11 Informationsflöden

M. von Platen. Föresatsen är den lates njutningsmedel.

11.1 Allmänt

Den direkta (och 'discretionary') åtkomstkontrollen löser inte alla frågeställningar.

Exempel. Om subjekt *A* har läsrättighet till filen *F* och *B* saknar sådan hindrar inget att *A* kopierar *F*'s *innehåll* till ett objekt *F'* som är läsbart av *B*. För att förhindra detta kringgående av rättigheter behövs flödeskontroll.

Till varje subjekt s och till varje objekt *o* associeras en etikett ('label', 'class') *c(s)* respektive *c(o)*. Etiketterna kan ordnas linjärt (≤), tex öppen, konfidentiell, hemlig, topphemlig, eller mer generellt via en partiell ordning (↵). En sådan är en relation som är

- reflexiv: $c(a) \hookleftarrow c(a)$, för alla *a*,

- antisymmetrisk: $c(a) \hookleftarrow c(b)$ och $c(b) \hookleftarrow c(a)$ medför $c(a) = c(b)$, och

- transitiv: $c(a) \hookleftarrow c(b)$ och $c(b) \hookleftarrow c(c)$ medför $c(a) \hookleftarrow c(c)$.

En obligatorisk ('mandatory') och med avseende på konfidentialitet flödessäker åtkomstkontroll införs genom att kontrollera att information inte kan flöda "nedåt" utan bara "uppåt". En säkerhetskärna ('security kernel') specificeras i grunden precis som den allmänna modellen för åtkomstkontroll men kompletteras med flera säkerhetsklasser.

11.2 Relationer

Här kommer en kort repetition av (eller möjligen introduktion till) de vanligaste begreppen om relationer, m m.

i. Definitioner Givet två (ändliga) mängder *X* och *Y* kan man bilda den sk *Cartesiska produkten*

$$X \times Y \equiv_{\Delta} \{ (x, y), x \in X, y \in Y \},$$

d v s mängden av par där första komponenten tillhör *X* och den andra komponenten tillhör *Y*. I många sammanhang är *X* och *Y* samma mängd. Det ger en relation över *X*.

En (binär) *relation R* över *X* och *Y* (eller från *X* till *Y*) är en delmängd av $X \times Y$; $X \times Y \supset R$.

För att uttrycka att paret $(x, y) \in R$ skriver man ofta

$$x \; R \; y$$

eller ibland (sällan) $y = R(x)$. Mängden *X* kallas relationens definitionsområde ('domain') och *Y* dess värdeförråd ('range').

En funktion är en relation sådan att $x \; R \; y$ och $x \; R \; z$ implicerar $y \equiv z$: "Entydigt funktionsvärde". Här är det normala skrivsättet $y = R(x)$.

Relationer brukar ibland betecknas med specialsymboler som tex ≤, <, ~, →, etc.

ii. Representationer Tre vanliga sätt att illustrera/representera en relation på X är:

- Linjär form: Här räknas helt enkelt de par upp som ingår i relationen. Ett alternativ är att göra en entydig karakteristik tex via mängdoperationer och logiska konnektiv.

- Grafform: Här placeras elementen $x, y, ... \in X$ ut på ett plan och en båge ritas från x till y precis då $x\ R\ y$.

- Matrisform: Här införs ett matriselement M_{ij} för varje par (x_i, x_j) där $x_i, x_j \in X$ och matrisen definieras av $M_{ij} = 1 = true$ om $(x_i, x_j) \in R$ och $M_{ij} = 0 = false$ annars.

iii. Typer av relationer. Här kommer definitioner på några vanliga typer av relationer.

Namn	*Definition (för alla x, y, z gäller)*
Reflexiv	$x\ R\ x$
Irreflexiv	$not\ x\ R\ x$
Symmetrisk	$x\ R\ y \Rightarrow y\ R\ x.$
Antisymmetrisk	$x\ R\ y, y\ R\ x \Rightarrow x \equiv y.$
Asymmetrisk	$x\ R\ y \Rightarrow not\ y\ R\ x.$
Transitiv	$x\ R\ y, y\ R\ z \Rightarrow x\ R\ z.$

Tabell 11.1. Typer av relationer

Det finns många samband mellan dessa typer av relationer och karakteristika för motsvarande matris/graf.

