Étude des sous-graphes communs des graphes de dépendances d'appels système pour la classification de logiciels malveillants

Stage L3

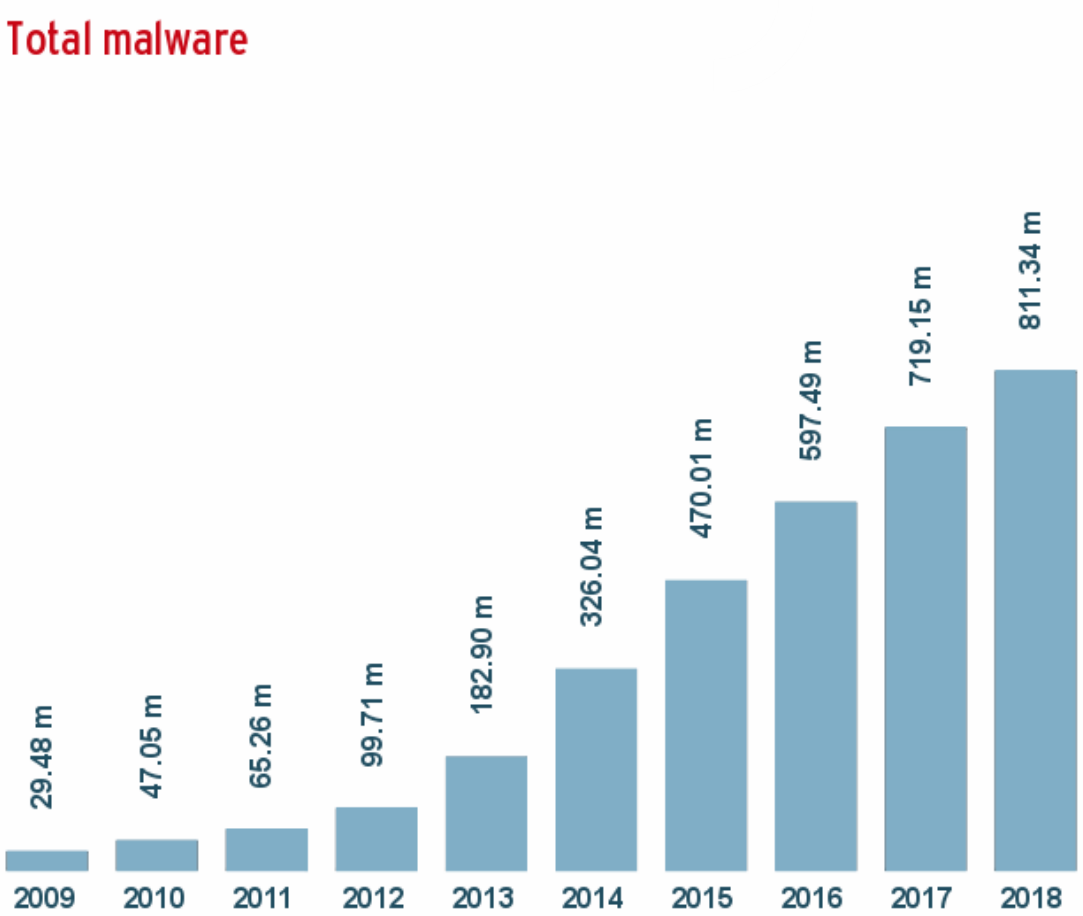
Dylan Marinho

sous la direction de Jean Quilbeuf et Axel Legay

TAMIS, Inria

4 septembre 2018

Augmentation du nombre du nouveaux *malwares*



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

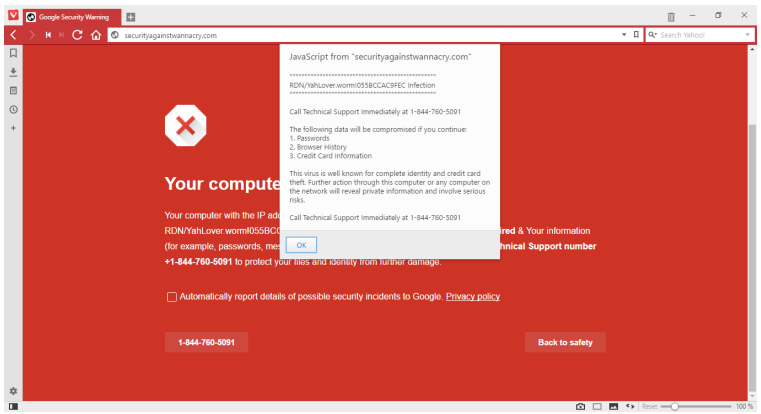
Conséquences des *malwares* sur la société

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 des *malwares*

- Besoin d'automatiser la détection de *malwares*
- Plusieurs méthodes
  - analyse syntaxique
    - Recherche de chaînes d'octets
    - Étude de l'en-tête et des sections
    - Pas d'analyse du comportement
  - analyse dynamique
    - Exécution dans une *sandbox*
    - Production d'une trace
  - analyse *concolique* (*conc*rete-symb*olic*)
    - Analyse de plusieurs traces
    - Pas besoin de *sandbox*

- Comportement caractérisé par les appels externes
- Approche par des *System Call Dependency Graphs* (SCDGs)
  - Transformer la trace en un graphe de dépendances d'appels
- Mais qui a ses limites. On ne sait pas :
  - si le comportement du logiciel est bien capturé
  - si une amélioration est possible

Contributions

- Analyse des graphes et des sous-graphes produits (analyse *concolique*)
- Deux méthodes de pré-traitement
- Évaluation expérimentale

Plan

Construction des *SCDGs*

Code – Trace

```
const char *pathname = 'file.txt';
const char *pathname2 = 'file2.txt';
int f = open(*pathname,flags);
int f2 = open(*pathname2,flags2);
ssize_t n = read(f,buf,1024);
ssize_t p = write(f2,buf,1024);
close(f);
close(f2);
```

