Introduction à la sécurité – Cours 11
Modèles de flux d'information

Catalin Dima

Contrôle d'accès de type Harrison-Ruzzo-Ullman

- Modèle très puissant.
- Indécidable en général.
- Sous-classes décidables.
- Contrôle **discretionnaire** : les utilisateurs décident de la propagation des droits.

Contrôle d'accès discrétionnaire et chevaux de Troie

- Les modèles de type matrice d'accès font confiance aux utilisateurs.
  - Situation : Alice, Bob et Charlie sont les utilisateurs d'un système informatique.
  - Alice crée un fichier f.
  - Alice permet à Bob de lire ce fichier, mais pas a Charlie.
  - Mais Bob va pouvoir *copier* le contenu du f dans copie$_f$.
  - ...et donner le droit de lecture à copie$_f$ à Charlie (car Bob est le propriétaire de copie$_f$!).
  - Même pire, Bob peut être honnête, mais employer un éditeur fourni par Charlie, qui portera un cheval de Troie et créera lui-même la copie de f pour Charlie.
- Ces situations ne peuvent pas être contrôlées dans les modèles de matrices d'accès.
- Une solution à ces problèmes doivent comprendre une *sémantique* des droits d'accès.
- On recherche un mécanisme *centralisé* pour étiqueter les sujets/objets et permettre/interdire l'accès en conséquence.
- *Cible* : flux d'informations.

Treillis du flux d'information

- Modèle militaire : *niveaux de sécurité* (Top Secret, Sécret, Classifié, Non-Classifié) permettant ou interdisant le flux d'information.
- **Système de sécurité** : $(SC, \rightarrow, \otimes)$:
  - SC : classes de sécurité.
  - $\rightarrow \subseteq SC \times SC$: rélation de type "*l'information peut couler*".
  - $\times : SC \times SC \rightarrow SC$: opérateur de combinaison de classes.
- *Axiomes de Denning* :
  - SC est finite.
  - $(SC, \rightarrow)$ est un ordre partiel.
  - SC possède un "plus petit" élément par rapport à $\rightarrow$.
  - $\otimes$ est l'opérateur "*plus petite borne sup*" ($\simeq$ union).
- Chaque sujet et objet se voit attribué une étiquette de sécurité
  - Classification pour les objets, autorisation pour les sujets.

Le modèle de Bell et LaPadula

- L'interêt principal est sur le **flux d'informations**.
  - Relation $\rightarrow$: flux d'information permis
  - Exemple : $SC = \{TS, S, C, N\}$ où $N \rightarrow C \rightarrow S \rightarrow TS$ et reflexivité.
  - $N \rightarrow TS$ veut dire que le flux d'informations est permis du niveau Non-clasifié au niveau Top Secret.
  - $TS \not\rightarrow C$ veut dire que le flux d'information n'est pas permis du niveau Top Secret au niveau Confidentiel.
- L'idée de base est d'augmenter les capacités d'un système discrétionnaire de contrôle d'accès par un mécanisme centralisé de contrôle.

Formalisation

- Modèle discretionnaire : $S, O, M : S \times O \to 2^R$.
- $R = \{$read, execute, write, append$\}$.
- Treillis de classes de sécurité : $(SC, \to, \otimes)$.
- Autorisation : $f_S : S \to SC$.
- Classification $f_O : O \setminus S \to SC$.
- Ensemble d'accès courants : $\mathcal{B} \subseteq S \times O \times R$.
- **État** du système : (b, M, f) où
  - $b, M \subseteq S \times O \times R$ (b accès courant, M matrice discrétionnaire de contrôle d'accès).
  - $f = (f_S, f_O)$.

