Principe du suivi de flux d'information au sein du système d'exploitation
application à l'analyse de malware Android

Valérie Viet Triem Tong

CentraleSupelec
Inria Cidre
valerie.viettriemtong@centralesupelec.fr

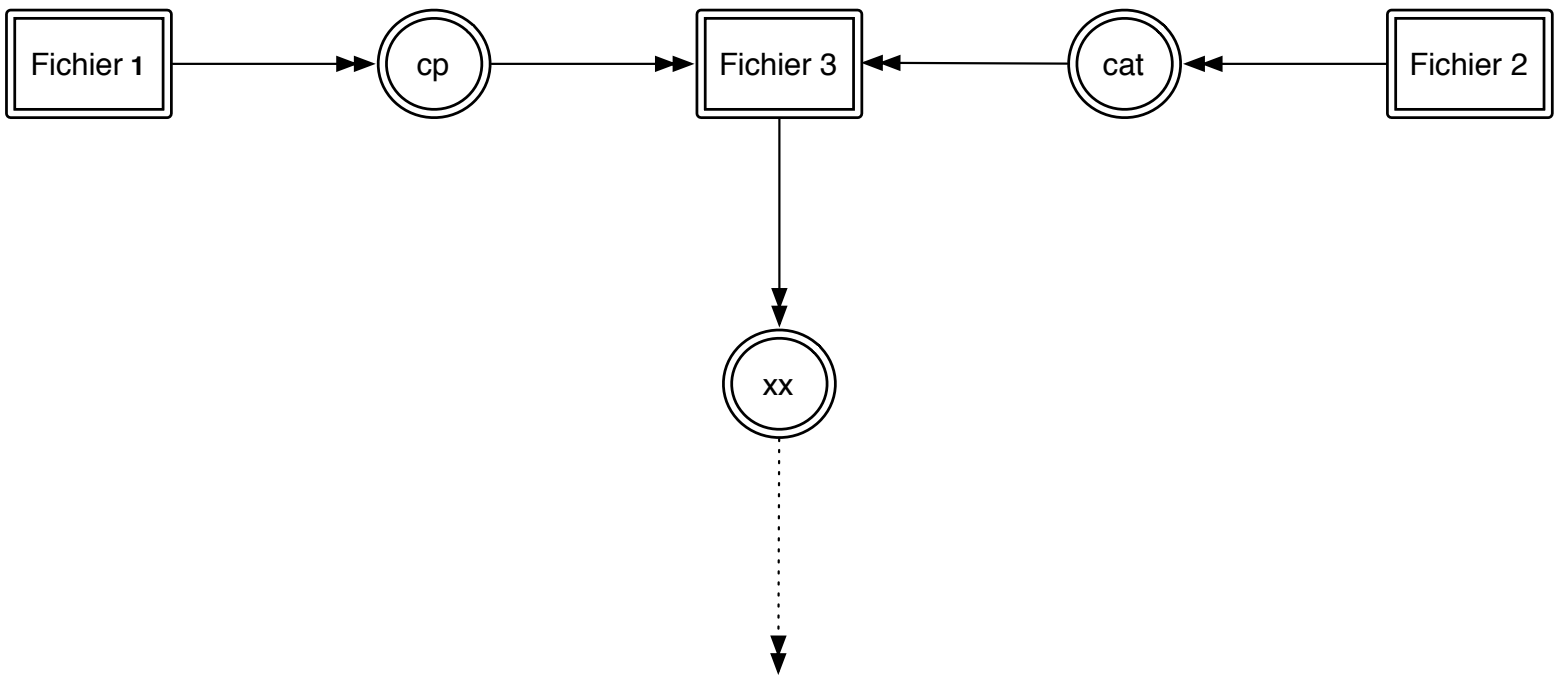
Janvier 2017

Suivre les flux d'information dans un système d'exploitation

pour savoir comment l'information se dissémine

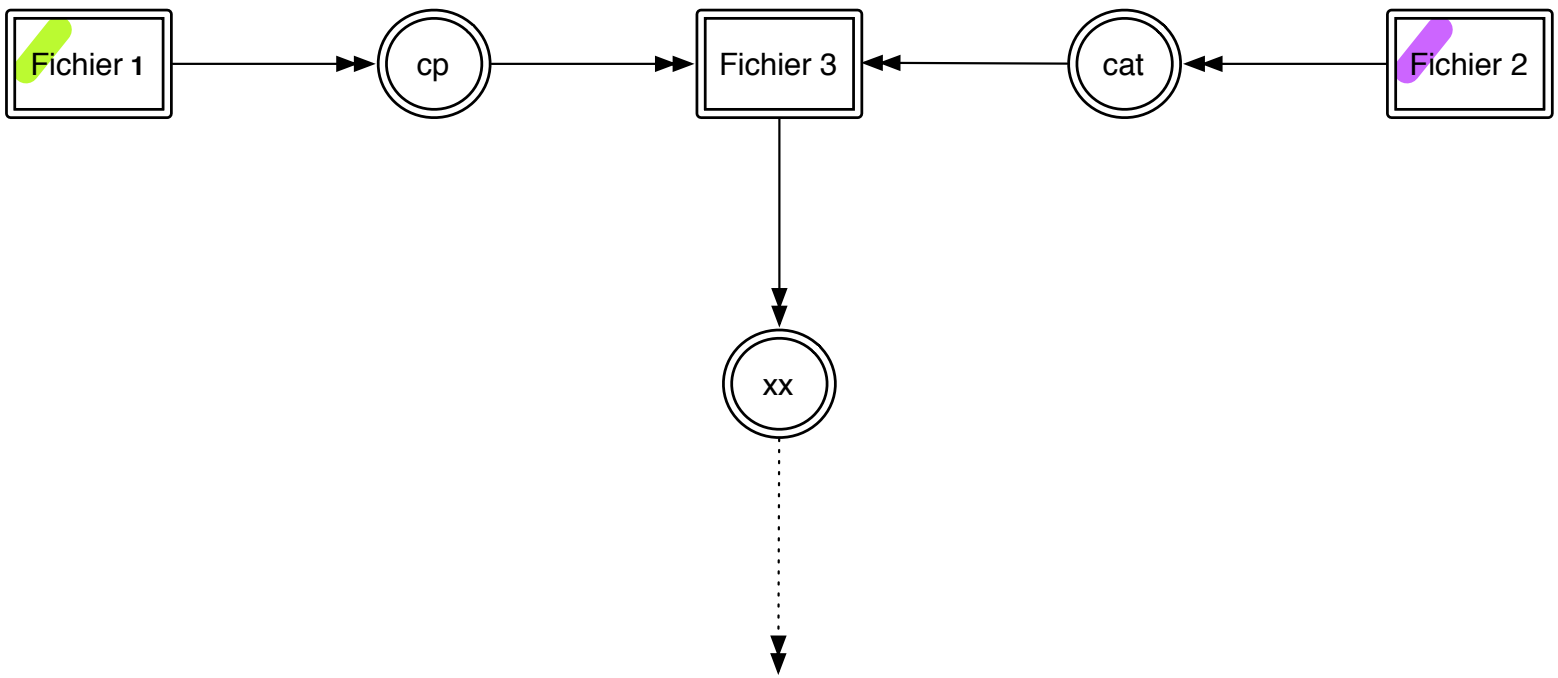
Principe du suivi de flux d'information

1. Une marque est attachée à chaque information considérée comme sensible
2. Les marques sont propagées à chaque observation de flux



