Prévention de fuites d'information dans des applications interactives

Andrew Bedford, Stephen Chong, Josée Desharnais, Scott Moore, Nadia Tawbi

Université Laval & Harvard University

Février 2015

La sécurité

- Quand est-ce qu'un programme est "sécuritaire" ?
  - Confidentialité
  - Intégrité
  - Disponibilité
- C'est important quelque soit l'application (blog, banque, centrale nucléaire, ...) !
- Des milliards de dollars sont investis à chaque année

Et pourtant...

Quelques statistiques

Vulnerability	Vulnerable sites among those surveyed
Cross-site scripting	31.5%
Information leakage	23.3%
Predictable resource location	10.2%
SQL injection	7.9%
Insufficient access control	1.5%
HTTP response splitting	0.8%

(Web Application Security Consortium, 2008)

Injection SQL - Exemple

```
statement = "SELECT * FROM users WHERE name ='" + userName +
      "';"
```

```
userName = "' or '1'='1"
-->
SELECT * FROM users WHERE name = '' OR '1'='1';

userName = "a';DROP TABLE users; SELECT * FROM userinfo
    WHERE 't' = 't';"
-->
SELECT * FROM users WHERE name = 'a';DROP TABLE users;
    SELECT * FROM userinfo WHERE 't' = 't';
```

Flot d'information de l'utilisateur à la base de donnée

Cross-site scripting (XSS) - Exemple

- Par l'URL :
  http ://www.a-video-site.org ?q=keyboard cat
  Results for "keyboard cat"

```
?q=<script type='text/javascript'>alert('MOUAHAHAHAHAH')
    ;</script>
```

- Partout où on génère du HTML à partir d'une entrée d'un utilisateur (post dans un forum, une signature, un nom d'utilisateur, ...)

Quelques observations

1. En ce moment, c'est au développeur à qui revient la responsabilité de prendre en compte ces flots d'informations
2. Une seule faille peut compromettre tout le système
3. La complexité et l'envergure des applications ne cesse d'augmenter

Ce qu'il nous faut : des outils nous permettant de détecter et prévenir automatiquement ce genre de failles de sécurité !

Notre travail

Afin de pouvoir éventuellement construire de tels outils, nous avons choisit d'étudier comment on pourrait détecter et prévenir les fuites d'informations dans un petit language impératif intéractif.

(variables)	x	$\in$	$\mathcal{V}$
(nombres entiers)	n	$\in$	$\mathbb{Z}$
(canaux)	ch	$\in$	C
(expressions)	e	$::=$	$x \mid n \mid ch \mid e_1$ **op** e_2
(commandes)	c	$::=$	**skip** $\mid x := e \mid c_1; c_2 \mid$
			if e **then** c_1 **else** c_2 **end** $\mid$
			while e **do** c **end** $\mid$
			receive x_1 **from** $x_2 \mid$ **send** x_1 **to** x_2

Notes : On utilise 0 pour *faux* et non-zéro pour *vrai*. Les programmes communiquent uniquement à travers des canaux de communication (fichier, clavier, écran, connection réseau, utilisateur, etc.).

Exemples de fuites

```
(* Flots explicites non-securitaires *)
send password to publicHTML;
x := password + 1010101;
send x to publicHTML;
```

```
if privateValue > 0 then
  (* Flot implicite non-securitaire *)
  send 42 to publicChannel
end
```

Lorsque `privateValue > 0`, `publicChannel` $= [42]$

Lorsque `privateValue <= 0`, `publicChannel` $= []$

Exemple de fuites

Vu que notre approche étudie les programmes intéractifs, elle doit être sensible au progrès.

```
send 0 to publicFile;
receive secretValue from topSecretFile;
while secretValue >= 0 do
  skip (*Boucle infinie*)
end;
send 1 to publicFile (*Fuite*)
```

Lorsque `secretValue >= 0`, `publicFile` = [0]

Lorsque `secretValue < 0`, `publicFile` = [0,1]

Problème : Comment prévenir ces fuites ?

En utilisant le contrôle de flots d'information ! L'idée de base est d'associer à l'information elle-même une étiquette correspondant à son niveau de sécurité et de la propager. On utilise :

- *L* (*Low*) : Pour l'information public
- *H* (*High*) : Pour l'information privée
- *U* (*Unknown*) : Pour l'information de niveau inconnu statiquement

```
receive x from publicFile; (*x est L*)
receive x from topSecretFile; (*x est H*)
y := x + 5; (*y est H*)
x := 0; (*x est L*)
if randomValue
  then c := publicFile (* c est un canal L *)
  else c := topSecretFile (* c est un canal H *)
end;
receive z from c (*z est U (statiquement)*)
```

Le contrôle de flots d'information

Pour éviter les fuites, on applique une politique de sécurité appelée la non-interférence.