Några exempel är följande.

- För en reflexiv/irreflexiv relation gäller att dess matris har 1-or/0-or på diagonalen.

- En symmetrisk relation har en symmetrisk matris $M = M^T$, där M^T är den till M transponerade matrisen; rader byts mot kolumner och vice versa.

- En reflexiv relation har en slinga i varje nod i graf- formen.

- En irreflexiv relation saknar slingor.

- I en symmetrisk relations graf är alla bågar dubbelriktade.

- Notera att en asymmetrisk relation alltid är irreflexiv.

Några vanliga relationsnamn framgår av följande sammanfattning.

Namn	*Karakteristika*
Strikt partiell ordning	Asymmetrisk, irreflexiv, transitiv
Partiell ordning	Reflexiv, antisymmetrisk, transitiv
Ekvivalensrelation	Reflexiv, symmetrisk, transitiv
Kompatibilitetsrelation	Reflexiv, symmetrisk
Total ordning (kedja)	Alla element är linjärt ordnade

Tabell 11.2. Fler typer av relationer

En total ordning är ett specialfall av en partiell ordning. I många tillämpningar vill man inte ordna alla symboler i "nummerordning", eftersom flera av de objekt som symbolerna representerar är oberoende; tex 'concurrent' eller lika säkerhetskänsliga.

Två relationer P och R är *konsistenta* om $P \cup R = P + R$ kan utvidgas till en total ordning.

Detta är möjligt precis då dess transitiva hölje (se nedan) är irreflexivt, dvs då dess graf är utan slingor.

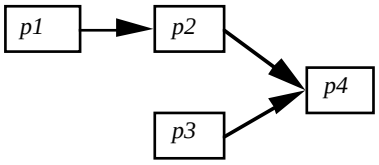
Exempel: En relation E, en exekveringsordning, är korrekt om och endast om den är konsistent med programordningen P. E är i sin tur en acyklisk orientering av den sk konfliktrelationen C, som definieras av "interfererande minnesreferenser" och som är symmetrisk och irreflexiv (men inte nödvändigtvis transitiv). Om man vill ta hänsyn till atomära operationer kan man representera dessa mha en ekvivalensrelation.

Exempel: Följande figur 11.1 visar ett annat exempel på en relation.

En relation hb ('happended before') defineras över en mängd av händelser eller processer p1, p2, p3 och p4 via följande uppräkning:

hb = < (p1, p2), (p2, p4), p3, p4)>

Motsvarande graf-form blir *Motsvarande matrisform blir*



p1	*0*	*1*	*0*	*0*
p2	*0*	*0*	*0*	*1*
p3	*0*	*0*	*0*	*1*
p4	*0*	*0*	*0*	*0*
	p1	*p2*	*p3*	*p4*

Relationen är irreflexiv, aymmetrisk och transitiv, dvs en strikt partiell ordning.

Figur 11.1. En relation

En ekvivalensrelation inducerar en uppdelning av elementen i sk ekvivalensklasser bestående av klasser av element som är ekvivalenta. Utgå från ett element x och låt $[x]_R$ bestå av alla element som är (R-) ekvivalenta med x. Föreningsmängden av alla ekvivalensklasser är X.

Exempel:

Ett exempel på en relation som genererar ekvivalensklasser är ekvivalensrelationen $a \equiv b$ (*mod n*). För tex $n = 3$ och $b = 1$ är tex

$$[2]_{mod\ 3} = \{2, 5, 8, \ldots\} = \{ 3k + 2 \mid k = 0, 1, 2, \ldots \}.$$

Ett annat exempel är en 'hash'-funktion h. Ekvivalenta symboler eller adresser 'hash'as till samma länkade överskottslista (samma h-värde) respektive avbildas associativt till samma 'cache'-rad.

iv. Operationer på relationer Givet en relation P från X till Y och en relation R från Y till Z kan man bilda *sammansättningen*

$$P \bullet R = \{ (x, z) : x \in X, z \in Z, (\exists\, y)\ ((x, y) \in P,\ (y, z) \in R\ )\}.$$

En sammansättning av P med sig själv upprepade gånger skrivs också

$$P^n = P \bullet P \bullet (n \text{ gånger}) \bullet P.$$

$P^0 = I$ står för identitetsrelationen. Sammansättning på matrisrepresentationen ges av matrismultiplikation. Eftersom en relation är en mängd kan vi bilda *förenings*- och snitt-*mängd*erna mellan två relationer. Föreningsmängden av P och R skrivs oftast $P + R$.