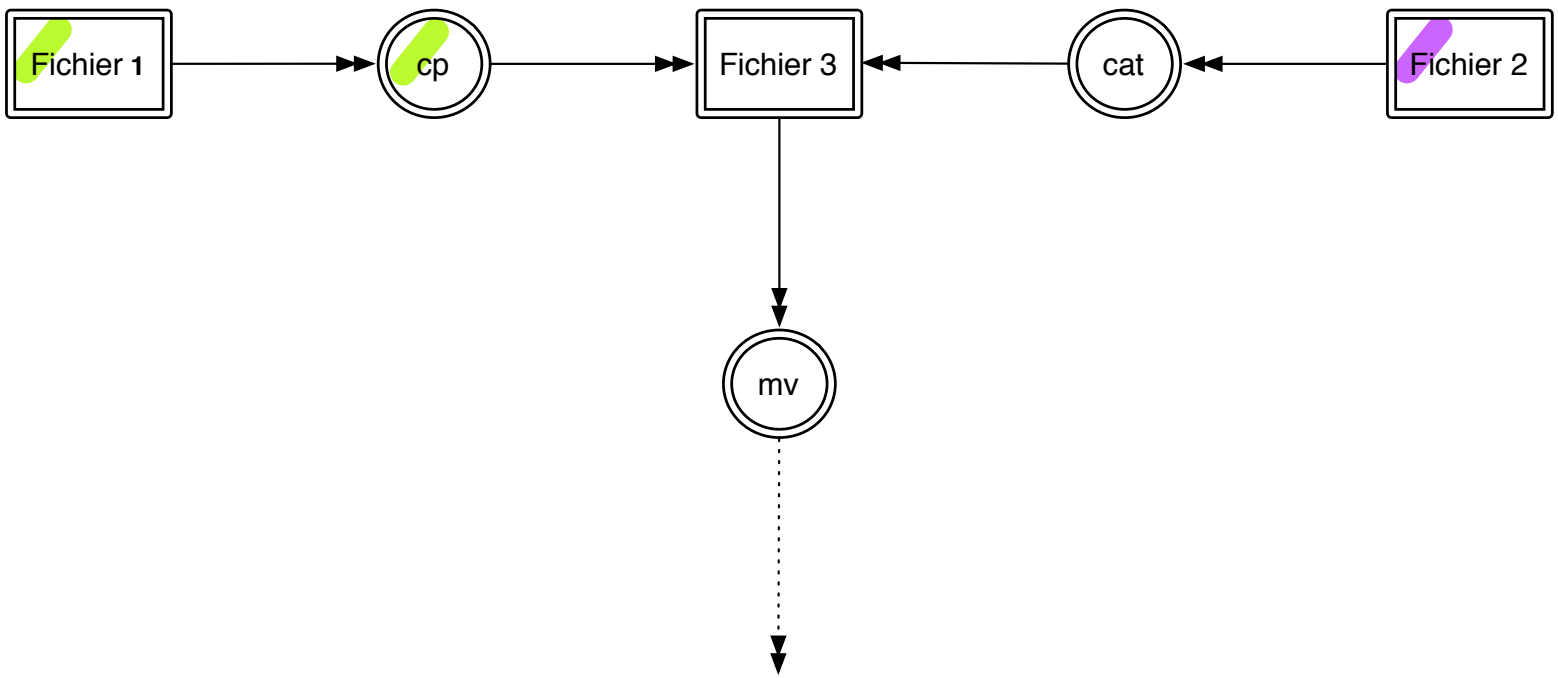
Principe du suivi de flux d'information

1. Une marque est attachée à chaque information considérée comme sensible
2. Les marques sont propagées à chaque observation de flux



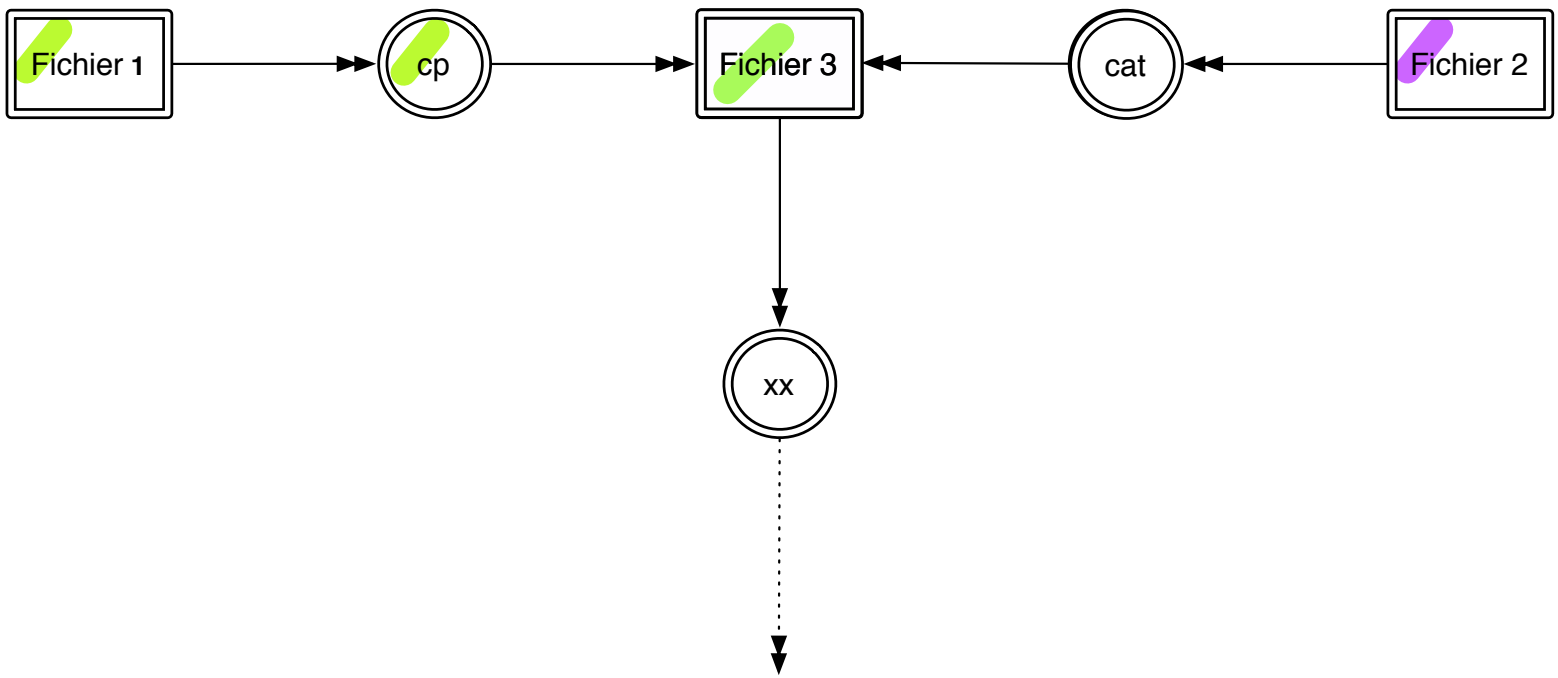
Principe du suivi de flux d'information

1. Une marque est attachée à chaque information considérée comme sensible
2. Les marques sont propagées à chaque observation de flux



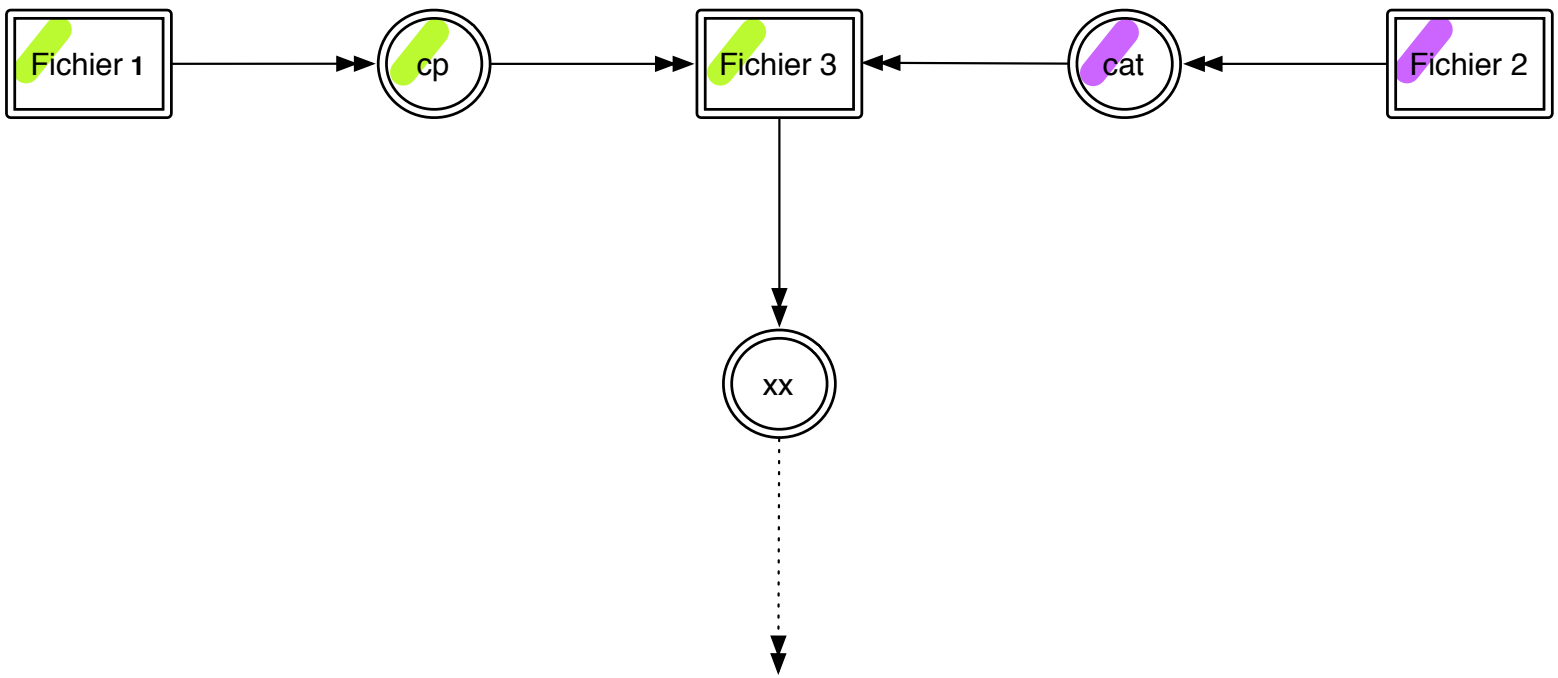
Principe du suivi de flux d'information

1. Une marque est attachée à chaque information considérée comme sensible
2. Les marques sont propagées à chaque observation de flux