Propriétés de sécurité

- **Propriété de sécurité simple (no read-up, NRU) :** Un sujet peut *lire* un objet si et seulement si son autorisation domine la classification de l'objet :

$$\forall(b, M, f), \forall(s, o, \text{read}) \in b, f_S(s) \geq f_O(o)$$

  Similaire pour "execute".
- **Propriété $\circledast$ (no write-down, NWD)** : Un sujet peut *écrire* dans un objet si et seulement si son autorisation est dominée par la classification de l'objet :

$$\forall(b, M, f), \forall(s, o, \text{write}) \in b, f_S(s) \leq f_O(o)$$

  Similaire pour "append".
- **Propriété de sécurité discrétionnaire (DS))** : La matrice discrétionnaire M devrait être conforme aux règles d'accès

$$\forall(b, M, f), \forall(s, o, \text{write}) \in b, a \in M(s, o)$$

Propriétés de sécurité

- **État sûr** = état satisfaisant NRU et NWD.
- **Transition sûre** : changement d'état $((b, M, f), (b', M', f'))$ pour lequel les propriétés suivantes sont satisfaites :
  1. Tout $(s, o, r) \in b' \setminus b$ satisfait NRU et NWD.
  2. Tout $(s, o, r) \in b$ qui ne satisfait pas NRU ou NWD n'appartient pas au b'.
- **Théorème de sécurité "basique"** : Si le système part d'un état sûr et passe par des transitions sûres, alors tous les états subséquents seront sûrs.

Bell–LaPadula empêche certains chevaux de Troie

- Solution pour Alice, Bob et Charlie :
  - On considère deux classes de sécurité : *Secret* et *Non-classifié*.
  - On place Alice et Bob dans la classe *Secret* et Charlie dans *Non-classifié*.
  - Le cheval de Troie de Bob va travailler en tant que sujet *Secret*, car créé par un sujet *Secret* par la propriété $\circledast$!
  - En conséquence, le cheval de Troie ne pourra pas créer de fichier de classe *Non-classifié*, par la même propriété $\circledast$.
  - Finalement, Charlie ne pourra lire aucun fichier créé par Alice, Bob ou le cheval de Troie par la propriété de sécurité simple.

Bell–LaPadula et le changement de classes de sécurité

- Le modèle donné ne fait pas référence aux modifications possibles des étiquettes de sécurité – principe de **tranquilité**.
- Le principe de tranquilité peut être relaxé, mais certaines extensions peuvent causer un flux d'informations non-désiré :
  - Diminuer la classification des objets peut entraîner la fuite d'information.
  - Augmenter, en tant que sujet dans la classe C, la classification des objets des classes dominées par C peut aussi entraîner la fuite d'information – l'objet disparaît de l'horizon des sujets de niveau de sécurité inférieur.
  - Par contre, augmenter, en tant que sujet de classe C, le niveau de sécurité des objets de *la même classe* ne crée pas de flux d'information contraire non-permis.

Bell–LaPadula et canaux couverts

- Les modèles de treillis n'empêchent pas l'existence des *canaux couverts* :
  - Un sujet secret A acquiert une large quantité de mémoire dans le système.
  - Un autre utilisateur B non-secret peut avoir une idée de cette quantité en demandant lui-même de la mémoire au système.
  - Si le système ne peut pas lui fournir assez, il peut déduire des informations sur la taille de la mémoire acquise par A.
- La découverte des canaux couverts est hors de portée des modèles basés sur les treillis.

Flux d'information dans des systèmes à ressources partagées

- Le modèle Bell-LaPadula ne prend pas en compte l'existence de ressources partagées entre les classes différentes de sécurité.
  - Mémoire limitée – flux d'information par *déni de service*.
  - Accès au même fichier en *exclusion mutuelle* (problème lecteurs-rédacteurs).
  - Exécution sur une même machine des opérations des niveaux différents, avec apparition de priorités
    - Flux d'information par *temporisation*.
  - Autres situations de flux d'information *implicite*.
- Un modèle plus fin pour l'analyse des *programmes* écrits dans un langage de programmation.

