Themen zur Computersicherheit

Autorisierung

PD Dr. Reinhard Bündgen
buendgen@de.ibm.com

Ziel der Autorisierung

Zugriffskontrolle

- Kontrolle des Zugriffs auf Ressourcen
- Wer darf mit welchen Ressourcen was tun?
  - Subjekte: Teilnehmer, Rollen, Prozesse
  - Objekte: Dateien, Sockets, Programme, Prozesse
  - Zugriffsarten: lesen, schreiben, verändern, ausführen, öffnen, erzeugen, tracen, …
- Durchsetzung der Zugriffskontrolle
  - von Nutzer bestimmt (discretionary access control – DAC)
  - vom System bestimmt (mandatory access control – MAC)

Modelle zur Autorisierung

- Zugriffsmatrix
- Rollenbasierter Zugriff
- Bell-LaPadula

Zugriffsmatrix

Definition Seien

- t ein Zeitpunkt
- O_t eine Menge von Objekten zum Zeitpunkt t
- S_t eine Menge von Subjekten zum Zeitpunkt t mit $S_t \subseteq O_t$
- A eine endliche Menge von Zugriffsrechten

dann ist M_t: S_t x $O_t \rightarrow 2^A$ eine *Zugriffsmatrix zum Zeitpunkt t* wobei $M_t(s,o) = \{a_1,...,a_n\}$ heißt, das Subjekt s hat zum Zeitpunkt t die Zugriffsrechte $a_1,...,a_n$ für Objekt o.

Notation
2^A: Potenzmenge von A

Beispiel einer Zugriffsmatrix

	Datei 1	Datei 2	Uwe	Prozess 1	Prozess 2	Prozess 3	Prozess 4
Uwe						execute	
Prozess 1					control, send		control
Prozess 2				wait, signal		control	
Prozess 3	read, write	write, owner			receive		send
Prozess 4			create, delete			send receive	

Dynamische Zugriffsmatrizen

- Statische Zugriffsmatrix: $M = M_t$ für alle Zeitpunkte t
- Dynamische Zugriffsmatrix: M_t kann sich mit der Zeit ändern
  - Änderung der Rechte:
    - $M_{t'} = \text{change}(M_t, M^+, M^-)$ mit $M_{t'}(s,o) = M_t(s,o) \cup M^+(s,o) \setminus M^-(s,o)$ für alle s,o und $t' > t$
  - Änderung der Subjekt und Objektmengen
  - Beispiele
    - Änderung von Dateizugriffsrechten (chmod)
    - Erzeugung / Löschung von Dateien, Prozessen

Beschreibung von Sicherheitseigenschaften

Safety-Problem

- Die Relation $\models_{S'}$ beschreibt eine mögliche Änderung einer Zugriffsmatrix dadurch, dass ein Subjekt $s \in S' \subseteq S$ eine ihm erlaubte Operation ausführt.
- Sei $\models_{S'}^*$ die reflexiv-transitive Hülle von $\models_{S'}$
- Frage: Sei a eine unzulässiges Zugriffsrecht auf das Objekt o für das Subjekt s. Gibt es zu gegebener Zustandsmatrix M_t zum Zeitpunkt t einen Zeitpunkt t' und Operationen von Subjekten in S', so dass $M_t \models_{S'}^* M_{t'}$, so dass $a \in M_{t'}(s,o)$?
- Das Safety-Problem ist unentscheidbar.

Soll-Ist Vergleiche

- Gegeben eine möglicherweise informelle Spezifikation der gewünschten Zugriffsrechte und Zugriffsbeschränkungen: Erfüllt Ein System mit gegebener Zugriffsmatrix diese Spezifikation?
- Zugriffsmatrix beschreibt keine Zugriffsbeschränkungen.
- Zugriffsrechte können in Zugriffsmatrix implizit beschrieben werden
  - execute $\in$ M(Uwe, Process3), read $\in$ M(Prozess3, Datei1) => Uwe kann Datei1 lesen

Rollenbasiertes Zugriffsmodell

- role based access control (RBAC)

Definition Seien

- U eine Menge von Teilnehmern,
- O eine Menge von Objekten,
- R eine Menge von Rollen,
- A eine Menge von Rechten,
- ur: $U \rightarrow 2^{R,}$ mit s darf Rolle r einnehmen wenn $r \in ur(u)$,
- ra: $R \rightarrow 2^{A}$ mit r hat Recht a wenn $a \in ra(r)$,
- ses $\subseteq U \times 2^{R}$ die Menge der aktuellen Sitzungen, wobei $R' \subseteq ur(u)$ für alle $(u,R') \in ses$

Dann gilt für die Sitzung (u,R'), dass u die Rechte $\cup_{r \in R'} ra(r)$ hat.