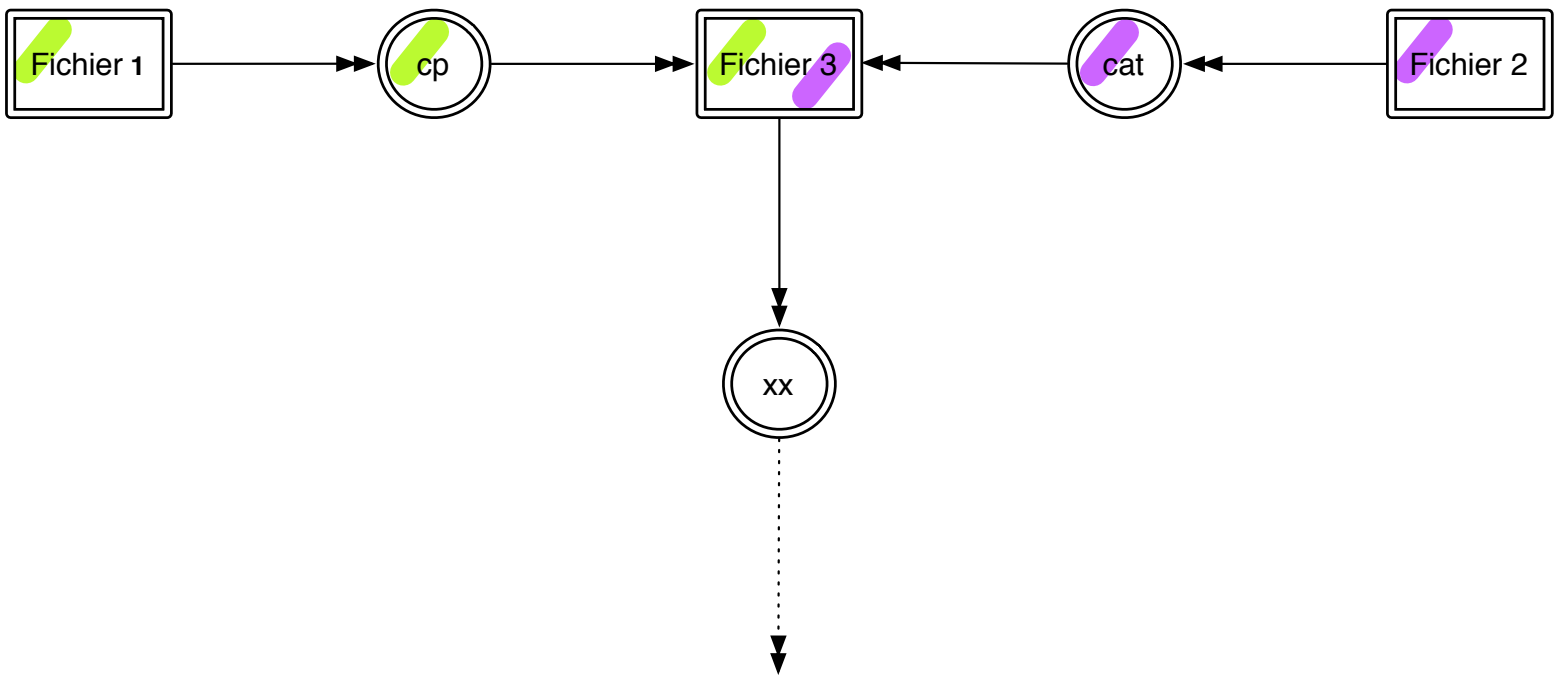
Principe du suivi de flux d'information

1. Une marque est attachée à chaque information considérée comme sensible
2. Les marques sont propagées à chaque observation de flux



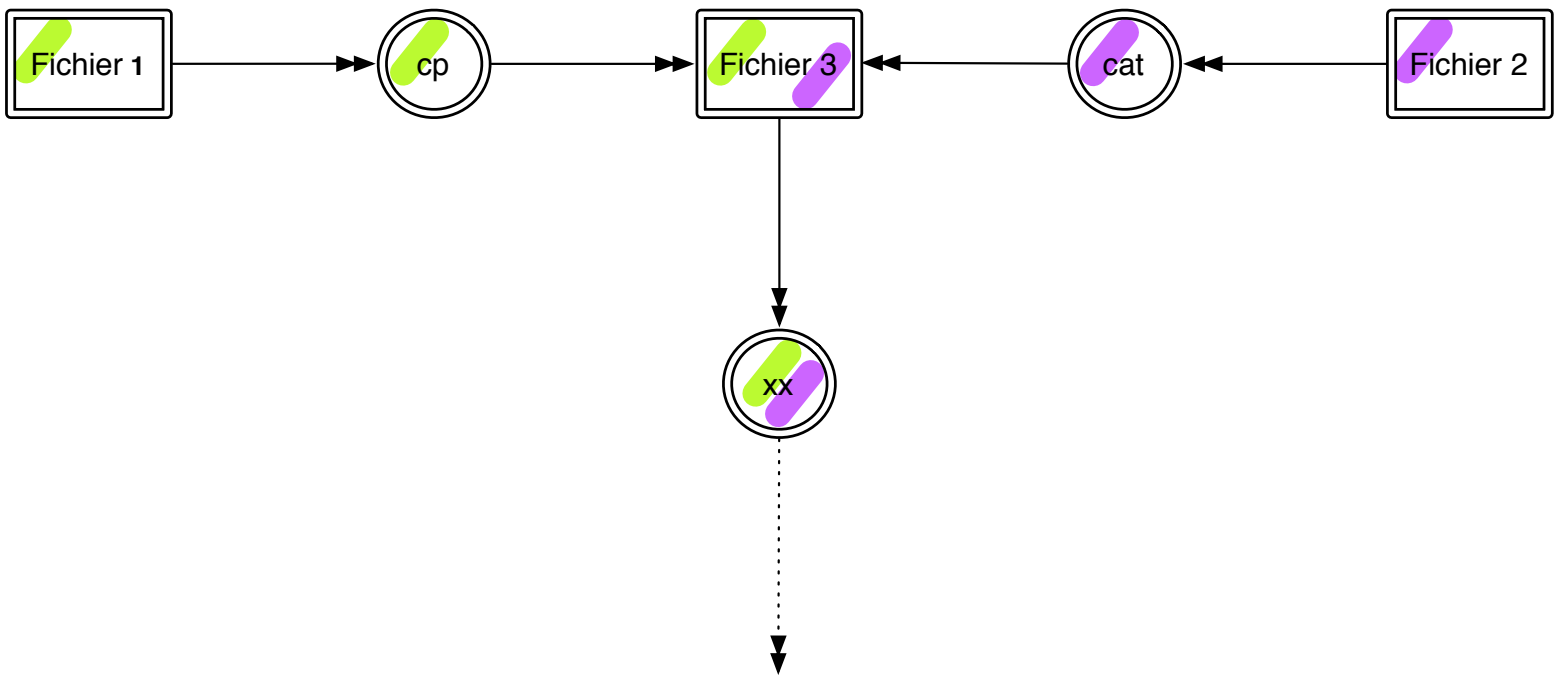
Principe du suivi de flux d'information

1. Une marque est attachée à chaque information considérée comme sensible
2. Les marques sont propagées à chaque observation de flux



Principe du suivi de flux d'information

1. Une marque est attachée à chaque information considérée comme sensible
2. Les marques sont propagées à chaque observation de flux



Modèle générique pour le contrôle de flux d'information

Expliciter la notion d'information

Les informations sensibles sont identifiées
puis caractérisées par un identifiant $i_1, i_2, \ldots i_n \in \mathcal{I}$

Exemple

C'était le sens des marques vertes et violettes.

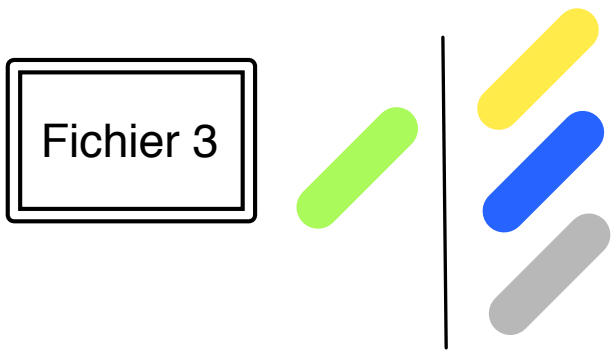
Modèle générique pour le contrôle de flux d'information

Définir une politique de flux d'information à grain fin

Une politique définit pour chaque conteneur c les mélanges d'information autorisés $\mathbb{P} : c \mapsto \mathscr{P}(\mathscr{P}(\mathcal{I}))$

Exemple : une telle politique peut spécifier que le fichier 3 peut contenir de l'information

- marquée verte
- ou tout mélange de jaune, bleu, gris

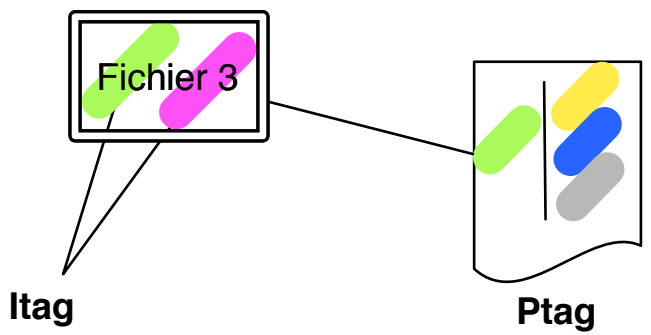


Modèle générique pour le contrôle de flux d'information

Cette politique est mise en œuvre par la surveillance des flux d'information et des conteneurs modifiés

- Les conteneurs d'information surveillés sont caractérisés par :
  - la source des informations contenues *dénotée par* $itag \in \mathscr{P}(\mathcal{I})$
  - les informations autorisées *dénotées par* $ptag \in \mathscr{P}(\mathscr{P}(\mathcal{I}))$

Exemple : le fichier 3 est donc caractérisé par l'origine de son contenu et par sa politique

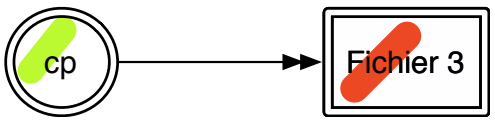


Cette politique est mise en œuvre par la surveillance des flux d'information et des conteneurs modifiés

