



Betriebssysteme (BS)

Systemsicherheit

<http://ess.cs.tu-dortmund.de/DE/Teaching/SS2017/BS/>

Olaf Spinczyk

olaf.spinczyk@tu-dortmund.de

<http://ess.cs.tu-dortmund.de/~os>





Inhalt

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



Inhalt

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



Sicherheitsprobleme

- Begriffsdefinition „Sicherheit“
 - *Safety*
 - Schutz vor Risiken durch (Software-)Fehler, Störungen oder Ausfällen
 - *Security*
 - Schutz von Menschen und Rechnern vor intendierten Fehlern (Angriffen)
- Beide Themenbereiche sind für Systemsoftware relevant
 - Im Folgenden geht es um Sicherheit im Sinne von *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

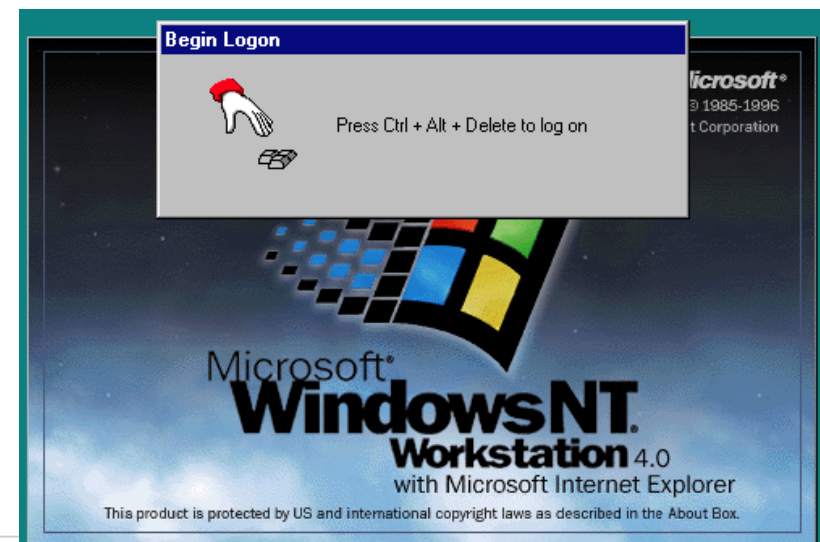
unerwünschte Dinge zu tun.

- 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**),
 - usw.
- Unterscheidung zwischen Angriffen von
 - innen
 - außen



Beispiel: Login-Attrappe

- Angreifer startet ein Benutzerprogramm, das am Bildschirm einen Login-Screen simuliert.
- Der ahnungslose Benutzer tippt Benutzername und sein privates Passwort.
 - Angreiferprogramm speichert Benutzername und Passwort in Datei.
 - Angreiferprogramm 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. CTRL-ALT-DEL 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
- Virusarten
 - Bootsektor-Virus: beim Systemstart
 - Makro-Virus: in skriptbaren Programmen wie Word, Excel
 - Durch Dokumente verbreitet!
 - Ausführbares Programm als Virus
- Verbreitung durch
 - Austausch von Datenträgern
 - Mail-Attachments
 - Webseiten



Beispiel: *Social Engineering*

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



Arten von „Schädlingen“ (1)

- **Viren**

- Durch Anwender unabsichtlich verbreitete Programme
- schleusen sich in andere Programme ein
- reproduzieren sich damit

- **Würmer**

- warten nicht, von Anwender auf neues System verbreitet zu werden
- versuchen, aktiv in neue Systeme einzudringen
- Ausnutzung von Sicherheitslücken auf Zielsystemen

- **Trojanische Pferde** („Trojaner“)

- Programm, das als nützliche Anwendung getarnt ist
- Zudem wird ohne Wissen des Anwenders andere Funktion erfüllt, 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
- Fallbeispiele
- Zusammenfassung



Ziele

- Schutz gespeicherter Information vor
 - Diebstahl
 - unerwünschter Manipulation
 - Verletzung der Vertraulichkeit
- 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
- grundlegenden Schutzmechanismen sollen ohne großen Aufwand überprüft werden können.



