Android malware

Attacks & Counterattacks

Valérie Viet Triem Tong

CentraleSupelec
Equipe INRIA Cidre/CentraleSupelec/CNRS/Université de Rennes 1
valerie.viettriemtong@centralesupelec.fr

Séminaire Aristote - Ecole Polytechnique
1er décembre 2016

CentraleSupélec

This talk focuses on *Security of Android devices*

Mobile Malware

- a popular application may be : 50 000 000 to 100 000 000 downloads
- it is easy to submit apps on Google Play
- 1,400,000 apps in 2014 (Google Play)
- + 1200 % of malicious apps in 2012

Why these malware have appeared ?

Because they are a simple way to make money

- by sending text message to premium numbers
- by taking control of the device
- by spying the user
- in ransoming the user

What are these Android malware ?

Most of the time

- contaminate the device through an Android application
- hidden in a benign application (repackaging)
- few pieces of code among a lot of lines of java byte code

How to get rid of these malware ?

By static analysis

- detect that a code is malicious *without executing it*
- detect → analyze → react

By dynamic analysis

- detect that a code is malicious *by observing its execution*
- detect → analyze → react

Challenges in static analysis

Malware easily circumvent (automatic) static analyzer

- who try to detect that a code is malicious *without executing it*

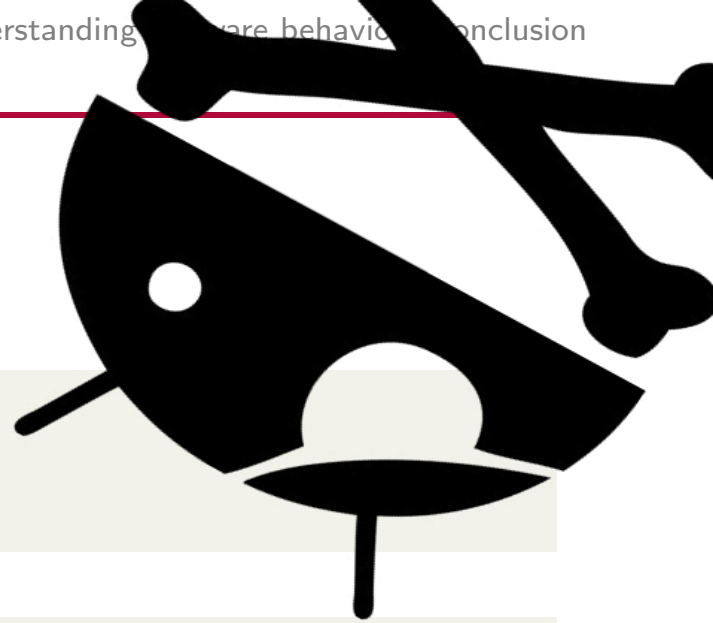
Studied pieces of code have to be *at least readable* by the analyzer

It's thus really easy to evade static analyzers

Malware authors simply use

- obfuscation
- reflection
- encryption
- dynamic loading
- ...

For instance ..



```java
// without reflection
Foo foo = new Foo();
foo.hello();
```

```java
// with reflection and string obfuscation
String var1 = "Y2xhc3NwYXRoLkZvbw==";
Object obj = Class.forName(new String (Base64.decode(
    var1, Base64.DEFAULT))).newInstance();
String var2 = "aGVsbG8=";
Method m = obj.getClass().getDeclaredMethod (new String
    (Base64.decode(var2, Base64.DEFAULT))), new Class
    <?>[0]);
m.invoke(obj);
```

Challenges in dynamic analysis

Malware easily circumvent (automatic) dynamic analyzer

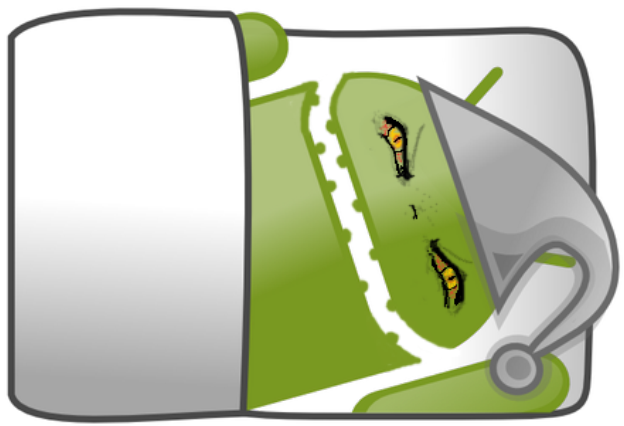
- who try to detect that a code is malicious *by observing its execution*