Un flux d'information $A \rightarrow B$ modifie la source des informations contenues dans B,

- soit l'information venue de A écrase la précédente

$$B.itag = A.itag$$



- soit cette information est ajoutée

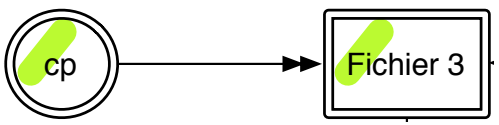
$$B.itag = B.itag \cup A.itag$$

Cette politique est mise en œuvre par la surveillance des flux d'information et des conteneurs modifiés

Un flux d'information $A \rightarrow B$ modifie la source des informations contenues dans B,

- soit l'information venue de A écrase la précédente

$$B.itag = A.itag$$



- soit cette information est ajoutée

$$B.itag = B.itag \cup A.itag$$

Cette politique est mise en œuvre par la surveillance des flux d'information et des conteneurs modifiés

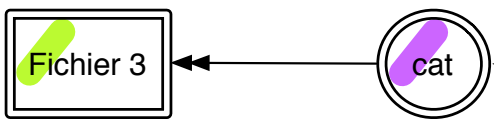
Un flux d'information $A \rightarrow B$ modifie la source des informations contenues dans B,

- soit l'information venue de A écrase la précédente

$$B.itag = A.itag$$

- soit cette information est ajoutée

$$B.itag = B.itag \cup A.itag$$



Cette politique est mise en œuvre par la surveillance des flux d'information et des conteneurs modifiés

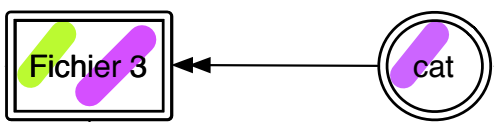
Un flux d'information $A \rightarrow B$ modifie la source des informations contenues dans B,

- soit l'information venue de A écrase la précédente

$$B.itag = A.itag$$

- soit cette information est ajoutée

$$B.itag = B.itag \cup A.itag$$

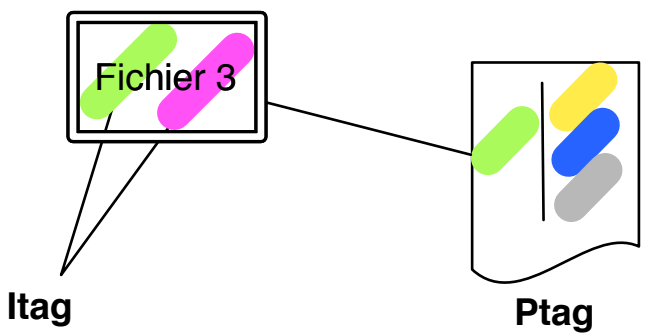


Cette politique est mise en œuvre par la surveillance des flux d'information et des conteneurs modifiés

Un flux $A \rightarrow B$ était conforme à la politique ssi

$$\exists m \in B.ptag,\ B.itag \subseteq m$$

Exemple : le dernier flux vers le fichier 3 n'était pas autorisé



Spécification d'un moniteur de flux d'information

Un moniteur de flux d'information est un logiciel qui

1. implémente les *itag* et *ptag*
2. modifie la valeur des *itag* à chaque flux observé
3. peut vérifier si ce flux était autorisé

Il doit donc être appelé par le système qu'il surveille à chaque opération causant un flux.

Implémentations d'un moniteur de flux d'information

(Andro)Blare, moniteurs de flux au niveau du système d'exploitation (Android)Linux

- Les *itag* et *ptag* sont dans les attributs étendus du système de fichier.
- Les appels système causant un flux d'information déclenchent la modification des *itag*.

Un tel moniteur peut être implémenté à d'autres niveaux d'observation, pour surveiller la propagation de l'information entre

- Les structures de données d'un programme
- Différents services web
- Différentes machines

Le moniteur observe

chaque appel au système :

`(read, write, fork, execve, ...)`

engendrant un flux d'information entre les conteneurs d'information du système d'exploitation.

`(fichier, socket, processus)`

Il a donc une vue locale et précise des flux d'information explicites dans le système.

Le moniteur enregistre

chaque flux observé :

`[DATE] [ORIGINE] [DESTINATION] [ITAG IMPLIQUE]`

Ce qui génère des journaux extrêmement volumineux.

Comprendre la propagation de l'information

Pour comprendre l'impact d'une application dans le système

- Quels fichiers a - t - elle modifié ?
- Quels processus a - t - elle crée ?
- Vers quelles IP a - t - elle communiqué ?

Et quel a été le comportement de ces objets eux-même ?

A la fin de l'exécution, où se trouve l'information détenue initialement par l'application ?

Les réponses à toutes ces questions se trouvent dans les journaux d'un moniteur de flux

Représentation et compréhension de comportements

Exploitons la mémoire d'un moniteur de flux

par analyse manuelle experte, par caractérisation automatique

Apport : extraire la connaissance des journaux d'un moniteur

Un Graphe de Flux System (SFG) est un multi-graphe orienté

chaque nœud est étiqueté par

- le type ;
- le nom ;
- l'identifiant système

du conteneur d'information qu'il représente

chaque arc est étiqueté par

- les dates de l'observation ;
- l'identifiant de l'information impliquée

des flux d'information qu'il représente

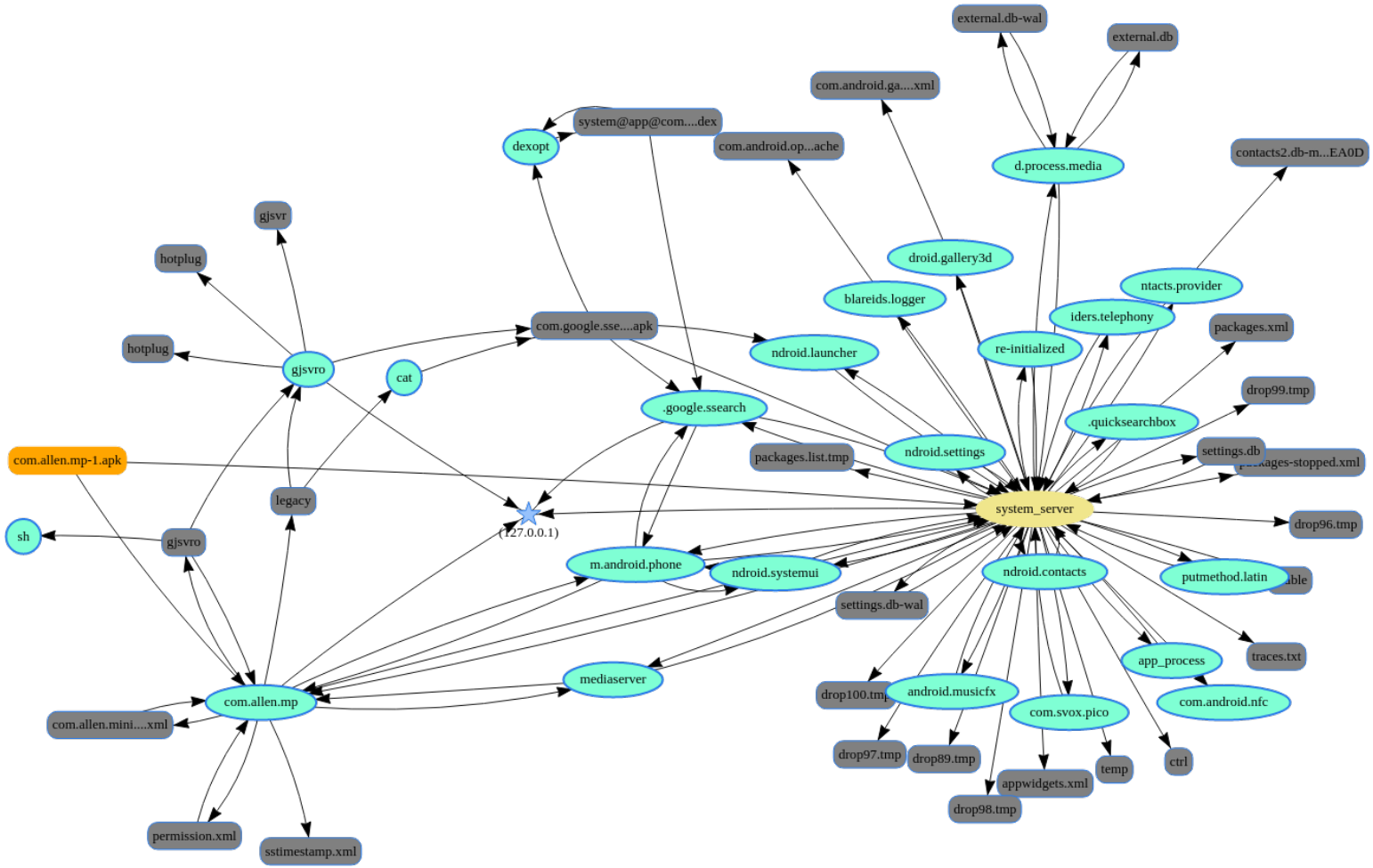
Représenter un comportement

Un SFG décrit l'ensemble des flux d'information impliquant un contenu influencé par l'information marquée initialement

Si la seule information marquée est le code d'une application

Nous apprenons comment cette application interagit avec son environnement pendant l'exécution surveillée.