Det *transitiva höljet* till en relation över en mängd med *n* komponenter *R* är

$$R^+ = R + R^2 + R^3 + ... + R^n .$$

Det *reflexiva transitiva höljet* är

$$R^* = I + R^+.$$

Relationen R^+ är den minsta transitiva relation som innehåller *R*.

Warshalls algoritm för att bilda det reflexiva transitiva höljet `rp` till givet `r` är som följer.

```
for i , j := 1 to n do   rp (i,j) := r (i,j);

     for k := 1 to n do
        for i := 1 to n do
           for j := 1 to n do
              if not rp (i,j)
              then  rp (i,j) := rp (i,k) and rp (k,j);
```

Tabell 11.3. Warshalls algoritm

R^+ och R^* svarar mot en upprepad tillämpning av *R*; en eller fler respektive 0 eller fler upprepningar.

11.3 Modeller

11.3.1 Bell-LaPadula

Varje objekt och subjekt tilldelas förutom rättigheter i accessmatrisen en fix etikett (linjärt ordnade) och enligt den modell som föreslagits av Bell och LaPadula skall följande axiom gälla. Vi antar att ≤ är en total ordning; etiketterna *c(x)* kan tolkas som naturliga tal.

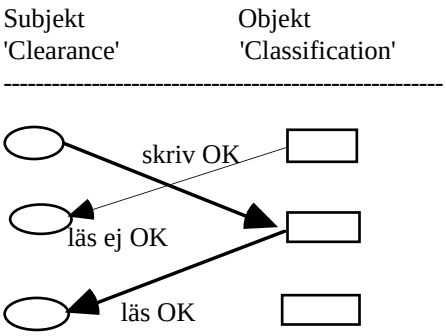
- En process *p* kan inte läsa ett object *o* försåvitt inte $c(o) \leq c(p)$.
Detta kallas vanligen '*simple security*'. ('no read up').

- En process kan inte läsa ett objekt *x* och skriva ett objekt *y* om inte $c(x) \leq c(y)$.
Detta kallas vanligen '**-property* ' eller 'confinement property' ('no write down').

Detta är hårdare krav än nödvändigt. Processen låter inte nödvändigtvis information flöda från *x* till *y*. I praktiken räcker det att kräva att $c(x) \leq c(y)$ i de fall då *y* är funktionellt beroende av *x*. Modellen tillåter inte heller dynamiska (databeroende, tidsberoende) eller partiellt ordnade etiketter.

Observera också att denna flödeskontroll avser konfidentialitetsaspekten. Integritet berörs inte: En process i klassen "öppen" tillåts skriva nonsens i ett objekt i klassen "topphemlig"!

Figur 11.2 illustrerar Bells och LaPadulas modell.



Figur 11.2 Flödeskontroll via BLP-modellen

11.3.2 Biba

En modell för integritet som kan sägas vara dual till BLP är den sk Biba-modellen. Den bygger på integritetsklasser *i(x)* och består av följande regler.

- En process *p* kan modifiera ett objekt o endast ifall i(o) ≤ *i(p)*.
Detta kallas vanligen '*simple integrity*'.

- Om en process *s* kan läsa ett objekt *o*, så kan *s* skriva ett objekt *y* endast ifall *i(y)* ≤ *i(o)*.
Detta kallas vanligen '**-integrity property* '.

Att samtidigt garantera både konfidentialitet och integritet med enbart dessa metoder (BLP och Biba) ger starka inskränkningar!

11.3.3 Denning

För att hantera dynamiska säkerhetsklasser (etiketter) och för att bättre kunna validera processer som hanterar information av olika klass har Dorothy Denning utvecklat en modell i vilken processens interna tillståndsövergångar studeras.

Vi börjar med att studera flöden ur kvalitativa aspekter.

i. Allmänt. Modellen bygger på lattices definierade av en partiell ordning (↵) av säkerhetsklasser med minimum ('greatest lower bound', infimum, ⊗) och maximum ('least upper bound', supremum, ⊕).

Ett lattice är en partiellt ordnad mängd där varje par har ett entydigt minimum och maximum.

