Détection de Malwares par graphes de comportement

XTRA 2018

Dylan Marinho
sous la direction de Jean Quilbeuf, Inria

ISTIC, Université de Rennes 1
DIT, École Normale Supérieure de Rennes

16 mai 2018

Fatemeh Karbalaie, Ashkan Sami, and Mansour Ahmadi.
Semantic malware detection by deploying graph mining.
IJCSI International Journal of Computer Science Issues, Janvier 2012.

Introduction

- Difficulté pour définir ce qu'est un *malware*
  - application logicielle qui fait des actions non souhaitées par l'utilisateur...
  - *In fine*, au cas par cas
- Différentes méthodes pour analyser des *malwares* :
  - par l'analyse de signatures
  - par l'analyse statique du code binaire
  - par une analyse dynamique dans une *sand box*

Introduction

- par l'analyse de signatures
  - \+ Rapidité
  - \+ Peu de faux positifs
  - – Facilités d'offuscation

Introduction

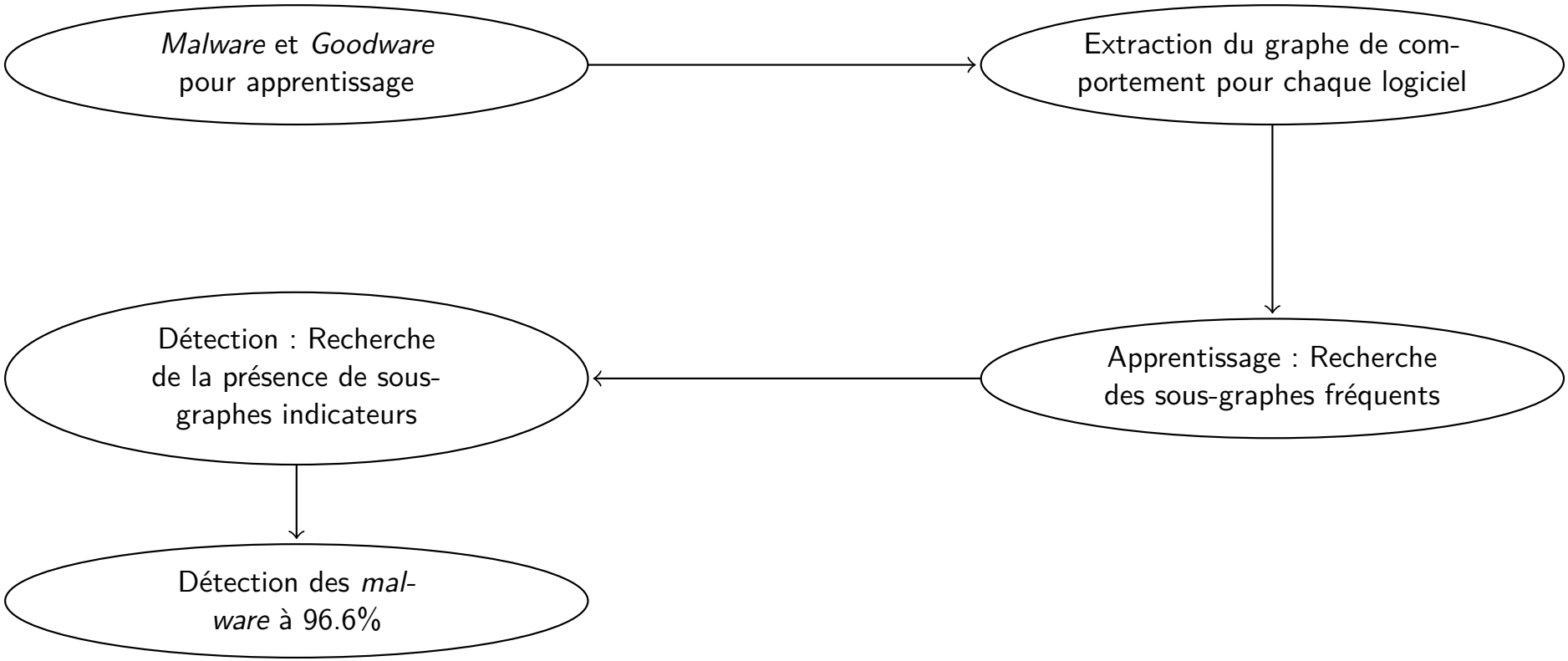
- par l'analyse statique du code binaire
  - \+ Plus précis que les signatures
  - – Grand coût en ressources (humains ou machine)
  - – Quelques techniques d'offuscation

