Sicherheitsmodelle

Es gibt mehr als rwxr-xr-x

somebody

Terminolodingsda

- Subjekte: handelnde Entitäten
  (z.B. Prozesse, Hardwarekomponenten,
  Netzwerk-[Interfaces|Sockets])

- Objekte: von Subjekten benutzte Ressourcen.
  (z.B. Dateien, bei IPC andere Prozesse,
  Netzwerk-[Interfaces|Sockets])

Definition TCB

TCB == Trusted Computing Base

TCSEC1983 definiert TCB als

”the totality of protection mechanisms within a computer system, including hardware, firmware and software, the combination of which is responsible for enforcing a security policy ”.

Einfacher: Alle Systemkomponenten die korrekt funktionieren müssen um sich auf überhaupt irgendwas verlassen zu können.

Reference Monitor

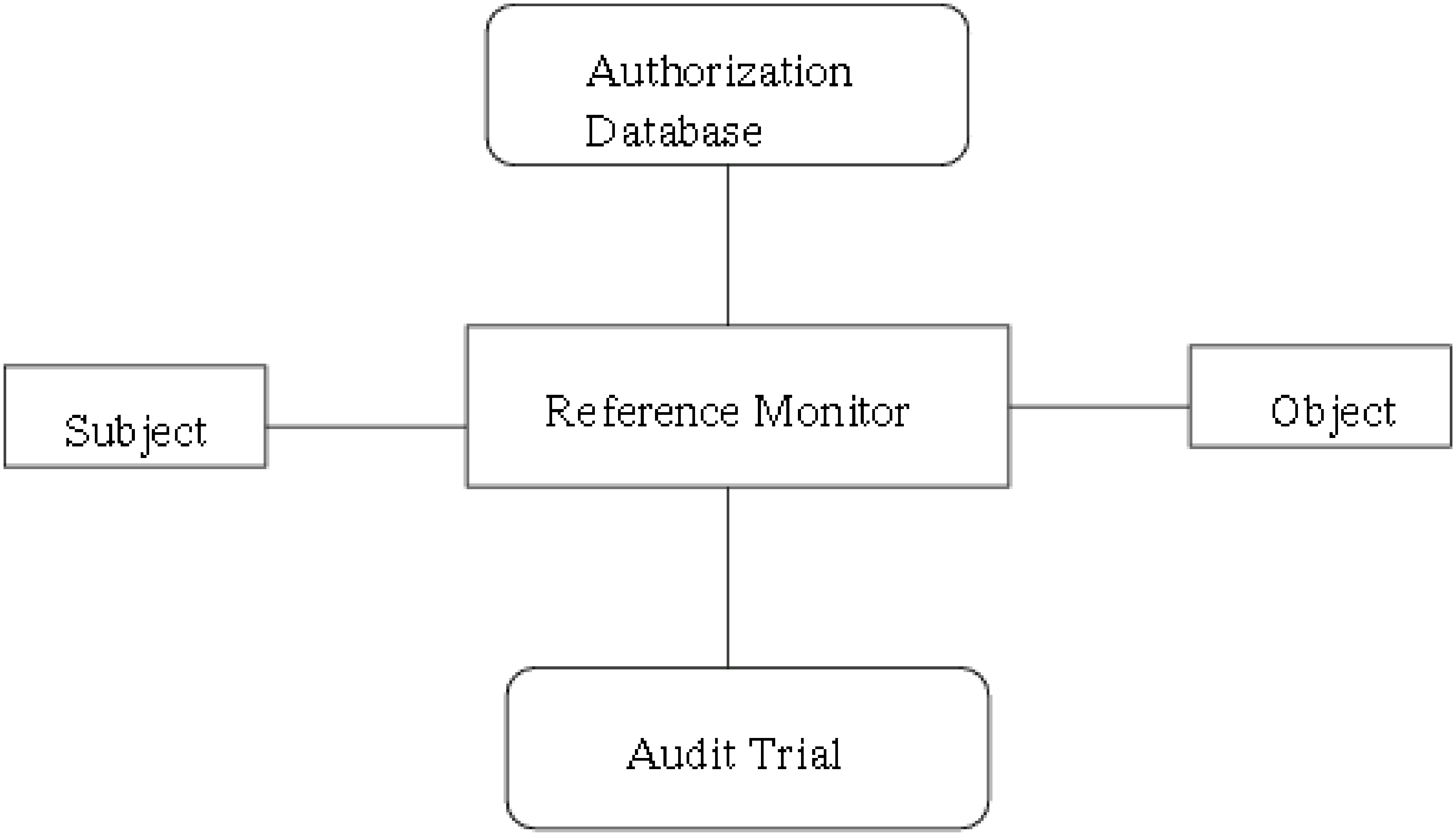


Figure 3.1: Reference Monitor Conceptual Model

Access Control Matrix

	File 1	File 2	File 3	File 4	Account 1	Account 2
John	Own R W		Own R W		Inquiry Credit	
Alice	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Bob	R W	R		Own R W		Inquiry Debit

Figure 2: An Access Matrix

DAC

DAC == Discretionary Access Control

Der ”Owner” eines Objektes hat die Kontrolle darüber wie die Zugriffsrechte gesetzt sind. ACLs sind sind übliches

DAC Schema.
(Unix rwxr-xr-x == ACL mit drei Eintraegen!) Probleme:

- prinzipiell Trojaner anfällig.
- keine Information-Flow-Policies möglich.

MAC

MAC == Mandatory Access Control

Ansatz: Die Regeln welches Subjekt was auf welchen Objekten machen darf werden von aussen (oben) vorgegeben.

Kann gegen Trojaner helfen, da boeswillige Software nicht mal eben die Zugriffsrechte auf andere Objekte ändern kann.

MLS als MAC policy // Information-Flow

MLS == Multi Level Security

- Jedes Subjekt/Objekt hat ein Label. (clearance / classification)
- Label sind angeordnet. (Op's: $L_1 = L_2$ und $L_1 > L_2$ (dominates))
- Beispiel: (TopSecret, Secret, Confidential, Public)
- Regeln: NoReadUp // NoWriteDown.
- Zusätzlich "compartments" um Themengebiete unterscheiden zu koennen.

=> Informationsfluss nur "nach Oben"

Covert Channel Problematik: MLS

In Systemen gibt es haufenweise ”shared resources” zwischen untersch. clearances. => Covert Channel.

Z.B.:

- ENOSPACELEFT ist ein Ein-Bit-Kommunikationskanal
- Morsen durch die Systemlast.
- Timing beim Plattenzugriff; ... ...

CovertChannel Analyse ist aufwändig! Wirklich aufwändig! Und auch noch Fehleranfällig.

Covert Channel Problematik: MLS

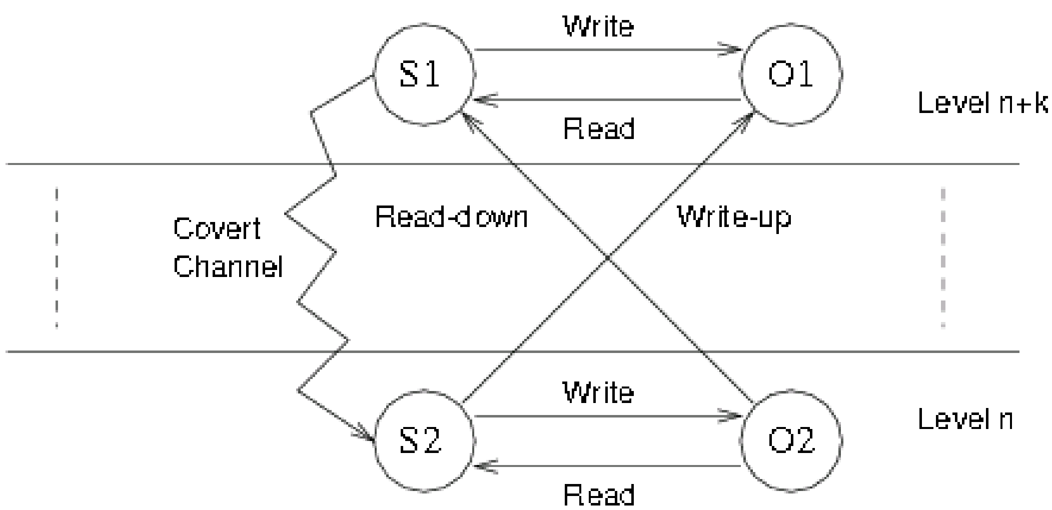


Fig. 1. Information flows in multilevel security.