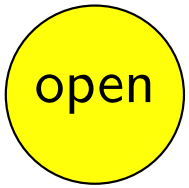
SCDG

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);
```

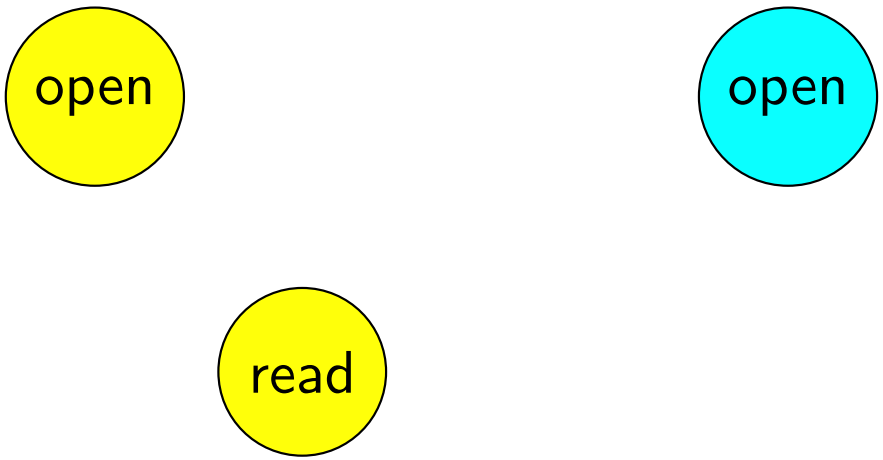
SCDG

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

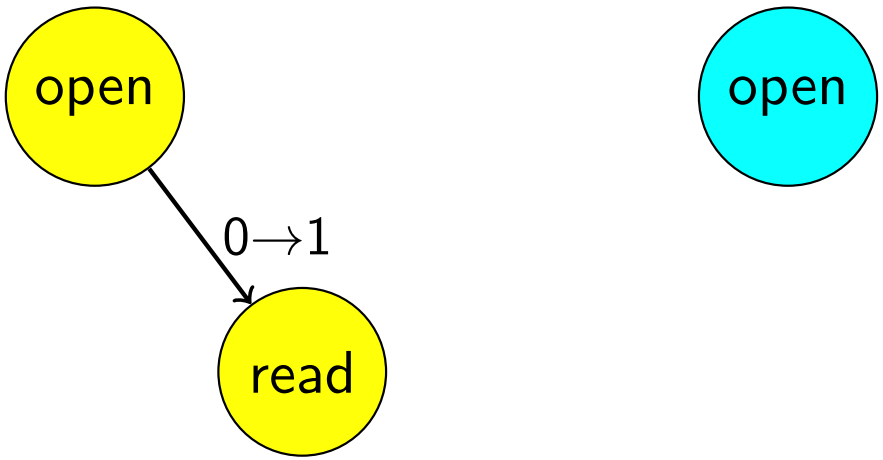


Construction des *SCDGs*

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