- Existensen av ⊕ medför att om $c(x_i) \leftarrow c(y)$ så finns en entydig klass $c(x) = \oplus\, c(x_i)$ så att $c(x) \leftarrow c(y)$. Det betyder att alla flöden från x_i till *y* är tillåtna om $c(x) \leftarrow c(y)$. Detta kan tillämpas på explicita flöden:

y := *expression* (x_1, ..., x_n);
Kontrollera att $\oplus\, c(x_i) \leftarrow c\,(y)$.

En sats *y* := *x* ger ett explicit flöde $x \rightarrow y$.

- Existensen av ⊗ medför att om $c(x) \leftarrow c(y_i)$ så finns en entydig klass $c(y) = \otimes\, c(y_i)$ så att $c(x) \leftarrow c(y)$. Det betyder att alla flöden från *x* till y_i är tillåtna om $c(x) \leftarrow c(y)$. Detta kan tillämpas på implicita flöden:

if *predicate*(*x*) **then** y_1 := ...; ...; y_n := ...; **end if**;

Kontrollera att $c(predicate(x)) \hookleftarrow \otimes c(y_i)$.

I en sats **if** *x* **then** *y* := 1; ger en observation av *y* information om *x*; dvs det föreligger ett implicit flöde $x \rightarrow y$.

Som vid tex programverifiering kan vi sedan sätta upp axiom för alla programspråks-konstruktioner, som alla kan uttryckas med

- EN "tillräckligt bra" aritmetisk/logisk operation (tex nand); jfr 'expression' ovan
- EN villkorlig hopp-instruktion; jfr if ovan,

och "tillräckligt med minne" att lagra resultaten i (Turing).

Konstruktioner som sekventiella sammansättningar säkerställs genom att utnyttja transitivitetsegenskapen.

Certifieringen kan ske under 'compile time' (statiskt) eller under exekveringen (dynamiskt).

ii. Exempel. Satsen **if** *x* > *y* **then** *z* := *w*; *i* := *k* + *j* **end if**; kräver att

$c(w) \hookleftarrow c(z)$ och $c(k) \oplus c(j) \hookleftarrow c(i)$ -- tilldelningarna; expicita flöden

$c(x) \oplus c(y) \hookleftarrow c(z) \otimes c(i)$ -- if-satsen; implicita flöden

Notera att kontrollen implementeras av en 'security kernel'. Tekniken är alltså avsedd för applikationer och inte för att användas *i* en operativsystemkärna.

Notera också att det som kan uppfattas som (enkla) variabler tex i exemplet också kan stå för mer sammansatta objekt som tex filer och databaser.

iii. Certifieringsregler (kvalitativa regler). Följande tabell är en sammanfattning av regler för kontroll av auktoriserade flöden med avseende på några vanliga program-konstruktioner.

iv. Flödessäker åtkomstkontroll. En grov kontroll som ibland räcker är att kräva för en process *p* och dess in- och utdata $\{x_i\}$ respektive $\{y_i\}$ att

$$\oplus_i c(x_i) \leq c(p) \leq \otimes_i c(y_i).$$

En begränsning med denna metod är att den inte "kan skilja på" vad som är känsliga klasser och inte.

En finmaskigare kontroll erhålls om man särskiljer de enskilda objektens klasser.

Sats	*Kontroll av*
`var := exp`	-- *c(exp)* ≤ *c(var)*
`begin` `s1, ..., sn;` **`end`**	-- *s*1, *s*2, ..., *sn*
`if` `exp` **`then`** `s1` [**`else`** `s2`]	-- *s*1 [och *s*2] -- *c(exp)* ≤ c*(a)*, där *c(a)* = *c(s1)* [⊗ *c(s2)*] och där *c(si)* = ⊗{*c(z)* | *z* tilldelas värde i *si*}
`while` `exp` **`do`** `s`	-- Satsen terminerar -- *s* -- *c(exp)* ≤ *c(s)*, där *c(s)* = ⊗{ *c(a)* | info flödar till *a* i satsen *s*}

Tabell 11.4. Certifieringsregler

v. Informationsflöden hos program (kvantitativa mått). Satser eller instruktioner i program eller kommandosekvenser representerar också kanaler, vilket gör att kapacitetsresonemang kan användas om man vill kvantifiera flödet av information vid analys av program map informationsflöden som för t ex BLP- och D. Dennings modell.