Définition

(Non-interférence) On dit qu'un programme est non-interférent lorsque l'information confidentielle n'affecte pas le comportement public du système (dans notre cas, les outputs aux canaux publics). Si un programme ne respecte pas cette propriété, alors on dira que ce programme a une fuite d'information.

Confidentialité : Information $H \not\rightarrow$ Sorties de niveau L
Intégrité : Information $L \not\rightarrow$ Sorties de niveau H

On applique cette politique en utilisant un système de type.

1. Si le code ne contient pas de fuites évidentes, alors il est instrumenté. C'est-à-dire qu'il est traduit dans un language cible supportant des variables étiquettes et des commande sont insérées pour "tracker" et contrôler les flots d'informations lors de l'exécution.
2. Le code est ensuite évalué partiellement (optimisation)

$$\text{(S-Chan)}\ \frac{\mathit{levelOfChan}(nch) = \ell}{\Gamma \vdash nch : (int_\ell\ chan)_L} \qquad \text{(S-Int)}\ \Gamma \vdash n : int_L \qquad \text{(S-Var)}\ \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \text{(S-Op)}\ \frac{\Gamma \vdash e_1 : int_{\ell_1} \qquad \Gamma \vdash e_2 : int_{\ell_2}}{\Gamma \vdash e_1\ \mathbf{op}\ e_2 : int_{\ell_1 \sqcup \ell_2}}$$

$$\text{(S-Skip)}\ \Gamma, pc, hc \vdash\ \mathbf{skip}\ :\ T, hc, \Gamma, \mathsf{skip} \qquad \text{(S-Assign)}\ \frac{\Gamma \vdash e : \sigma_\ell}{\Gamma, pc, hc \vdash x := e :\ T, hc, \Gamma[x \mapsto \sigma_{pc \sqcup \ell}], \mathsf{genassign}}$$

$$\text{(S-Recv)}\ \frac{\Gamma(x_2) = (int_\ell\ chan)_{\ell'}}{\Gamma, pc, hc \vdash \mathbf{receive}\ x_1\ \mathbf{from}\ x_2 :\ T, hc, \Gamma[x_1 \mapsto int_{(pc \sqcup \ell \sqcup \ell')}, x_2 \mapsto (int_\ell\ chan)_{\ell' \sqcup pc}], \mathsf{genrec}}$$

$$\text{(S-Send1)}\ \frac{\Gamma(x_1) = int_{\ell_1} \qquad \Gamma(x_2) = (int_\ell\ chan)_{\ell_2} \qquad (pc \sqcup hc \sqcup \ell_1 \sqcup \ell_2) \sqsubseteq_s \ell}{\Gamma, pc, hc \vdash \mathbf{send}\ x_1\ \mathbf{to}\ x_2 :\ T, hc, \Gamma, \mathsf{send}\ x_1\ \mathsf{to}\ x_2}$$

$$\text{(S-Send2)}\ \frac{\Gamma(x_1) = int_{\ell_1} \qquad \Gamma(x_2) = (int_\ell\ chan)_{\ell_2} \qquad (pc \sqcup hc \sqcup \ell_1 \sqcup \ell_2) \sqsubseteq_m \ell \qquad (pc \sqcup hc \sqcup \ell_1 \sqcup \ell_2) \not\sqsubseteq_s \ell}{\Gamma, pc, hc \vdash \mathbf{send}\ x_1\ \mathbf{to}\ x_2 : T, hc \sqcup \ell_2, \Gamma, \mathsf{gensend}}$$

$$\text{(S-If)}\ \frac{\begin{array}{c}\Gamma \vdash e : int_\ell \qquad h_3 = \sqcup_{j \in \{1,2\}} d(\Gamma, pc \sqcup \ell, cmd_j) \\ \bot \notin \mathrm{ran}(\Gamma_1 \sqcup \Gamma_2) \qquad h = (h_1 \sqcup h_2 \sqcup h_3 \sqcup level(t_1 \oplus_\ell t_2)) \qquad \Gamma, pc \sqcup \ell, hc \vdash cmd_j : t_j, h_j, \Gamma_j, [\![cmd_j]\!] \quad j \in \{1,2\}\end{array}}{\Gamma, pc, hc \vdash\ \mathbf{if}\ e\ \mathbf{then}\ cmd_1\ \mathbf{else}\ cmd_2\ \mathbf{end} : (t_1 \oplus_\ell t_2), h, \Gamma_1 \sqcup \Gamma_2, \mathsf{genif}}$$