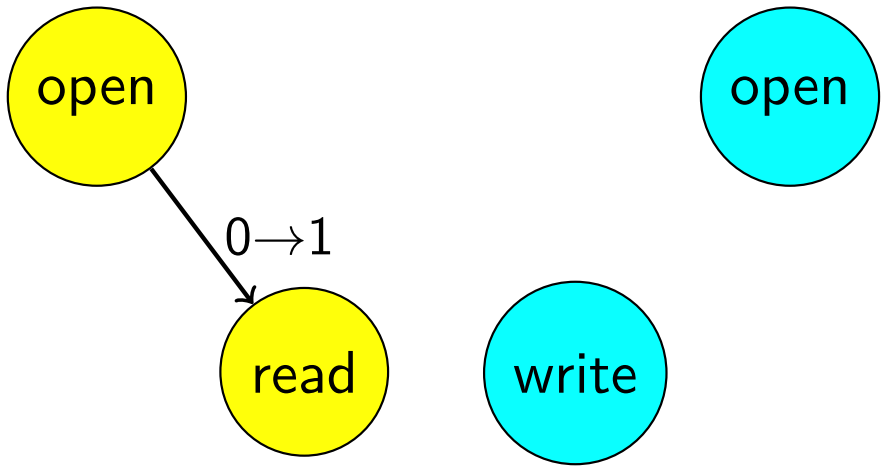


Construction des *SCDGs*

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

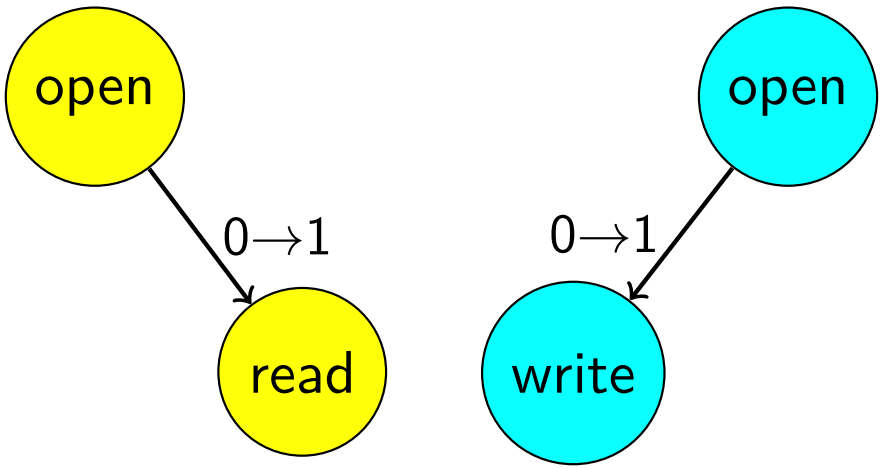


Construction des *SCDGs*

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

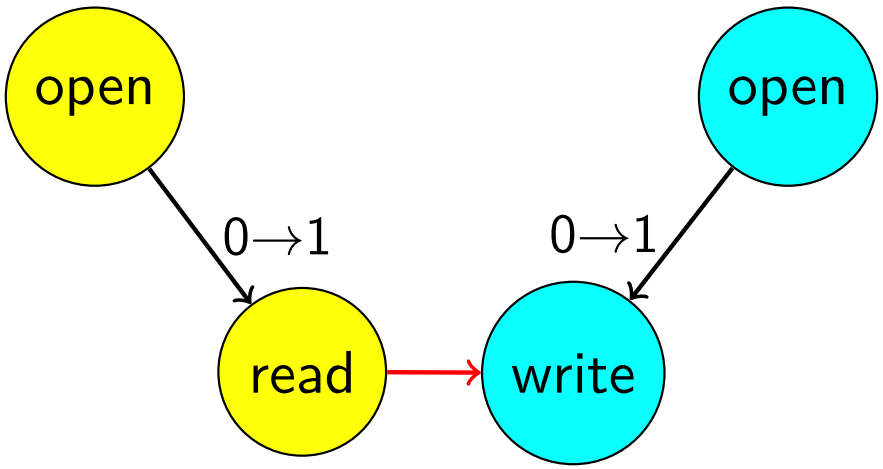


Construction des *SCDGs*

