G Sicherheitsmodelle

- Ziel: Klassifikation und Einordnung bekannter und weiterer Sicherheitskonzepte
- Fokus in diesem Abschnitt: Zugriffskontrolle
- Beispiel für reale Umsetzung: SE-Linux
- Formalisierung von Sicherheitsmodellen:
  Grundlage für verifizierbare Aussagen
  - ➤ Problem: komplette Sicherheitsmodelle komplexer Anwendungen meist nicht handhabbar
  - ➤ aber: Grundlage für den Nachweis von Eigenschaften konkreter Konzepte

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.1 Objekte und Subjekte

- **Objekte**: die zu schützenden Einheiten des Modells
- Problem: häufig sehr grob-granulare Festlegung der Objekte
  - ➤ z. B. Dateien, Directories
  - ➥ unvollständige Erfassung sicherheitsrelevanter Eigenschaften
    - ➤ z. B. Anlege-/Löschrecht für Directories
    - ➤ Folge: Sicherheitslücken
  - ➥ Zusammenfassung unterschiedlicher Einheiten zu einem Objekt
    - ➤ Verletzung des "need-to-know"-Prinzips

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.1 Objekte und Subjekte (2)

- **Subjekte**: greifen auf Objekte zu
- Problem: grob-granulare Subjekte
  - ➤ Benutzer, Benutzergruppen
  - ➥ Rechte werden an ganze Gruppen vergeben ohne individuelle Bedürfnisse zu berücksichtigen
- Problem: Verbindung Subjekt - reale Person
  - ➤ bei Rechtevergabe erfolgt Bezug auf Vertrauenswürdigkeit einer Person
  - ➤ Rechteausübung erfolgt durch Software, die für die Person agiert
    - ? ist die Software so vertrauenswürdig wie die Person

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.1 Objekte und Subjekte (3)

- Anwendungsspezifische Granularität
  - ◆ Zugriffsrechte für Objekte beliebiger Granularität festlegbar
    - ➤ Persistente Objekte einer Anwendung
    - ➤ Sätze einer Datenbank
- Anwendungsspezifische Rechte
  - ◆ nicht nur systemweite Standard-Rechte
    - ➤ z. B. Rechte zum Aufruf bestimmter Methoden (Kontrolle z. B. mit Hilfe des Typ-Systems)
- Feingranulare Subjekte
  - ◆ nicht nur Benutzer und Benutzergruppen
    - ➤ zeitlich- und Rollen-abhängige Rechte

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.2 Zugriffsrechte

- universelle Rechte
  - ➤ erlauben allgemeine, nicht objektspezifische Operationen
  - ➤ z. B. read/write auf Dateien
    - – unabhängig von der Semantik des Dateiinhalts
  - ➤ Realisierung: konventionelle Dateisysteme
- objektspezifische Rechte
  - ➤ Einschränkung von Zugriffsmöglichkeiten auf einen festgelegten funktionalen Kontext
  - ➤ Manipulation eines Objekts nur gemäß der festgelegten Semantik der erlaubten Operation (Typ-spezifische Rechte)
  - ➤ Realisierung:
    - – objektorientierte Middleware
    - – Datenbanksysteme

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.3 Zugriffskontrollstrategien

1 Benutzer-bestimmbare Zugriffskontrollstrategien

Discretionary Access Control (DAC)

- Eigentümer eines Objekts ist für die Rechtevergabe verantwortlich
- Vergabe auf Basis einzelner Objekte
  - ➤ z. B. Datei-Zugriffsrechte unter UNIX (*chmod*)
- Festlegung Objekt-bezogener Sicherheitseigenschaften
- ★ keine System-globalen Sicherheitseigenschaften möglich / festlegbar
  - ➤ Abhängigkeiten zwischen Objekten werden bei Festlegung der Zugriffsrechte nicht zwangsläufig berücksichtigt
    - – Benutzung, Kommunikation, Kooperation zwischen Objekten
      - ➥ Inkonsistenzen in der Gesamtstrategie (z. B. unbeabsichtigte Informationsflüsse)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Systembestimmte Zugriffskontrollstrategien

Mandatory Access Control (MAC)

- Systemweite Regeln dominieren über Benutzer-bestimmte Regeln
  - ➤ Zugriff wird verweigert, auch wenn ihn benutzerspezifische Regeln erlauben würden
  - ➤ Benutzer kann systemweite Regeln weiter einschränken