Studied pieces of code have to be *at least executed* by the analyzer

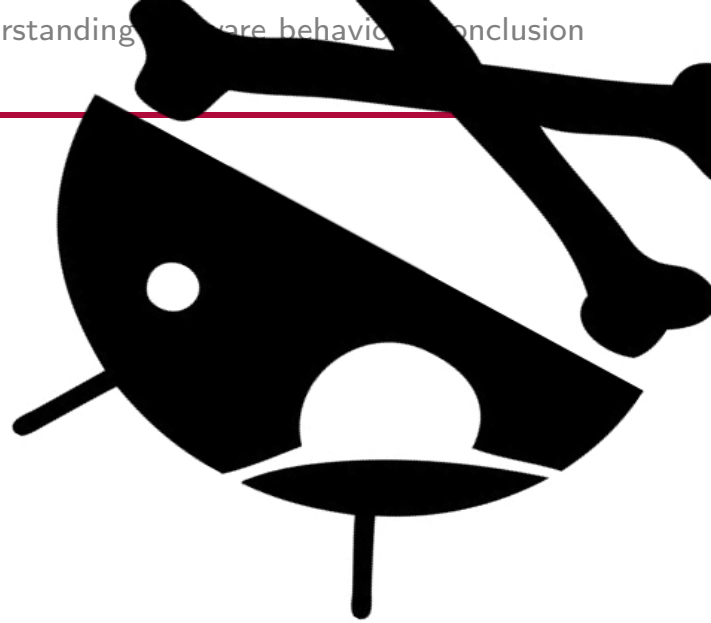
It's thus really easy to evade dynamic analyzers

Malware authors simply delay the attack

- wait for a period of time
- wait for a user event
- wait for a remote order
- wait *something*
- ...



For instance ..



Dynamic analysis evasion tries to ensure that malicious code is not monitored.

```
1 if getDeviceID().contains("000000000") ||
     getFileStreamPath("/sys/qemu_trace").exists()
2     stay_dormant(); // Possibly Emulator
3 else
4     do_evil();      // Device
```

Roadmap

Dynamic analysis is a promising approach but is not enough studied.

It mainly faces to the following challenges

1. Suspicious code triggering
2. Capturing malware behavior
3. Attack explanation

Triggering suspicious behaviors

Defeating malware protections against dynamic analysis

Protections against dynamic analysis

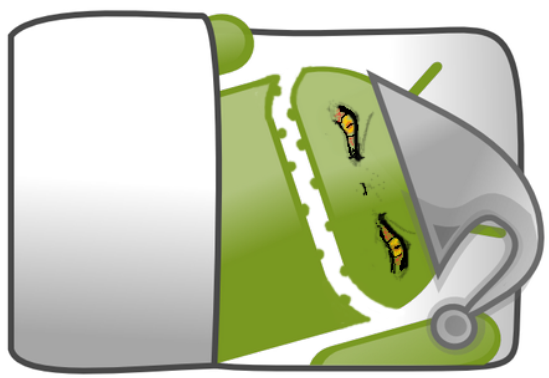
Launching infected app is not sufficient

to observe malicious executions

Malware don't run on demand

and wait for a triggering event

- a period of time ;
- a message from a remote server ;
- a system event ;
- a user event ;



Defeating protections against dynamic analysis

Best Paper at the Int. Conf. on Malicious and Unwanted Software 2015

An Android malware

- has an entry point in the bytecode of the app
- can be dynamically loaded, encrypted, obfuscated
- can try to send SMS, make calls, obeys to a remote server

Aafer, Du & Yin [SecureComm2013] have identified

Some APIs as
`Android.telephony.SmsManager`, `Java.net.UrlConnection`,
`Android.telephony.TelephonyManager`, `Java.lang.Process`
more used by *malware* than by *goodware*

Suspicious code targeting (by static analysis of the bytecode)

For each method in the bytecode computes a risk score.
The more the method uses sensitive APIs, the higher is the score.