Qqs intuitions sur le flux d'information dans un langage de programmation

- Langage de programmation, variables dans deux classes de sécurité : H et L.
- On ne veut pas permettre aux utilisateurs du programme qui n'observent que des valeurs de classe L de déduire les valeurs des variables H.
  - $x : H$ et $y : L$.
  - $x := y$: flux d'information explicite.
  - `if` $x = 0$ `then` $y := 1$ else $y := 2$: flux d'information implicite.
  - De même pour `if` $x < 2$ `then` $y := 1$ `else` $y := 2$.
  - `if` $x = 0$ `then` $x := 1234 * 2345 * 3456 * 4567 * 5678....; y := 1$: flux d'information par temporisation.
- Principe de la **non-interference** (formulation du principe de non-fuite d'information) :

  Une variation dans les valeurs des variables de haut niveau ne devraient pas créer une variation dans les valeurs des variables de bas niveau.

Flux d'information selon Denning

- Cadre : treillis de classes de sécurité $(SC, \rightarrow)$, disposant ainsi de lub et de glb.
- Associations de classes de sécurité à chaque variable : $l : Vars \longrightarrow SC$.
- Conditions de conformité des classes de sécurité avec les instructions du programme.
- Constantes déclarées de classe *Low* $= \mathsf{lub}(SC)$.
- Déclarations :

$$x : \texttt{integer } \mathbf{class}\, \{C_1, \ldots, C_k\}$$

  - $l(x) = \mathsf{lub}(C_1, \ldots, C_k)$.
- Affectations :

$$y := f(x_1, \ldots, x_k)$$

  - $l(y) \geq \mathsf{lub}\big(l(x_1), \ldots, l(x_k)\big)$.
- Cas particulier : tableaux

$$y := a[i]$$

  - $l(y) \geq \mathsf{lub}\big(l(a), l(i)\big)$.
  - Le tableau possède une classification de sécurité C et l'indice en possède une autre !
  - Les éléments du tableau possèdent la même classification de sécurité.

Flux d'information selon Denning (2)

- Composition d'instructions :

$$\texttt{begin } S_1; S_2; \ldots; S_n \texttt{ end}$$

  - Les contraintes sur les classifications des variables sont l'union des contraintes sur chaque instruction.

- Tests :

$$\texttt{if } f(x_1, \ldots, x_k) \texttt{ then } S_1 \texttt{ else } S_2$$

  - $\mathsf{lub}\big(l(x_1), \ldots, l(x_k)\big)$
    $\leq \mathsf{glb}\big\{l(y) \mid y \text{ est la cible d'une affectation dans } S_1 \texttt{ ou } S_2\big\}$
  - En plus, les contraintes impliquées par S_1 et S_2 doivent être satisfaites.

- Boucles :

$$\texttt{while } f(x_1, \ldots, x_k) \texttt{ do } S$$

  - $\mathsf{lub}\big(l(x_1), \ldots, l(x_k)\big) \leq \mathsf{glb}\big\{l(y) \mid y \text{ est la cible d'une affectation dans } S\big\}$
  - En plus, les contraintes impliquées par S doivent être satisfaites, et
  - ... la boucle doit s'arrêter (sinon flux d'information par temporisation !)

Flux d'informations selon Denning (3)

- Déclarations de procédures :

$$\texttt{procedure qqchose}(i_1 : \text{type}, i_2 : \text{type}, \ldots, \texttt{ out } o_1 : \text{type}, \ldots)$$

$$\{\ S \text{ // corps de la procédure } \}$$

  - Les paramètres $i_1, i_2, \ldots$ sont considérés *d'entrée*, donc pas d'affectation à ces paramètres dans S !
  - Les contraintes impliquées par S *sont considérées comme des équations à vérifier sur les classes de sécurité des variables* $i_1, i_2, \ldots, o_1, o_2, \ldots$.
    - Ces équations seront toujours d'une forme très simple : $i_k \leq o_m$ ou $o_k \leq o_m$ ou $i_k \leq C$ ou $o_m \leq C$ ou $o_m \geq C$ ou
    - On va noter alors ces équations $Eq(S)$.
- Appels de procédures :

$$\texttt{qqchose}(x_1, \ldots, x_m, y_1, \ldots, y_n)$$

  - Dans les équations $Eq(S)$, on remplace les variables $i_1, i_2, \ldots,$ et $o_1, o_2, \ldots$ avec $x_1, x_2, \ldots$ et, respectivement, $y_1, y_2, \ldots$ et on *vérifie* les