RBAC Beispiel

Beispiel : Bank

- Teilnehmer: Sonja, Uwe, Klaus
- Rollen: Kunde, Kassierer, Kundenbetreuer, Zweigstellenleiter, Kassenprüfer
- Objekte: Kundenkonten, Personaldaten, Kundendaten; Kreditdaten
- Rechte: Geld einzahlen, Geld abheben, Konto sperren, Kreditrahmen erhöhen
- ur(Sonja) = {Zweigstellenleiter, Kunde}
- ur(Uwe) = {Kassierer, Kundenbetreuer, Kunde}
- ur(Klaus) = {Kassenprüfer}
- Einschränkungen:
  - kein Teilnehmer darf sowohl die Rolle eines Kassierers als auch die eines Kassenprüfers haben (statischen Aufgabentrennung)
  - keine Sitzung darf sowohl die Rollen eines Kundenbetreuers als auch die eines Kunden haben (dynamische Aufgabentrennung)

Hierarchische rollenbasierte Modelle

- Gegeben eine partielle Ordnung ≥ über den Rollen mit aus $r_1 \geq r_2$ folgt, dass r_1 alle Zugriffsrechte von r_2 hat.
- Beispiel:
  - Zweigstellenleiter > Kassierer
  - Zweigstellenleiter > Kundenbetreuer
  - Kassenprüfer > Kassierer

partielle Ordnung:
- reflexiv,
- transitiv,
- anti-symmetrisch

Das Bell-LaPadula Modell

- 1973 von D. E. Bell und L. J. LaPadula (im Auftrag der US Air Force) entwickelt
- Oft auch mit Multi Level Security (MLS) bezeichnet
- Ziel Schutzvertraulicher Information
- Implementierungen:
  - trusted Solaris
  - SELinux
  - z/OS - RACF

Bell-LaPadula Modell - Definitionen

- aufbauend auf dynamischen Zugriffsmatrixmodell
- Zugriffsrechte: A = {read-only, append, execute, read-write, control}
- Geordnete Menge von Sicherheitsmarken M
  - z.B. Zugriffsklassifizierung
- Menge von Sicherheitskategorien K
  - z.B. Rollen mit Zugriff
- Geordnete Menge von Sicherheitsklassen $C = M \times 2^K$
  - mit $(m,k) \leq (m',k') \iff (m \leq m' \wedge k \subseteq k')$
- sc: $S \rightarrow C$ ist die Clearance eines Subjekts
- sc: $O \rightarrow C$ ist Klassifikation eines Objekts
- Ein Subjekt kann sich mit der aktuellen Sicherheitsklasse $sc_{akt}(s)$ einloggen wenn $sc_{akt}(s) \leq sc(s)$

BLP Beispiel Krankenhaus

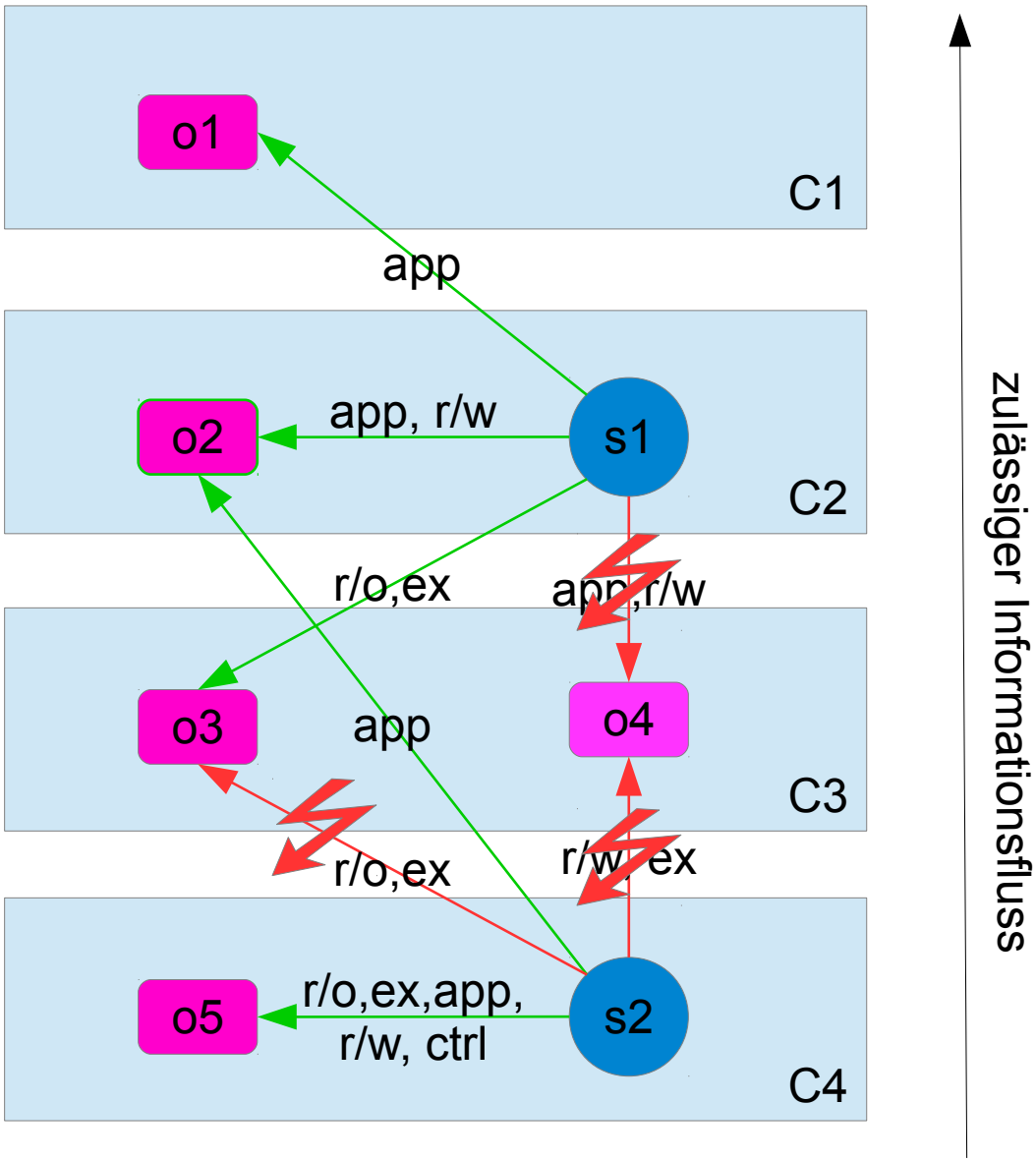
- S = { Achim, Claudia, Lotte, Frieder, Hans }
- O = { Krankenakte, Forschungsergebnis, Behandlungsplan, Gehaltsabrechnung }
- M = { geheim, vertraulich, unklassifiziert } mit geheim ≥ vertraulich ≥ unklassifiziert
- K = { Chefarzt, Arzt, Schwester, Patient, Verwaltung, Besucher }
- C = { (geheim, ∅), (geheim, {Chefarzt}), (geheim, {Chefarzt, Arzt}), (vertraulich, ∅), (vertraulich, {Arzt, Schwester}), (vertraulich, {Schwester}), (vertraulich, {Verwaltung}), … }
- Ordnung über C:
  - (geheim, ∅) ≥ (vertraulich, ∅),
  - (vertraulich, {Arzt, Schwester}) ≥ (vertraulich, {Arzt})
  - ¬ ((vertraulich, {Schwester}) ≥ (vertraulich, {Verwaltung}))
  - ¬ ((vertraulich, {Schwester}) ≤ (vertraulich, {Verwaltung}))