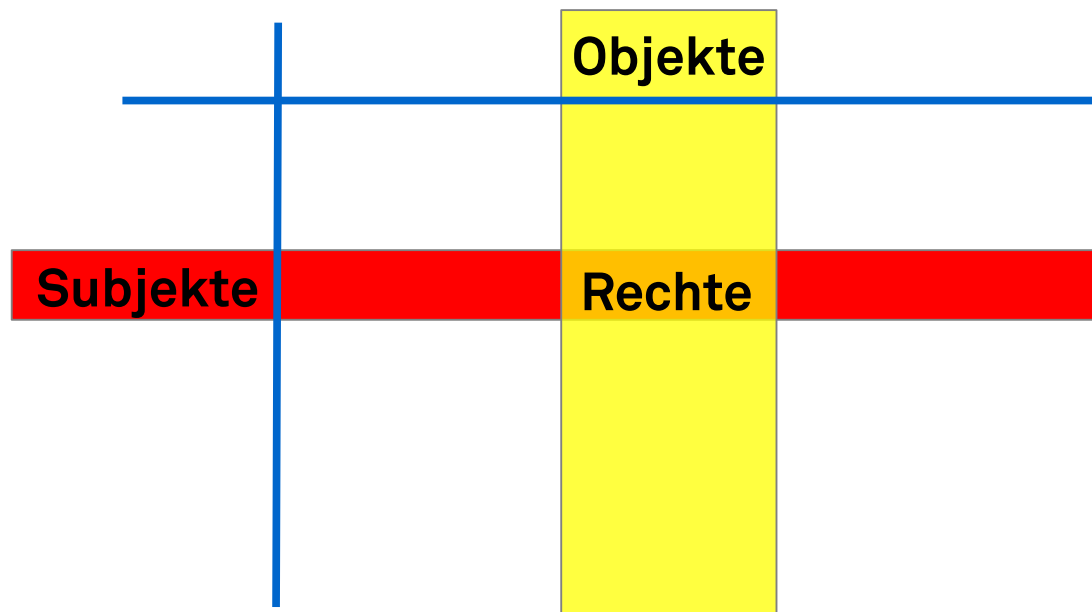
Entwurfsprinzipien

- **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: Unix „root“
- **Sichere Standardeinstellungen** („*fail-safe 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			