The scoring function points out malicious codes

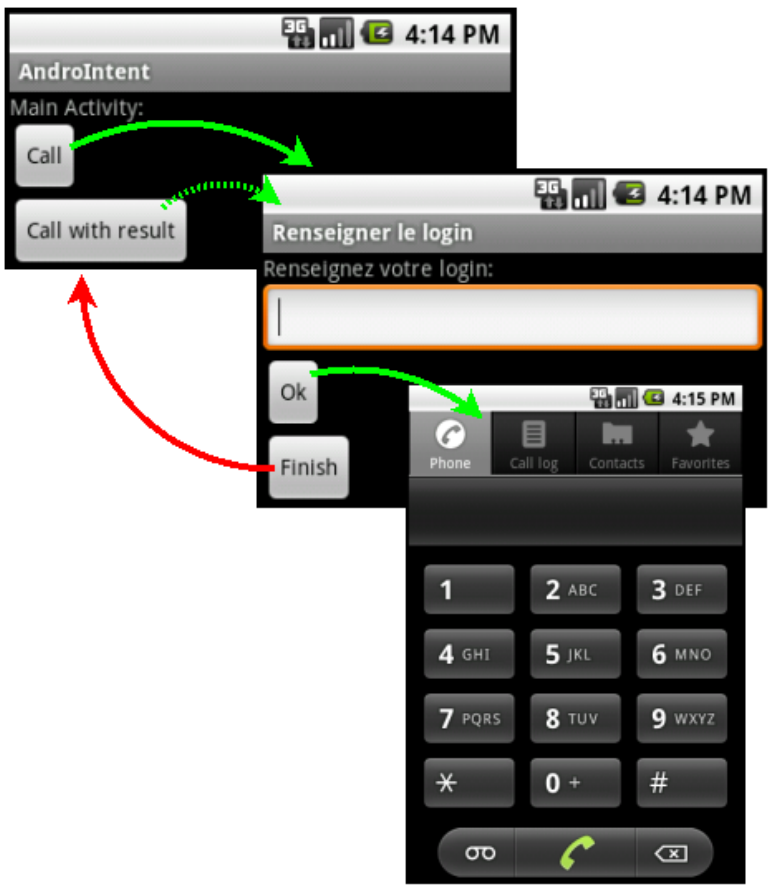
On our dataset of reversed malware

- 0.009% of methods have a score higher than 0
- 72% of them are (indeed) malicious

Playing with the infected app as GroddDroid

kharon.gforge.inria.fr/grodddroid.html

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



Is GroddDroid able to force malware to execute ?

GroddDroid reconstructs an execution path towards the suspicious code

GroddDroid modifies the bytecode and cancels the conditional jumps that could drive away from the malicious code

