



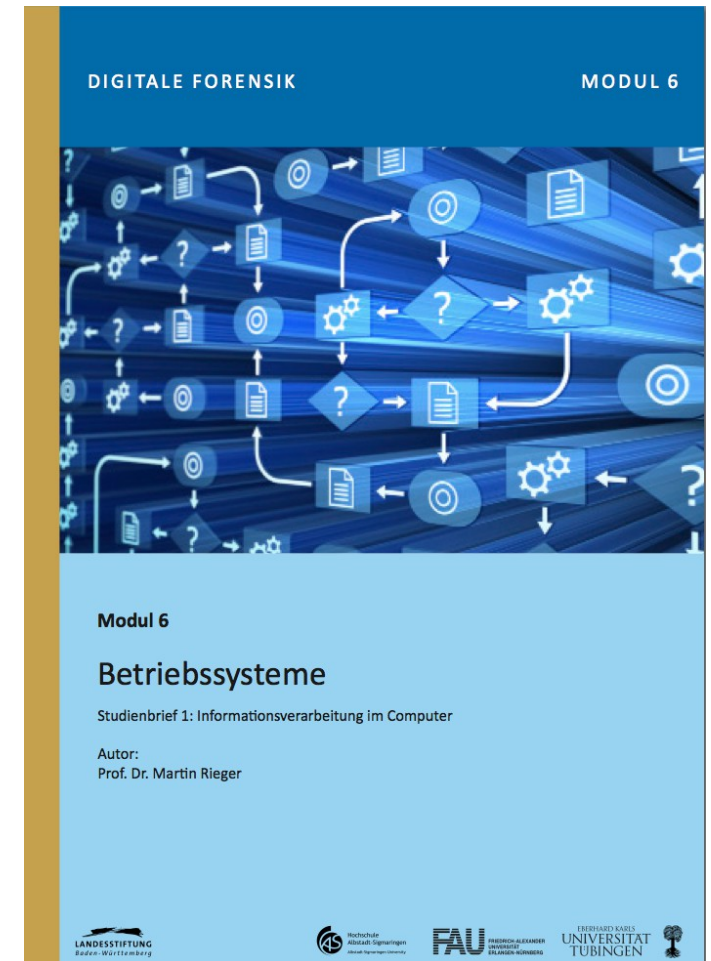
Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Betriebssysteme