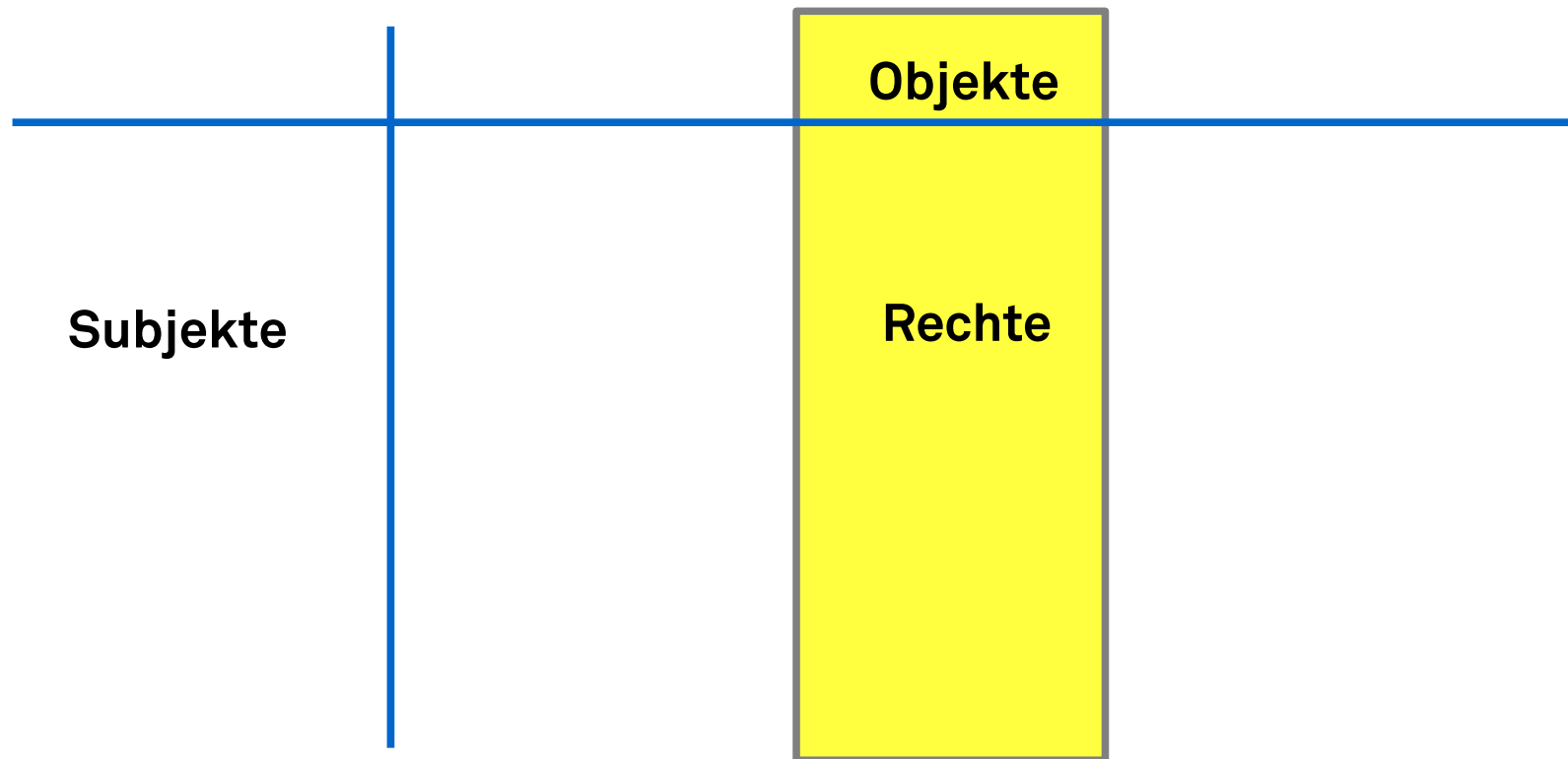
Varianten der Schutzmatrix

- 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



ACLs

- Spaltenweise Darstellung: **Access Control Lists**
- Festlegung für jedes Objekt, „was welches Subjekt damit tun darf“





ACLs

- 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--), (Maaïke, *, RW-)

File 3 (*, student, R--)

File 4 (Jelle, *, ---), (*, student, R--)

- Windows (ab NT)
 - Objekt: allow, 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

Datei.tex		
rw-	r - -	- - -
		Others
		Group: Staff
User: Michael		

Dateiattribute:

rwX

- Ausführen (**eX**ecute): ja/nein
- Schreiben (**W**rite): ja/nein
- Lesen (**R**ead): ja/nein



Problem: Rechte-Erweiterung

- Beispiel
 - *Highscore*-Liste: /home/me/spiele/tetris/Highscores
 - Programm: /home/me/bin/spiele/tetris
 - Jeder Spieler soll seinen *Highscore* selbst eintragen können

1. alle haben Schreibrecht

- zu viele Rechte (funktioniert nicht)
 - Jeder Benutzer könnte *Highscores* beliebig manipulieren

2. SetUID: nur “me” hat Schreibrecht

- Tetris-Programm mit „setuid“-Rechten
- sobald ein Prozess Tetris aufruft, erhält es als User-ID den *Owner* des ausführbaren Programms



Unix: Benutzer und Prozesse

- Jeder Prozess repräsentiert einen Benutzer
- Prozess-Attribute:
 - User ID (**uid**), Group ID (**gid**)
 - Effective-uid (**euid**), Effective-gid (**egid**)
 - Bestimmen beim Zugriff auf Dateien die Rechte eines Prozesses
- Nur wenige hochprivilegierte Prozesse dürfen uid und gid manipulieren
 - z.B. login-Prozess.
- Nach Überprüfung des Passwortes setzt Login-Prozess uid, gid, euid, egid.
 - Alle anderen Prozesse: Kinder des Login-Prozess.
- Kind-Prozesse erben Attribute von Eltern

Prozess
uid: Fritz
gid: Studis
euid: Fritz
egid: Studis



Unix-Lösung: setuid-Mechanismus

- Datei, die vertrauenswürdigen Programmcode (z.B. Tetris) enthält, besitzt Kennzeichnung als „SetUID“ (s-Bit)
 - im Verzeichnislisting „s“ statt „x“ für Executable
 - Es gibt auch (seltener verwendet) ein setgid-Bit.
- **exec** auf SetUID-Programme
 - ausführender Prozess erhält als effektive UID die UID des Owners des Programms
 - genauer: der Datei, die das Programm enthält
 - Prozessausführung erfolgt unter den Rechten dieses Benutzers, solange der Prozess läuft
 - Widerspricht dem Prinzip der geringsten Privilegierung
 - *Workaround*: Speziellen Nutzer für die Applikation einrichten; nicht 'root' verwenden
 - Guter Stil ist es, die erlangten SetUID-Rechte sobald wie möglich wieder abzugeben



