
- Zusammenfassung

Ziele der Rechteverwaltung

- **Schutz gespeicherter Information vor ...**
 - Diebstahl (Verletzung der **Vertraulichkeit**)
 - unerwünschter Manipulation (Verletzung der **Integrität**)
- **... in allen Mehrbenutzersystemen**
 - ... und jedes am Internet angeschlossene System ist **de facto** ein Mehrbenutzersystem!

Anforderungen

- alle **Objekte** eines Systems müssen eindeutig und fälschungssicher identifiziert werden
- (externer) **Benutzer** eines Systems muss eindeutig und fälschungssicher identifiziert werden
 - Authentifizierung
- Zugriff auf Objekte sollte nur über zugehörige **Objektverwaltung** geschehen
- Zugriff auf Objekte nur, wenn Zugreifer nötige **Rechte** hat
 - Rechte müssen fälschungssicher gespeichert werden
 - Weitergabe von Rechten darf nur kontrolliert erfolgen

Entwurfsprinzipien

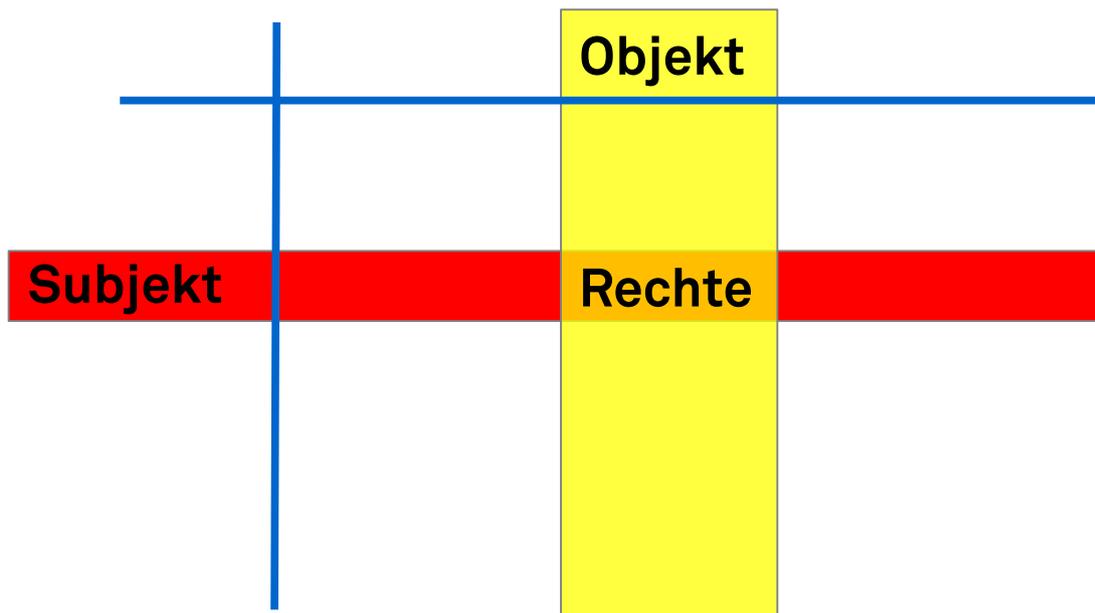
- **Prinzip der geringst-möglichen Privilegisierung** („*principle of least privilege*“)
 - Person oder Software-Komponenten nur die Rechte einräumen, die für die zu erbringende Funktionalität erforderlich sind
 - Verbot als Normalfall
 - **Gegenbeispiel:** Systemdienste mit „root-Rechten“ (Unix)

- **Sichere Standardeinstellungen** („*secure defaults*“)
 - **Beispiel:** neu installierte Server-Software

- **Separierung von Privilegien** („*separation of duties*“)
 - mehrfache Bedingungen für die Zulassung einer Operation

Zugriffsmatrix

- **Begriffe:**
 - **Subjekte** (Personen, Prozesse)
 - **Objekte** (Daten, Geräte, Prozesse, Speicher ...)
 - **Operationen** (Lesen, Schreiben, Löschen, Ausführen ...)
- **Frage:** Ist Operation(Subjekt, Objekt) zulässig?



Basismodell: Datei-/Prozessattribute

- Festlegungen in Bezug auf Benutzer:
 - Für welchen Benutzer arbeitet ein Prozess?
 - Welchem Benutzer gehört eine Datei (*owner*)?
 - Welche Rechte räumt ein Benutzer anderen und sich selbst an „seiner“ Datei ein?