First experiments

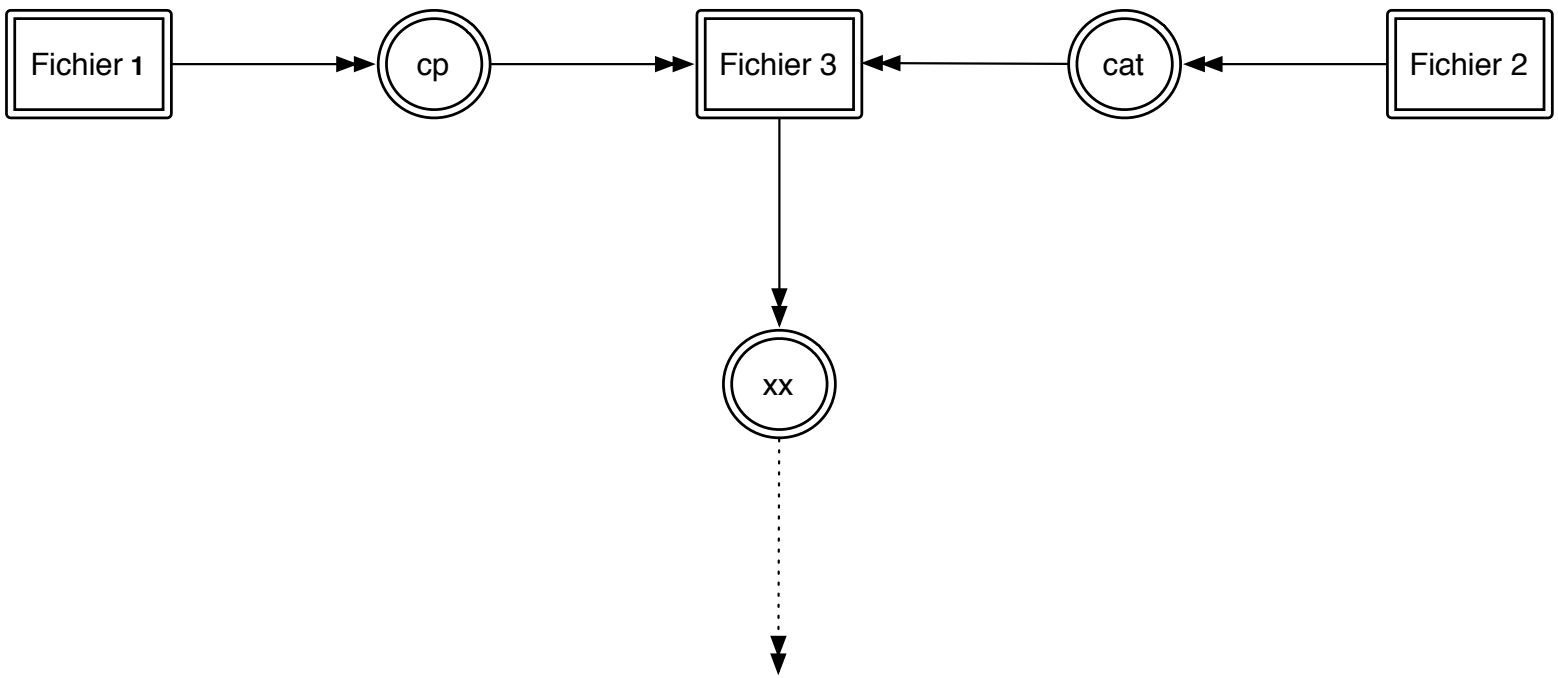
GroddDroid succeed to trigger 28 of the most scored units of code appearing on 100 malware