SS 2013

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

Foliensatz E
SB 5 (11.04.2013)
ACLs und Capabilities



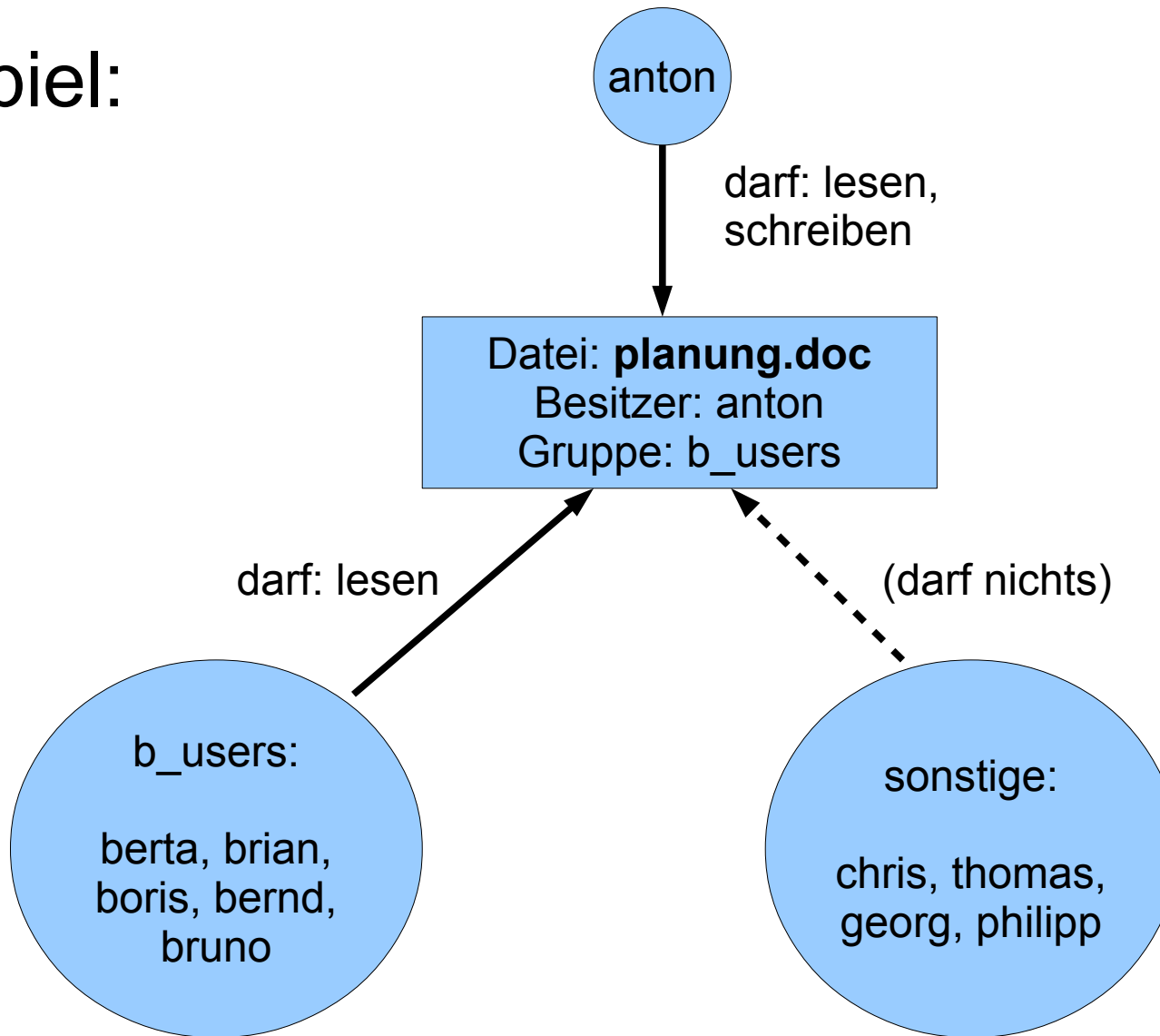


- aus Studienbrief 5 (5.8.4, 5.8.5)
- alternative Erklärung
- Referenzen:
 - Jonathan Shapiro: *What is a capability, anyway?*
<http://www.eros-os.org/essays/capintro.html>, 1999
 - Michael Bacarella: „Taking Advantage of Linux Capabilities“, *Linux Journal*, 2002,
<http://www.linuxjournal.com/article/5737>



- Zugriffsschutz für Dateien funktioniert traditionell (z. B. Unix) so:
 - Datei hat einen Besitzer
 - Datei ist einer Benutzergruppe zugeordnet
 - Der Besitzer legt für sich selbst und für die Gruppe Zugriffsrechte fest (klassisch: rwx)

Beispiel:



lesen =
lesen
kopieren

schreiben =
anhängen
verändern
löschen
Attribute ändern

- Zugriffsrechte dieser Art speichert man an der Datei (Unix: `rw-r-----` für das Beispiel)
- Allgemein nennt man solche Strukturen ACLs (Access Control Lists)
- komplexere Varianten speichern detailliertere Rechte
 - Rechte aufsplitten (lesen, Dateiexistenz abfragen, Dateigröße abfragen, Dateiattribute abfragen, anhängen, verändern, löschen, Zugriffsrechte ändern, ...)
 - Rechte für mehr Benutzer
berta: `r,append`; bernd: `r`; bruno: `rw,del` etc.
 - Rechte für mehrere Gruppen (`b_users`, `c_users`, ...)

- Unter Linux: getfacl, setfacl
 - Dateisystem muss mit Mount-Option `-o acl` eingebunden werden
 - verweisen vom Inode aus auf zusätzliche Rechte

```
[esser@quadamd:mnt]$ setfacl -m u:anton:rw
[esser@quadamd:mnt]$ ls -l datei.doc
-rw-rw-r--+ 1 esser users 0 2012-04-23 12:57 datei.doc
[esser@quadamd:mnt]$ getfacl datei.doc
# file: datei.doc
# owner: esser
# group: users
user::rw-
user:anton:rw-
group::r--
mask::rw-
other::r--
```

- Rechte für weitere Gruppen: `-m g:...`

```
[esser@quadamd:mnt]$ setfacl -m g:b_users:rw
[esser@quadamd:mnt]$ getfacl datei.doc
# file: datei.doc
# owner: esser
# group: users
user::rw-
user:anton:rw-
group::r--
group:b_users:rw-
mask::rw-
other::r--
```

```
berta@quadamd:/mnt/mnt$ id
uid=1003(berta) gid=1004(berta) Gruppen=1004(berta),1003(c_users)
berta@quadamd:/mnt/mnt$ ls -l datei.doc
-rw----- 1 esser users 19 2012-04-23 13:08 datei.doc
berta@quadamd:/mnt/mnt$ cat datei.doc
Das ist ein Test.
```

- Idee bei ACL ist immer, zu einer bestimmten Datei oder einem Ordner Rechte zu vergeben

- Columns of access control matrix

	<i>file1</i>	<i>file2</i>	<i>file3</i>
<i>Andy</i>	rx	r	rwo
<i>Betty</i>	rwxo	r	
<i>Charlie</i>	rx	rwo	w

ACLs:

- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

- Capabilities drehen diesen Ansatz um
 - betrachte Zeilen in der Rechte-Matrix
 - speichere Rechte beim Benutzer / Prozess
 - Rows of access control matrix

	<i>file1</i>	<i>file2</i>	<i>file3</i>
<i>Andy</i>	rx	r	rwo
<i>Betty</i>	rwxo	r	
<i>Charlie</i>	rx	rwo	w