- Rechte eines Prozesses an einer Datei
 - Attribute von Prozessen: User ID
 - Attribute von Dateien: Owner ID

	Datei 1	Datei 2	Datei 3
User 1			
User 2		Read	
User 3			
User 4			

Varianten der Schutzmatrix

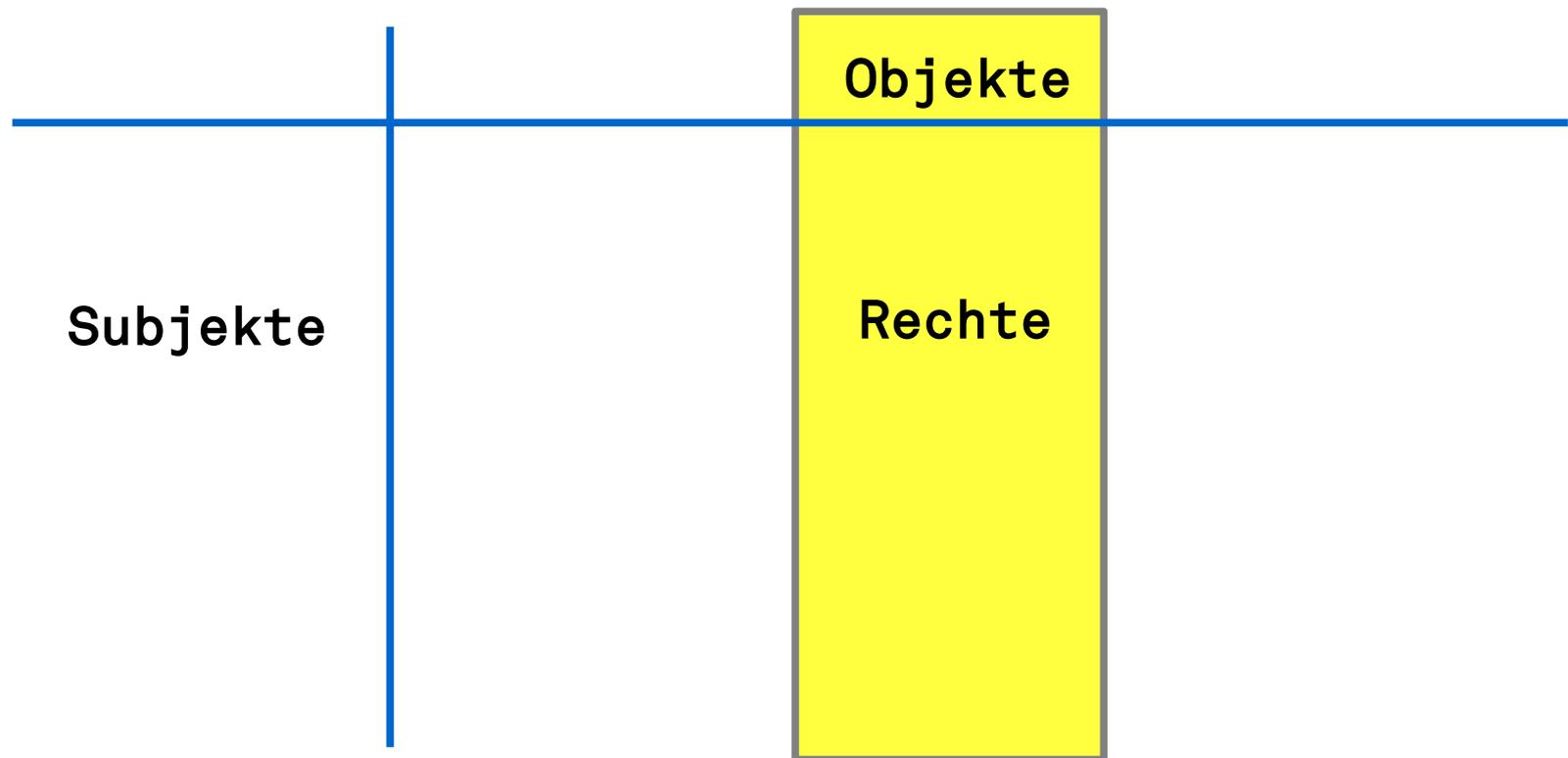
- spaltenweise: **ACL – Access Control List** (Zugriffssteuerliste)
 - bei jedem Zugriff wird beim Objekt auf der Basis der Identität des Absenders dessen Berechtigung geprüft