BIBA Integrity Model

BIBA ist MLS umgedreht: schützt die Integrität von Daten.
Schützt NICHT vor unberechtigtem lesen/rauskopieren.

Label z.B. (TCB, LocalSoftware, Suspicious, Garbage).

Effektiver Schutz gegen Viren/Wuermer.

LoMAC: Flowting Label BIBA

MLS / BIBA sind fixed label policies: Falsches Label => Zugriff verweigert.

Alternativ: Flowting Label BIBA == Low Watermak Policy == LoMAC

Subjekte werden bei eingehenden Informationsfluss automatisch ”gedowngraded”

Beispiel: Admin startet Editor(HighIntegrity) um BenutzerDatenbank(HighIntegrity) zu editieren. Boeser Mensch hat aber irgendwo eine .vimrc(LowerIntegrity) mit fiesen Makros hinterlegt. Editor wird dynamisch umgelabelt(LowerIntegrity) und darf nichts mehr in der /etc/shadow machen.
=> Admin plötzlich hellwach und im Alarmzustand.

Knobel Aufgabe

Warum kein flowting MLS?

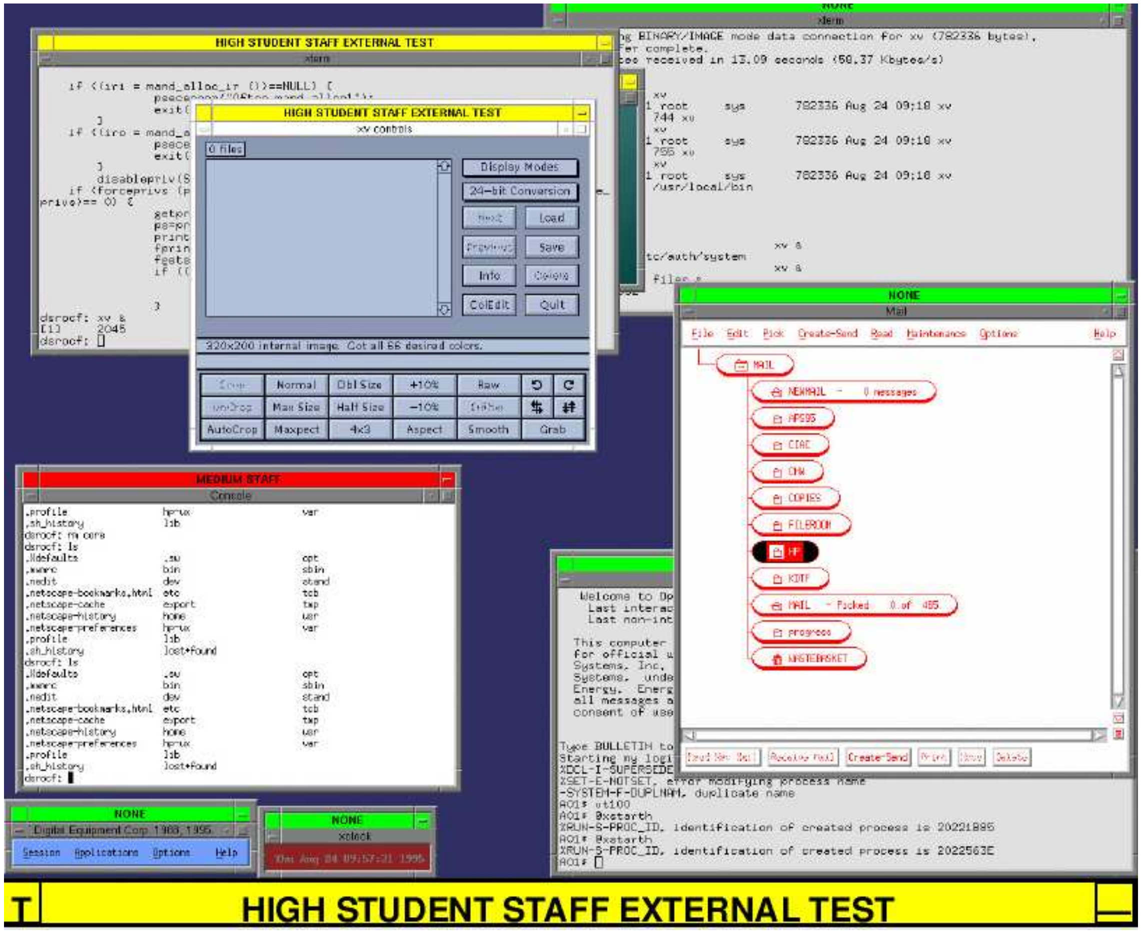
MLS/BIBA Praxisbezug

Nicht gegeben. Klingt alles auf dem Papier tolle – in der Praxis aber schwer zu handhaben.

Und, wichtig:

- richtig machen == Bequemlichkeitsverlust.
- nur grobe Einteilung in Klassen – kein POLA!

MLS/BIBA Praxisbezug



Domain Type Enforcement

DTE ist auch eine MAC policy

Subjekte bekommen ein Domain Label, Objekte ein Type Label

Table 1: A Domain Definition Table (DDT)

Domain/Type	System Object	Accounting Object	Engineering Object
Printer Driver	rwx	r--	r--
Accounting	---	rwx	---
Engineering	---	---	rwx

Table 2: A Domain Transition Table (DTT)

Domain/Domain	Printer Driver	Account-ing	Engineer-ing
Printer Driver	True	-	-
Accounting	True	True	-
Engineering	True	-	True

DTE fuer Unix == Flask Architecture

Sieht ein bisschen aus wie aus wie allg. Syntax um den ReferenceMonitor zu instruieren.