BLP Zugriffsregeln

- Ziel: verhindere unzulässige Informationsflüsse
- Simple security / no-read-up
  - $s \in S$ darf zum Zeitpunkt t den Zugriff $a \in$ {read, execute} auf Objekt $o \in O$ durchführen, wenn $a \in M_t(s,o) \wedge sc(s) \geq sc(o)$
- ✱-Eigenschaft / no-write-down
  - $s \in S$ darf zum Zeitpunkt t den append Zugriff auf Objekt $o \in O$ durchführen, wenn append $\in M_t(s,o) \wedge sc(s) \leq sc(o)$
  - $s \in S$ darf zum Zeitpunkt t den read-write Zugriff auf Objekt $o \in O$ durchführen, wenn read-write $\in M_t(s,o) \wedge sc(s) = sc(o)$

Beispiel BLP Zugriffsregeln

- C1, C2, C3, C4 ∈C
- C1 > C2 > C3 > C4
- s1, s2 ∈S
- o1, o2, o3, o4, o5 ∈O
- Zugriffe
  - app = append
  - r/w = read-write
  - r/o = read-only
  - ex = execute



Diskussion von BLP

- Information sammelt sich „oben“
  - vertrauenswürdige Subjekte (trusted subjects) dürfen ✱-Eigenschaft verletzen
- append wird Subjekt mit niedrigster Klassifikation nie verboten (außer über Zugriffsmatrix)
- Informationsfluss über verdeckte Kanäle
  - z.B. über (Nicht)Existenz von Objekten auf die nur mit append zugegriffen werden kann
- Modellierung unterschiedlicher Sitzungen/Rollen eine Teilnehmers zu unterschiedlichen Zeitpunkten
  - Chinese Wall Model