- zeilenweise: **Capabilities** (Zugriffsberechtigungen)
 - bei jedem Zugriff wird etwas geprüft, was Subjekte besitzen und bei Bedarf weitergeben können

- regelbasiert: **Mandatory access control**
 - bei jedem Zugriff werden Regeln ausgewertet

Access Control Lists (1)

- Spaltenweise Darstellung: **ACL – Access Control List**
- Festlegung für jedes **Objekt**, was welches **Subjekt** damit tun darf



Access Control Lists (2)

■ Setzen der ACLs darf:

- Wer einen ACL-Eintrag für dieses Recht hat
- Erzeuger der Datei

■ Beispiel Multics: Tripel (Nutzer, Gruppe, Rechte)

```
File 0 (Jan, *, RWX)
File 1 (Jan, system, RWX)
File 2 (Jan, *, RW-), (Els, staff, R--), (Maaike, *, RW-)
File 3 (*, student, R--)
File 4 (Jelle, *, ---), (*, student, R--)
```

■ Beispiel Windows (ab NT):

Pro Objekt eine Liste von *Access Control Entries* (ACEs)

- *Trustee* (Nutzer oder Gruppe)
- Rechte (jew. mit *allow* oder *deny*): *full control, modify, read, execute, ...*

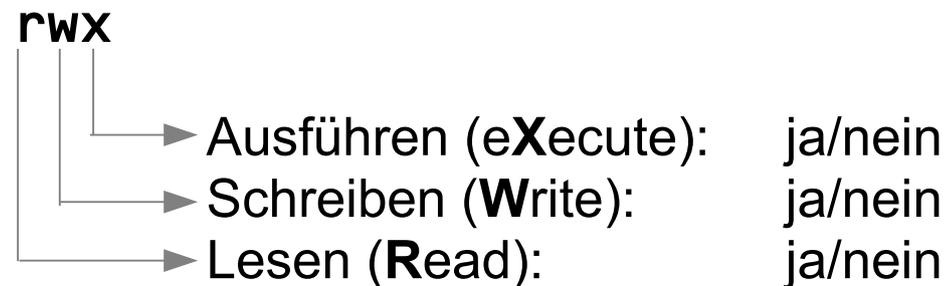
Unix-Zugriffsrechte

- **Unix:** rudimentäre Zugriffssteuerlisten
 - Prozess: User ID, Group ID
 - Datei: Owner, Group
 - Rechte in Bezug auf User (Owner), Group, Others

- **Neuere Unix-Systeme implementieren auch ACLs**
 - siehe get/setfacl (1)
 - **Problem:** Dateisystem-Integration, Kompatibilität mit Anwendungen