SFG obtenu à partir de la surveillance d'une application Android

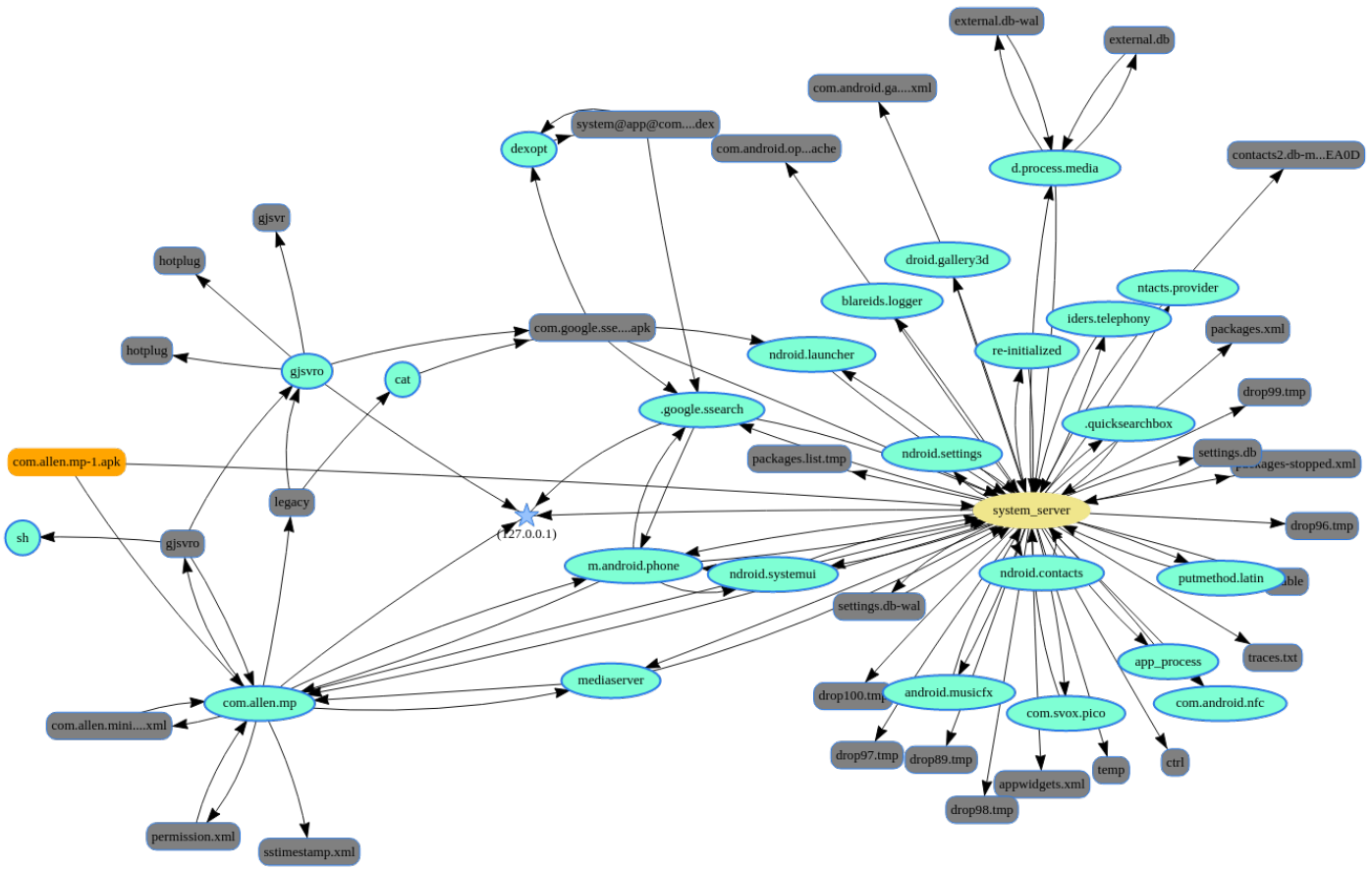


Il se révèle peu profond, peu interconnecté, de taille raisonnable

Comprendre une attaque en l'ayant observée

Ce graphe provient d'une application malveillante

Il contient les flux d'information causés par l'attaque.



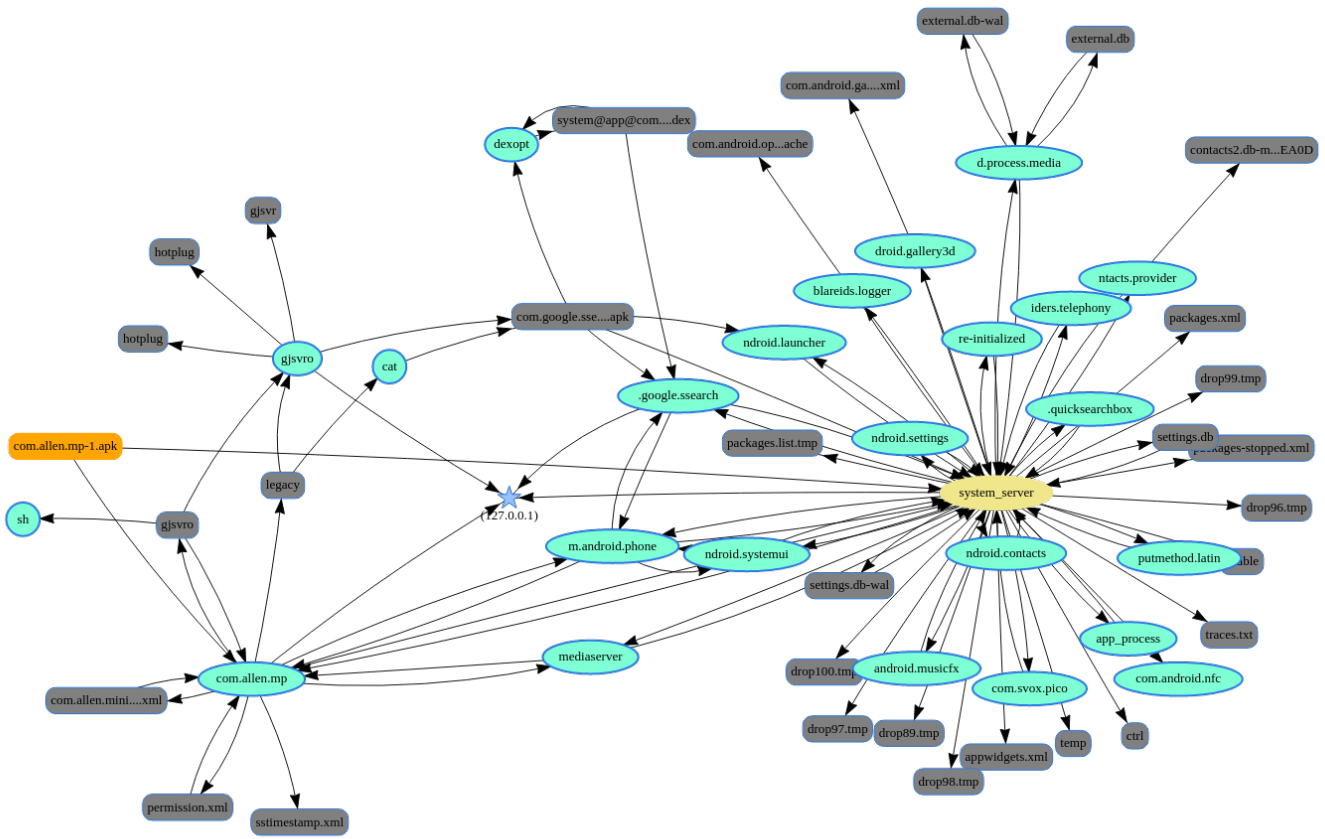
Représentation et compréhension de comportements

