Étude des sous-graphes communs des Graphes de Dépendance d'Appels Systèmes pour la classification de logiciels malveillants

Intervention pour les TS2 – Lycée Notre Dame des Missions

Dylan Marinho

7 Janvier 2019

Mon parcours

Juin 2015 Baccalauréat Scientifique

Mon parcours

Juin 2015	Baccalauréat Scientifique
2015 – 2017	Classes Préparatoires aux Grandes Écoles (MPSI – MP*) Lycée Marcelin Berthelot (94)

Mon parcours

Juin 2015	Baccalauréat Scientifique
2015 – 2017	Classes Préparatoires aux Grandes Écoles (MPSI – MP*) Lycée Marcelin Berthelot (94)
2017 – 2018	Licence Recherche & Innovation ENS Rennes (1A) et Université de Rennes 1 (L3)

Mon parcours

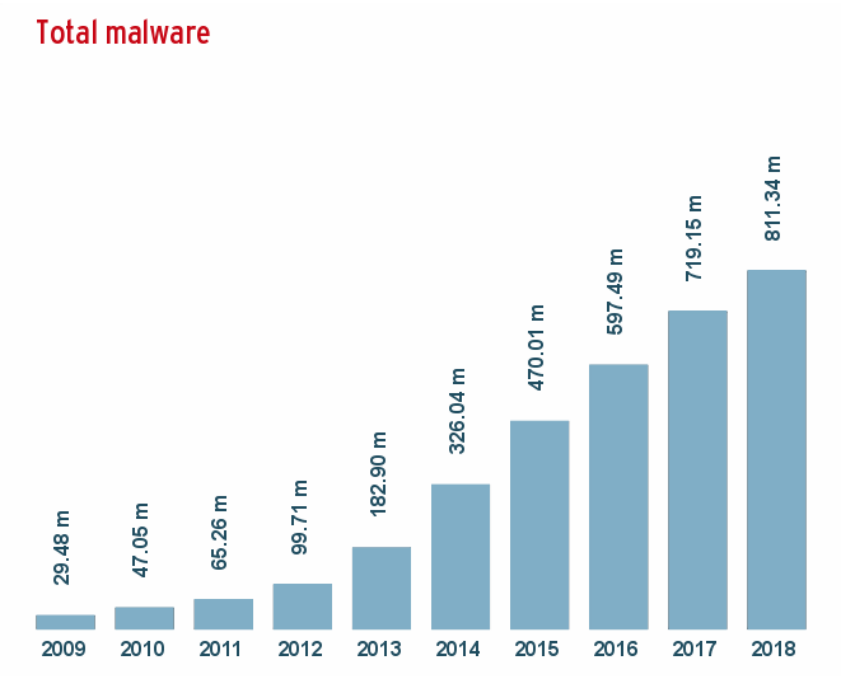
Juin 2015	Baccalauréat Scientifique
2015 – 2017	Classes Préparatoires aux Grandes Écoles (MPSI – MP*) Lycée Marcelin Berthelot (94)
2017 – 2018	Licence Recherche & Innovation ENS Rennes (1A) et Université de Rennes 1 (L3)
2018 – 2020	Master Science Informatique (SIF) ENS Rennes et Université de Rennes 1

Mon parcours

Juin 2015	Baccalauréat Scientifique
2015 – 2017	Classes Préparatoires aux Grandes Écoles (MPSI – MP*) Lycée Marcelin Berthelot (94)
2017 – 2018	Licence Recherche & Innovation ENS Rennes (1A) et Université de Rennes 1 (L3)
2018 – 2020	Master Science Informatique (SIF) ENS Rennes et Université de Rennes 1
2017 – 2020	Magistère Informatique de l'ENS de Rennes

Plan

Augmentation du nombre du nouveaux *malwares*



https ://www.av-test.org/en/statistics/malware/

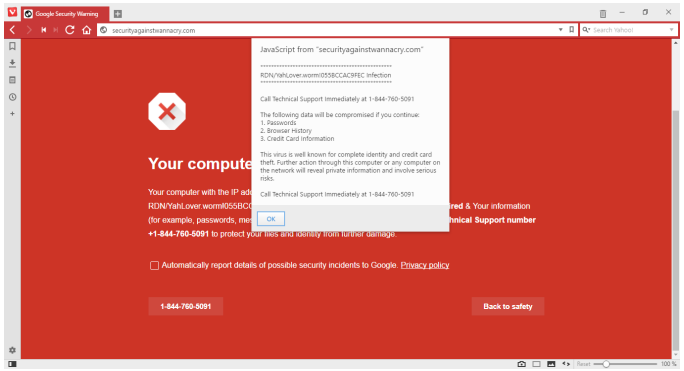
Conséquences des *malwares* sur la société