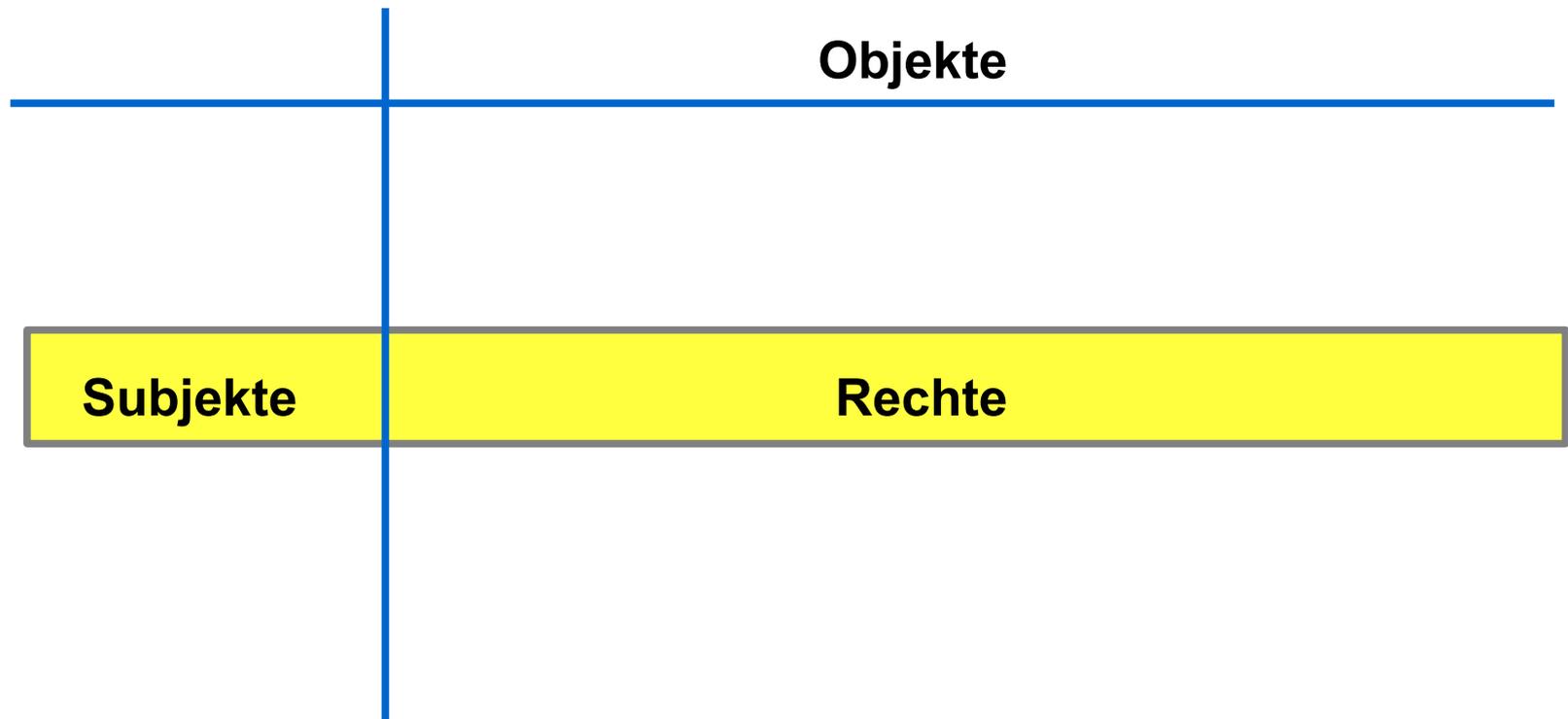
vorlesung.tex		
rw-	r--	---
		Others
	Group: staff	
User: ulbrich		

Dateiattribute:



Capabilities

- Zeilenweise Darstellung der Schutzmatrix: **Capability**
- Festlegung für jedes Subjekt, wie es auf welche Objekte zugreifen darf



Capabilities: Beispiel

- Rudimentäre Form: Unix-Dateideskriptoren
- Weitergabe durch **fork**-Systemaufruf
 - Ermöglicht Zugriff auf Dateien ohne erneute Prüfung der UNIX-Zugriffsrechte

Prozessleitblock

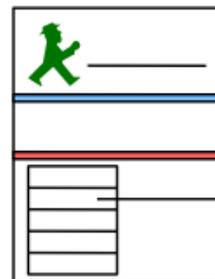
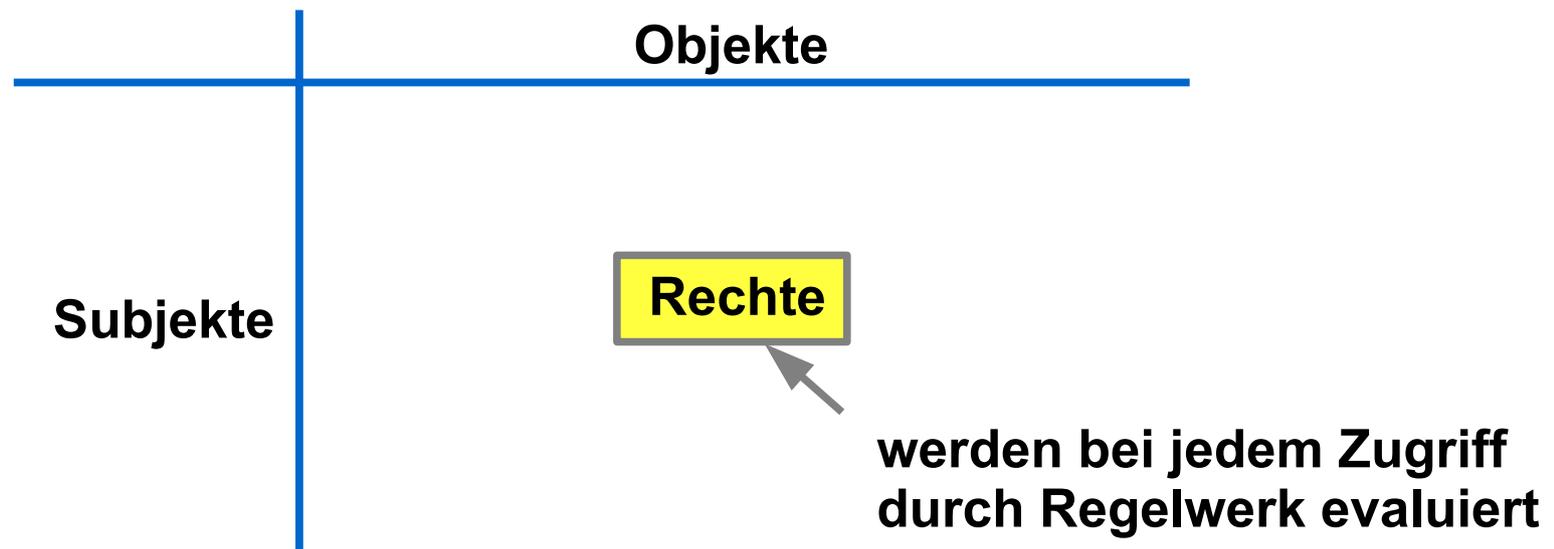