Übung

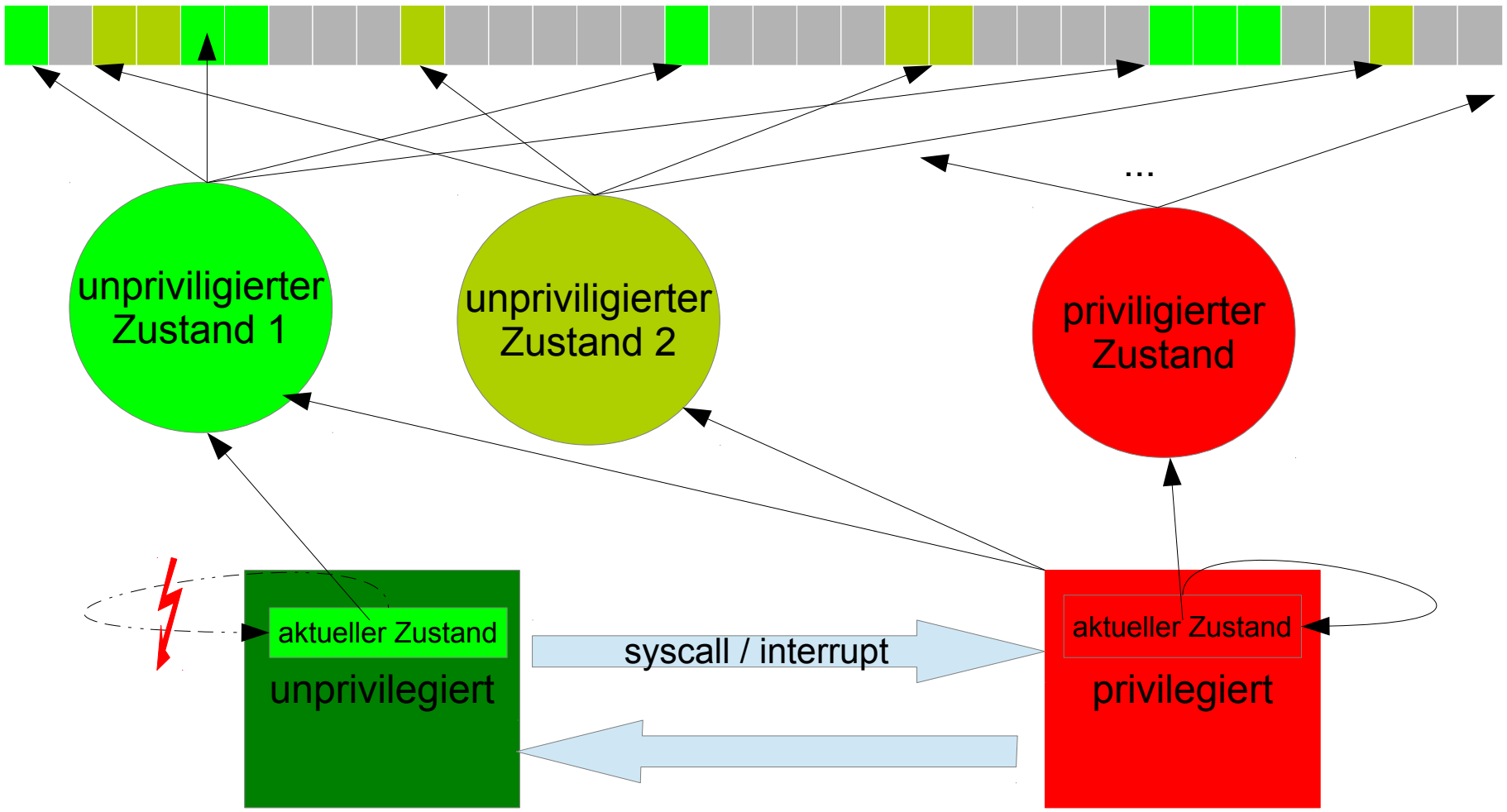
Fallbeispiel Universität

- Rollen: Professoren, Wiss. Mitarbeiter, Prüfungsausschuss, Sekretäre, Studierende, Angestellte im Studentensekretariat, Eltern der Studierenden, Polizisten, Journalisten, ...
- Objekte: Forschungsergebnisse, Vorlesungsfolien, Übungsblätter, Noten Studierendenausweise, e-mail-Adressen, Fachbereichsbudget, …
- Wer soll Zugriff auf welche Daten (Objekte) haben?
- Welche Zugriffsrechtmodellierung passt am besten?

HW Kontrollmechanismen (MAC)

- privilegierter/nicht privilegierter Modus der CPU
  - privilegierte Instruktionen
  - Moduswechsel nur über spezielle Ereignisse (Unterbrechung, Systemruf) möglich
- Virtuelle Adressräume
  - im nicht-privilegierten Modus werden nur virtuelle Adressenbenutzt
  - Adressraumverwaltung nur im privilegiertem Modus möglich
- Unified Extensible Firmware Interface (UEFI) Secure Boot
- Trusted Platform Module (TPMs)
- Hardware Security Module (HSMs)

CPU Modi & Virtueller Speicher



Traditionelle Unix-Zugriffskontrolle

- Discretionary Access Control (DAC)
- Jede Datei hat einen Besitzer und eine Gruppe
- Jeder Teilnehmer ist Mitglied einer oder mehrerer Gruppen (/etc/group)
- Jede Datei kennt drei Zugriffsarten: read, write, execute (r,w,x)
  - die jeweils für den Besitzer, die Gruppe der Datei oder alle Teilnehmer des Systems getrennt vergeben oder verwehrt werden
  - für Verzeichnisse beschreibt der execute Zugriff, den Durchgriff durch das Verzeichnis
- setuid und setgid Bits für ausführbare Dateien:
  - Prozess, der Datei ausführt, bekommt Dateibesitzer als effektive uid bzw Dateigruppe als effektive gid
- setgid Bit im Verzeichnis:
  - Dateien im Verzeichnis erben den Besitzer des Verzeichnisses
- sticky Bit im Verzeichnis:
  - Teilnehmer darf eine Datei in dem Verzeichnis nur dann löschen, wenn er Schreibzugriff auf das Verzeichnis hat und zusätzlich entweder Besitzer der Datei oder des Verzeichnisses ist oder Superuser ist.

NFSv4 ACLs

- Viele neue Unix System unterstützen NFSv4 Zugriffssteuerungslisten (access control lists, ACLs) via extended attributes
- z.B. AIX, FreeBSD, MAC OS X, Solaris (ZFS), Linux (Ext3, Ext4)
- ext3:
  - Montageoption „-o acl“
  - setfacl Werkzeug verwaltet ACLs
    - setfacl -m u:buendgen:rw /home/vorlesung/script.tex
  - getfacl zeigt ACLs
- MAC OS X
  - z.B ls -l -e

Beschränkung der Allmacht von Root

- setuid Programme
- Systeme ohne Root Login
  - z.B. OS X, Ubuntu
  - sudo
- Linux Capabilities
- SELinux

Linux Capabilities (I)

- Voraussetzungen zur Unterstützung von Capabilities:
  - Linuxkern überprüft für jede privilegierte Operation, ob aufrufender Thread über ausreichende capabilities verfügt
  - Linuxkern unterstützt Systemrufe um die Capabilities eines Threads zu inspizieren bzw zu ändern
  - Das Dateisystem muss die Assoziation von Capabilities mit ausführbaren Dateien unterstützten

Linux Capabilities (II)

