

Détection de flux d'information illégaux

en vue de détection d'intrusion

Guillaume Hiet
Ludovic Me
Benjamin Morin
Valérie Viet Triem Tong

équipe SSIR
Supelec

jeudi 7 février 2008

Problème considéré

Insuffisance du contrôle d'accès

- Le contrôle d'accès discrétionnaire *contrôle* l'accès aux conteneurs d'information
- Mais ne gère pas l'accès à l'information une fois sortie de son conteneur initial

exemple :

	F_1	F_2	F_3
Alice	{ <i>read</i> , <i>write</i> }		{ <i>read</i> , <i>write</i> }
Bob	{ <i>read</i> }	{ <i>read</i> , <i>write</i> }	
Charlie		{ <i>read</i> , <i>write</i> }	

Petit exemple



F1



F2



F3

Etat 0

	F_1	F_2	F_3
Alice	{ <i>read, write</i> }		{ <i>read, write</i> }
Bob	{ <i>read</i> }	{ <i>read, write</i> }	
Charlie		{ <i>read, write</i> }	

Petit exemple



F1



F2



F3

Etat 1

Alice lit le contenu de F3 et l'écrit dans F1,

	F_1	F_2	F_3
Alice	{ <i>read, write</i> }		{ <i>read, write</i> }
Bob	{ <i>read</i> }	{ <i>read, write</i> }	
Charlie		{ <i>read, write</i> }	

Petit exemple



F1



F2



F3

Etat 2

Bob lit le contenu de F1, et l'écrit dans F2,

	F_1	F_2	F_3
Alice	{ <i>read, write</i> }		{ <i>read, write</i> }
Bob	{ <i>read</i> }	{ <i>read, write</i> }	
Charlie		{ <i>read, write</i> }	

Petit exemple



F1



F2



F3

Etat 3

Charlie lit le contenu de F2 : c'est une opération

	F_1	F_2	F_3
Alice	{read, write}		{read, write}
Bob	{read}	{read, write}	
Charlie		{read, write}	

- autorisée du point de vue *contrôle d'accès au conteneur*
- interdite du point de vue *contrôle d'accès à l'information*

Notre approche

Nous proposons d'observer les flux d'information dans le système en vue de **détecter des intrusions**

- Une *intrusion* se caractérise par un **flux d'information illégal**
- Pour les détecter, il faut savoir
 - 1 d'où l'information provient
 - 2 si elle est autorisée à se trouver dans le conteneur

**Nous souhaitons lever une alerte
si et seulement si
un flux d'information illégal a eu lieu**

Travaux connexes

Il existe beaucoup de travaux sur l'observation des flux d'information

- principalement au niveau des langages de programmation (Jif, FlowCaml..)
- pas de *vue globale du système d'information*
- pas de lien entre les diverses applications du système

Notre approche

Plus précisément, notre approche repose sur

- 1 Une *politique de flux* directement dérivée des règles de contrôle d'accès en vigueur sur le système
- 2 Pour chaque conteneur, deux *tags* dérivés de la politique qui permettront de connaître
 - l'origine de l'information
 - les droits attachés à cette information

c'est une approche dynamique

Plan

- 1 **Principe de détection**
 - Définir de la politique de flux
 - Positionner des tags de sécurité
 - Faire évoluer les tags en fonction des flux
- 2 Détection de l'intrusion donnée en exemple
- 3 Implémentation du modèle
- 4 Conclusions et perspectives

Définition de la politique de flux

La politique de flux

s'exprime par des couples de la forme

$$(\mathcal{I}, \mathcal{C})$$

où les \mathcal{I} sont des ensembles informations
et les \mathcal{C} des ensembles de conteneurs d'information

Un couple $(\mathcal{I}, \mathcal{C})$ appartenant à la politique de flux signifie que

toute information de \mathcal{I} a le droit de se trouver
dans n'importe quel conteneur de \mathcal{C}

La politique de flux est déduite du contrôle d'accès

	F_1	F_2	F_3
Alice	{ <i>read</i> , <i>write</i> }		{ <i>read</i> , <i>write</i> }
Bob	{ <i>read</i> }	{ <i>read</i> , <i>write</i> }	
Charlie		{ <i>read</i> , <i>write</i> }	

Cette matrice induit

- 6 conteneurs $F_1, F_2, F_3, F_A, F_B, F_C$
- 6 contenus initiaux $f_1, f_2, f_3, f_A, f_B, f_C$

Construction de la politique

Une information f_i peut aller dans un conteneur F_j , s'il existe au moins un utilisateur