Cyberattaque mondiale "WannaCry", le ransomware qui chiffre les données

Actualité / Monde /


Rançongiciel WannaCry : le gouvernement britannique incrimine la Corée du Nord

Le virus avait touché un tiers des hôpitaux et cliniques d'Angleterre, ainsi que nombre des cabinets médicaux du pays.



lexpress.fr – numerama.com – lemonde.fr – bleepingcomputer.com

Nécessité de détection

- Besoin d'automatiser la détection de malwares

Nécessité de détection

- Besoin d'automatiser la détection de malwares
- Plusieurs méthodes existent :

Nécessité de détection

- Besoin d'automatiser la détection de malwares
- Plusieurs méthodes existent :
  - Analyse syntaxique
    - Recherche et étude de chaînes d'octets
    - Pas d'analyse de comportement

Nécessité de détection

- Besoin d'automatiser la détection de malwares
- Plusieurs méthodes existent :
  - Analyse syntaxique
    - Recherche et étude de chaînes d'octets
    - \- Pas d'analyse de comportement
  - Analyse dynamique
    - Exécution du programme dans une *sandbox*
    - Production d'une trace
    - \+ Analyse du comportement

Nécessité de détection

- Besoin d'automatiser la détection de malwares
- Plusieurs méthodes existent :
  - Analyse syntaxique
    - Recherche et étude de chaînes d'octets
    - \- Pas d'analyse de comportement
  - Analyse dynamique
    - Exécution du programme dans une *sandbox*
    - Production d'une trace
    - \+ Analyse du comportement
  - Analyse *concolique*
    - \+ Analyse de plusieurs traces
    - \+ Pas besoin de *sandbox*

Approche

Principe

Comportement caractérisé par les appels systèmes

Méthode

Approche par des *System Call Dependency Graphs* (SCDG)

Approche

Principe

Comportement caractérisé par les appels systèmes

Méthode

Approche par des *System Call Dependency Graphs* (SCDG)

Limites

- Le comportement est-il bien capturé ?
- Une amélioration est-elle possible ?

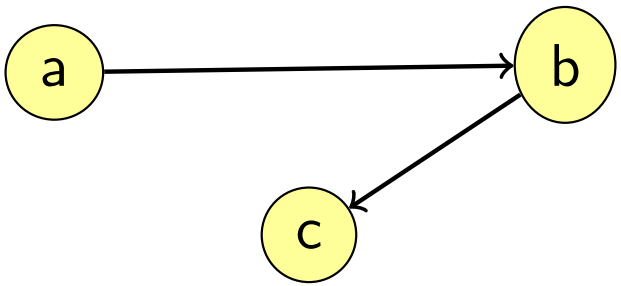
Plan

Définition d'un graphe

Definition (Graphe)

Un graphe $\mathcal{G}$ est un couple (S, A) où :

- S est l'ensemble des *sommets* ;
- $A \subseteq S \times S$ est l'ensemble des arêtes.