Beispiel: Highscore-Liste

Shell

uid: Fritz

gid: Studis

euid: Fritz

egid: Studis

tetris

r - s - - x - - -

Others

Group: Tetris

User: Michael

Highscores

rw - r - - - - -

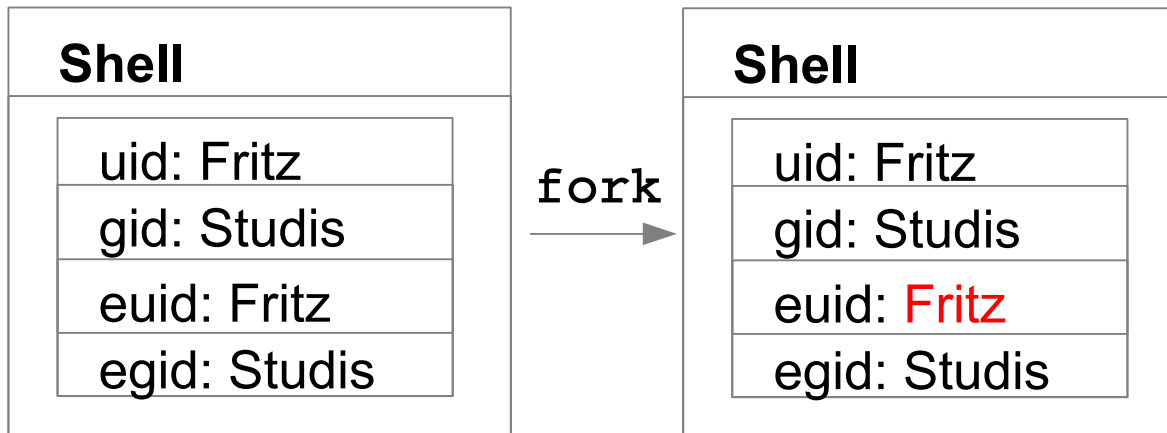
Others

Group: Tetris

User: Michael



Beispiel: Highscore-Liste



tetris		
r - s	- - x	- - -
		Others
		Group: Tetris
User: Michael		

Highscores		
r w -	r - -	- - -
		Others
		Group: Tetris
User: Michael		



Beispiel: Highscore-Liste



tetris		
r - s	- - x	- - -
		Others
		Group: Tetris
		User: Michael

Highscores		
rw -	r - -	- - -
		Others
		Group: Tetris
		User: Michael



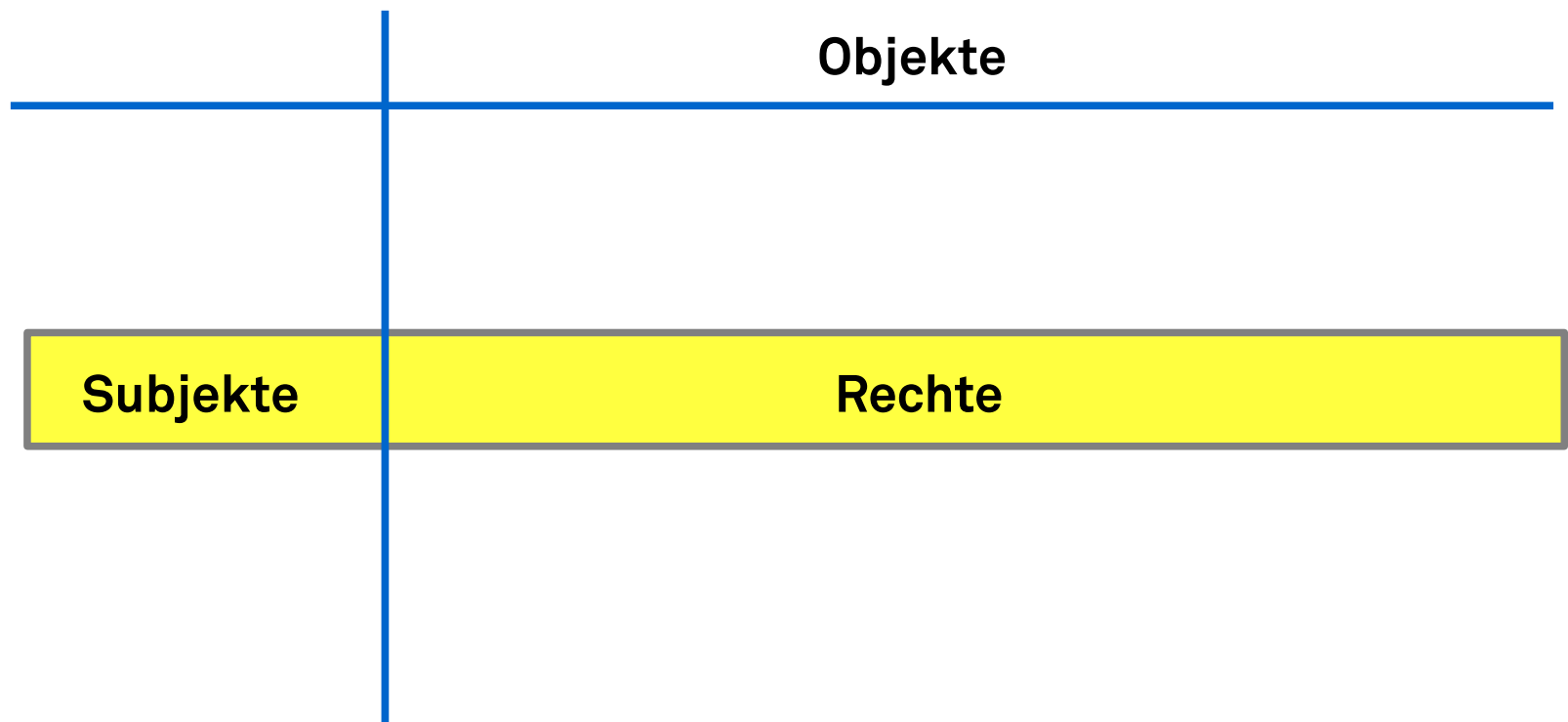
setuid-Probleme

- Erweiterung der Rechte eines Benutzers genau für den Fall der Benutzung dieses Programms
- „Besitzer“ des Programms vertraut dem Benutzer, wenn er dieses Programm nutzt
 - Besitzer kann Administrator, aber auch normaler Benutzer sein
- Problem: **Programmfehler**
 - können zu sehr großen Rechteerweiterungen führen