En sats α (tilldelning ('copy file'), **`if`**-sats (betingad kompilering), osv) genererar en tillståndsövergång $s \perp_\alpha s'$, där tillstånden *s* och *s'* betecknar värdetilldelningar av variabler före respektive efter α.

Ett programflöde $x_s \rightarrow_\alpha y_{s'}$ kan karakteriseras av reduktionen av osäkerheten hos *x*, dvs

$$I\ (\alpha, x, y, s, s') = H(x \mid y) - H\ (x \mid y') \qquad -- y' \equiv y_{s'}$$

Ibland existerar inte *y* före α och då blir $H(x \mid y) = H(x)$ i tillståndet *s*.

Kanalkapaciteten *C* för satsen definieras som vanligt som supremum av den ömsesidiga informationen

$$C = max_{p(s,\ x)}\ I\ (\alpha, x, y, s, s'),$$

där *p(s, x)* är sannolikhetsfördelningen för *x* i tillstånd *s*.

En tilldelning $y := x$ med $x \in [0, 15]$ där alla *x* är lika sannolika ger ett flöde om 4 (bitar) (förutsatt att *x* inte är känt i tillstånd *s*).

Ett uttryck och en tilldelning i en sats $x := y + z$, där *y* och *z* är lika sannolika i tex [0,15], reducerar osäkerheten avseende både *y* och *z* om *x* kan observeras efter satsen.

En sats $\underline{x := y \oplus z,}$ där *y* och *z* är lika sannolika i tex [0,15], reducerar däremot inte, osäkerheten om *y* och *z* vid observation av *x*; 'One time pad' ger ju prefekt sekretess!

En villkorssats

if $x \geq 8$ **then** $y := 1$ **end if**; -- y = 0 initialt

och $x \in [0, 15]$ där alla x är lika sannolika har ett flöde som kan beräknas:

$H(x \mid y) = H(x) = \underline{4}$, trivialt.

$H(x \mid y') = - \Sigma_{j=0,1}\, p(y' = j)\, \Sigma_{i=0,..,15}\, p(x = i \mid y' = j)\, log\, \{p(x = i \mid y' = j)\}$.

Men

$p(y' = 0) = p(y' = 1) = 1/2$

och

$p(x = i \mid y = 0) = 1/8$ för $0 \leq i < 8$ och $p(x = i \mid y = 0) = 0$ för övriga i.

och

$p(x = i \mid y = 1) = 1/8$ för $8 \leq i \leq 15$ och $p(x = i \mid y = 1) = 0$ för övriga i.

Enkla uträkningar ger $H(x \mid y') = \underline{3}$, så informationsflödet är 1 (bit).

En sats **if** $f(x)$ **then** $y := 1$ end if; där $y \in [0,1]$ kan ge högst 1 bits flöde eftersom y inte kan ge mer än en bits information om x.

11.3.4 Clark-Wilson

Denna modell bygger på regler för användande och underhåll av data/information för att åstadkomma integritet. Reglerna bygger i sin tur på sk välformade transaktioner (*wft*).

Data *D* indelas i två disjunkta klasser; skyddade ('constrained') *CDI* och övriga ('unconstrained) *UDI*. (I står för 'items'). Subjekt är i denna modell enheter som kan initiera s k transformationsprocedurer (*TP*).

Modellen omfattar 9 regler.

i. Det måste finnas en sk *IVP* ('integrity validation procedure') för varje *CDI*.

ii. En applikation av en *TP* på en *CDI* måste bevara denna *CDI*s integritet.

iii. En *CDI* kan bara ändras av en *TP*.

iv. Ett subjekt *s* kan bara initiera vissa *TP* (*t*) på vissa *CDI* (*c*). Enheter av formen <*s*, *t*, *c*> kallas ibland för *CW*-tripletter.

v. *CW*-tripletter måste implementera en "separation of duties"- policy.

vi. Vissa speciella *TP* utförda på *CDI*-er kan producera utdata.

vii. Varje *TP* måste generera tillräcklig information genom tillägg på en *CDI* för att kunna rekonstruera den process som initierade denna *TP*; EN 'audit log' måste skapas.

viii. Systemet måste autenticera subjekt som vill initiera *TP*.

ix. Systemet får tillåta endast speciella subjekt att ändra de listor som beskriver behörigheter.