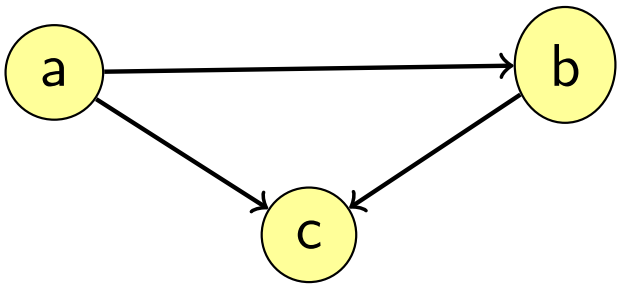


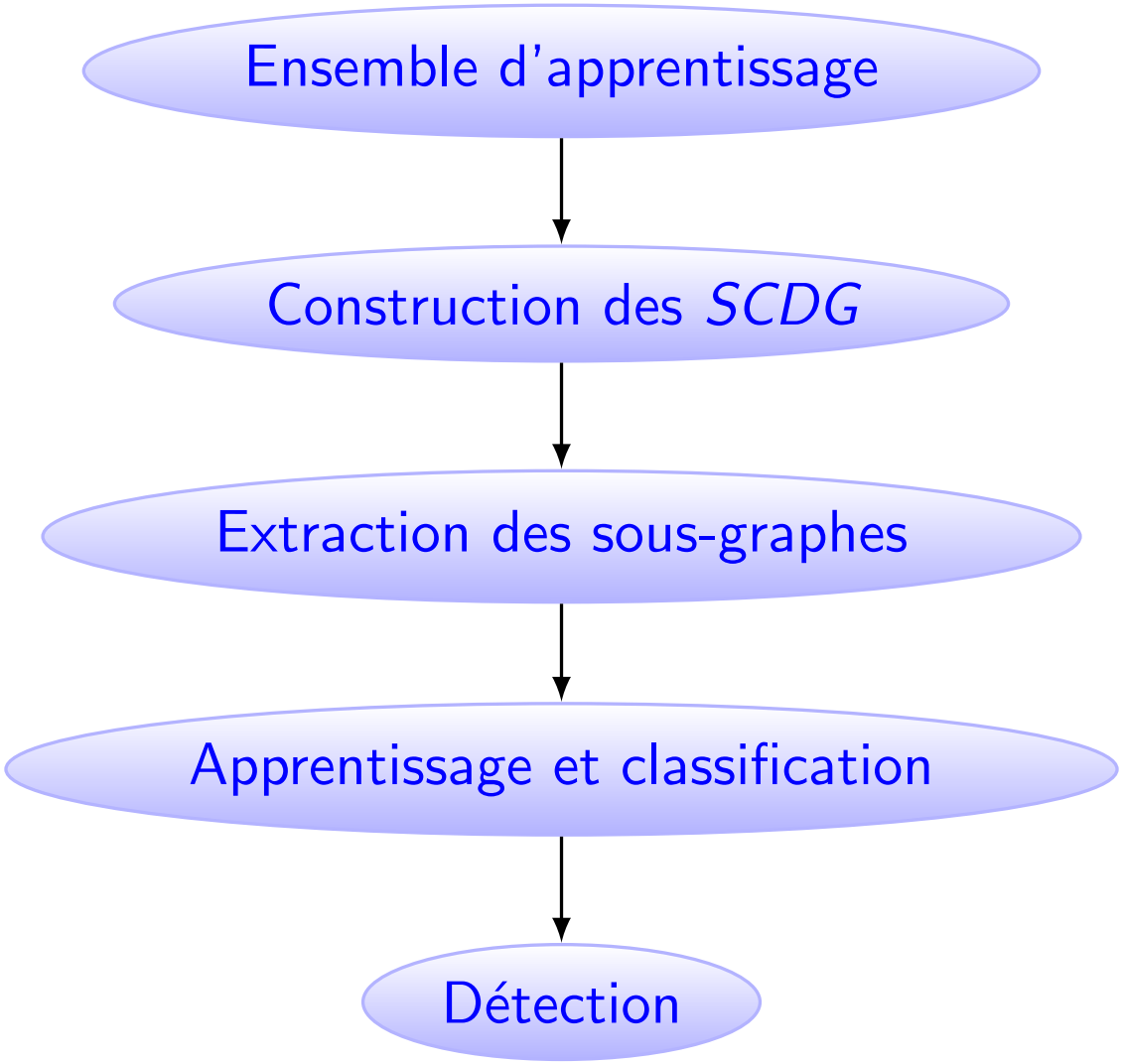
Définition d'un graphe

Definition (Graphe)

Un graphe $\mathcal{G}$ est un couple (S, A) où :

- S est l'ensemble des *sommets* ;
- $A \subseteq S \times S$ est l'ensemble des arêtes.




Ensemble d'apprentissage
Construction des *SCDG*
Extraction des sous-graphes
Apprentissage et classification
Détection

Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

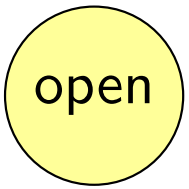
SCDG

Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

SCDG



Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

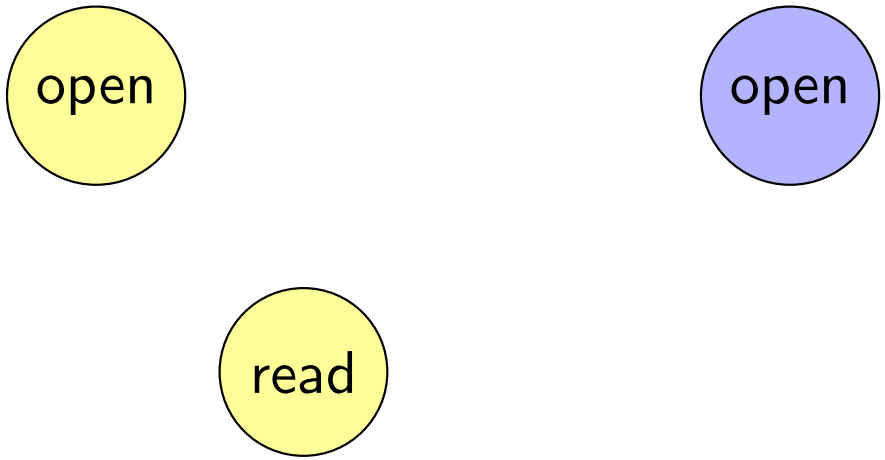
SCDG

Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

SCDG

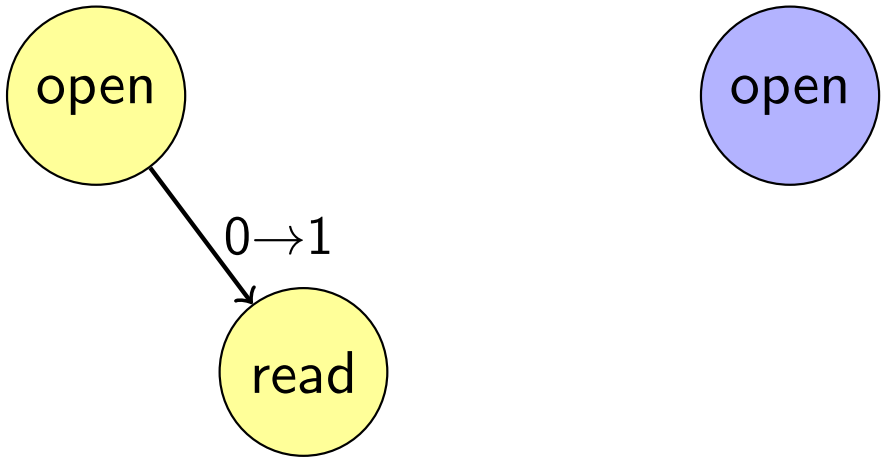


Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

SCDG

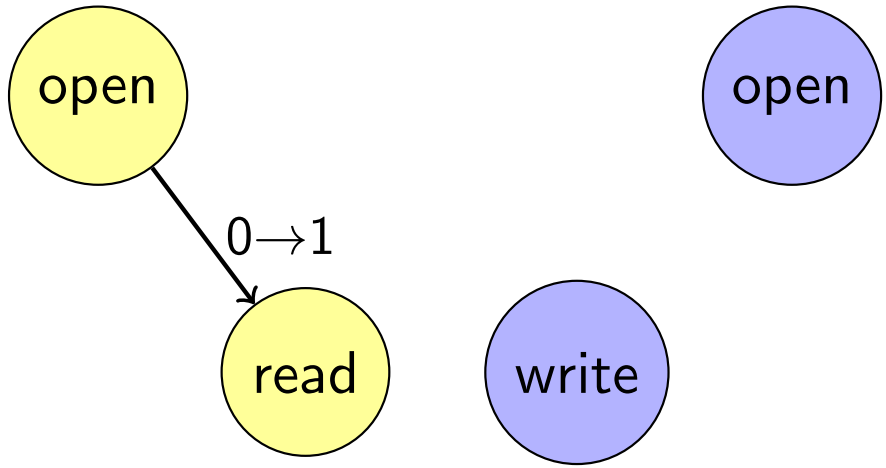


Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

SCDG

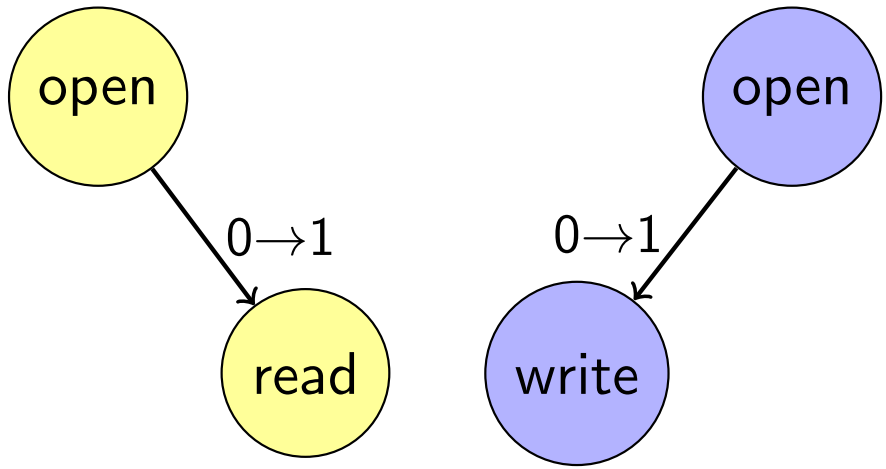


Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

SCDG

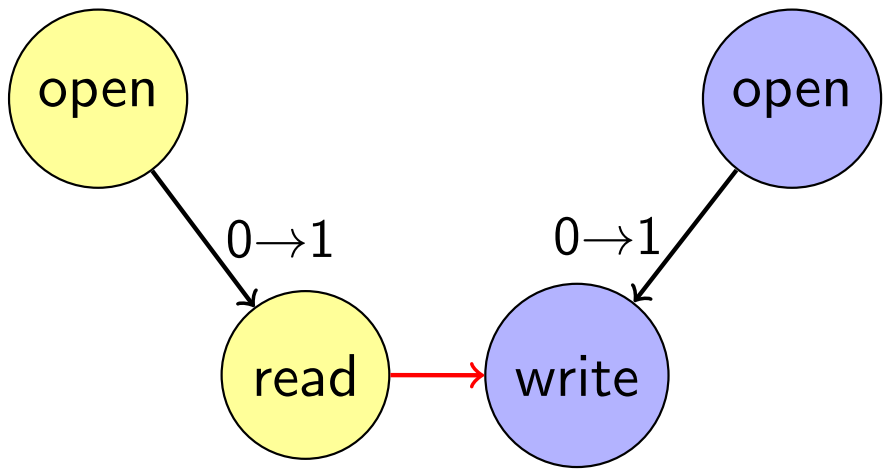


Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

SCDG

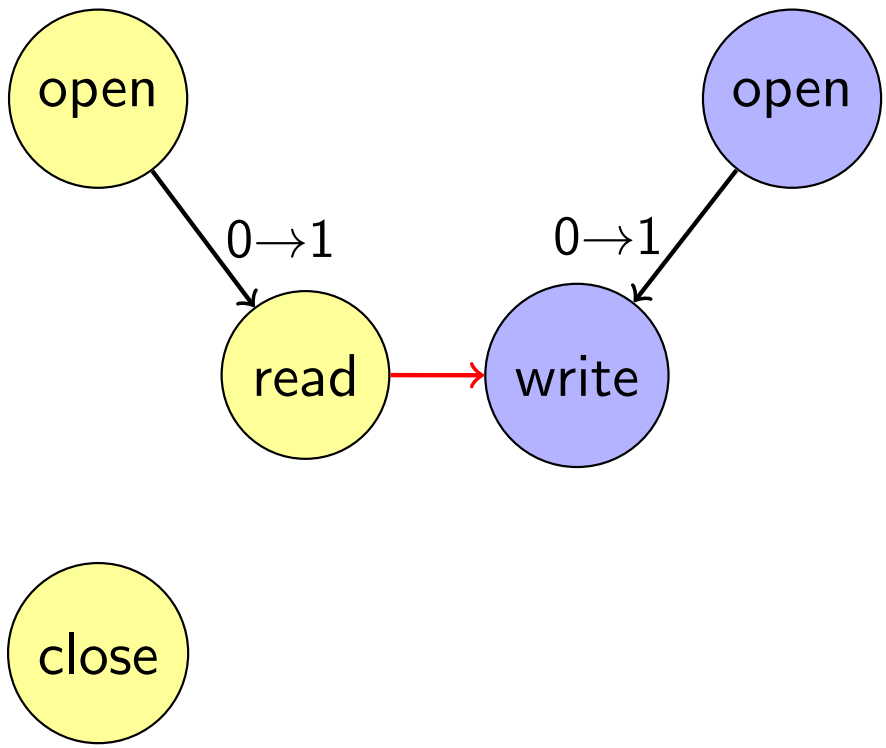


Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```

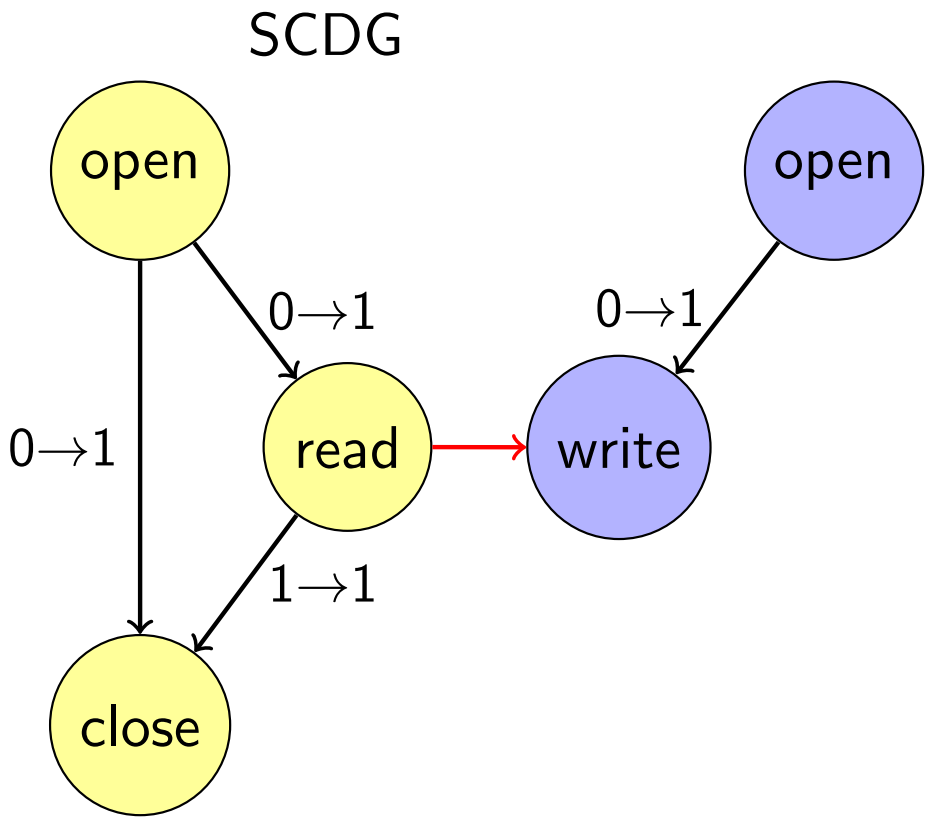
SCDG



Construction des *SCDGs*

Code – Trace

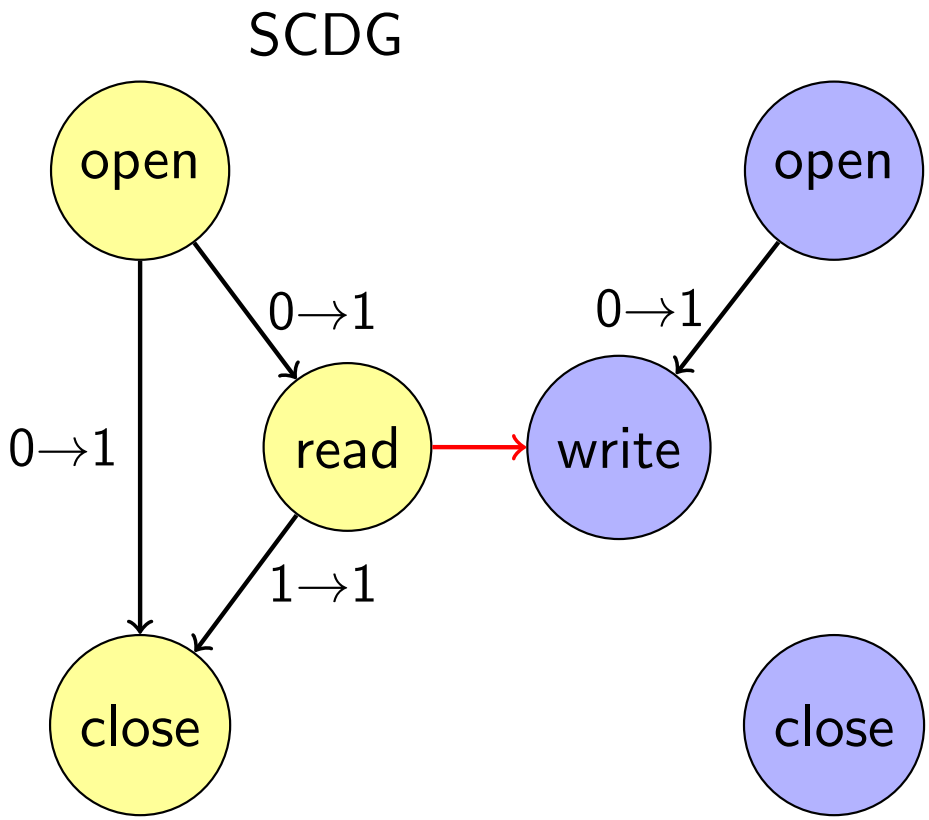
```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```



Construction des *SCDGs*

Code – Trace

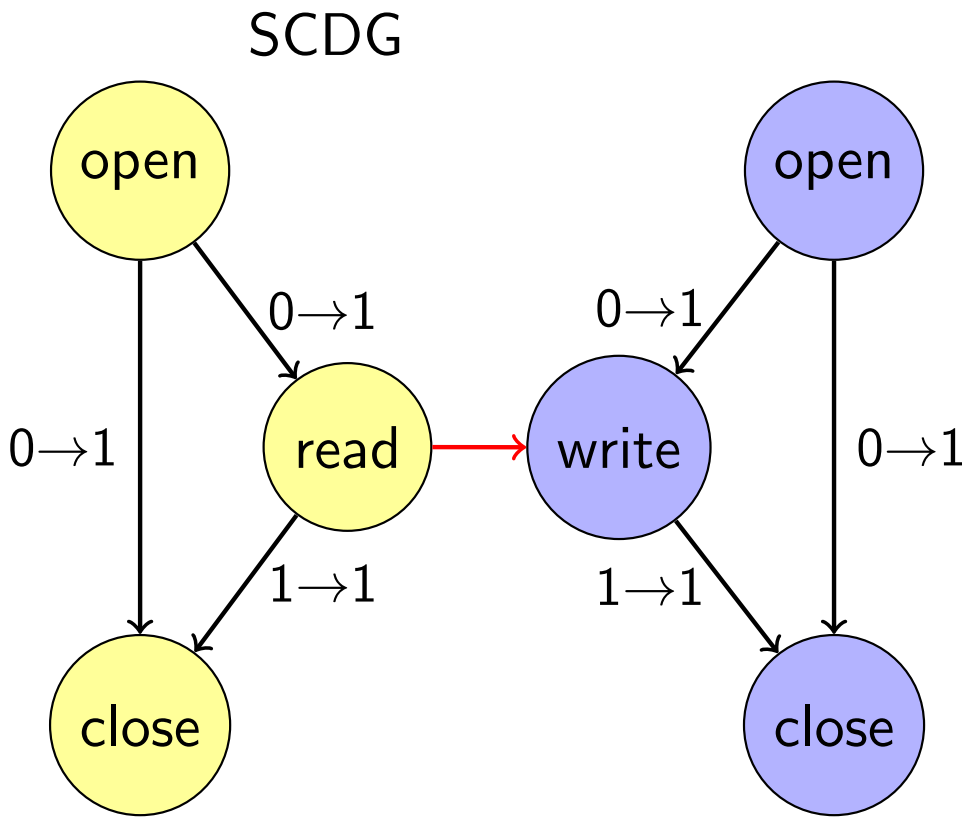
```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```