 - z.B. Shell-Aufruf aus einem solchen Programm heraus
 - Siehe Übung
- Praktische Erfahrung: immer noch zu viele Rechte!



Capabilities

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





Beispiel

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

Prozessleitblock

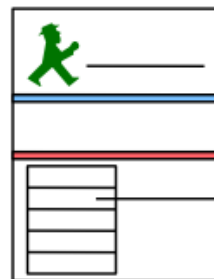


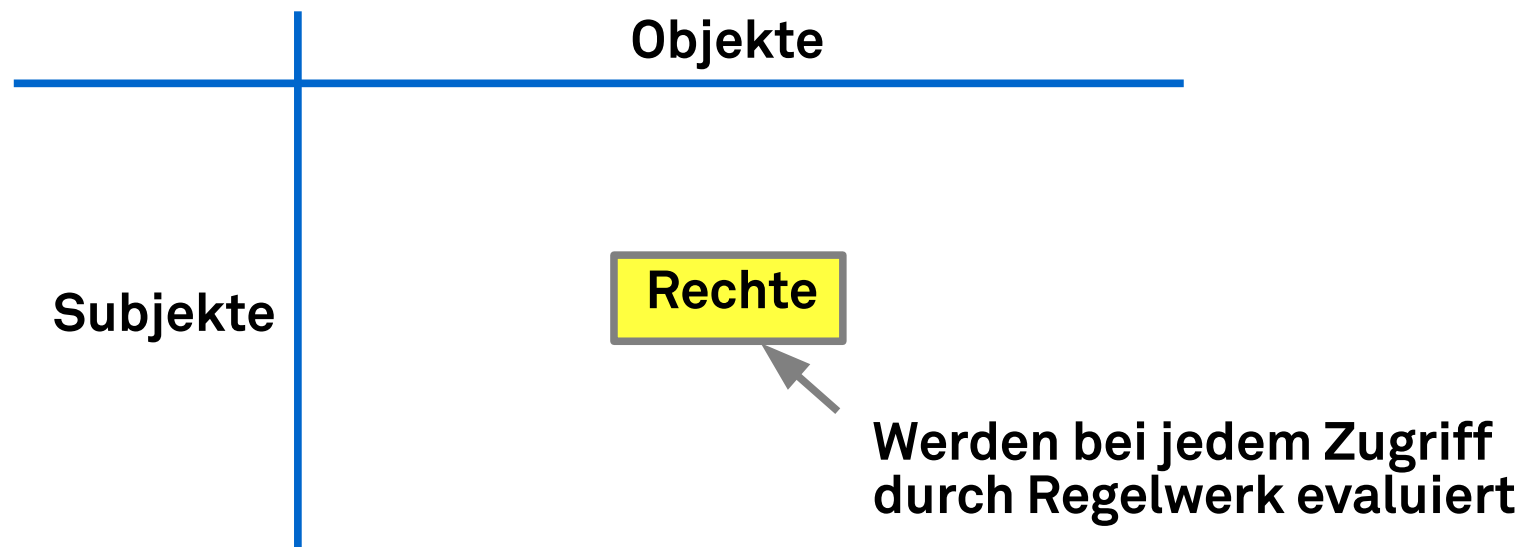
Tabelle offener
Dateien





Schutzmatrix regelbasiert

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





Inhalt

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



Systemsoftware und Sicherheit

- Schutz auf Hardware-Seite
 - MMU
 - Schutzringe
- ... ergänzt durch Schutz auf Systemsoftware-Seite
 - Alleinige Kontrolle der Hardware durch das Betriebssystem
 - Alleinige Kontrolle über alle Prozesse
 - Alleinige Kontrolle über alle Ressourcen
 - Bereitstellung von
 - Identifikationsmechanismen
 - Authentisierungsmechanismen
 - Privilegseparation
 - Kryptographische Sicherung von Informationen