Exploitons la mémoire d'un moniteur de flux

par analyse manuelle experte, par caractérisation automatique

Comprendre une attaque - analyse manuelle experte

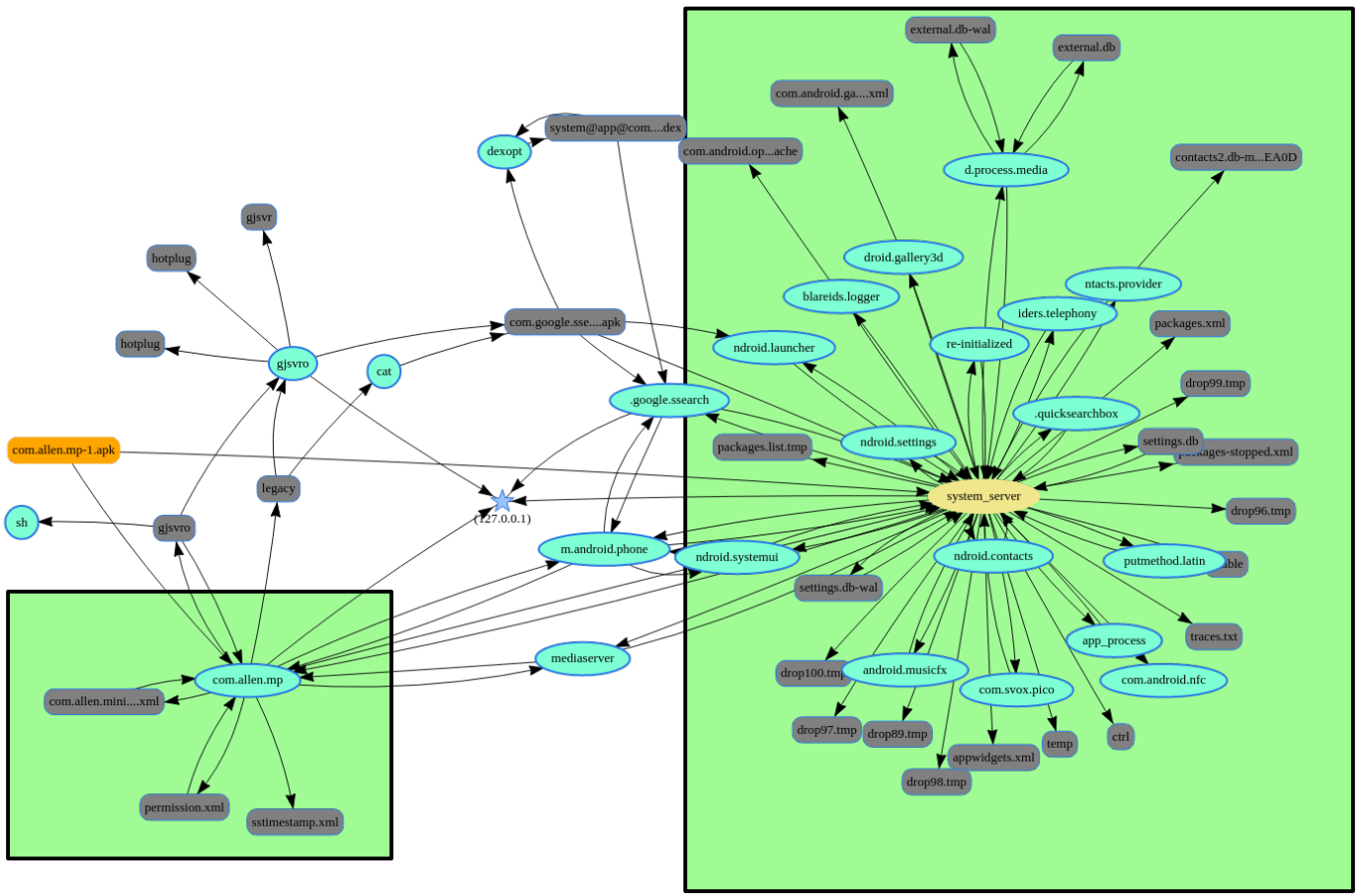
Certains sous-graphes reviennent régulièrement



Comprendre une attaque - analyse manuelle experte

Nous avons constaté que

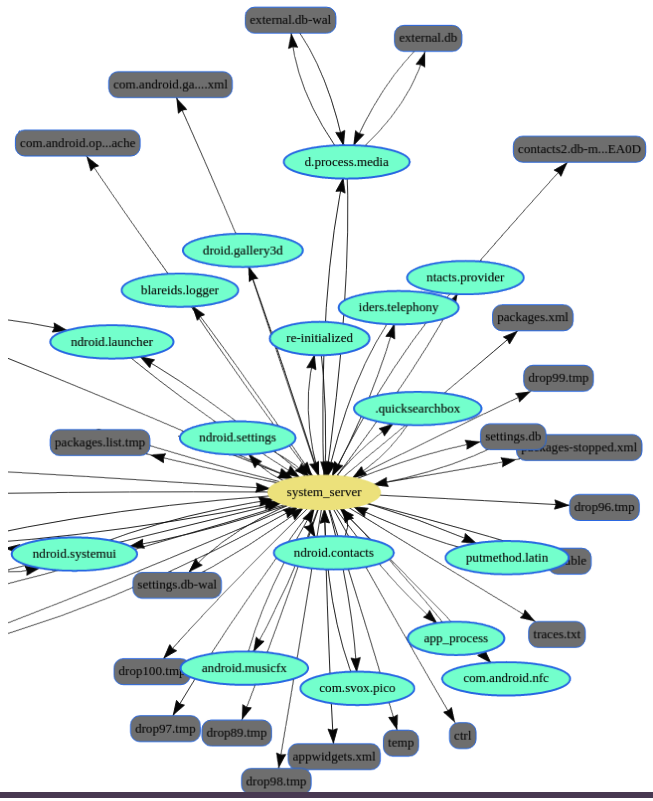
Certains sous-graphes revenaient régulièrement.



Sous graphes représentatifs de comportements habituels

Une application Android interagit avec le système *via* `system_server`

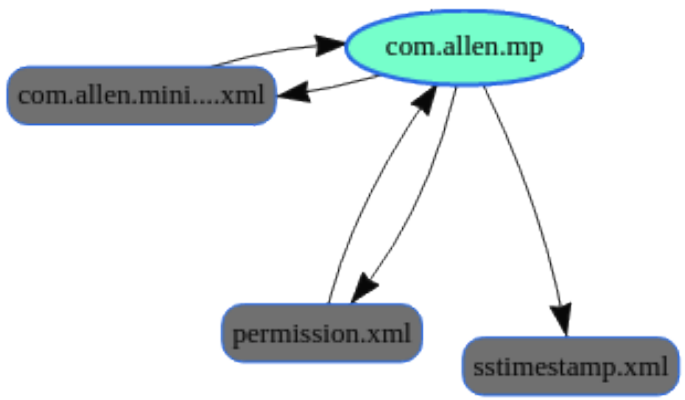
Pour accéder à des services du téléphone (sons, luminosité, contacts, localisation, SMS, appels, accès à d'autres applications.)



Sous-graphes représentatifs de comportements habituels

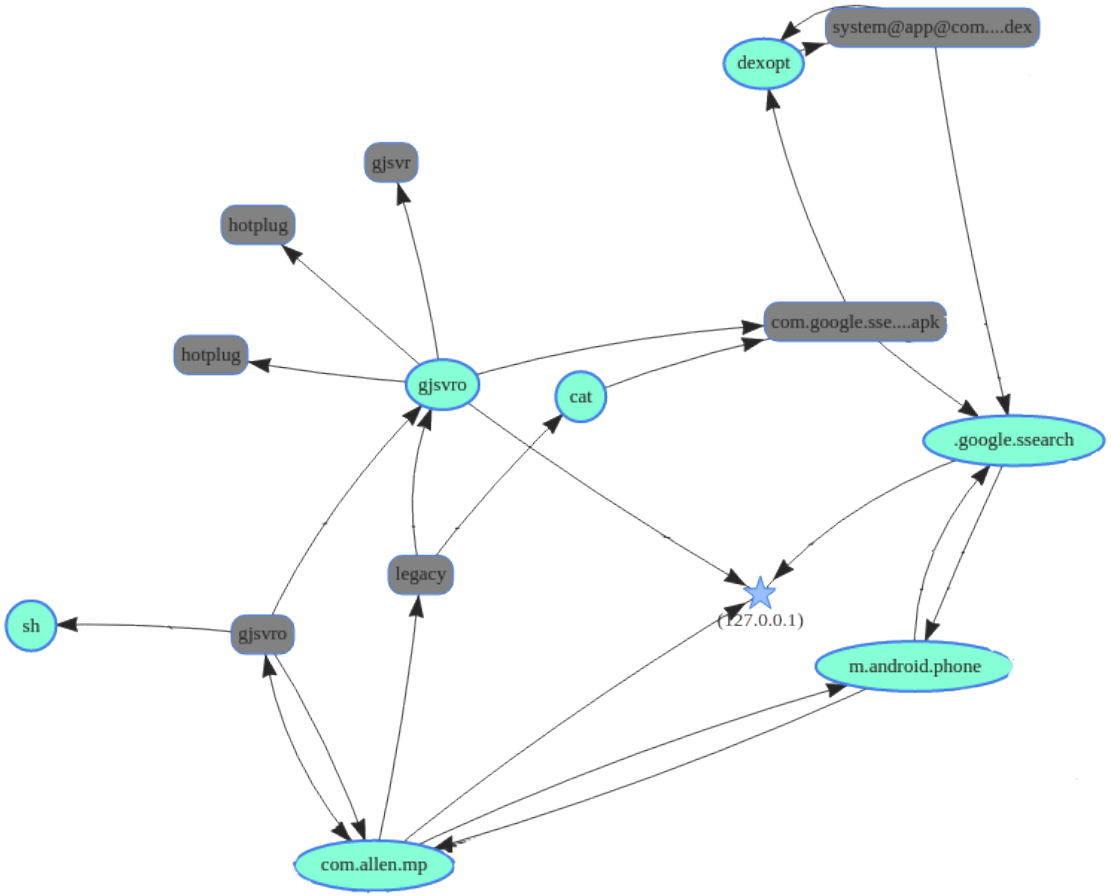
Une application Android peut écrire dans des fichiers de logs, de configuration

Donc nous ne pouvons pas juger ce sous-graphe *anormal*



Sous graphes représentatifs de comportements malveillants

Le sous-graphe restant contient le comportement malveillant



Représentation et compréhension de comportements

Exploitons la mémoire d'un moniteur de flux

par analyse manuelle experte, par caractérisation automatique

Extraction de signatures comportementales [NSS14, CSCloud15]

Hypothèse

Si des applications sont infectées par le même code malveillant, elles devraient exhiber des sous graphes égaux que l'on ne retrouve pas dans des applications bénignes.