- Capability: binärer Wert, der wenn gleich 1 eine Erlaubnis bezeichnet
  - Thread Capabilities
  - permitted T(p): maximal erreichbare Capabilities
  - inheritable T(i): Capabilities, die bei einem execve-Aufruf geerbt werden können
  - effective T(e): Capabilities, die der Kern überprüft bevor er eine Erlaubnis erteilt
  - cap_bset („bounding set“): beschränkt die durch execve erlangbaren capabilities
- File Capabilities (für ausführbare Programme)
  - permitted: fließen in die permitted capabilities des Threads ein
  - inheritable: ihre Schnittmenge mit den thread capabilities fließen in die permitted capabilities des threads ein
  - effective: 1 Bit, das wenn 1 besagt, dass die effective capabilities des Threads gesetzt werden sollen

Capabilitiy Änderungen (III)

- **mit cap_set_proc**
  - if ¬CAP_SETPCAP then
    - T'(i) ⊆ T(i) ∪ T(p)
    - T'(i) ⊆ T(i) ∪ T(b)
    - T'(e) ⊆ T(p)
    - T'(b) = T(b)
  - else
    - T'(p) ⊆ T(p), T'(b) ⊆ T(b)
- **UID Änderungen**
  - alle uids nach nicht-0:
    - T'(p) = Ø, T'(e) = Ø
  - euid nach nicht-0:
    - T'(e) = Ø
  - nicht-0 euid nach 0:
    - T'(e) = T(p)
- **exeve(F) (non-setuid Datei)**
  - T'(p) = (T(i) ∩ F(i)) ∪ (F(p) ∩ T(b))
  - if F(e) then
    - T'(e) = T'(p)
  - else
    - T'(e) = Ø
  - T'(i) = T(i)
  - T'(b) = T(b)
- **fork**
  - T'(p) = T(p)
  - T'(e) = T(e)
  - T'(i) = T(i)
  - T'(b) = T(b)

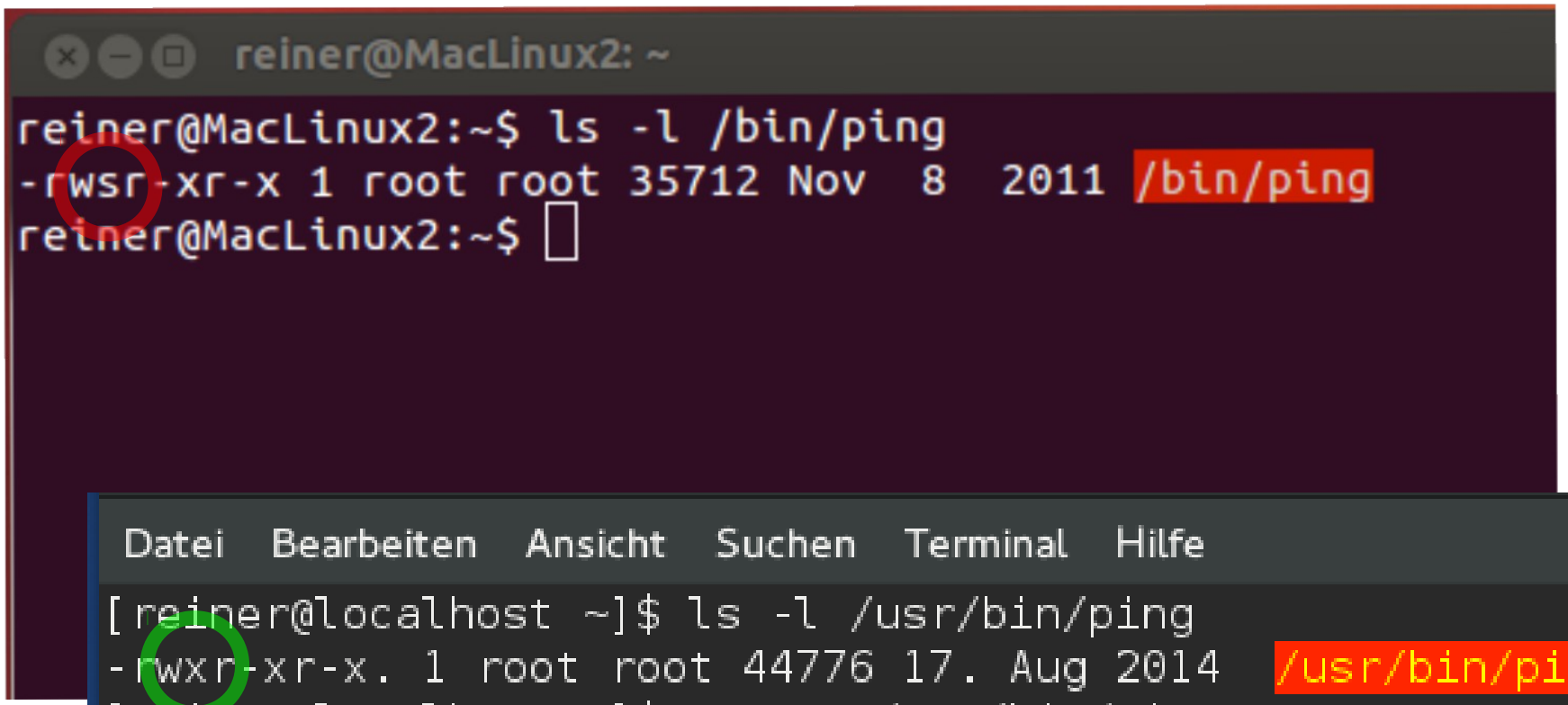
Notation
T: Thread Capability vor Änderung
T': Thread Capability nach Änderung
F: Datei Capability vor Änderung
F': Datei Capability nach Änderung

Capablities (IV)