C-Lists:

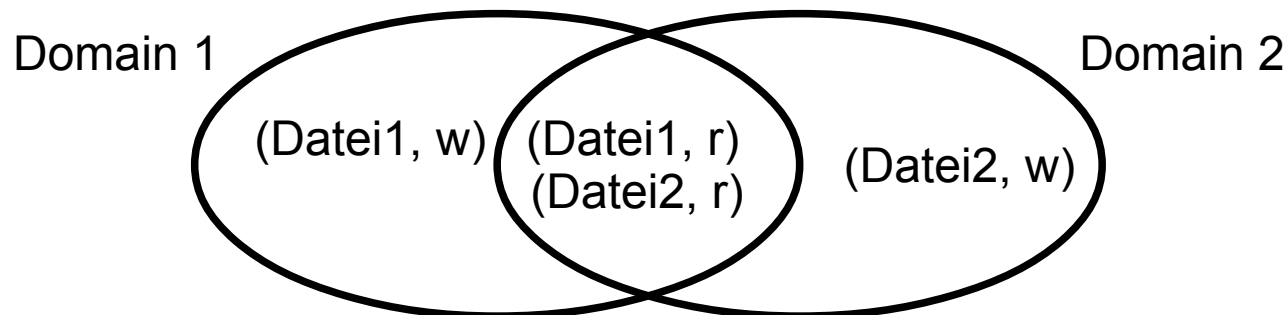
- Andy: { (file1, rx) (file2, r) (file3, rwo) }
- Betty: { (file1, rwxo) (file2, r) }
- Charlie: { (file1, rx) (file2, rwo) (file3, w) }

Quelle: Matt Bishop



- werden oft mit Schlüsseln verglichen
 - Schlüssel gewährt mir ein spezielles Nutzungsrecht
 - „valet parking key“: öffnet beim Auto Türen und startet Motor, öffnet *nicht* Kofferraum
 - Standardschlüssel: Vollzugriff
 - nicht relevant, wer den Schlüssel benutzt

- **Schutzdomäne (protection domain):**
abstraktes Konzept für Zugriffsrechte
 - Domain ist Sammlung von (Objekt, Zugriff)-Paaren,
z. B. { (Datei1, r), (Datei1, w), (Datei2, r) }



- Domain kann Benutzern zugeordnet sein (UNIX),
z. B.: anton \leftrightarrow Domain 1, berta \leftrightarrow Domain 2

- Wir haben also
 - Objekte (Datei1, Datei2, ...)
 - Zugriffsarten (r, w, x, ...)
 - Domänen, Benutzer
- Jedes Recht ist ein Tripel
 - z. B. (Anton, Datei1, r)
- ACLs und Capabilities: verschiedene Ansätze, um solche Informationen zu speichern



ACLs vs. Capabilities

- Prinzipiell: ACLs und Capabilities gleichwertig
 - kann zu einem Benutzer Liste aller erlaubten Datei/Zugriff-Kombinationen speichern
(Benutzer darf mit welchen Dateien was tun?)
 - kann zu einer Datei Liste aller zugelassenen Benutzer/Zugriff-Kombinationen speichern
(Datei kann von wem wie manipuliert werden?)
- Verwaltung der Rechte, Entzug von Rechten unterschiedlich aufwendig

- Beispiel: Benutzer sämtliche Rechte entziehen
 - ACLs: gesamtes Dateisystem durchsuchen und bei jeder Datei Rechte entfernen (aufwendig)
 - Cap.: bei Benutzer die Cap's entfernen (schnell)
- Beispiel: Zugriff auf Datei sperren
 - ACLs: Zugriffsrechte an Datei entfernen (schnell)
 - Cap.: Für alle Benutzer nach einer Cap. für diese Datei suchen und entfernen (eher aufwendig)
- (es gibt mehr Dateien als Benutzer)

- Der Begriff „Capabilities“ wird nicht einheitlich verwendet
- z. B. POSIX-Capabilities (UNIX):
 - steuert die Möglichkeit, Systemdienste zu nutzen:
 - Dateizugriff, Netzwerkzugriff etc.
 - Rechte werden an Programme vergeben (unabhängig vom Benutzer!)
 - z. B. `CAP_SYS_RAWIO`: Zugriff auf Hauptspeicher
 - nach Entfernen einer Cap. kein Wiederherstellen
- Cap.-basierte Betriebssysteme
 - Weitergeben von Rechten an andere Nutzer

- Unix: SUID-Bit für Kommando `passwd` (User braucht für Passwortänderung Zugriff auf Datei, die normal für ihn gesperrt ist; durch SUID-Bit kann er den Befehl `passwd` effektiv mit Root-Rechten ausführen → muss auf perfekte Implementierung von `passwd` vertrauen
 - Überprüfung der Berechtigung
 - kein „Ausbruch“ aus Programm möglich
- Recht „Netzwerkkonfiguration“ ohne totale Root-Rechte?
 - `sudo`
 - POSIX Capabilities