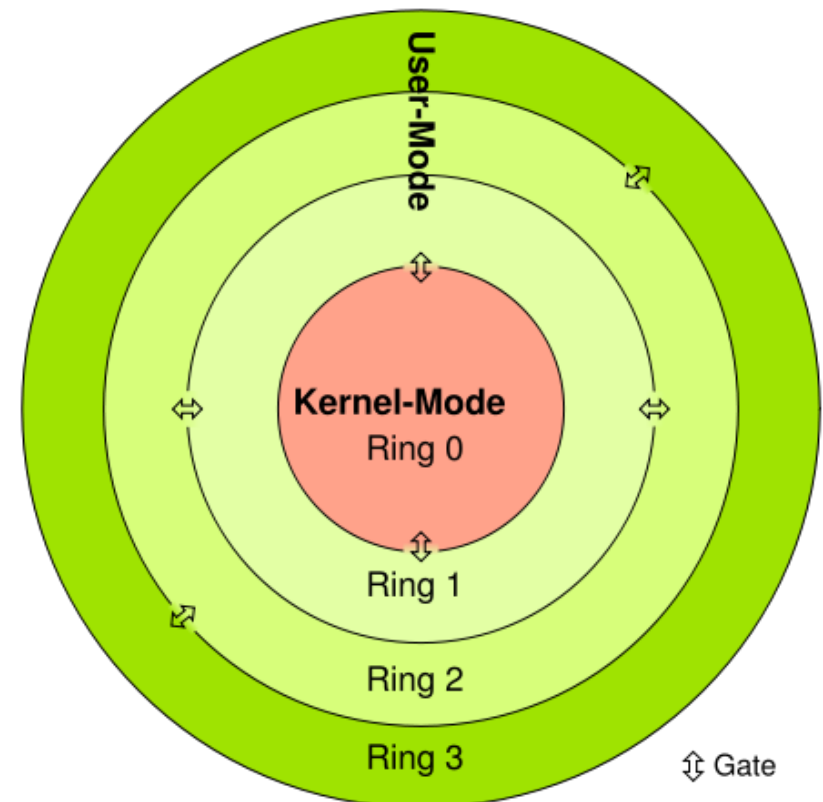
Hardwarebasierter Schutz: MMU

- **Memory Management Unit**
 - Hardwarekomponente der CPU, die Zugriff auf Speicherbereiche umsetzt und kontrolliert
 - Umsetzung von Prozess-Sicht (virtuelle Adressen) auf Hardware-Sicht (physikalische 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 phys. Adressräume unterschiedlicher Prozesse
 - Schutzbits für jede Seite, die bei jedem Zugriff kontrolliert werden
 - Lesen
 - Schreiben
 - Code ausführen



Schutzringe

- Privilegienkonzept
 - 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





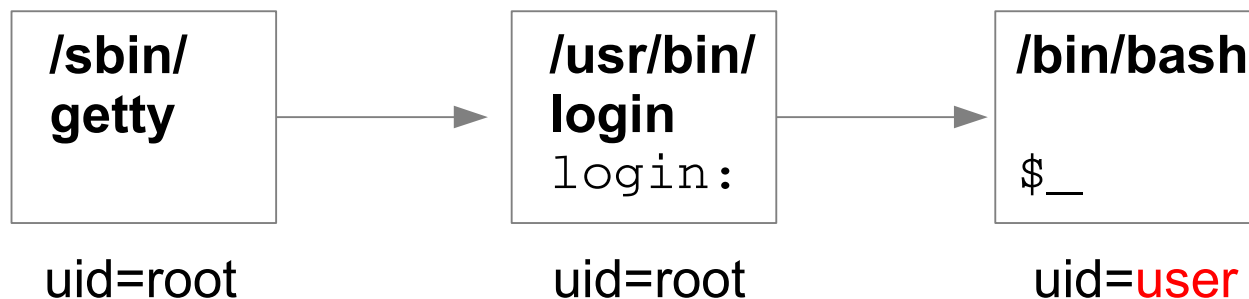
Softwarebasierter Schutz

- Identifikationsmechanismen
- Unix: Benutzeridentifikation, Gruppenidentifikation
 - Numerischer Wert
 - Umgesetzt in Texte (Usernamen, Gruppennamen) durch lookup in /etc/passwd
- Ressourcen haben zugeordnete Besitzer
- Superuser: uid = 0
 - Hat volle Rechte auf das System