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

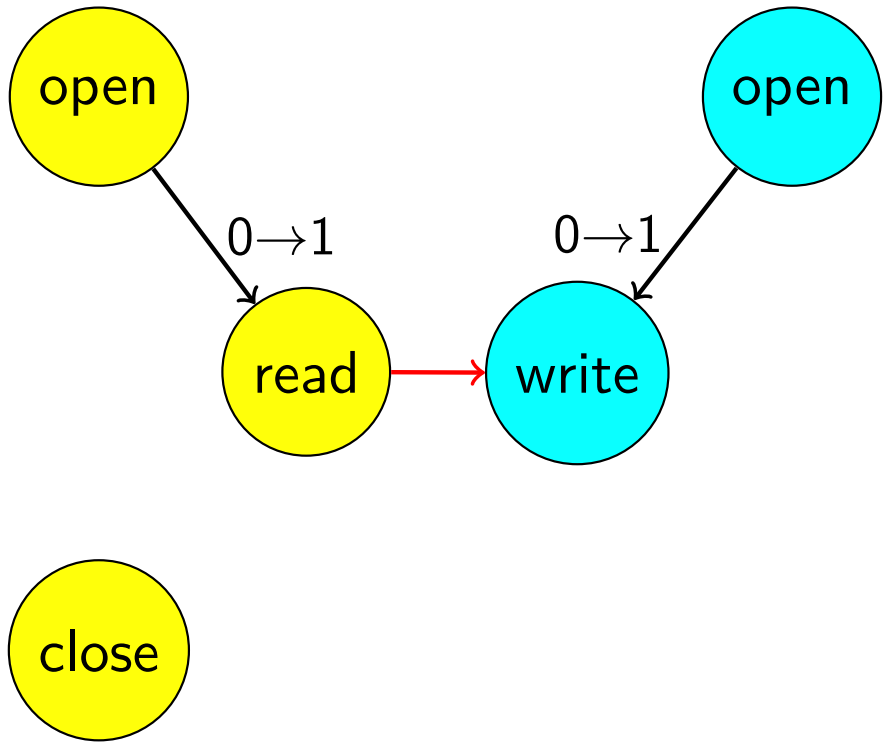


Construction des *SCDGs*

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

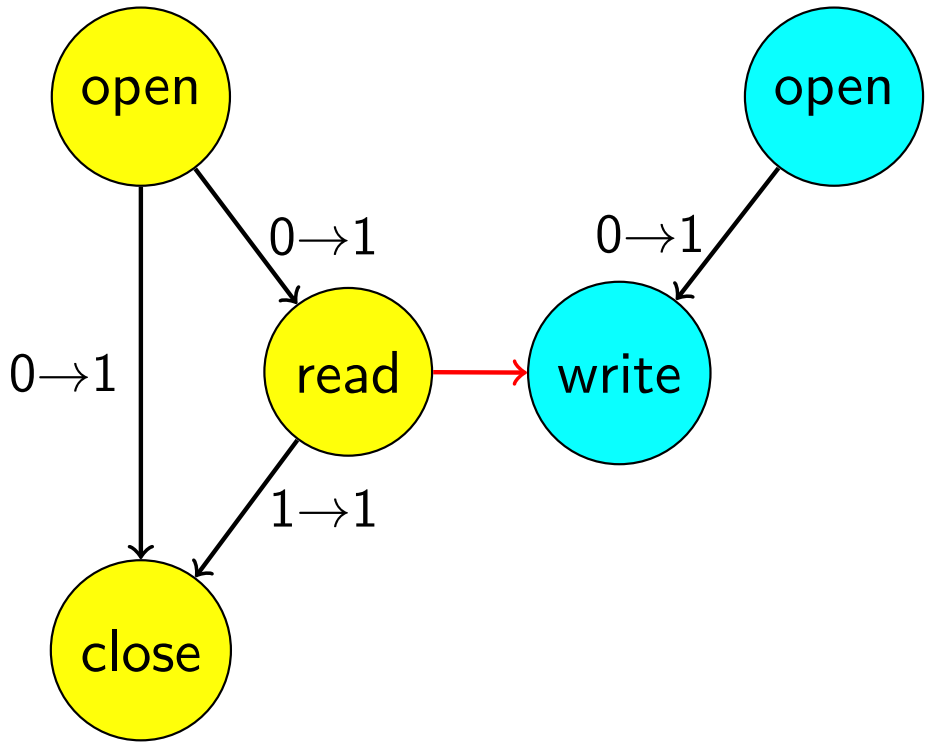


Construction des *SCDGs*

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);
```