3 Rollenbasierte Zugriffskontrollstrategien

Role-Based Access Control (RBAC)

- Rechtemanagement nicht auf Basis von Subjekten (Benutzern) sondern an den durchzuführenden Aufgaben (Rollen) orientiert
- Subjekte erhalten Rechte aufgrund der Rolle, die sie gerade ausführen
  - ➤ ggf. zeitlich variabel

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.4 Zugriffskontrollmodelle

1 Zugriffsmatrix-Modell (vgl. Abschnitt D.3)

- Schutzzustand des Systems wird durch Matrix beschrieben
  - ➤ Spalten = Objekte O
  - ➤ Zeilen = Subjekte S
  - ➤ Festlegung von Rechten R zu jedem Tupel (s, o)
- statische Zugriffsmatrix
  - ➤ Modellierung von Anwendungsproblemen mit a priori bekanntem Rechtezustand
  - ➤ z. B. für Sicherheitsstrategien einfacher Router (zustandslose Paketfilter-Firewalls)

	Port 21 (ftp)	Port 25 (smtp)	Port 53 (DNS)	Port 514 (rsh)	> 1023
ftp-Server	receive		send		
Mail-Rechner		receive			send
externer Rechner		send	send		

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Zugriffsmatrix-Modell (2)

- dynamische Zugriffsmatrix
  - ➤ Veränderungen der Matrix durch Operationen
    (z. B. **`create`**/**`destroy`** von Objekten oder Subjekten,
    `enter`/**`delete`** von Rechten)
- Sicherheitseigenschaften
  - ➤ Formalisierung von Schutzzuständen ermöglich Untersuchung von Sicherheitseigenschaften der modellierten Systeme
  - ◆ Safety-Problem
    $r \in M_t(s,o)$: In Schutzzustand M_t hat Subjekt s das Recht r an Objekt o
    - ➤ Objekt o hat das Recht r nicht.
      Ist sicher, dass es das Recht in Zukunft nicht erlangen kann?
      $r \notin M_t(s,o)$. $\nexists\ M_{t'}$ mit $t' > t$ und $r \in M_{t'}(s,o)$?
    - ➤ allgemein nicht entscheidbar,
      für konkrete Fälle meist aber Entscheidung möglich

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Zugriffsmatrix-Modell (3)

- … Sicherheitseigenschaften
  - ◆ Soll-Ist-Vergleiche
    - ➤ erfüllt ein modelliertes System (Ist-Eigenschaften) die gestellten Anforderungen (Soll-Eigenschaften)
    - ➤ Verifikation bei formaler Spezifikation der Eigenschaften möglich
  - ◆ Modellierung von Zugriffsrechten
    - ➤ Modellierung konkreter Zugriffsrecht-Konzepten (z. B. UNIX-Rechte) durch Kommandos zur Modifikation einer Zugriffsmatrix
    - ➤ Basis für formale Überprüfung von Konzepten
      (vgl. Eckert, IT-Sicherheit 5. Auflage, Seite 241 ff)
  - ◆ Einsatz von Zugriffsmatrix-Modellen
    - ➤ erlauben feingranulare Spezifikation von Objekten und Subjekten
    - ➤ Überprüfung von Eigenschaften mit Hilfe von Werkzeugen möglich (Basis: Model-Checking-Ansatz)
    - ➤ Theoretische Grundlage von Flask/SELinux

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Rollenbasierte Modelle

- Definition: $RBAC = (S, O, RL, P, sr, pr, session)$

  S endliche Menge der Benutzer des Systems

  O endliche Menge der zu schützenden Objekte

  RL endliche Menge von Rollen. Jede Rolle beschreibt eine Aufgabe und damit die Berechtigungen der Rollenmitglieder

  P endliche Menge der Zugriffsberechtigungen

  sr Relation der Rollenmitgliedschaft sr: $S \rightarrow 2^{RL}$
  $sr(s) = \{R_1, ..., R_n\} \Rightarrow$ Subjekt s darf in den Rollen $R_1 \ldots R_n$ aktiv sein

  pr Berechtigungsrelation pr: $RL \rightarrow 2^P$
  ordnet jeder Rolle $R \in RL$ die Zugriffsrechte zu, die ihre Mitglieder während ihrer Rollenaktivität wahrnehmen dürfen