Steget till en konkret implementering blir i det här fallet lång. Modellen kan emellertid kombineras med t ex Biba-modellen.

11.4 Exempel:Unix V/MLS

AT&T har lanserat en version *Unix V/MLS* ('multi level security') som bygger på BLP. Det är i grunden ett *Unix System V* med tillägg av funktioner för 'mandatory access control', MAC.

Etiketter implementeras via Unix's 'group identifier' och de innehåller

- 'level', ett 16 bitars heltal som definierar en linjär ordning (L).
- 'category', en 64 bitars 'boolean array' som definierar ett delmängdslattice (C).

Cartesiska produkten $L \times C$ blir också ett lattice och används för att etikettera subjekt och objekt.

Vi kan definiera $L \times C$ med följande regler.

1. $<a, c> \hookleftarrow <a', c'>$ om och endast om $a \leq a'$ och c är en delmängd av c'.
2. $<a, c> \oplus <a', c'> = <max\ (a, a'), c \cup c'>$.
3. $<a, c> \otimes <a', c'> = <min\ (a, a'), c \cap c'>$.
4. $low = <0, \{\}>$.
5. $high = <2^{16}, C>$.

Systemet hanterar filer, i-nodes, kataloger, ipc-mekanismer och processer.

För de olika operationerna på de olika objekten (objekttyp följer av operationstyp) måste dominansrelationer enligt följande tabell vara uppfyllda för att en referens ska kunna ske.

Operation	*Relation*
Read, Search, Execute	$s \geq o$
Write (Overwrite, Append)	$s = o$
Create, Link, Unlink	$s = o$
Read-file/I-node status	$s \geq o$
Change status	$s = o$
ReadIPC	$s \geq o$
WriteIPC	$s = o$
Send a signal	$s = o$

Lilla s står för klassen för inblandat subjekt och o för objektsklassen.

Systemadministratörerna definierar klasserna för objekten och subjekten.

Tabell 11.5. Unix V/MLS

11.5 Hemliga kanaler

Helt omöjligt blir det att i praktiken stänga *alla* oönskade kanaler i ett system. Delade skivminnen, medier och utnyttjandegrad hos system skapar alltid ett minimum av informationsflöde som inte var avsett. Jämför också sk subliminala kanaler i chiffer.

Exempel på "lustigheter": Kodning av ett meddelande med de minst signifikanta bitarna hos en sekvens bilder (en bit per bild).

Ett mer realistiskt exempel på en "synkroniseringskanal" (troligen 'covert') illustreras av följande parallella system av processer.

```
parbegin

process 1:   if x = 0 then signal (s0) else signal (s1)
process 2:   wait (s0); y := 1; signal (s1)
process 3:   wait (s1); y := 0; signal (s0);

parend
```

Figur 11.3. En synkroniseringskanal

Om $x = 0$ så exekverar process 2 före process 3 så y blir 0.

Om $x \neq 0$ så kommer process 3 före process 2 och y blir 1.

Systemet ger alltså ett flöde $x \rightarrow y$ trots att x och y inte är gemensamma.

Eftersom dock satser efter *wait*-operationerna är betingade av *signal*-operationer, så föreligger ett implicit flöde via semaforerna.

Ett rättframt sätt att förhindra sådana flöden är att kräva att $c(semaphore) \leq c(y)$, där y är sådana "betingade variabler".

Även variabler av annan typ än semaforer kan orsaka flöden om de är globala.

En signalerande process kan i allmänhet läcka information genom att orsaka en annan process att vänta en tid som är proportionell mot värdet av en variabel.

Någon allmän lösning på detta problem finns inte vad man vet.

11.6 Säkerhetskärnor

En säkerhetskärna integrerar samtliga säkerhetsfunktioner till en isolerad plats i arkitekturen. Ett exempel är en s k 'reference monitor' för åtkomstkontroll.

Kärnan ligger principiellt i gränslandet mellan operativsystemet och maskinvaran och är alltså inte direkt åtkomlig för användaren.

Några grundförutsättningar är att kärnan inte kan förstöras och inte förbipasseras. Vidare bör den efter konstruktionen valideras eller verifieras med avseende på funktionalitet. Många delar implementeras gärna i maskinvara. Den bör vara av minimal komplexitet och feltolerant.