Construction des *SCDGs*

Code – Trace

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,1024);
6  ssize_t p = write(f2,buf,1024);
7  close(f);
8  close(f2);
```



Apprentissage

Apprentissage

Pour chaque famille F de malwares, la phase d'apprentissage a deux étapes :

- l'extraction du *SCDG* de chaque malware
- l'extraction des sous-graphes fréquents de cette famille (signature *comportementale* de la famille)

Classification

Classification

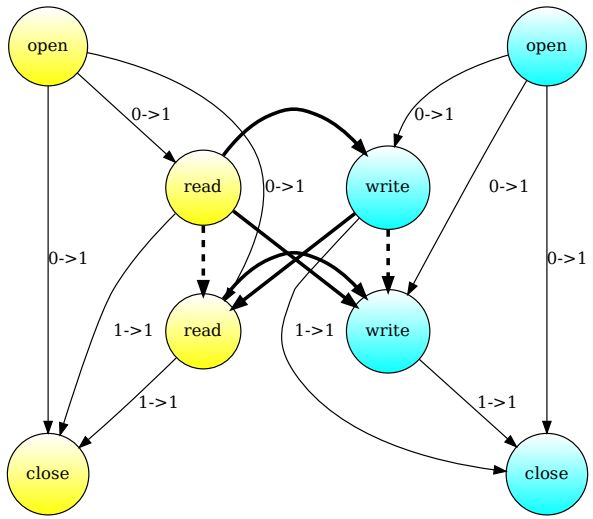
La classification est paramétrée par un seuil t et pour chaque binaire b :

- on extrait son *SCDG*, noté G
- pour chaque $sg \in \mathcal{G}$, on exécute gspan(G,sg)
- on calcule un score $\mathcal{M}(sg, b) = \frac{\text{size(gspan}(b,sg))}{\text{size}(sg)}$
- - si un sous-graphe obtient un score plus grand que t, le binaire b est classifié comme *malware* dans la famille où il obtient le score maximum
  - sinon il est classifié comme *cleanware*

Plan

Nombres d'arêtes