Capturing malware behavior

Monitoring information flows at system level

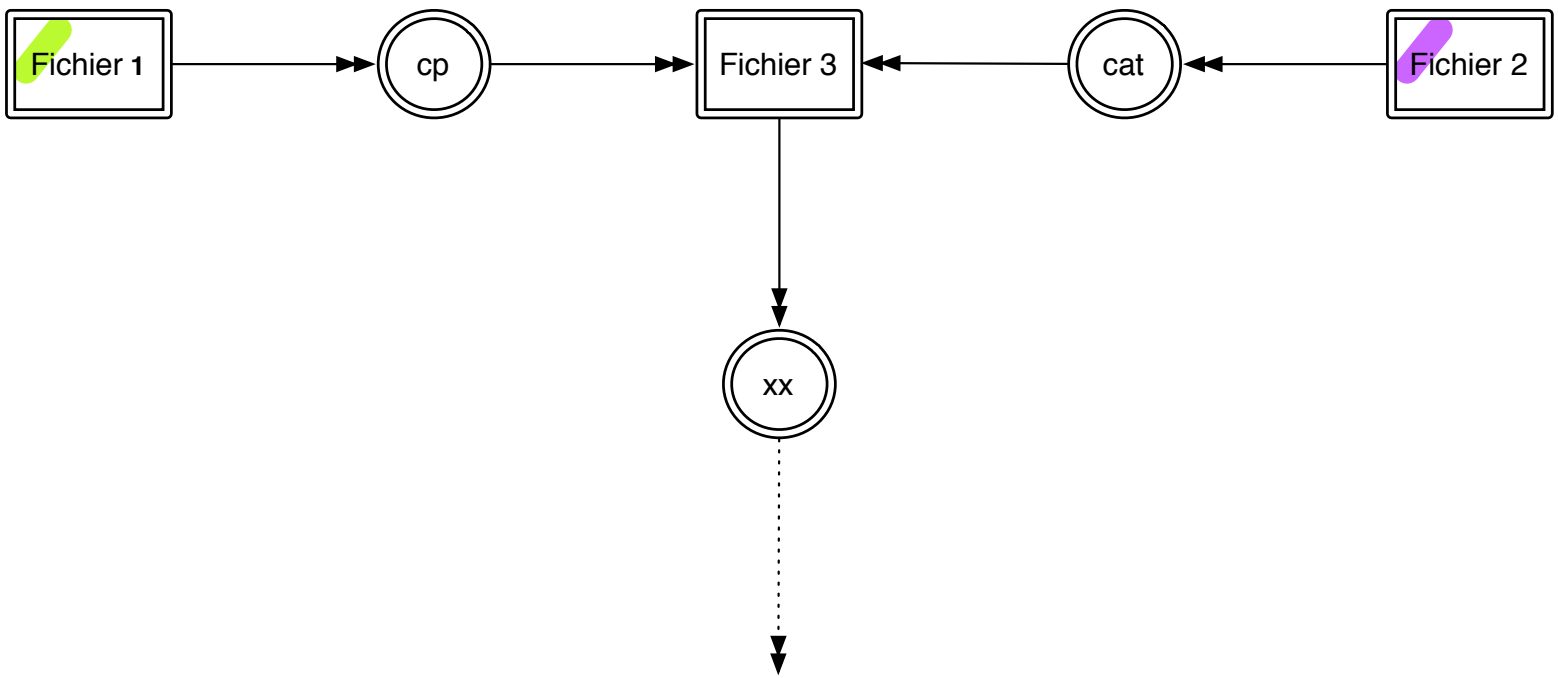
Principles

1. A mark is attached to each sensitive piece of information
2. Marks are propagated at each observation of a flow