SCDG

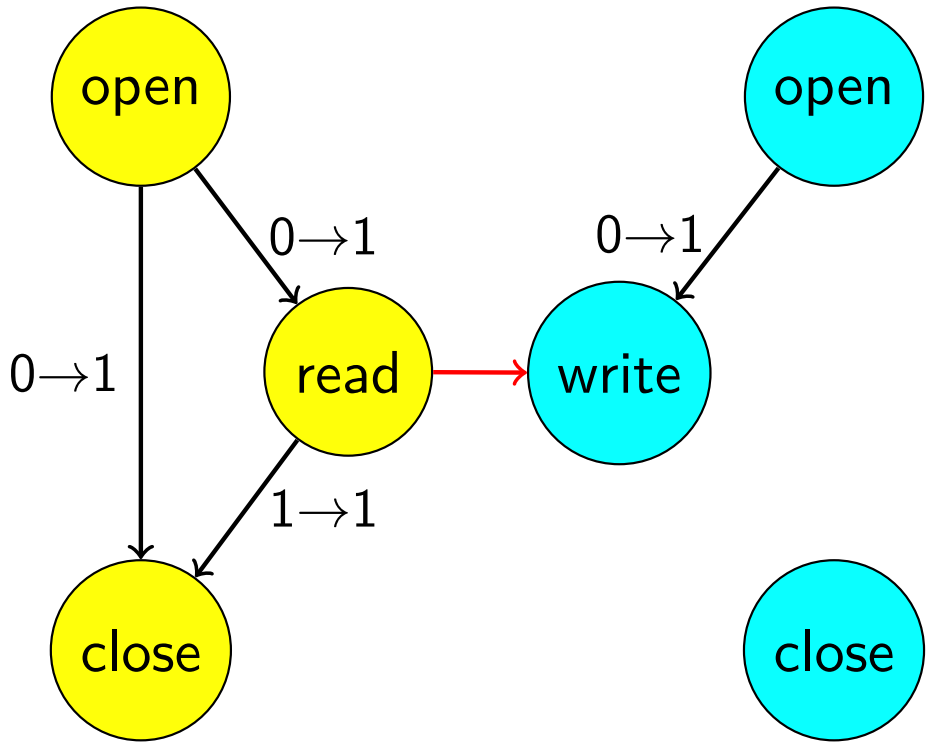


Construction des *SCDGs*

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

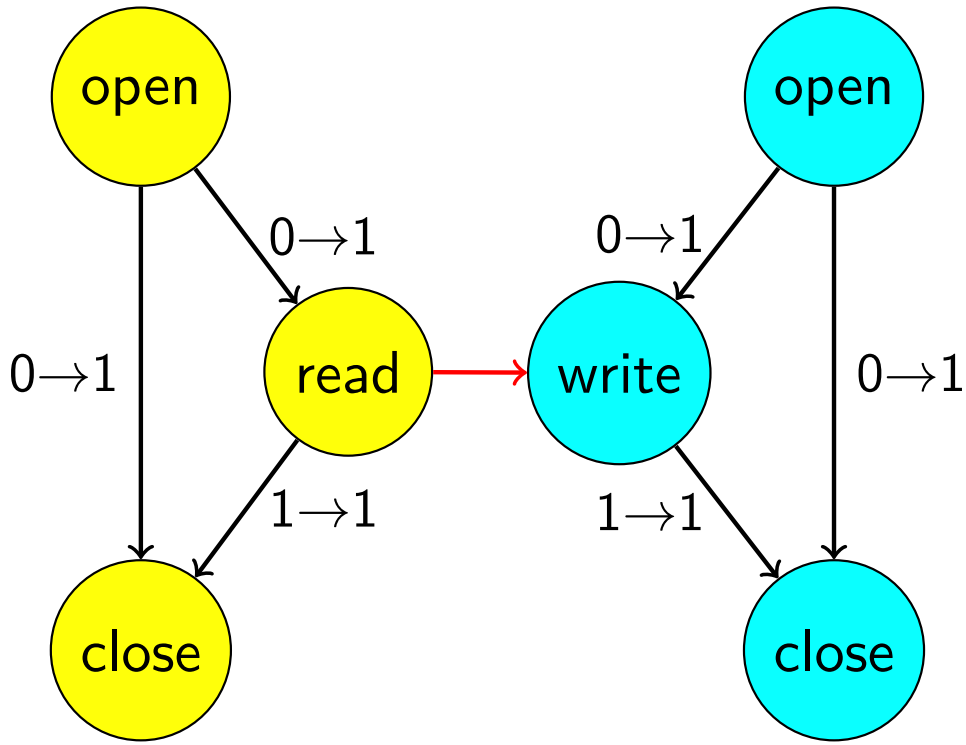


Construction des *SCDGs*