  $session$ Relation $session \subseteq S \times 2^{RL}$ beschreibt Sitzungen (s, rl) mit $rl \subseteq sr(s)$
  Für ein Subjekt s und eine Sitzung $(s, rl) \in session$ gilt:
  s besitzt alle Rechte der Rollen $R \in rl$
  Sitzung (s, rl): s ist aktiv in den Rollen $R \in rl$

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Rollenbasierte Modelle (2)

- Beispiel: Bank
  - Rollen: $RL = \{Zweigstellenleiter, Kassierer, Kundenbetreuer, Kunde\}$
  - Objekte: $O = \{Kundenkonten, Kreditdaten, Kundendaten\}$
  - Zugriffsrechte: $P = \{Konto\ sperren, Kreditrahmen\ festlegen, Einzahlung, ...\}$
  - Subjekte: $S = \{Huber, Meier\}$
  - Huber ist Zeigstellenleiter, Meier ist Kundenbetreuer, beide sind auch Kunden ihrer eigenen Bank
    $sr(Huber) = \{Zweigstellenleiter, Kunde\}$
    $sr(Meier) = \{Kundenbetreuer, Kunde\}$
  - Beispiel für Rechtevergabe
    $\{Konto\ sperren, Kreditrahmen\ festlegen\} \subseteq pr(Zweigstellenleiter)$
    $\{Einzahlung, Auszahlung\} \subseteq pr(Kunde)$

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Rollenbasierte Modelle (3)

- Hierarchische RBAC
  - ➤ Aufgaben und Zuständigkeiten sind in der Praxis häufig hierarchisch geordnet — z. B.
    Zweigstellenleiter ≥ Kassenprüfer
    Kassenprüfer ≥ Kundenbetreuer
    Kassenprüfer ≥ Kassierer
  - ➤ Problem: durch Rollenhierarchie erbt z. B. Kassenprüfer alle Rechte von Kassierer, obwohl er diese (z. B. Ein- und Auszahlung auf Kundenkonten) für seine Aufgabe nicht benötigt
- Beschränkte Rollenmitgliedschaft
  - ◆ Regeln zur Aufgabentrennung (*separation of duty*) beschreiben den wechselseitigen Ausschluss von Rollenmitgliedschaften
    - ➤ statische Aufgabentrennung $SSD \subseteq RL \times RL$ mit $(R_i, R_j) \in SSD \Leftrightarrow$ gleichzeitige Mitgliedschaft in den Rollen R_i und R_j ist ausgeschlossen (z. B. Kassenprüfer darf nicht Kassierer sein)
    - ➤ dynamische Aufgabentrennung $DSD \subseteq RL \times RL$ mit $(R_i, R_j) \in DSD \Leftrightarrow$ gleichzeitige Aktivität in den Rollen R_i und R_j ist unzulässig

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Chinese-Wall-Modell (Brewer-Nash-Modell)

- Hintergrund: unzulässige Ausnutzung von Insiderwissen bei Bank- und Börsentransaktionen verhindern
  - ➤ weiteres Einsatzfeld: Informationsfluss zwischen konkurrierenden Unternehmen in Unternehmensberatungen verhindern
- Idee: zukünftige Zugriffsmöglichkeiten eines Subjekts können durch seine Zugriffe in der Vergangenheit beschränkt werden
  - ➤ Basis: Zugriffsmatrix-Modell beschreibt die grundlegenden Rechte
  - ➤ Zugriffshistorie wird aufgezeichnet
  - ➤ Zugriffsentscheidungen auf Basis der Zugriffsmatrix + Zugriffshistorie
- Probleme:
  - ➤ Regeln meist zu restriktiv
  - ➤ Modell erfasst nur den Schutz von Vertraulichkeit, nicht aber Datenintegrität

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Bell - La Padula - Modell

- erstes vollständig formalisiertes Sicherheitsmodell
  - ➤ 1973 im Auftrag der US Air Force entwickelt
- Basis: dynamisches Zugriffsmatrix-Modell
  $R = \{read\text{-}only, append, execute, read\text{-}write, control\}$
  (*control* erlaubt Rechteweitergabe und -rücknahme)
- ***Multi-Level-Security (MLS)*** - Modell
  - ➤ Subjekte und Objekte werden mit einer Sicherheitsstufe markiert
- regelt den Fluss von Informationen
  - ◆ Simple-Security-Regel (*no-read-up* Regel)
    - ➤ ein Subjekt auf Sicherheitsstufe *k* kann nur Objekte von Sicherheitsstufe ≤ *k* lesen oder ausführen
  - ◆ *-Regel (*no-write-down* Regel)
    - ➤ ein Subjekt auf Sicherheitsstufe *k* kann nur auf Objekte von Sicherheitsstufe ≥ k schreiben

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Bell - La Padula - Modell (2)

- sichert nur Datenvertraulichkeit, nicht aber Datenintegrität!