$$\text{(S-Loop)}\ \frac{\begin{array}{c}O(e, cmd, \Gamma \sqcup \Gamma') = t' \qquad h = d(\Gamma, pc \sqcup \ell_1, cmd) \qquad \ell = level(t) \qquad \ell' = level(t') \\ \bot \notin \mathrm{ran}(\Gamma \sqcup \Gamma') \qquad \Gamma \sqcup \Gamma' \vdash e : int_{\ell_1} \qquad \Gamma \sqcup \Gamma', (pc \sqcup \ell_1), (hc \sqcup \ell \sqcup h') \vdash cmd : t, h', \Gamma', [\![cmd]\!]\end{array}}{\Gamma, pc, hc \vdash \mathbf{while}\ e\ \mathbf{do}\ cmd\ \mathbf{end} : t', h \sqcup h' \sqcup \ell', \Gamma \sqcup \Gamma', \mathsf{genwhile}}$$

$$\text{(S-Seq1)}\ \frac{\Gamma, pc, hc \vdash cmd_1 : D, h, \Gamma_1, [\![cmd_1]\!]}{\Gamma, pc, hc \vdash cmd_1; cmd_2 : D, h, \Gamma_1, [\![cmd_1]\!]} \qquad \text{(S-Seq2)}\ \frac{\begin{array}{c}t_1 \neq D \qquad \Gamma, pc, hc \vdash cmd_1 : t_1, h_1, \Gamma_1, [\![cmd_1]\!] \\ \Gamma_1, pc, h_1 \vdash cmd_2 : t_2, h_2, \Gamma_2, [\![cmd_2]\!]\end{array}}{\Gamma, pc, hc \vdash cmd_1; cmd_2 : t_1 \S t_2, h_2, \Gamma_2, [\![cmd_1]\!]; [\![cmd_2]\!]}$$

- Γ est un environnement de typage (map les variables à leur types)

```
x := 0; (*Γ(x) = int_L*)
receive x from secretFile; (*Γ(x) = int_H*)
c := secretFile (*Γ(c) = (int_H chan)_L*)
```

- *pc* est le program context, une variable niveau. Il permet de détecter les fuites implicites Il représente le contexte dans lequel on est présentement.

```
(*pc = L*)
if privateValue then
  (*pc = H*)
  send 42 to publicChannel
end
```

- *hc* est le halting context, une variable niveau. Il permet de détecter les fuites par progrès. Il représente le niveau d'information qui aurait pu halter le progrès du programme. En d'autres mots, ...
- Afin de pouvoir mettre à jour le *hc*, chaque commande à un type de terminaison qui lui est associé.

Les types de terminaisons sont les suivants $\{T, D, M_L, M_U, M_H\}$. Où :

T : si la commande termine toujours

D : si la commande diverge toujours

M_L : si la terminaison de la commande est inconnue statiquement mais dépend seulement d'information de niveau L

M_H : si la terminaison de la commande est inconnue statiquement mais dépend seulement d'information de niveau H

M_U : si la terminaison de la commande est inconnue statiquement mais dépend à la fois d'information de niveau U.

Exemple de terminaisons

```
x := 1; (*T*)
(*hc = L*)
send 1 to publicChannel
```

```
while 1 do skip end; (*D*)
(*hc = L*)
send secret to publicChannel
```

```
while secret >=0 do
  skip
 end; (*M_H*)
 (*hc = H*)
 send 1 to publicChannel
```

```
while secret >= 0 do
  secret := secret - 1;
end; (*T*)
(*hc = L*)
send 1 to publicChannel
```

Terminaison des boucles

Pour déterminer la terminaison des boucles, on utilise un oracle.

Démonstration

- Notre approche en bref :
  1. Vérifier avec notre système de type si le code contient des fuites évidentes. Si c'est le cas, on rejette.
  2. Sinon, le programme est instrumenté (de manière prouvée correct).
  3. Le code est ensuite évalué partiellement.
- Grâce à l'oracle et au niveau U, on accepte plus de programmes que les autres approches similaires.
- Le fait que l'on réécrit notre programme pour qu'il soit non-interférent veut dire qu'il peut être optimisé.

Questions ?