Softwarebasierter Schutz

- Authentisierungsmechanismen
 - Unix login
 - Abfrage von Benutzernamen und Passwort
 - Verifikation des Passworts mit im System hinterlegten Passwort
 - Entweder durch Verschlüsselung des eingegebenen Passworts und Vergleich mit dem hinterlegten verschlüsselten Wertes
 - 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**
- Fallbeispiele
- Zusammenfassung



Softwarefehler

- *Trade-off*: Performance ↔ Sicherheit
- C, C++, Assembler: „*unmanaged*“ Sprachen
 - Pointer, Arraygrenzen, Wertebereiche-Overflow
- C#, Java: „*managed*“ Sprachen
 - Für Systementwicklung ungeeignet!
 - ... warum?
 - Aber auch Probleme in managed Sprachen!
- Probleme
 - Pufferüberläufe (siehe Übung)
 - Wertebereichsüberläufe
- Statistik zu Fehlern
 - Durchschnittlich ein Fehler pro 1000 lines of code
 - Unabhängig von der Implementierungssprache!



Wertebereiche

- 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 ... +127
- Die zugehörige binäre Berechnung sieht wie folgt aus:


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

```
01111111 (a)
+00000011 (b)
-----
10000010 (Ergebnis
           ist negativ!)
```
- Es sind nur die unteren 8 Bit signifikant
 - also Ergebnis = -126!



Wertebereiche

- 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));
```

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



Heap-Überlauf

- *Heap* („Halde“): der Speicherbereich für dynamisch allozierte Daten (z.B. durch malloc angefordert)
- Puffer-Überläufe im *Heap* können problematisch sein
 - Separat mit malloc angeforderte Speicherbereiche liegen hintereinander im Hauptspeicher
 - Es findet keine Überlaufskontrolle statt
 - Durch Angabe falscher Größen von Datenbereichen kann ein Angreifer andere Daten auf dem *Heap* überschreiben
- Beispiel: Microsoft JPEG GDI+ (MS04-028)
 - Größenangaben in JPEG-Bilddateien wurden nicht überprüft
 - „Normale“ Bilddateien enthalten gültige Werte
 - Führen nicht zu Fehlverhalten
 - Manipulierte Bilddateien enthalten ungültige Werte
 - Überschreiben andere Daten auf dem Heap



Heap-Überlauf

```
#define BUFSIZE 16
#define OVERSIZE 8 /* overflow buf2 by OVERSIZE bytes */

int main(void) {
    u_long diff;
    char *buf1 = malloc(BUFSIZE),
          *buf2 = malloc(BUFSIZE);

    diff = (u_long)buf2 - (u_long)buf1;
    printf("buf1 = %p, buf2 = %p, diff = 0x%x\n", buf1, buf2, diff);

    memset(buf2, 'A', BUFSIZE-1);
    buf2[BUFSIZE-1] = '\0';

    printf("before overflow: buf2 = %s\n", buf2);
    memset(buf1, 'B', (u_int)(diff + OVERSIZE));
    printf("after overflow: buf2 = %s\n", buf2);
    return 0;
}
```



Ergebnis ...

- Überschreiten des Wertebereichs um 8 Bytes