5 Biba - Modell

- umgekehrte Eigenschaften zum Schutz von Datenintegrität
  - ◆ Simple-Integrity-Regel und Integrity-*-Regel

➥ einfache MLS-Modelle in der Praxis ungenügend
- ◆ aber konzeptionelle Basis für heutige MAC-Systeme
- ◆ Einsatz des Bell-LaPadula-Modells mit Erweiterungen in heutigen Betriebssystemen
  - ➤ SELinux
  - ➤ AppArmor
  - ➤ z/OS (IBM), Solaris 10 mit Trusted Extensions

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

G.5 Beispiel SELinux

1 Überblick

- Erweiterung von Linux um flexiblen MAC-Mechanismus
  - ➤ ***Type Enforcement***
    - jedem Subjekt und Objekt ist ein
      Security Context = user : role : type
      zugeordnet
    - Objektzugriff erfordert Autorisierung des Subjekt-Typs (= *Domain*) für den Objekt-Typ – unabhängig von dem tatsächlichen Benutzer
    - Autorisierung erfolgt auf Basis einer *SELinux Policy* (Datei, die alle Regeln des SELinux-Kerns enthält)
    - Flexible Regeln durch rollenbasierte Zugriffskontrolle
    - zusätzliche MLS durch Definition mehrere Sicherheitsebenen möglich (entsprechend Bell-LaPadula-Modell)
- *Type Enforcement* erfolgt zusätzlich zu den herkömmlichen UNIX-Rechteprüfungen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Zugriffkontrolle durch Type Enforcement

- Zugriffe in SELinux müssen explizit erlaubt werden
  - ➤ Default: kein Zugriff (unabhängig von den Linux-Rechten!)
  - ◆ allow-Regeln spezifizieren den Zugriff eines Subjekt-Typs (Domain) auf einen Objekt-Typ

source type(s)	Domain eines Prozesses der auf Objekt zugreifen möchte
target type(s)	Typ eines Objekts auf das der Prozess zugreifen soll
object class(es)	Die Objektklasse(n), für die der Zugriff erlaubt ist
permission(s)	

- Beispiel 1:
```
allow user_t bin_t : file {read, execute, getattr};
```
  - ➤ Prozess der Domain `user_t` dürfen auf `file`-Objekte des Typs `bin_t` mit den Rechten `read`, `execute` oder `getattr` zugreifen
- Beispiel 2:
```
audit user_t classified_t : file {read, unlink};
```
  - ➤ Eskaliere Zugriffe auf geschützte Dateien

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Zugriffkontrolle durch Type Enforcement

- Beispiel 3: Realisierung des Programms `/usr/bin/passwd`
  - benötigt Schreibrechte auf `/etc/shadow` um verschlüsselte Passwörter zu ersetzen
  - normale Vorgehensweise:
    - `/usr/bin/passwd` hat root-s-Bit
    - `/etc/shadow` ist nur für root les- und schreibbar
    - Problem: alle anderen Programme die mit `root`-uid ausgeführt werden könnten ebenfalls `/etc/shadow` modifizieren
  - Standard-UNIX-Situation:
    - Programm benötigt zum Ablauf besondere Privilegien
    - s-Bit bewirkt Umschalten der euid im Rahmen des `execve`-Systemaufrufs
    - eigentlich sehr grob-granulare Lösung
      – alle oder viele (Gruppe) dürfen das Programm ausführen
      – das Programm hat - egal wer es ausführt - alle Rechte des Dateibesitzers

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Domain Transition

- SELinux: Verfeinerte Objekte und Subjektdefinitionen:
  - Typ `passwd_t` als Domain für `/usr/bin/passwd`
  - Typ `shadow_t` als Typ für `/etc/shadow`
  - Regel:

```
allow passwd_t shadow_t : file {ioctl read write create getattr setattr
lock relabelfrom relabelto append unlink link rename}
```

- SELinux erlaubt Verfeinerung der Fragestellung
  - wie bekommt der Prozess, der `/usr/bin/passwd` ausführt den Domain-Typ `passwd_t`?
  - mehrere Teilprobleme:
    1. welche Domains dürfen `/usr/bin/passwd` ausführen?
    2. welche Prozesse (d. h. Instanzen welcher Programme) dürfen den Domain-Typ `passwd_t` bekommen?
    3. welche Domains dürfen bei Ausführung eines solchen Programms in den Domain `passwd_t` überführt werden?