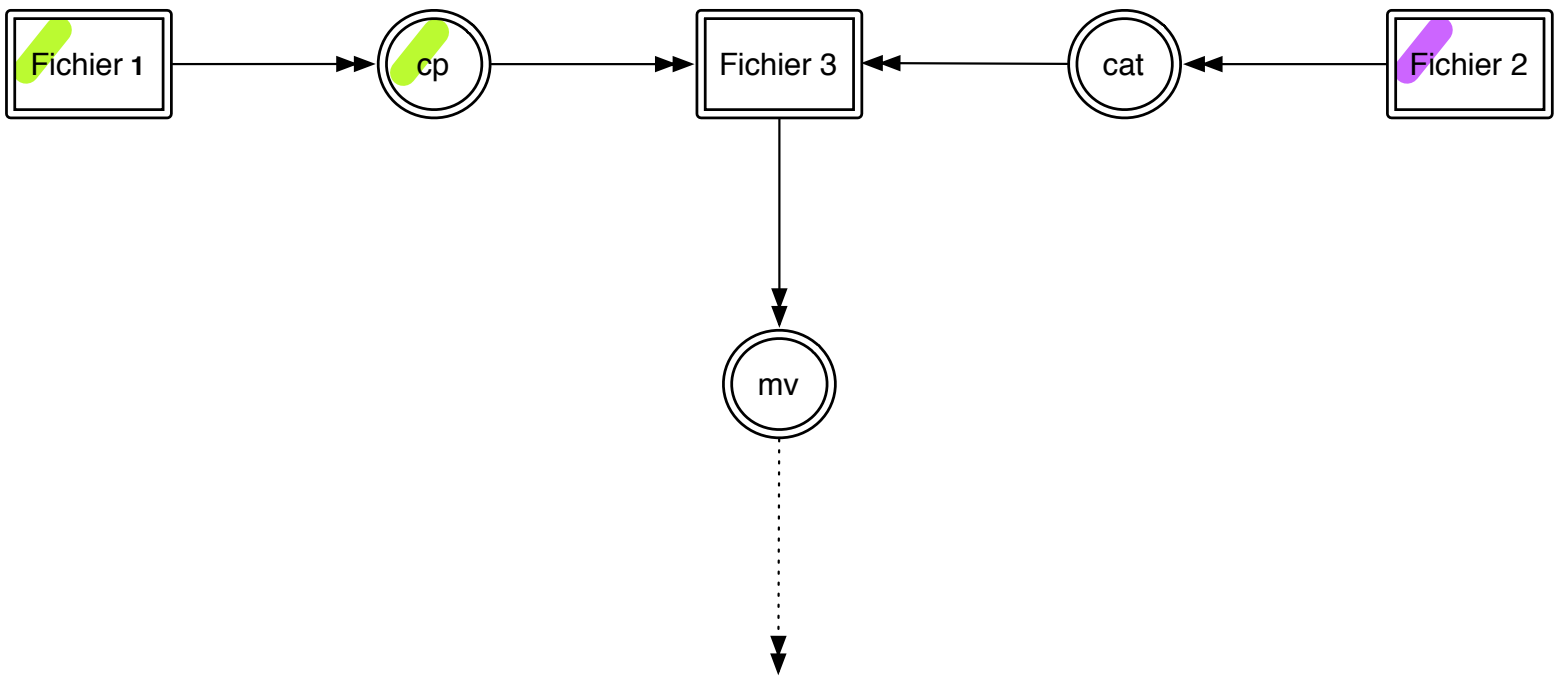
Principles

1. A mark is attached to each sensitive piece of information
2. Marks are propagated at each observation of a flow



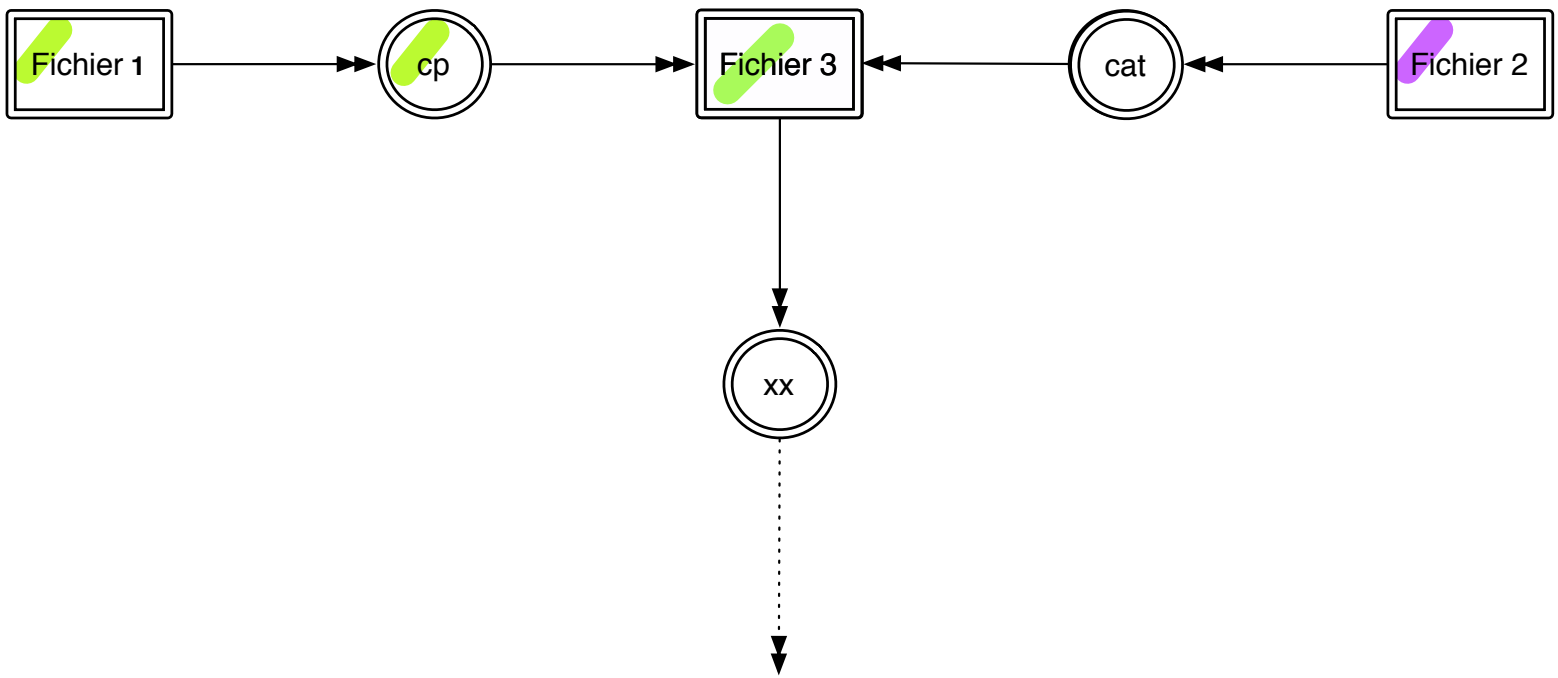
Principles

1. A mark is attached to each sensitive piece of information