Flux d'information selon Denning (4)

- Concurrence :
  - Ce qui est spécifique concerne l'utilisation des **sémaphores** :

$$\texttt{begin } S_1; \texttt{wait}(s); S_2 \texttt{ end}$$

  - Pour tout sémaphore y,

$$l(y) \leq \{l(z) \mid z \text{ cible d'une affectation dans les instructions qui suivent le } \texttt{wait}(y)\}$$

Flux d'informations selon Denning (5)

– **Théorème :** Si l'affectation des classes de sécurité aux variables dans un programme respecte toutes les contraintes ci-dessus, alors le programme ne permet pas de flux d'information non-désiré d'une variable x à une variable y avec $l(x) \geq l(y)$.
– *Preuve :* théorie des types + réécriture.

Autres modèles de flux d'information dans les programmes

- **Program slicing :** calculer les intérdépendences entre les instructions du programme.
- **Tranche** (angl. *slice*) d'un programme par rapport à un critère γ = un sous-ensemble des instructions du programme qui sont relevantes du point de vue du critère γ.
- **Program slicing** par rapport à un critère γ = action de trouver les tranches relevantes d.p.d.v. du critère γ.
- Très utilisé dans le debogage des programmes, dans la parallélisation, dans le test, etc.
- ... mais peut devenir une source d'inspiration dans l'analyse de flux d'information.

Program slicing selon Weiser

- *Définition plus formelle* : un sous-ensemble Q d'un programme P représente une tranche de P par rapport à un critère γ si P et Q calculent les mêmes résultats par rapport à γ.
- Exemple de critère : valeur d'une variable v après une instruction λ.
- Pour calculer la tranche *minimale*, on doit trouver de manière itérative les instructions qui affectent la valeur de v.
- Exemple rapide pour un critère du genre (λ, v).

Formalisation du program slicing à la Weiser (1)

- Le graphe de contrôle d'un programme :
  - Un noeud pour toute instruction/branchement dans le programme.
  - Arêtes connectant les noeuds selon le flux de contrôle.
- $\mathsf{REF}(n)$ = ensemble de variables référencées (= utilisées) dans le node n.
- $\mathsf{DEF}(n)$ = ensemble de variables définies (= qui prennent des nouvelles valeurs) dans le node n.
- Noeud n est **flux-dépendend** du noeud n' s'il existe $x \in \mathsf{DEF}(n) \cap \mathsf{REF}(n')$ et il existe un chemin orienté de n à n'.
- Noeud n est **post-dominé** par noeud n' si tous les chemins d'exécution qui commencent en n et qui mènent au noeud final du programme passent par n'.
- Noeud n est **contrôlé** par le noeud n' si
  - Il existe un chemin de n' à n pour lequel tout noeud n'' sur ce chemin est post-dominé par n et
  - n' n'est pas post-dominé par n.
- Pour tout noeud de branchement b, le **domaine d'influence** de b est l'ensemble de noeuds qui sont contrôlés par b. Notation : $INFL(b)$.

Formalisation du program slicing à la Weiser (2)

- Soit $\gamma = (n, V)$ un critère (V est un ensemble de variables).
- On cherche à calculer, pour chaque noeud m, les *variables relevantes* $R_\gamma(m)$ qui influencent les variables dans V juste avant d'exécuter le noeud n.
  - Influence = valeurs différentes pour au moins une variable dans $R_\gamma(m)$ implique valeur différente pour au moins une variable dans V.
  - Comme pour la noninterférence ! sauf que c'est calculé à la volée.
- On calcule d'abord, pour tout noeud m, les variables directement relevantes pour γ au noeud m, $R^0_\gamma(m)$:
  - Pour le noeud cible, $R^0_\gamma(n) = V$ (bien-sûr !)
  - Pour tout $m \longrightarrow k$ dans le graphe de contrôle, $R^0_\gamma(m)$ comprend toutes les variables v pour lesquelles
    1. soit $v \in R^0_\gamma(k)$ et $v \notin \mathsf{DEF}(m)$,
    2. soit $v \in \mathsf{REF}(m)$ et $\mathsf{DEF}(m) \cap R^0_\gamma(k) \neq \emptyset$.
  - À partir de R^0_γ on peut calculer l'ensemble d'instructions directement relevantes pour γ :