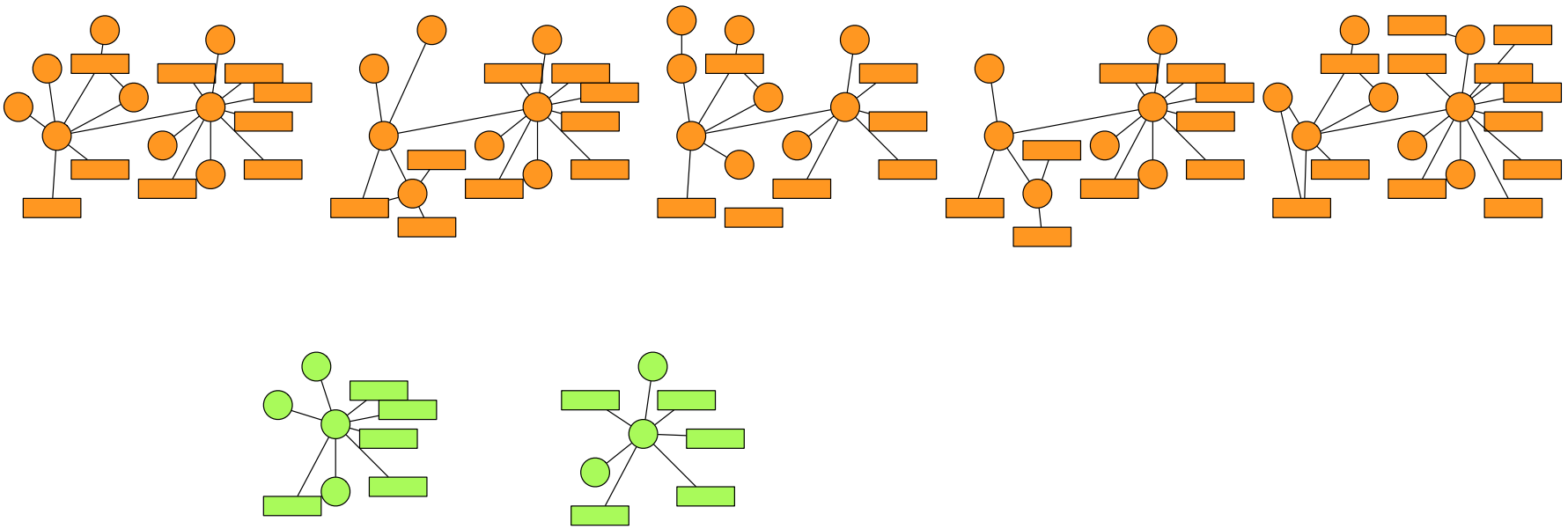
Expérience

$\mathcal{C}_{malware}$ collection de SFG malveillants :

- différentes applications infectée par des *malware*
- au moins deux exemplaires de chaque *malware*
- comportement malveillant déclenché

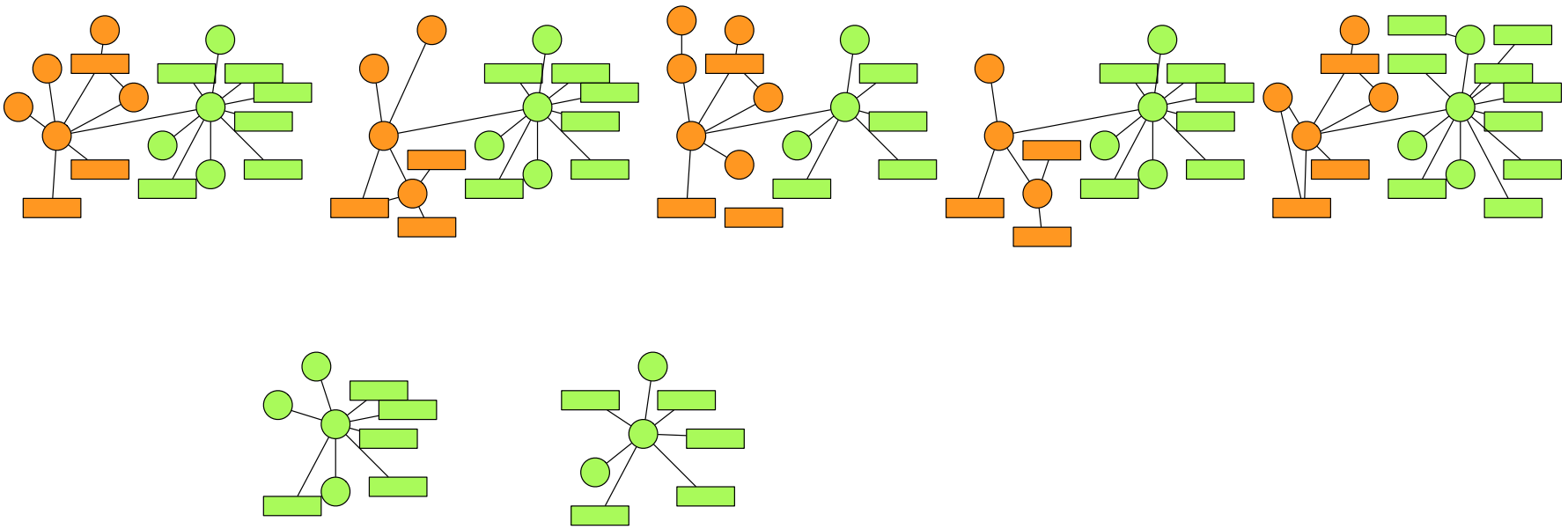
$\mathcal{C}_{goodware}$ collection de SFG jugé non malveillants

Extraction de signatures comportementales



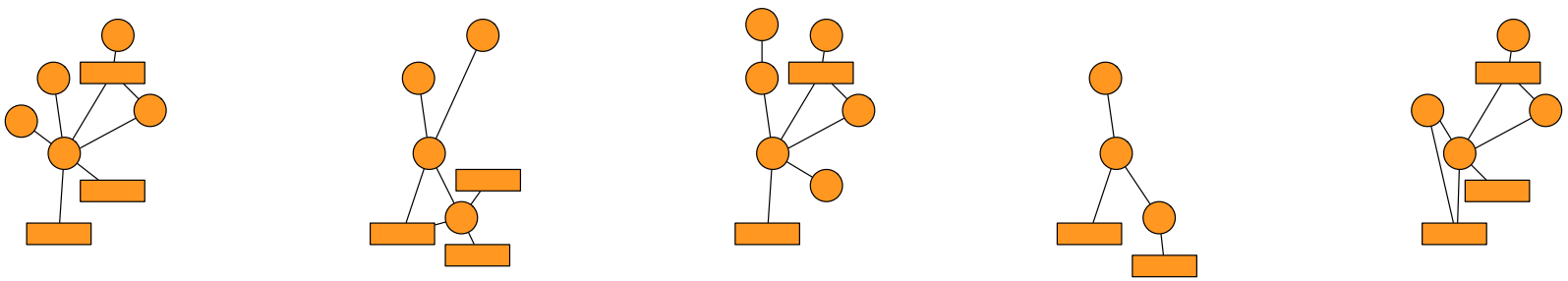
$\mathcal{C}_{malware}$ et $\mathcal{C}_{goodware}$

Extraction de signatures comportementales



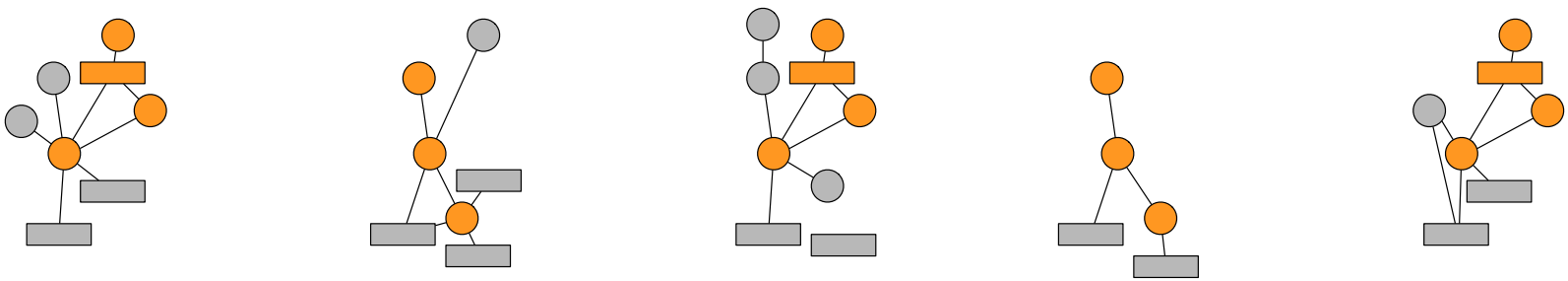
Enlever de $\mathcal{C}_{malware}$ les arcs apparaissant dans $\mathcal{C}_{goodware}$