=> AccessControlMatrix clusterung

- pro: sehr mächtig, fast bel. flexibel.
- pro: guter Integritätz Schutz
- con: wer will das konfigurieren? Verleichsweise tiefe Eingriffe in System Binaries.
- con: Information-Flow Kontrolle nur mit aufwändigen Policies möglich.

m4 Makros "vereinfachen" die Konfiguration.

DTE fuer Unix == Se(Linux|BSD)

- Wird gerade fleissig portiert
- Es sind mehrere Tools angekündigt, die DTE Pol.. analysieren und high-level Fragen beantworten können sollen.

Grosses Problem:

Policy gilt immer für’s System – keine Subpolicy. => ich kann als Benutzer nicht die Policy verändern. (MAC halt)

back to DAC

Es gibt noch capabilities:

UNIX capabilities nach POSIX 1003.1e sind KEINE wirklichen capabilities. Sie sind das was VAX/VMS Privileges genannt hat:
Flags die anzeigen, ob ein Subjekt bestimmte Dinge tun darf. Privileges können vererbt werden .... oder nicht.

Wurden erfunden (und benutzt!) um root unter UNIX aufzudröseln. (uid == 0 hat dann keine besondere Bedeutung mehr)

Beispiele: CAP_DAC_OVERRIDE, CAP_NET_RAW, ...
siehe /usr/include/linux/capabilities.h, bzw. (TrustedBSD) /usr/include/sys/capabilities.h

Posix 1e capabilities

Jeder Prozess hat drei Flag-Sets:

- E effektive capabilities: aktuell gültige caps. (vergl. euid)
- I inheritable capabilities: werden bei exec(2) vererbt
- P permitted capabilities: diese koennen mit capset in E oder I gesetzt werden.

Posix 1e capabilities und so...

Vererbungsregeln bei exec(2):
(vor exec X_0, nach exec X_1, Dateisystem X_f)

$I_1 = I_0 \&\& I_f$
$P_1 = P_f || (I_1 \&\& P_0)$
$E_1 = E_f \&\& P_1$

oder:

$I_1 = I_0$
$P_1 = (P_f \&\& Z) || (I_0 \&\& I_f)$
$E_1 = E_f \&\& P_1$