En TCB, 'trusted computing base', är mängden av maskin- och programvara ansvarig för säkerheten.

En TCB är funktionellt lik en säkerhetskärna men kan vara distribuerad i systemet. En TCB kan dessutom byggas i flera skikt. TCB-minimering ger enklare validering.

Honeywells Secure Communications Processor (SCOMP) konstrueras väsentligen som en säkerhets-front-end till Multics. SCOMP består av en program- och en maskinvarudel vardera arbetande mot en "databas"; de kallas "security attributes" respektive "descriptor base".

Ett anrop (kommando eller systemanrop) från användaren tas om hand av programvarudelen, som ger en fysisk accessförfrågan riktad till maskinvarudelen. Denna svarar med att godta eller vägra utföra anropet.

Programvarudelen innehåller funktioner för accesskontroll, filtjänster, konsistenskontroller för filer och 'auditing'.

Säkerhetskärnor återfinns också för datornät. För TCP/IP finns tex en variant MLS/TCP för 'multi-level security'.

NSA har också "sponsrat" ett projekt/datornät kallat SDNS, Secure Data Network System.

SDNS bygger på OSI-modellen och innehåller kryptering på länk-, nät- och transportskikten ('end-to-end') och protokoll för e-post och nyckelhantering på skikt 7.

Ah, ett citat från F. T. Grampp & R. H. Morris [GM84]:

> It is easy to run a secure computer system.
> You merely have to disconnect all dial-up
> connections and permit only direct-wired terminals,
> put the machine and its terminals in a shielded room,
> and post a guard at the door.

Noter

Bell-LaPadulamodellen är från [BLP73], Bibamodellen från [Bib77] och Dennings modell från [DDen76] och [Den82]. Amorosos bok innehåller också mycken information [Am94]. Unix V/MLS beskrivs i [FW88].

Övningar

11.1. Betrakta satsen **if** x > k **then** y := 1 **end if**; med $x \in [1, 2m]$ likformigt fördelat och $y = 0$ initialt. Bestäm $H(X \mid Y')$ efter det att satsen utförts. Visa att mängden information som överförs är maximal då $k = m$.

11.2. Betrakta satsen **if** (x = 1) **and** (y = 1) **then** z := 1 **end if**; där x och y båda är 0 eller 1 lika sannolikt och där $z = 0$ initialt. Beräkna $H(X \mid Z')$ och $H(Y \mid Z')$.

11.3. Visa med hjälp av ett syntaxträd hur följande sats certifieras med avseende på informationsflöden.

```
while a > 0 do
   a := a - x;
   b := a * y;
end while;
```

11.4. Visa att Cartesiska produkten av delmängslattice och ett totalt ordnat lattice också är ett lattice.

Anm. Denna konstruktion används i Unix V/MLS.

11.5. Betrakta satsen `if x > 0 then y := 1 end if;` där $y = 0$ initialt och

$p(x = 0) = 0.5$, $p(x = 1) = 0.25$ och $p(x = 2) = 0.25$.

Bestäm $H(X) - H(X \mid Y')$.

11.6. Konstruera en certifieringsregel för informationsflöden för en **`case`**/`switch`-sats.

11.7. I ett visst system har man fastlagt följande policy för informationsflöden med avseende på objekts etiketter ('classes'): $\underline{a} \leq \underline{b}$, $\underline{a} \leq \underline{c}$, $\underline{b} \leq \underline{d}$ och $\underline{c} \leq \underline{d}$. Är följande sats tillåten ?

a. `if a = b then c := a;`
b. `begin b := a; d := b; c := d; end;`
c. `c := a + b + c + d;`

11.8. Betrakta satsen `if (x = 1) and (y = 1) then z := 1; end if;` där $x, y \in [0, 1]$, båda lika sannolika och där $z = 0$ initialt.

a. Beräkna ekvivokationen $H(x \mid z')$ efter det att satsen exekverats.
b. Vilka villkor måste 'labels' för x, y och z uppfylla för att satsen ska vara godkänd ?

11.9. Bestäm informationsflödet $x \rightarrow y$ för satsen $y := x^{p-1} \bmod p$, då x är likformigt fördelad i $[1, p]$ och $y = 0$ initialt.