```
1  const char *pathname = 'file.txt';
2  const char *pathname2 = 'file2.txt';
3  int f = open(*pathname,flags);
4  int f2 = open(*pathname2,flags2);
5  ssize_t n = read(f,buf,512);
6  ssize_t p = write(f2,buf,512);
7  ssize_t n2 = read(f,buf,512);
8  ssize_t p2 = write(f2,buf,512);
9  close(f);
10 close(f2);
```

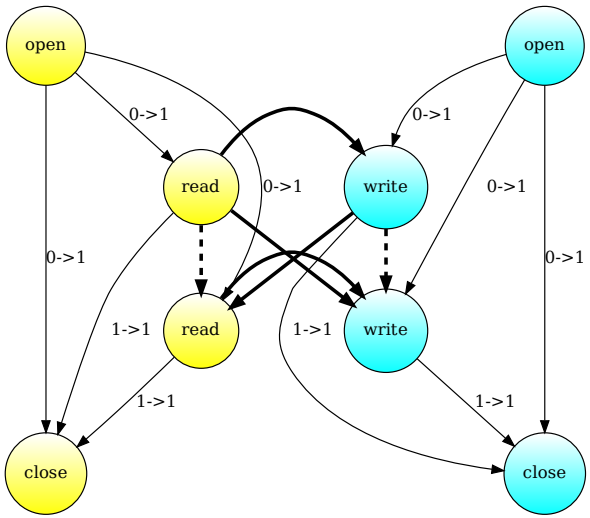


Nombre d'arêtes pour n cycles read write :

$$E(n) = 7n^2 + 2$$

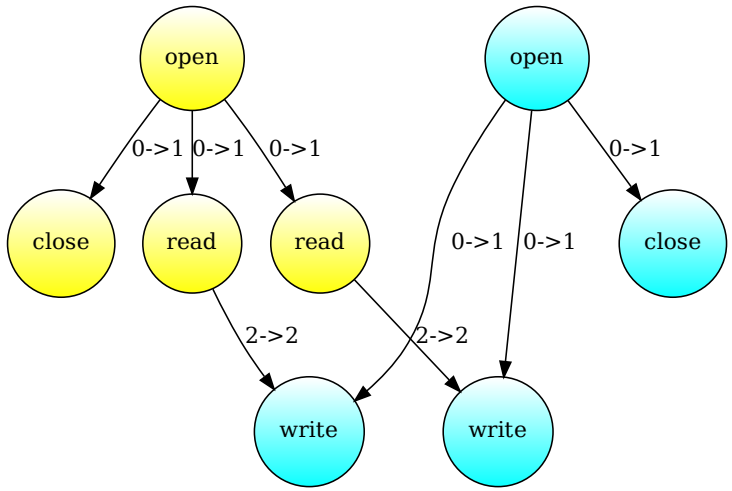
Nombre d'arêtes

Graphe obtenu



$$E(n) = 7n^2 + 2$$

Graphe « idéal »



$$E(n) = 3n + 2$$

On obtiendra $E(n) = \frac{1}{2}n^2 + \frac{5}{2}n + 2$

Petits entiers

Signatures

- **int** open(**const char** ∗pathname, **int** flags);
- **int** socketcall(**int** call, **unsigned long** ∗args);

```
1   f = open(...)
2   ...
3   s = socketcall(3, ...)
```

Plan

Argument direction

On associe a un argument d'un appel système une direction : Input, Output

Exemple

On considère l'appel système (read, A_r, r_r)

ssize_t read(**int** fd, **void** ∗buf, size_t count)

- $\text{dir}(0, \text{read}) = \text{Output}$
- $\text{dir}(1, \text{read}) = \text{Input} \ : \text{fd}$
- $\text{dir}(2, \text{read}) = \text{Output} \ : \text{buf}$
- $\text{dir}(3, \text{read}) = \text{Input} \ : \text{count}$

Plan

Argument direction – Preprocessing

On supprime en moyenne :

- 75,13% des arêtes sur les *Mirais*
- 93,31% des arêtes sur les *cleanwares*
- 86,98% des arêtes sur l'ensemble de la collection

Argument direction – Expériences

	Nouveaux résultats	Résultats précédents
Taux de faux-positifs	0.49%	0%
Taux de faux-négatifs	2.41%	2.88%
F-0.5 score	99.11%	99.41%
Temps d'apprentissage	1.77s	2 503s
Temps de classification	2.56s	315.59s

Résultats des expériences

Perte en $F_{0.5}$ score : 0.3 points
Gain de vitesse :

- 1 414x en apprentissage
- 123x en classification

Conclusion

- Analyse à partir des graphes extraits précédemment
- Mise en place de méthodes de pré-traitement
  - Amélioration en temps avec la méthode « *Argument direction* » : 1 414x en apprentissage, 123x en classification
- *Future work* : Comparaison avec de la *taint analysis*