- autorisé à lire F_j et donc à accéder à f_i
- autorisé à écrire dans F_j

La politique de flux est déduite du contrôle d'accès

	F_1	F_2	F_3
Alice	{ <i>read</i> , <i>write</i> }		{ <i>read</i> , <i>write</i> }
Bob	{ <i>read</i> }	{ <i>read</i> , <i>write</i> }	
Charlie		{ <i>read</i> , <i>write</i> }	

Cette politique de sécurité induit la politique de flux suivante :

(dû à Alice) ($\{f_1, f_3, f_A\}, \{F_1, F_3, F_A\}$)

(dû à Bob) ($\{f_1, f_2, f_B\}, \{F_2, F_B\}$)

(dû à Charlie) ($\{f_2, f_C\}, \{F_2, F_C\}$)

Positionner des tags de sécurité

Les tags sont définis à l'initialisation du système et évolueront suivant les flux d'information. A chaque conteneur d'information, on va associer 2 tags

- un *Read security tag* concernant les droits liés à l'information contenue :

Quels sont les mélanges d'information autorisés, pour quels conteneurs ?

- un *Write security tag* concernant les droits liés au conteneur

Quelles informations, ce conteneur peut-il accueillir ?

**Une information est autorisée à aller dans un conteneur
si et seulement si**

l'intersection de des deux tags de ce conteneur reste non vide

Tags de sécurité

Les tags sont définis à l'initialisation du système et évolueront suivant les flux d'information.

Notons

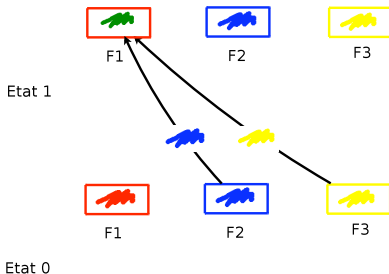
- \mathcal{P} la politique de flux.
- f l'information **initialement** contenue dans un conteneur F
- le **Read security tag** de F est

$$\{(\mathcal{I}, \mathcal{C}) \in \mathcal{P} \mid f \in \mathcal{I}\}$$

- le **Write security tag** de F est

$$\{(\mathcal{I}, \mathcal{C}) \in \mathcal{P} \mid F \in \mathcal{C}\}$$

Evolution des tags de sécurité



- le *Write security tag* ne change pas,
- le *Read security tag* est l'**intersection** des *Read security tags* liés aux conteneurs **accédés**

Le flux était légal si l'intersection reste non vide

Plan

- 1 Principe de détection
- 2 Détection de l'intrusion donnée en exemple**
- 3 Implémentation du modèle
- 4 Conclusions et perspectives

Retour sur notre exemple

	F_1	F_2	F_3
Alice	{ <i>read</i> , <i>write</i> }		{ <i>read</i> , <i>write</i> }
Bob	{ <i>read</i> }	{ <i>read</i> , <i>write</i> }	
Charlie		{ <i>read</i> , <i>write</i> }	

Cette politique de sécurité induit la politique de flux suivante :

(dû à Alice) ($\{f_1, f_3, f_A\}, \{F_1, F_3, F_A\}$) noté C_1

(dû à Bob) ($\{f_1, f_2, f_B\}, \{F_2, F_B\}$) noté C_2

(dû à Charlie) ($\{f_2, f_C\}, \{F_2, F_C\}$) noté C_3

Positionnement des *Read security tags*

à l'initialisation

le *Read security tag* d'un F_i est

$$\{(I, C) \in \{C_1, C_2, C_3\} \mid f_i \in I\}$$

$$C_1 = (\{f_1, f_3, f_A\}, \{F_1, F_A\}), \quad C_2 = (\{f_1, f_2, f_B\}, \{F_2, F_B\}), \\ C_3 = (\{f_2, f_C\}, \{F_2, F_C\})$$



F1



F2



F3

$$T_{read}(F_1) = \{C_1, C_2\}, \quad T_{read}(F_2) = \{C_2, C_3\}, \quad T_{read}(F_3) = \{C_1\}$$

Positionnement des *Write security tags*

à l'initialisation

le *Write security tag* d'un F_i est

$$\{(I, C) \in \{C_1, C_2, C_3\} | F_i \in I\}$$

$$C_1 = (\{f_1, f_3, f_A\}, \{F_1, F_A\}), C_2 = (\{f_1, f_2, f_B\}, \{F_2, F_B\}), \\ C_3 = (\{f_2, f_C\}, \{F_2, F_C\})$$



F1



F2



F3

$$T_{write}(F_1) = \{C_1, C_2\}, \quad T_{write}(F_2) = \{C_2, C_3\}, \quad T_{write}(F_3) = C_1$$