Tabelle offener
Dateien



Schutzmatrix regelbasiert

- **Mandatory Access Control (MAC):** regelbas. Zugriffssteuerung
- **Konzept:**
 - Subjekte und Objekte haben Attribute (*labels*)
 - Entscheidung über Zugriff anhand von Regeln
- Implementierung in sog. Sicherheitskernen, z.B. SELinux



Inhalt

- Überblick über Sicherheitsprobleme
- Rechteverwaltung
- **Systemsoftware und Sicherheit**
- Softwarefehler
- Zusammenfassung

Systemsoftware und Sicherheit

■ Schutz auf **Hardware**-Seite

- MMU (Speicher)
- Schutzringe (CPU)

■ ... ergänzt durch Schutz auf **Systemsoftware**-Seite

- Betriebssystem hat **alleinige Kontrolle**
 - der Hardware
 - über alle Prozesse
 - über alle Ressourcen
- Bereitstellung von
 - Identifikationsmechanismen
 - Authentisierungsmechanismen
 - Privilegseparation
 - Kryptographische Sicherung von Informationen

Hardwarebasierter Schutz: MMU

■ **Memory Management Unit**

- Hardwarekomponente der CPU, die Zugriff auf Speicherbereiche umsetzt und kontrolliert
- Umsetzung von Prozess-Sicht (virtuelle Adressen) auf Hardware-Sicht (physische Adressen)

■ Einteilung des Hauptspeichers in Seiten (*pages*)

■ **Schutz durch ...**

- **Einblendung** nur der genau benötigten Menge an Speicherseiten des Hauptspeichers in den virtuellen Adressraum eines Prozesses
- **Isolation** der physikalischen Adressräume unterschiedlicher Prozesse
- **Schutzbits** für jede Seite, die bei jedem Zugriff kontrolliert werden
 - Lesen/Schreiben/Code ausführen
 - Zugriff im User-Mode/Supervisor-Mode