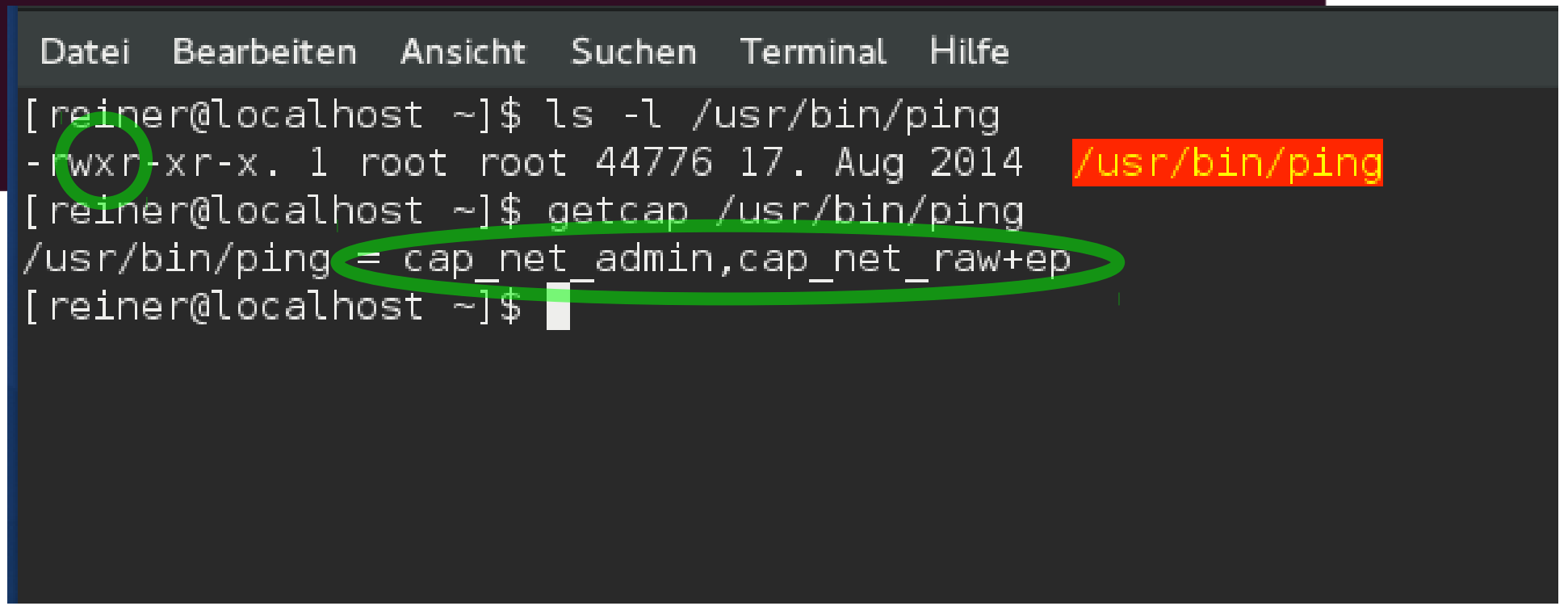
- Problem: nicht-root Threads können keine Datei Capabilities erben, wenn F(i) leer
- neues „ambient“ Capability Set T(a) mit:
  - $T(a) \subseteq T(p) \cap T(i)$
- **exeve(F)**
  - if F is setuid or setgid or has file caps then
    - $T'(a) = \emptyset$
  - else
    - $T'(a) = T(a)$
  - $T'(p) = (T(i) \cap F(i)) \cup (F(p) \cap T(b)) \cup T'(a)$
  - if F(e) then
    - $T'(e) = T'(p)$
  - else
    - $T'(e) = T'(a)$
  - $T'(i) = T(i)$
  - $T'(b) = T(b)$

Capabilities Beispiel

Ubuntu



Fedora



Dokumentation zu Linux Capabilities

- man (7) capabilities
- über das ambient capability set:
  - https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=58319057b7847667f0c9585b9de0e893b0fdb08
  - http://lwn.net/Articles/632520/

FLASK / SELinux

- Entwickelt von der NSA
- Flask / SELinux: http://www.nsa.gov/research/selinux/docs.shtml
- Unterstützte MAC Formen
  - Type Enforcement (TE) durch Regelwerk (policy) gesteuert
  - RBAC
  - Multi-Level Security (MLS) / Multi-Category Security (MCS) gemäß BLP

SELinux Komponenten

- Subjekte (Prozesse, Teilnehmer)
- Objekte (Prozesse, Dateien, Dateisysteme, Netzwerke, Ports, HW, …)
  - gehört zu einer Objektklasse, die die anwendbaren Zugriffstypen definiert (z.B. Klasse file: read, write, append, ...)
- Sicherheitskontext String der Form *user*:*role*:*type*[:*level*]
  - *user*: SELinux user ≠ Unix user, assoziiert mit einer oder mehreren SE Linux Rollen
  - *role*: SE Linux Rolle, assoziiert mit einem oder mehreren SE Linux Typen
    - nur für Prozesse relevant
    - alle Dateien haben die generische Rolle object_r
  - *type*: SE Linux Typ
    - für Subjekt: definiert auf welche Objektezugegriffen werden kann
    - für Objekt: definiert welche Subjekte auf Objekt zugreifen dürfen
  - *level*: eine Sicherheitsklasse, oder ein Bereich von Sicherheitsklassen (BLP)
    - optional