Positionnement des *security tags*



F1



F2



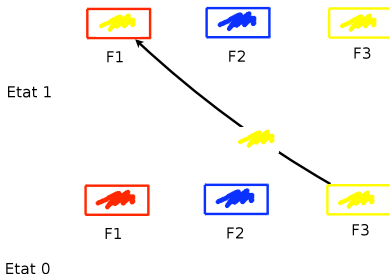
F3

Etat 0

$$T_{read}(F_1) = \{C_1, C_2\}, \quad T_{read}(F_2) = \{C_2, C_3\}, \quad T_{read}(F_3) = \{C_1\}$$

$$T_{write}(F_1) = \{C_1, C_2\}, \quad T_{write}(F_2) = \{C_2, C_3\}, \quad T_{write}(F_3) = \{C_1\}$$

Evolution des tags suivant les flux

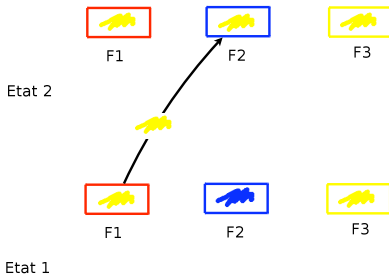


à l'état 1, Alice a lu le contenu de F_3 et l'a placé dans F_1

$$T_{\text{read}}(F_1) = \{C_1\}, \quad T_{\text{read}}(F_2) = \{C_2, C_3\}, \quad T_{\text{read}}(F_3) = \{C_1\}$$

$$T_{\text{write}}(F_1) = \{C_1, C_2\}, \quad T_{\text{write}}(F_2) = \{C_2, C_3\}, \quad T_{\text{write}}(F_3) = \{C_1\}$$

Evolution des tags suivant les flux



à l'état 2, Bob a lu le contenu de F_1 et l'a placé dans F_2

$$T_{read}(F_1) = \{C_1\}, \quad T_{read}(F_2) = \{C_1\}, \quad T_{read}(F_3) = \{C_1\}$$

$$T_{write}(F_1) = \{C_1, C_2\}, \quad T_{write}(F_2) = \{C_2, C_3\}, \quad T_{write}(F_3) = \{C_1\}$$

Plan

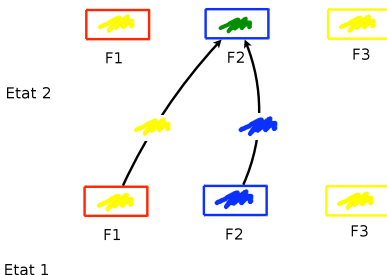
- 1 Principe de détection
- 2 Détection de l'intrusion donnée en exemple
- 3 Implémentation du modèle**
 - Au niveau système
 - Au niveau langage
- 4 Conclusions et perspectives

Conclusion sur le modèle

Détection des flux illégaux

Nous avons montré qu'un flux est légal si l'intersection des tags de tous les objets modifiés est non vide.

Ce résultat repose sur l'hypothèse d'une observation exacte
i.e. : le résultat d'un flux dépend effectivement de tous les objets utilisés en entrée



Implémentation du modèle

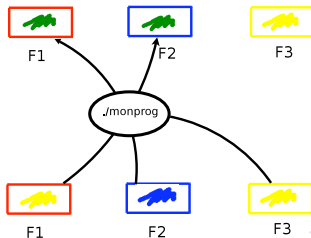
Première implémentation : au niveau système

- Implémentation sous la forme d'un patch pour le noyau linux
- L'observation des appels systèmes *read* et *write* modélisent l'observation des flux
- Les applications sont vues comme des boites noires : toutes les entrées dépendent de toutes les sorties

Implémentation du modèle

Bilan de l'implémentation au niveau système

- + Valide l'approche
- + Permet une observation *globale* des flux
- + Permet de détecter des attaques *non-préalablement connues*
- A une granularité trop grosse au niveau des applications



Implémentation du modèle : au niveau langage

Seconde implémentation : au niveau langage

- Suivi dynamique des flux d'information dans des programmes Java
- Suivi fait au niveau bytecode
- Permet de raffiner l'évolution des tags au niveau des objets systèmes
- Les deux implémentation coopèrent

Plan

- 1 Principe de détection
- 2 Détection de l'intrusion donnée en exemple
- 3 Implémentation du modèle
- 4 Conclusions et perspectives**

Bilan

Une méthode de détection d'intrusion

- repose sur l'observation des flux d'information
- utilise une politique de flux dérivée des règles de contrôle d'accès
- caractérise une intrusion comme un flux illégal
- est correct et complète si l'observation des flux est exacte
- les implémentations raffinent au fur et à mesure l'observation des flux.

Perspectives

Pistes de travail

- Etudier le lien entre politiques de sécurité et politique de flux
- raffiner encore l'observation des flux, par interprétation abstraite ?