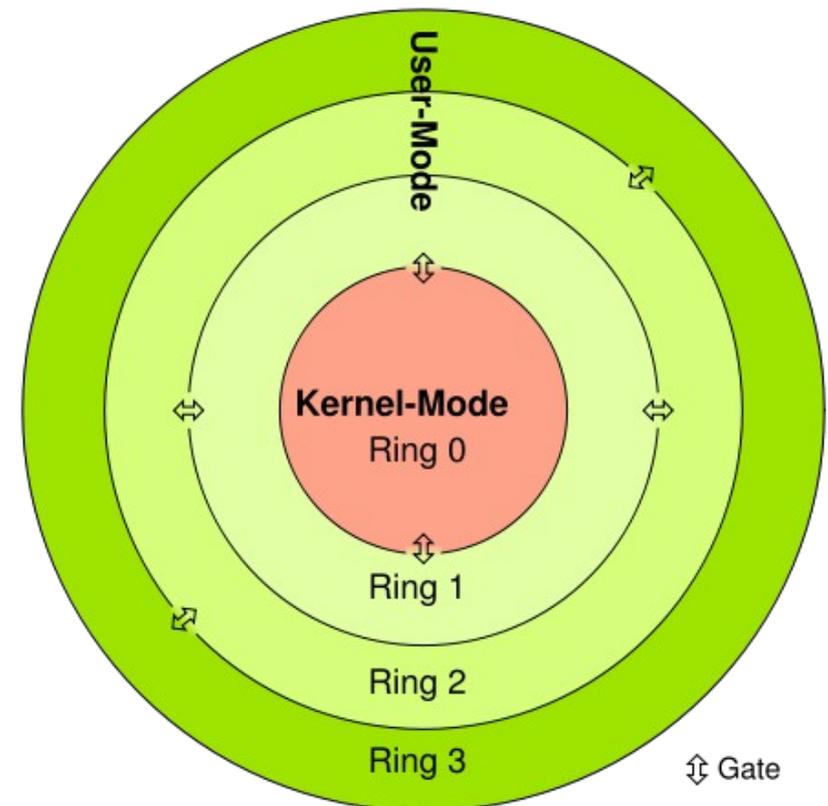
Schutzringe

■ Privilegienkonzept, hier in der x86-Variante

- Ausführung von Code ist bestimmtem Schutzring zugeordnet
- Code in Ring 0 hat Zugriff auf alle Ressourcen des Systems
- User-Programme laufen in Ring 3
- Ringe 1 u. 2 für BS-nahen Code
 - z.B. Gerätetreiber

■ Ringe schränken ein ...

- den nutzbaren *Befehlssatz* des Prozessors
 - z.B. keine Interruptsperrern in Ring > 0
- den zugreifbaren *Adressbereich* für den Prozess
 - Sperre von I/O-Zugriffen



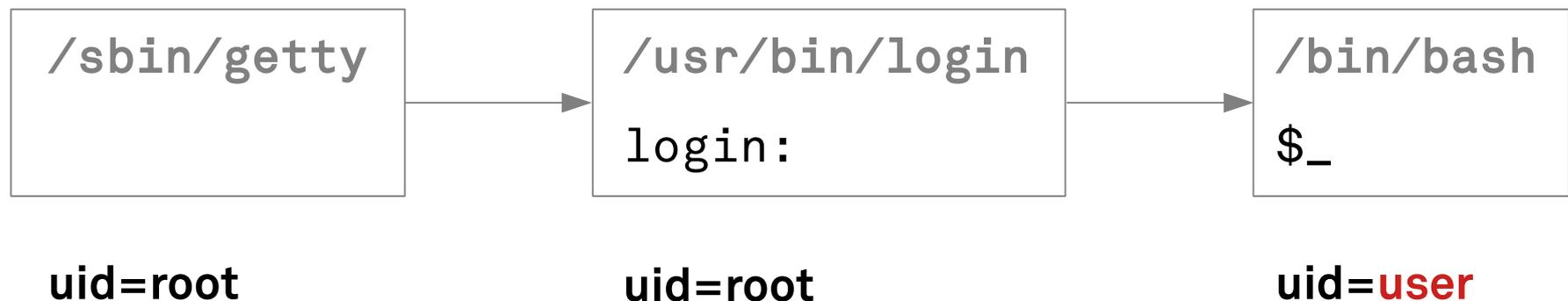
Softwarebasierter Schutz

- Identifikationsmechanismen
- **Unix:** Benutzeridentifikation, Gruppenidentifikation
 - Numerischer Wert
 - Übersetzung in Namen (Usernamen, Gruppennamen) durch Zuordnungstabellen in `/etc/passwd` bzw. `/etc/group`
- Ressourcen haben zugeordnete Besitzer
- Superuser: `uid = 0`
 - Hat volle Rechte auf das System

Softwarebasierter Schutz

Authentifizierungsmechanismen

- **Beispiel:** Unix login
- Abfrage von Benutzernamen und Passwort
- Verifikation des Passworts mit im System hinterlegtem Passwort
 - Entweder durch Verschlüsselung des eingegebenen Passworts und Vergleich mit dem hinterlegten verschlüsselten Wert
 - oder durch Verifikation eines Hash-Wertes
- Der login-Prozess startet dann das erste Benutzerprogramm (z.B. eine Shell) mit der uid und gid, die zum eingegebenen Benutzernamen gehören.