MAC-Durchsetzung im OS Kern

- Wird überprüft, wenn Zugriffsrecht gemäß Unix DAC gegeben
- Viele Systemrufe des Linux Kern sind mit „hooks“ für Linux Security Module (LSMs) instrumentiert
- SE Linux ist ein solches LSM
- Sicherheitsserver assoziiert Sicherheitskontexte aktiver Kernobjekte mit Sicherheits-Ids (SIDs)

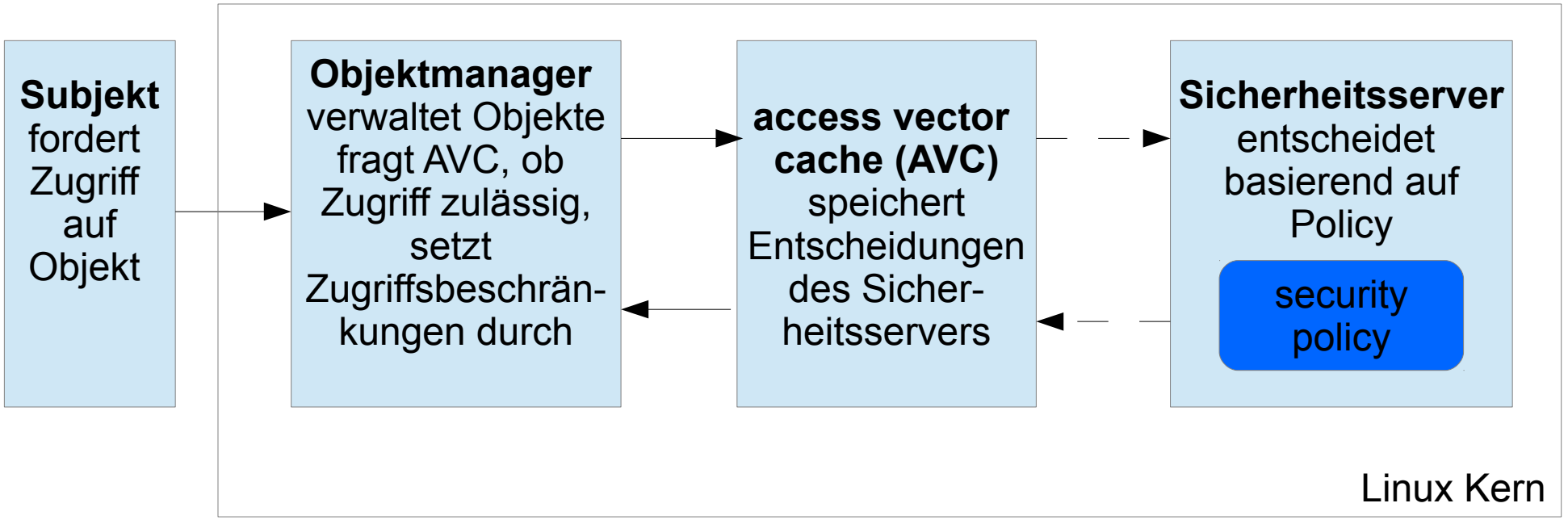


Illustration aus The SELinux Notebook

- Processing a system call: Figure 2.3

Objektmarkierungen

- jedes Objekt hat eine Sicherheitsmarkierung (security label)
  - Sicherheitskontext bzw SID
- für neues Objekt erfragt Objektmanager eine Sicherheitsmarkierung vom Sicherheitsserver
- Markierungsentscheidung basiert auf
  - Markierung des erzeugenden Subjekts
  - Markierung eines verwendeten Objektes
  - von der Klasse des erzeugten Objektes
  - Beispiel
    - `fd=open(„/home/buendgen/beispiel.txt“, …);`
    - Markierung von `fd` hängt ab von der Markierung des Prozesses, der open aufruft
    - von der Markierung der Datei `/home/buendgen/beispiel.txt`
    - von der Tatsache, dass `fd` zur Klasse file descriptor gehört
  - Details der Entscheidung werden durch policies (oder Defaults) beschrieben
  - Sicherheitsmarkierungen von Dateiobjekten werden persistent in Dateisystem gespeichert

Zugriffsentscheidungen

- Zugriffsentscheidung
  - Paar von Sicherheitsmarkierungen
  - Pro Zugriffsart ist eine Zugriffsentscheidung nötig
  - Berechnete Zugriffsentscheidungen werden im AVC gespeichert
- Manche Operation umfassen mehrere Zugriffsarten (möglicherweise zwischen mehreren Objekten)
  - Beispiel
    - die unlink(Datei) Operation benötigt
      1. die unlink Erlaubnis für die Datei und
      2. die remove_name Erlaubnis für das Verzeichnis, in der die Datei liegt

Zugriffsrechte (Auswahl)

- Prozesse
  - transition of security label
  - entrypoint
  - execute
  - inherit open files
  - send signals
  - trace other processes
  - Prozessverwaltung: fork, wait, sched, setpgid, setpriority, …
- Netzwerk
  - Kommunication zwischen sockets
  - send/receive Nachrichten über Netzwerkschnittstellen
  - connectto / acceptfrom
  - port number association
- Dateisysteme/Dateien
  - mount
  - mounton
  - Dienste: statfs, creat, stat, link, rename, unlink, rmdir
  - append
  - write
  - add_name (Verzeichnis)
  - remove_name (Verzeichnis)
  - reparent

Ausgewählte Zugriffsrechte (gemäß FLASK Paper*)