```
root /w00w00/heap/examples/basic]# ./heap1
buf1 = 0x804e000, buf2 = 0x804eff0, diff = 0xff0 bytes
before overflow: buf2 = AAAAAAAAAAAAAAAAAA
after overflow: buf2 = BBBBBBBBAAAAAAAA
```

- buf1 überschreitet seine Grenze und erreicht den Heap-Bereich, in dem buf2 steht
- Dieser Heap-Bereich von buf2 ist immer noch gültiger Speicher
 - Also bricht das Programm nicht ab, sondern manipuliert Daten!



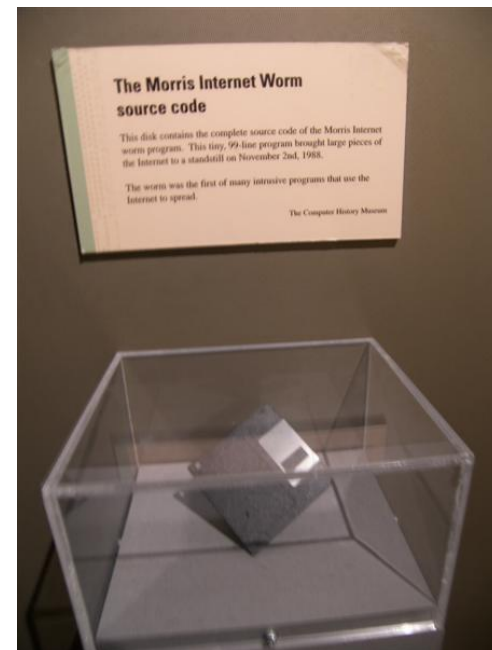
Inhalt

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



Unix Morris Worm (sendmail)

- Einer der ersten über Internet verbreiteten Würmer
- Geschrieben von einem Studenten der Cornell University, Robert Tappan Morris, und am 2. November 1988 vom MIT aus aktiviert
 - Vom MIT aus, damit der wahre Ursprung verschleiert werden konnte
 - Robert Tappan Morris ist heute *Professor* am MIT! :-)
- Nutzt Lücke im sendmail-System aus
 - *Buffer Overflow* in `gets()`
 - Geschrieben, um Größe des Internet zu bestimmen, infizierte jedes System nur einmal
 - ... hatte aber einen fatalen Bug in der Replikation!
- 6000 Unix-Systeme infiziert
 - Schaden zwischen US\$10 und US\$100 Millionen
 - ... 3 Jahre Haft auf Bewährung und US\$10.000 Geldstrafe für den Autor ...





Michelangelo-Virus

- 1991 zum ersten Mal in Neuseeland entdeckt
- Bootsektor-Virus, infiziert u.a. MS-DOS-Systeme
 - Benutzt nur BIOS-Funktionen, keine DOS-Systemcalls
- Zeitgesteuertes Virus, aktiv am 6. März
- Überschreibt die ersten 100 Sektoren der (ersten) Festplatte mit Nullen
- Verbreitung über Bootsektoren von eingelegten Disketten
 - Installiert im Bootsektor der Festplatte
- Einer der ersten Viren, die großes Medieninteresse hervorriefen
 - ... das aber spektakulär übertrieben war ;-)
- Auslieferung kommerzieller Software mit Virus im Bootsektor
 - Heute: Viren auf USB-Sticks, Handys mit USB, ... ab Produktion



Sony BMG Rootkit

- Software auf mit Digital „Rights“ Management (DRM) versehenen, kopiergeschützten CDs
 - Filtertreiber für CD-ROM-Laufwerke sowie für die IDE-Treiber, durch die der Zugriff auf Medien kontrolliert werden
 - Installation ohne Information oder Genehmigung des Benutzers
- Kontrolle der Verwendung von Daten der Sony BMG
 - Unter Windows-Systemen
- Verborgен vor Analyse mit *Rootkit*-Funktionen
 - taucht weder in der Software-Liste der Systemsteuerung auf, noch lässt sie sich über einen Uninstaller deinstallieren
 - versteckt nicht nur die zugehörigen Dateien, Verzeichnisse, Prozesse und *Registry*-Schlüssel, sondern global alles, was mit \$sys\$ im Namen anfängt
 - Andere Schadsoftware kann sich damit einfach durch entsprechende Namensgebung mit Hilfe des *Rootkits* tarnen



Blue Pill – VM-basiertes Rootkit

- Entdeckung und Entfernung von *Rootkits* auf Betriebssystemebene ist möglich
 - Allerdings aufwendig
- Ziel: „unauffindbares“ *Rootkit*
- „Blue Pill“ soll einen PC ohne Neustart des Systems unter die Kontrolle eines *Rootkits* bringen
 - Ausnutzung von Hardware-Virtualisierungstechniken aktueller CPUs
 - keine Leistungseinbußen des Rechners
 - alle Geräte, wie etwa Grafikkarten, sind für das Betriebssystem weiterhin voll zugänglich
- Unauffindbar, da das Betriebssystem nicht merkt, dass es in einer virtuellen Maschine läuft
 - ... aber es gibt doch Seiteneffekte, die es erlauben, auch solche *Rootkits* zu entdecken



Fazit

- Sicherheit in vernetzten Umgebungen immer relevanter
 - Extrem hoher Schaden durch Viren, *Phishing*, *Botnets*, ...
 - Auch erfahrene Computerbenutzer sind nicht sicher!
- Sicherheitsüberprüfungen in Code unerlässlich
 - Automatisierte Tests finden nicht alle Fehler
 - Manuelle *Audits* sind weiterhin erforderlich
 - Dennoch sind Sicherheitsprobleme unvermeidlich
 - 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 ...