Softwarebasierter Schutz

■ Kryptographische Sicherung von Informationen

- z.B. Passworte der Systembenutzer DES-verschlüsselt
- Ursprünglich in Unix: `/etc/passwd`

```
root:4t6f4rt3423:0:0:System Administrator:/var/root:/bin/sh
daemon:ge53r3rfrg:1:1:System Services:/var/root:/usr/bin/false
me:1x3Fe5$gRd:1000:1000:Michael Engel:/home/me:/bin/bash
```

- *Problem*: verschlüsselte Passworte für alle Benutzer lesbar
 - ... und mit genügend Zeit auch durch „brute force“-Angriff zu knacken
 - fertige Tools wie z.B. „John the Ripper“

■ Heute: Nur Benutzerinformationen in `/etc/passwd`

- Passworte stehen separat in `/etc/shadow`!

```
-rw-r--r-- 1 root root 1353 May 28 22:43 /etc/passwd
-rw-r----- 1 root shadow 901 May 28 22:43 /etc/shadow
```

Inhalt

- Überblick über Sicherheitsprobleme
- Rechteverwaltung
- Systemsoftware und Sicherheit
- Softwarefehler
- Zusammenfassung

Softwarefehler

- **Trade-off:** Performance ↔ Sicherheit
- C, C++, Assembler: sogenannte „*unmanaged*“-Sprachen
 - u.a. Pointer, keine Prüfung von Arraygrenzen, Wertebereiche-Overflow
- C#, Java: „*managed*“-Sprachen
 - Für Systementwicklung leider i.d.R. ungeeignet.
 - In *managed*-Sprachen gibt es auch Probleme, z.B. mit Speicherlecks!
- **Häufige Probleme:**
 - Pufferüberläufe
 - Wertebereichsüberläufe
- **Statistik:** Durchschnittlich 1 Fehler pro 1000 Quellcodezeilen

Beispiel: Wertebereiche (1)

- **Problem:** Ganzzahlen werden durch Bitstrings mit begrenzter Bitanzahl dargestellt
- Beispiel: char in C
 - Wird als 8-Bit-Wert dargestellt
 - Wertebereich: $-2^7 \dots +2^7 - 1$
 - ... oder $-128 \dots +127$
- Die zugehörige binäre Berechnung sieht wie folgt aus (*):
 - Es sind nur die unteren 8 Bit signifikant
 - **Ergebnis = -126!**

```
char a = 127;
char b = 3;
char Ergebnis = a + b;
```

```
01111111 (a)
+00000011 (b)
-----
10000010 (Ergebnis
           ist negativ!)
```

(*) Überlauf bei Vorzeichen-behafteten Datentypen ist in C „undefined“, **kann** aber auf quasi allen gängigen Architekturen zu einem *Wrap-Around* (zum „negativen Ende“ des Darstellungsbereichs) führen.

Beispiel: Wertebereiche (2)

- Folgender Code führt zu Problemen:

```
char string[127] = "Hallo Welt!\n"
char a = 127;
char b = 3;

...

char myfunc(char *string, char index) {
    return string[index];
}

...
printf("%x", myfunc(string, a+b));
```

- Bester Fall: **Segmentation Fault**
- Schlimmer: **Auslesen von „benachbarten“ Daten!**

Inhalt

- Überblick über Sicherheitsprobleme
- Rechteverwaltung
- Systemsoftware und Sicherheit
- Softwarefehler
- Zusammenfassung

Fazit

- **Sicherheit in vernetzten Umgebungen immer relevanter**
 - Extrem hoher Schaden durch Viren, Phishing, Botnets, Ransomware, ...
 - Auch erfahrene Computerbenutzer sind nicht sicher!
- **Sicherheitsüberprüfungen in Code unerlässlich**
 - Automatisierte Tests finden nicht alle Fehler
 - Manuelle Audits sind weiterhin erforderlich
 - Dennoch sind Sicherheitsprobleme weiterhin häufig.
 - Systemsoftware muss also ständig aktualisiert werden.
- **„Hase-und-Igel“-Spiel**
 - „Zero Day Exploits“, also neu entdeckte und nicht veröffentlichte Sicherheitslücken, sind extrem gefährlich.
 - Reaktionszeiten von Systemherstellern im Bereich von Stunden bis Monaten ...