Introduction

- par une analyse dynamique dans une *sandbox*
  - \+ Comportement réel du programme
  - \+ Relativement rapide
  - – Techniques pour détecter une *sandbox*

C'est la méthode que l'on choisit ici

Principe général



Modélisation de la sémantique

Modélisation

On s'intéresse ici aux dépendances entre les appels système effectués

Construction du graphe de comportement

- Les nœuds sont les appels système
- Une flèche est présente entre les appels A et B si une valeur est partagée entre ces appels

Une première amélioration

Limite

Une telle modélisation crée des graphes de très grande taille

Amélioration

Afin de réduire la taille des graphes :

- on restreint le nombre d'appels système étudiés
- on ne s'intéresse pas aux valeurs de retour 0 et 1

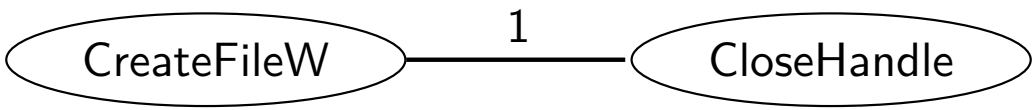
Appels intéressants

Bibliothèques retenues pour l'étude

- *kernel32.dll*
- *user32.dll*
- *ws_s32.dll*
- *advapi32.dll*
- *wininet.dll*
- *CreateProcess.dll*

Un exemple de construction

Appel système	Paramètres	Valeur de retour
CreateFileW	*lpFileName* : $0 \times 00415F2C$	$\mathbf{0 \times 000025A8}$
CloseHandle	*hObject* : $\mathbf{0 \times 000025A8}$	0×00000001



Définition

Definition (Recherche de sous-graphes fréquents)

Soit $GS = \{G_i | i \in \{1, ..., n\}\}$ un ensemble de graphes, un graphe g et un support $s \in \mathbb{N}$. Considérons alors

$$\zeta(g, G) = \begin{cases} 1 & \text{si } g \text{ est isomorphe à un sous-graphe de } G \\ 0 & \text{si } g \text{ n'est pas isomorphe à un sous-graphe de } G \end{cases}$$

et ensuite $\sigma(g, GS) = \sum_{G \in GS} \zeta(g, G)$.
$\sigma(g, GS)$ décrit le nombre d'occurrences de g (à isomorphisme près) comme sous-graphes d'éléments de GS.
Rechercher les sous-graphes fréquents d'une collection consiste à trouver tous les graphes g tels que $\sigma(g, GS) \geq s$

Algorithme utilisé

gSpan (*graph-based substructure pattern mining*)

Permet de chercher des sous-graphes courants avec une certaine fréquence minimale

Idée de l'algorithme

Construire un ordre lexicographique sur les graphes à partir d'un parcours en profondeur (*DFS*)

Utilisation

On utilisera *gSpan* avec un support variant de 4% à 9%

Classification

Idée

Les sous-graphes communs à une famille sont caractéristiques de cette famille

- Apprentissage :
  - Recherche des sous-graphes communs sur le jeu d'apprentissage
  - Construction, pour chaque binaire, d'un vecteur ayant une composante par sous-graphe, valant 0 ou 1
  - Application d'un algorithme de *random forest*
- Détection :
  - Construction du vecteur pour l'échantillon à classifier
  - Utilisation de *random forest*

Échantillon

- Échantillon composé de 404 *malwares* et 349 *goodwares*
- Répartition des *malwares* :

Nom	Nombre
Constructor	188
Porte dérobée	162
Exploit	54

- 10−validation croisée

Détection du système (en pourcentages)

Support	Rappel	Précision	Faux négatifs [1]	F-mesure [2]
4	89,9	88,2	10,1	89,1
5	88,7	87,4	11,3	88
6	89,9	87,8	10,1	88,8
7	94,6	96,3	5,4	95,4
8	96,1	98,7	3,9	97,4
9	96,6	98,7	3,4	97,6

(en pourcentages)

1. Pourcentage par rapport aux nombre de *malware* à détecter (404).
2. F-mesure $= 2\frac{\text{précision} \times \text{rappel}}{\text{précision} + \text{rappel}}$

Conclusion

- Méthode intéressante et efficace
- Possibilité d'amélioration au niveau de l'extraction des sous-graphes : *comment s'assurer que le comportement malicieux a bien été capturé ?*
- Une autre amélioration : *Peut-on interpréter les sous-graphes issus de l'analyse d'une famille de malwares ?*