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Domain Transition (2)

- SELinux-Lösung:
  1. Ausführungsrecht für `/usr/bin/passwd`
     - ➤ "normale" Benutzerprozesse laufen in Domain `user_t`
     - ➤ `/usr/bin/passwd` hat Typ `passwd_exec_t`
     - ➤ `allow user_t passwd_exec_t : file (getattr execute);`
       "normale" Benutzerprozesse (Prozesse in Domain `user_t`) dürfen *`stat(2)`* und *`execve(2)`* auf /usr/bin/passwd ausführen
  2. "Potentielles s-Bit" für das Programm /usr/bin/passwd
     - ➤ `allow passwd_t passwd_exec_t : file entrypoint;`
       Prozesse, die das Programm `/usr/bin/passwd` ausführen, dürfen in Domain `passwd_t` überführt werden
     - ➤ strikte Kontrolle, welche Programme eine Domain betreten dürfen!
  3. s-Bit-Nutzungsrecht für Domain `user_t`
     - ➤ `allow user_t passwd_t : process transition;`
       Prozesse in Domain `user_t` dürfen in Domain `passwd_t` überführt werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Domain Transition (3)

- explizit
  - ➤ vor einem execve-Aufruf wird der Security Context dafür gesetzt (*`setexeccon(3)`*)
  - ➤ Problem: aufrufendes Programm hat meist kein explizites Wissen über den besonderen Rechtebedarf des ausgeführten Programms:
    s-Bit-Semantik setzt euid automatisch um - aufrufendes Programm
- implizit : process passwd_t
  - ➤ spezielle Regel in der SELinux Poliy
  - ➤ `type_transition user_t passwd_exec_t : process passwd_t;`
    wenn ein Prozess der Domain `user_t`
    ein Objekt mit Typ `passwd_exec_t` ausführt,
    wird die Domain des Prozesses automatisch nach `passwd_t` überführt.
    Die Regel bezieht sich auf die Objektklasse `process`
    - – ein Prozess (normalerweise ein Subjekt) ist hier Objekt im Sinn der Regel - d. h. die Regel bezieht sich auf Prozesse

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Rollen

- Mechanismus zur Einschränkung von Domain-Zuordnungen oder Domain-Transitionen
  - ➤ Gegensatz zu RBAC-Modellen, die Rollen mit Rechten verknüpfen!
  - ➤ Rechte in SELinux ergeben sich nur aus *Type-Enforcement*-Mechanismus
  - ➤ Rollen schränken Rechte ein
- Zuordnung von Linux-Benutzern (uid) zu Rollen
  - ➤ `user joe roles { user_r, system_r };`
    Prozesse des Benutzers `joe` dürfen die Rollen `user_r` und `system_r` einnehmen
- Kompatibilität von Domain-Typen und Rollen
  - ➤ `role user_r types user_t;`
    `role user_r types passwd_t;`
    die Domains `user_t` und `passwd_t` sind mit der Rolle `user_r` kompatibel
    => Prozesse in der Rolle `user_r` dürfen eine Domain-Transition zwischen `user_t` und `passwd_t` durchführen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Rollen (2)

- Ziele
  1. Entkopplung von Linux-Benutzer-Ids und SELinux-Domains
     - ➤ große Benutzerzahl, viele Domains
     - ➤ wenige "typische" Rollen fassen die Rechte auf die Typen zusammen ("normaler Benutzer", "Administrator", ...)
     - ➤ Zuordnung der Rollen zu den Benutzern überschaubar
  2. Rechte eines Benutzers werden auf die gerade "aktive" Rolle beschränkt
     - ➤ jeder Prozess hat jeweils eine "aktive" Rolle
       Domaintransitionen sind auf dazu kompatible Domains beschränkt
     - ➤ Wechsel der "aktiven" Rolle erlaubt temporären Rechtewechsel (ähnlich wie su/sudo-Kommando, aber feiner abstimmbar)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Multi-Level-Security