Extraction de signatures comportementales



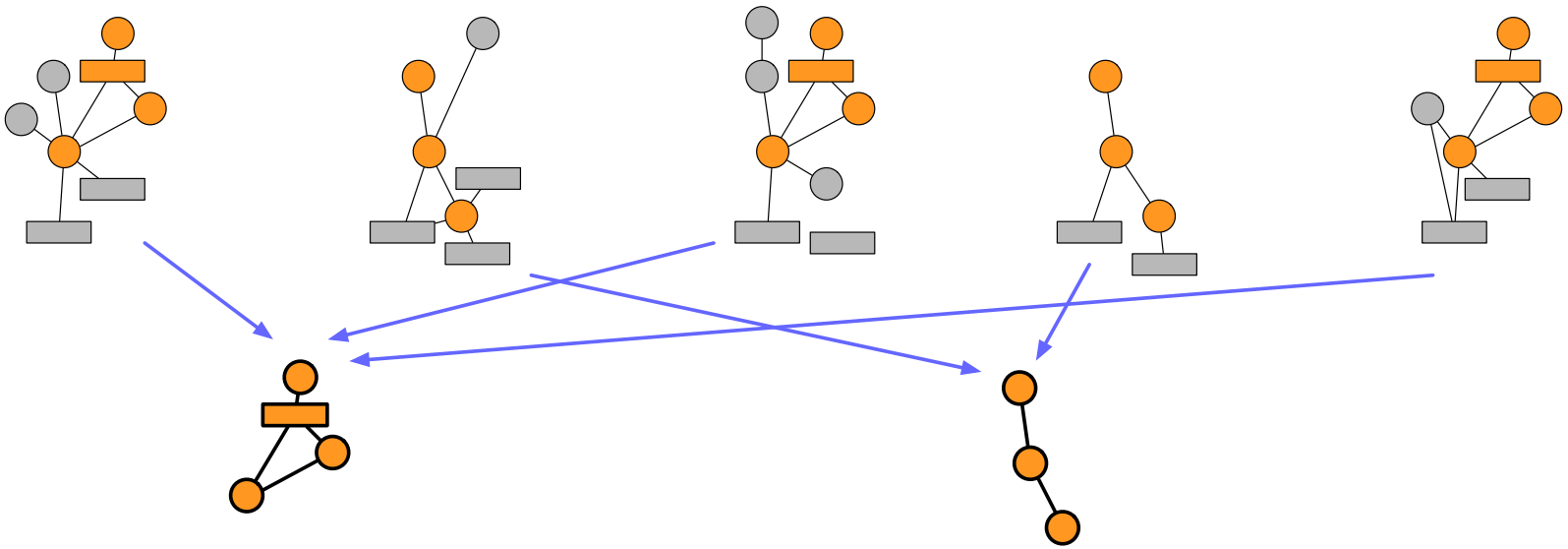
Enlever de $\mathcal{C}_{malware}$ les arcs apparaissant dans $\mathcal{C}_{goodware}$

Extraction de signatures comportementales



Chercher les plus grands graphes égaux
à au moins deux sous-graphes des graphes dans $\mathcal{C}_{malware}$

Extraction de signatures comportementales



Chercher les plus grands graphes égaux
à au moins deux sous-graphes des graphes dans $\mathcal{C}_{malware}$

Extraction de signatures comportementales [NSS14, CSCloud15]

Hypothèse

Si des applications sont infectées par le même code malveillant, elles devraient exhiber des sous graphes égaux que l'on ne retrouve pas dans des applications bénignes.

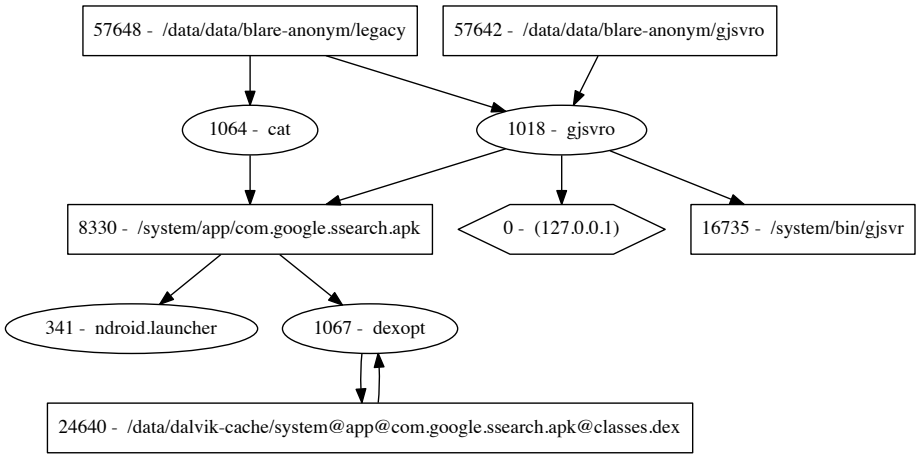
Hypothèse vérifiée dans les expériences menées

Nous avons pu exhiber des sous-SFGs signatures des codes malveillants

DroidKungFu (2011), DroidKungFu2 (2011),
BadNews (2013), JsmsHider (2011)

avec 19 SFG dans $\mathcal{C}_{malware}$, 5 SFG dans $\mathcal{C}_{goodware}$

Exemple de signature obtenue : DroidKungFu2 (2011)



Ce sous-graphe

- ne se retrouve jamais dans les SFG d'applications bénignes étudiées (70)
- se trouve toujours dans les SFG d'applications infectées par DroidKungFu2 que nous avions

Cela a été le cas pour les autres signatures calculées.

Bilan sur la représentation de comportements

Un *graphe de flux système* capture comment une information influence son environnement durant une exécution.

La caractérisation de *malware* en utilisant ces graphes est une approche prometteuse.

Néanmoins, cette approche est intéressante tant que

- Le même code malveillant apparaît au moins deux fois,
  $\rightarrow$ sinon l'algorithme de classification échoue
- Il n'y pas obfuscation comportementale
  $\rightarrow$ les signatures pourraient être très petites, voire vides
- Un comportement malveillant est effectivement observé :
  $\rightarrow$ sinon l'analyse dynamique est inutile

Des Graphes de Flux Système pour caractériser des comportements malveillants. [Projet Labex CominLabs Kharon (2015-2018)]

Décider

si une application est malveillante
(ou non) en étudiant son graphe de flux système.

Pour cela

- savoir reconnaître des comportements malveillants connus ✓
- savoir identifier des comportements bénins

Challenges

- disposer de graphes pertinents
- disposer de graphes complets

Graphes de Flux Système pertinents / complets

Graphes qui permettent décider si l'application est malveillante

Par exemple, si :

- un graphe présente un comportement malveillant
  $\rightarrow$ il suffit d'observer une exécution malveillante
  (graphe pertinent)
- un graphe ne présente aucun comportement malveillant
  $\rightarrow$ il faut d'observer tous les comportements
  (graphe complet)

Pour obtenir un graphe pertinent

Il suffit d'avoir su observer l'exécution du code malveillant

Playing with the infected app as GroddDroid

1. Collects graphical elements
2. Explores the app by clicking on the buttons
3. Can go back
4. Can launch the app again
5. Detects loops
6. Until he has explored all the different activities
7. Can force the malicious code if needed

Déclenchement de comportements malveillants

Annuler les protections contre l'analyse dynamique

Les applications malveillantes se protègent de l'analyse dynamique

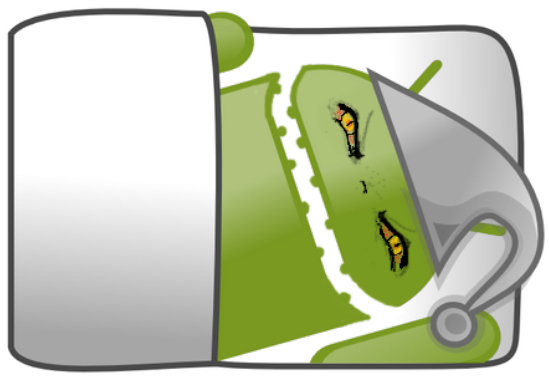
Il ne suffit pas de lancer une application infectée

pour voir le code malveillant s'exécuter

Beaucoup de codes malveillants sont dormants

et attendent *quelque chose* avant de se déclencher

- dix minutes, une semaine ;
- un message d'un serveur distant ;
- un événement système ;
- un événement utilisateur ;



Défaire les protections contre l'analyse dynamique

[Malcon'15]

Caractérisques d'un code Android malveillant

- a un point d'entrée dans le bytecode de l'application
- peut être chiffré, obfusqué, chargé dynamiquement
- peut chercher à envoyer des SMS, effectuer des appels, dialoguer avec un serveur distant

Aafer, Du & Yin [SecureComm2013] ont identifié

Un ensemble d'APIs dont
`Android.telephony.SmsManager`, `Java.net.UrlConnection`,
`Android.telephony.TelephonyManager`, `Java.lang.Process`
plus utilisées par les *malware* que par les *goodware*

1. travail encadré avec JF Lalande

Identification du code suspect par analyse statique

Score de risque pour chaque méthode du bytecode

Plus une méthode utilise des APIs sensibles
plus son *score de risque* augmente.

Hypothèse

Une partie du code malveillant présente des méthodes avec un score de risque élevé

Expérience

Nous avons vérifié cette hypothèse sur une collection de malware que nous avons totalement retro-analysés.

- La fonction de score pointe 0.009% des méthodes du bytecode
- Ce code est effectivement malveillant 72% des cas

Reconstruire un chemin d'exécution vers le code jugé suspect

Modifier le bytecode

en élaguant le graphe de flots de contrôle inter-procédural.

Détails

- Calculer le graphe de flot de contrôle inter-procédural
- Reconstruire un chemin menant d'un point d'entrée vers le code jugé suspect
- Annuler les choix s'éloignant du code suspect le long de ce chemin
- Reconstruire l'application et exécuter l'application

Expérience

Ce travail déclenche le code malveillant dans 28% des cas.

Reconstruire un chemin d'exécution vers le code jugé suspect

Modifier le bytecode

en élaguant le graphe de flots de contrôle inter-procédural.

Détails

- Calculer le graphe de flot de contrôle inter-procédural
- Reconstruire un chemin menant d'un point d'entrée vers le code jugé suspect
- Annuler les choix s'éloignant du code suspect le long de ce chemin
- Reconstruire l'application et exécuter l'application

Expérience (juin 2015)

GroddDroid, déclenche le code malveillant dans 28% des cas.

Conclusion

Suivre les flux d'information

- est un bon outil pour la sécurité
- nécessite de savoir implémenter un moniteur pour le niveau d'observation choisi

Analyser les malware Android

nécessite des outils d'analyse statiques et dynamique