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



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

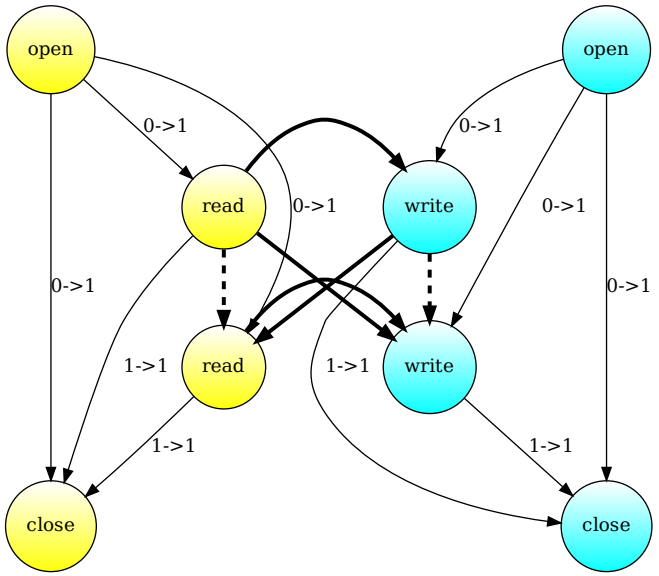
La classification est paramétrée par un seuil t et pour chaque binaire b :