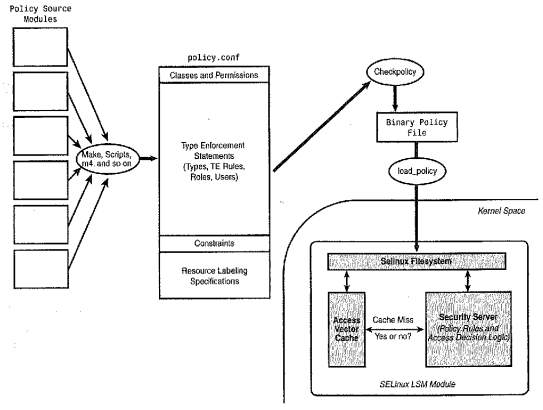
- Variante des Bell-LaPadula-Modells
- *Security Context* wird um Sicherheitsstufe(n) SL erweitert
  - $SL = (s, C)$
    s Sensitivity, aus einer geordneten Menge von Werten
    $C = \{c1, ..., cn\}$ Kategorie-Werte, ungeordnete Werte
  - Relationen auf Sicherheitsstufen
    dom SL1 dominates SL2 $\Leftrightarrow$ s1 $\geq$ s2 $\wedge$ C1 $\supseteq$ C2
    domby SL1 is dominated by SL2 $\Leftrightarrow$ s1 $\leq$ s2 $\wedge$ C1 $\subseteq$ C2
    eq SL1 is equal to S2 $\Leftrightarrow$ s1 = s2 $\wedge$ C1 = C2
    incomp SL1 is incomparable to S2 $\Leftrightarrow$ C1 $\not\subset$ C2 $\wedge$ C2 $\not\subset$ C1
- Basis-Regeln
  - Prozess darf Objekt lesen wenn $SL_{proc}\ dom\ SL_{Obj}$
  - Prozess darf Objekt schreiben wenn $SL_{Proc}\ domby\ SL_{Obj}$
- Erweiterung: vertrauenswürdige Anwendungen dürfen Sicherheitsstufen von Objekten (z. B. Dateien) verändern

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 SELinux Kern-Architektur

- Basis: LSM (*Linux Security Modules*) Framework
  - Anknüpfungspunkte für Sicherheitsabfragen in allen sicherheitsrelevanten Systemaufrufen
  - LSM-Aufrufe erfolgen nach den "normalen" Zugriffsrecht-Abfragen (DAC)

8 Installation von SELinux Policies



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Feingranulare Rechte-Spezifikation bei MAC

- MAC: Rechte werden systemweit für Objektmengen spezifiziert
  - allow-Regeln
    `allow Domain-Typ(en) Objekt-Typ(en) : Objekt-Klasse(n) Recht(e);`
- Objekt-Klasse erlaubt Zusammenfassung/Differenzierung von Objekten
  - *file*, *blk_file*, *chr_file*, *dir*, *lnk_file*, …
  - *socket*, *tcp_socket*, *unix_stream_socket*, ...
  - *msg*, *sem*, *shm*
  - *capability*, *process*, *security*, *system*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Feingranulare Rechte-Spezifikation bei MAC (2)

- Rechte für die Nutzung von Systemschnittstellen
  - im Gegensatz zu Standard-UNIX-Rechten (rwx) sehr fein-granular
- Beispiele für Objekt-Klasse *file*
  - *read*, *write*, *append*, *execute*, *ioctl*, *create*, *rename*, *unlink*,
  - *setattr* (für `chmod`), *getattr* (für `stat`),
  - *entrypoint* (s-Bit der Datei darf genutzt werden = Datei darf für Domain-Transition genutzt werden)
  - *execute_no_trans* (Datei wird in Domain des Aufrufers ausgeführt - ohne Domain-Transition)
  - …
- Beispiele für Objekt-Klasse *process*
  - *fork*, *sigkill* (darf SIGKILL versenden),
  - *execstack* (Prozess darf Code auf Stack ausführen)
  - …

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Feingranulare Rechte-Spezifikation bei MAC (3)

- Typen
  - repräsentieren Ressourcen in Bezug auf Sicherheit
  - Zuordnung erfolgt über Security Context
  - typischerweise sehr große Zahl von Typen in einem SELinux-System
- Attribute
  - Konzept zur Gruppierung von Typen
  - allow-Regeln können sich auf Attribute statt auf Typen beziehen

10 Literatur

MMC07. Frank Mayer, Karl MacMillan, David Caplan. *SELinux by Example*. Prentice Hall, 2007.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.