Call	Steuerunganforderungen			
	Klasse des Arg.	**Erlaubnis**	**Source SID**	**Target SID**
fork	process	fork	current	current
execve	dir	search	current	path
	file	execute	current	file
	process	transition	current	new
	process	entrypoint	new	file
	process	execute	new	file
	process	ptrace	parent	new
	fd	inherit	new	fd
write	fd	setattr	current	fd
	file	write	current	file
	file	append	current	file
connect	socket	connect	current	client socket
	socket	connectto	client socket	server socket
	netif	tcp_send	client socket	netif
	node	tcp_send	client socket	node
	netif	tcp_recv	server socket	netif
	node	tcp_recv	server socket	node

*) Loscocco & Smalley „Integrating Flexible Support for Security Policies into the Linux Operating System“, 2001

Policy-Sprache (Ausschnitt)

Typ Regeln

- Typübergang (type transition) für Prozesse:
  - `type_transition` T1 T2`:` `process` T3
  - Prozess vom Typ T1, der Programmdatei vom Typ T2 ausführt, erhält den Typ T3
  - z.B. `type_transition initrc_t acct_exec_t: process acct_t`
- TE-Zugriffsvektorregeln
  - `allow` T1 T2 `:` C P
  - Subjekt vom Typ T1 hat Erlaubnis P auf Objekte der Klasse C und des Typs T2
  - z.B. `allow initrc_t acct_exec_t: file execute`

Zusicherungen

- TE Zugriffsvektorzusicherungen
  - `neverallow`

Nebenbedingungen (Constraints)

- `constrain` C P E
- Subjekte der Klasse C bekommen die der Erlaubnis P nur wenn die Bedingung E erfüllt ist
- z.B, `constrain process transition (u1==u2 or t1==system_t)`

Illustration aus The SELinux Notebook

- Abschnitt 2.12.1 Domain Transition
  - Seiten 43ff
  - Figure 2.7

User und Rollen in SELinux

- Rollenmitgliedschaft
  - `user` U `roles` R
  - user U hat Rolle(n) R
  - z.B. `user sysadm_u roles { sysadm_r storageadm_r }`
- Typassoziation einer Rolle
  - `role` R `types {` T1 T2 ... `}`
  - Rolle R umfasst die Typen T1, T2, ...
  - z.B. `role user_r types { mail_t edit_t browse_t }`
- Rollendominanz
  - `dominance { role` R1 `{ role` R2 `}; }`
  - R2 ist Teilrolle von R1
  - z.B. `dominance { role big_r { role small_r}; }`
- Rollenübergang
  - `allow` R1 R2
  - bei Transition kann die Rolle R1 in die Rolle R2 übergehen
  - z.B `allow sysadm_r secadm_r`

Illustration aus The SELinux Notebook

- RBAC
  - Seiten 25, Figure 2.4

Multi Level Security (MLS) / Multi Category Security (MCS)

- user:role:type:sensitivity[:category,…] [- sensitivity[:category, …]]
  - sensitivity: BLP Marke
  - level: sensitivity[:category,…] BLP Sicherheitsklasse
- BLP Ordnung über Sicherheitsklassen
  - dominance constraints
    - `mlsconstrain` class access `(`L1 `dom`|`domby` L2`)`
      - `mlsconstrain file read (`l1 `dom` l2`);` # no read up
      - `mlsconstrain file write (`l1 `domby` l2`;` # no write down
- Beispiel: dynamisches Labeln von virtuellen Maschinen
  - VM1 process: system_u,system_r,svirt_tcg_t:s0:c585,c813
  - VM1 image: system_u,system_r,svirt_image_t:s0:c585,c813
  - VM2 process: system_u,system_r,svirt_tcg_t:s0:c535,c601
  - VM2 image: system_u,system_r,svirt_image_t:s0:c535,c601
  - shared image: system_u,system_r,svirt_image_t:s0

Anzeigen der SELinux Konfiguration

SELinux Modi: enforcing, permissive, disabled

Dateien

```
# ls -Z file1

-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
```

Prozesse

```
# ps -eZ
system_u:system_r:dhcpc_t:s0             1869 ?  00:00:00 dhclient
system_u:system_r:sshd_t:s0-s0:c0.c1023  1882 ?  00:00:00 sshd
system_u:system_r:gpm_t:s0               1964 ?  00:00:00 gpm
```

Zuordnung von Linux Teilnehmern zu SELinux usern

```
# semanage login -l
Login Name                SELinux User              MLS/MCS Range
__default__               unconfined_u              s0-s0:c0.c1023
root                      unconfined_u              s0-s0:c0.c1023
system_u                  system_u                  s0-s0:c0.c1023
```

Illustration aus The SELinux Notebook

- Überblick über SELinux Ökosystem
- Seite 20, Figure 2.2

SELinux Dokumentation

- FLASK Paper
  - Loscocco & Smalley „Integrating Flexible Support for Security Policies into the Linux Operating System“, 2001
  - https://www.nsa.gov/research/_files/publications/security_policies_linux_os.pdf
- The SELinux Notebook
  - frei verfügbares pdf
  - http://freecomputerbooks.com/books/The_SELinux_Notebook-4th_Edition.pdf
- SELinux Wiki
  - http://selinuxproject.org/page/Main_Page
- SELinux in Fedora
  - https://fedoraproject.org/wiki/SELinux

Audit

- protokollieren aller Sicherheitsrelevanten Aktionen: audit (log)
- Audit-Logfunktion, darf nicht unerkannt abgestellt werden können
- Audit-Logs dürfen nicht modifizierbar sein
  - Zeitstempel, Sequenznummern
  - senden an ferne Systeme
  - signieren
- Audit-Logs dürfen keine Geheimnisse enthalten