$$S^0_\gamma = \{m \mid \text{ pour tout noeud } k, \text{ si } m \longrightarrow k \text{ alors } \mathsf{DEF}(m) \cap R^0_\gamma(k) \neq \emptyset\}$$

Formalisation du program slicing à la Weiser (3)

- Ensuite on cherche les variables qui influent *de manière indirecte* les variables dans le critère γ.
- Ces sont les variables qui se trouvent dans les instructions de test ou les conditions des boucles.
- On retrouve les noeuds de branchement qui sont relevants par leur influence sur les noeuds de S^0_γ :

$$B^0_\gamma = \{b \mid b \text{ noeud de branchement et } S^0_\gamma \cap INFL(b) \neq \emptyset\}$$

- Et on cherche maintenant les noeuds relevants pour le critère $(b, \mathsf{REF}(b))$ pour tout $b \in B^0_\gamma$.
- Alors les variables influant de manière indirecte *de premier degré* le critère γ au noeud n sont :

$$R^1_\gamma(n) = R^0_\gamma(n) \cup \bigcup_{b \in B^0_\gamma} R^0_{(b,\mathsf{REF}(b))}(n)$$

- On peut donc construire les noeuds qui influent de manière indirecte de premier degré le critère γ :

$$S^1_\gamma = B^0_\gamma \cup \{m \mid \text{ pour tout } k, \text{ si } m \longrightarrow k \text{ alors } \mathsf{DEF}(m) \cap R^1_\gamma(k) \neq \emptyset\}$$

Formalisation du program slicing à la Weiser (4)

- On continue ce processus, avec la construction de B^1_γ, R^2_γ, S^2_γ, etc. jusqu'au moment où

$$S^{j+1}_\gamma = S^j_\gamma$$

- L'ensemble S^j_γ obtenu alors représente la tranche minimale pour le critère γ !
- ... et R^j_γ l'ensemble de variables relevantes, à chaque noeud, pour le critère γ.
- Pour conformité,

$$\begin{aligned} B^j_\gamma &= \{b \mid b \text{ noeud de branchement et } S^j_\gamma \cap INFL(b) \neq \emptyset\} \\ R^{j+1}_\gamma(n) &= R^j_\gamma(n) \cup \bigcup_{b \in B^j_\gamma} R^j_{(b,\mathsf{REF}(b))}(n) \\ S^{j+1}_\gamma &= B^j_\gamma \cup \{m \mid \text{ pour tout } m, \text{ si } m \longrightarrow k \text{ alors } \mathsf{DEF}(m) \cap R^j_\gamma(k) \neq \emptyset\} \end{aligned}$$

Program slicing

- Technique effective permettant de découvrir les dépendances de flux de données.
- Peut être utilisée en tant qu détection de flux d'information non-désiré :
  - Si $v_1 \in R^j_{(n,v_2)}(n')$ et $l(v_1) \not\leq l(v_2)$ alors flux d'information non-autorisé !
- Peut être effectué aussi par des techniques en avant.
- **Problème** : on rejette des programmes du genre

$$\texttt{if } x = 0 \texttt{ then } y = 1 \texttt{ else } y = 1$$

  où il n'y a pas de flux d'information !
- Donc il y a des programmes qui respectent le principe de non-interference et qui sont rejettés par les techniques de type program slicing et même de type Denning.

Exemple

```
1. b := 1;
2. c := 2;
3. d := 3;
4. a := d;
5. if (a) {
6.     d := b+d;
7.     c := b+d;
     } else {
8.     b++;
9.    d := b+1;
     }
10. a := b+c;
```