- on extrait son *SCDG*, noté G
- pour chaque $sg \in \mathcal{G}$, on exécute gspan(G,sg)
- on calcule un score $\mathcal{M}(sg, b) = \frac{\text{size}(\text{gspan}(b,sg))}{\text{size}(sg)}$
- - 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

```
const char *pathname = 'file.txt';
const char *pathname2 = 'file2.txt';
int f = open(*pathname,flags);
int f2 = open(*pathname2,flags2);
ssize_t n = read(f,buf,512);
ssize_t p = write(f2,buf,512);
ssize_t n2 = read(f,buf,512);
ssize_t p2 = write(f2,buf,512);
close(f);
close(f2);
```

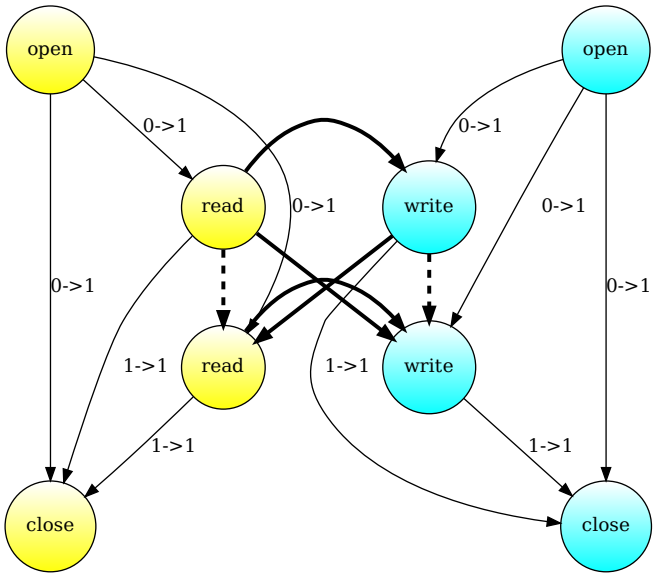


Nombre d'arêtes pour n cycles read write :

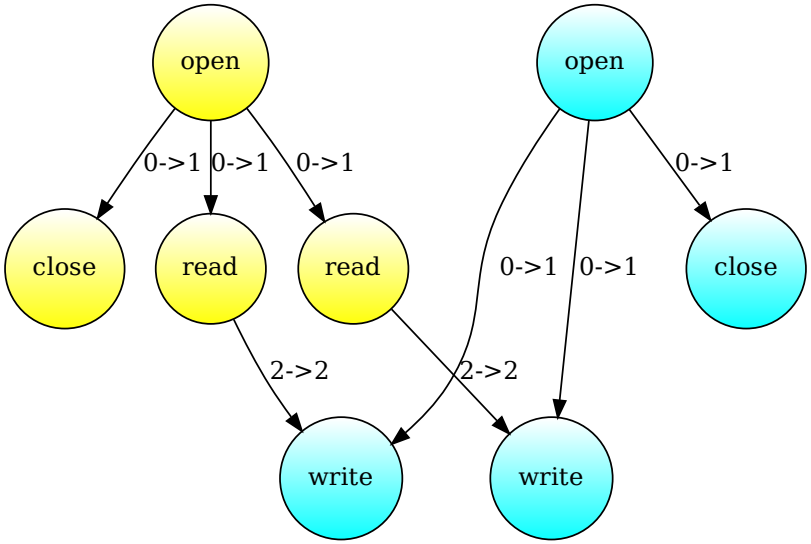
$$E(n) = 7n^2 + 2$$

Nombre d'arêtes

Graphe obtenu



Graphe idéal



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

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

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);

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

Plan

Edge consistency

- On définit l'ensemble *ET* d'un graphe comme l'ensemble des triplets (label de départ, label d'arrivée, label de l'arête)
- On classe ensuite ses éléments :
  Consistent, Meaningless, Indeterminate

Exemple

Signatures

- **int** close(**int** fd);
- **int** socketcall(**int** call, **unsigned long** *args);
- **int** fcntl(**int** fd, **int** cmd, ... /**arg* */);

Exemples

$$(\text{fcntl}, \text{close}, 1\text{->}1) \mapsto \text{consistent}$$
$$(\text{fcntl}, \text{socketcall}, 2\text{->}1) \mapsto \text{meaningless}$$
$$(\text{fcntl}, \text{fcntl}, 0\text{->}3) \mapsto \text{indeterminate}$$

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)

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

Plan

Méthodologie de validation

- Méthodes testées sur les graphes issus de *Mirai*
- Deux phases :

  Exploration de paramètres sur un nombre restreint de *data-sets*

  Expériences sur 500 *data-sets* obtenus aléatoirement (80% des Mirais utilisés pour l'apprentissage, 20% pour la classification)
- Les résultats seront comparés au dernier article de l'équipe [1] :
  - $F_{0.5}$ score de 99.41%
  - Temps d'apprentissage 2 503s (approx. 12.4s par graphe)
  - Temps de classification 315.59s (approx. 1.56s par graphe)

1. Najah Ben Said et al. "Detection of Mirai by Syntactic and Behavioral Analysis". *ISSRE 2018.* Accepted, to appear. IEEE Computer Society, 2018.

Plan

Edge consistency – Preprocessing

Consistent	*Meaningless*	*Indeterminate*	Total
59	76	55	190

Classification des *edge types*

- Suppression de 39% des arêtes des *Mirais*
- Non exécuté sur les *cleanwares*

Edge consistency – Exploration de paramètres

	Total base de données	Apprentissage	Classification
Mirais	503	403	100
Cleanwares	516	403	100

Répartition des *data-sets*

Gspan	Support	# Sous graphes	Seuil	F-0.5 score
directed	0.5	25	0.45 - 0.55	98.13%
directed	0.5	30	0.45 - 0.55	98.13%
directed	0.5	40	0.51 - 0.55	98.13%

Meilleurs résultats de l'exploration de paramètres

Gspan	Apprentissage	Classification	Classification par graphe
undirected	8.36	454.04	2.27
directed	4.95	764.75	3.82

Temps obtenus pour les expériences (en s)

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 – Exploration de paramètres

	Total base de données	Apprentissage	Classification
Mirais	500	400	100
Cleanwares	204	0	100

Répartition des *data-sets*

Gspan	Support	# Sous graphes	Seuil	F-0.5 score
directed	0.8	1	0.29 - 0.42	99.16%
undirected	0.8	1	0.29 - 0.42	99.16%
directed	0.8	2	0.34 - 0.42	99.16%
undirected	0.8	2	0.34 - 0.42	99.16%
directed	0.8	5	0.34 - 0.50	99.16%
undirected	0.8	5	0.34 - 0.50	99.16%
directed	0.8	10	0.41 - 0.50	99.16%
undirected	0.8	10	0.41 - 0.50	99.16%

Meilleurs résultats

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 deux méthodes de pré-traitement
  - Amélioration relative sur la méthode 1 : mise de côté
  - Amélioration en temps avec la méthode 2 : 1 414x en apprentissage, 123x en classification
- *Future work* : Comparaison avec de la *taint analysis*