2. Marks are propagated at each observation of a flow



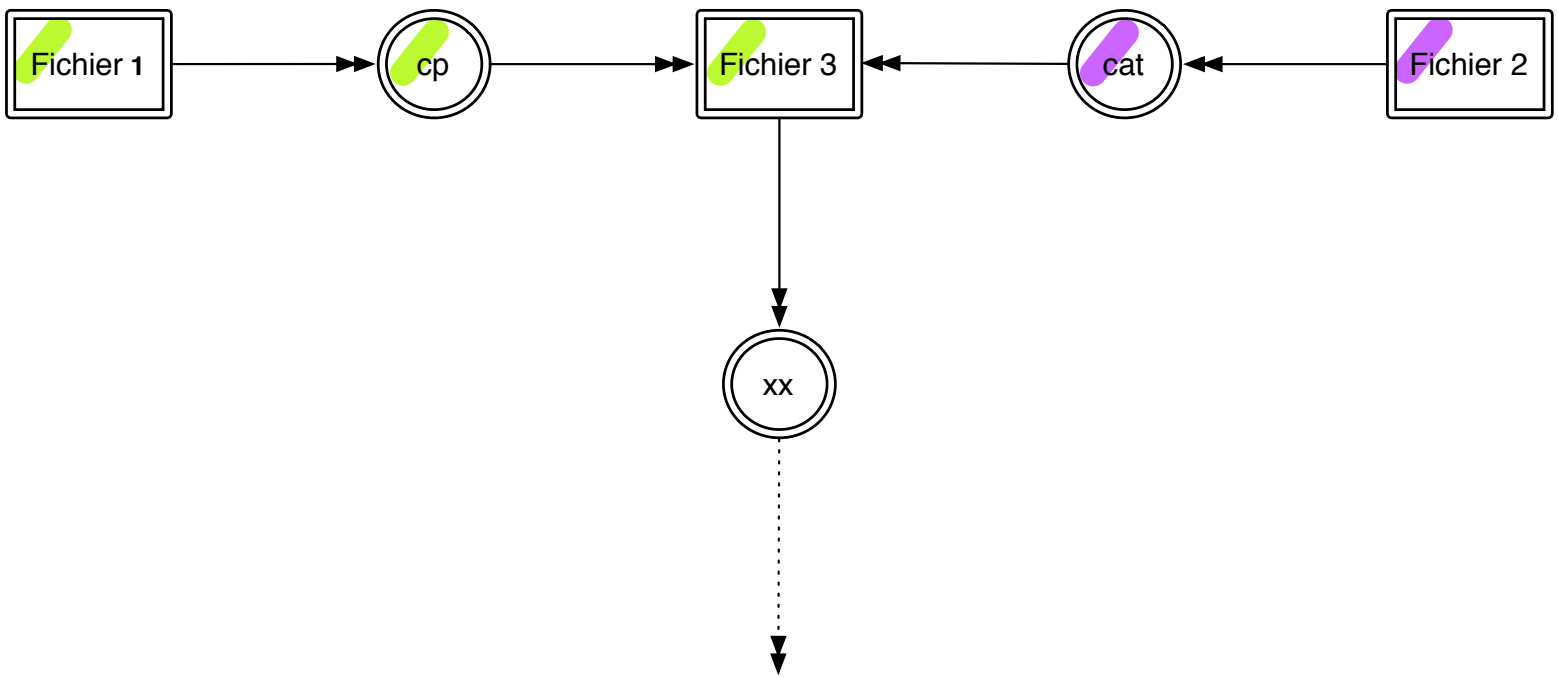
Principles

1. A mark is attached to each sensitive piece of information
2. Marks are propagated at each observation of a flow



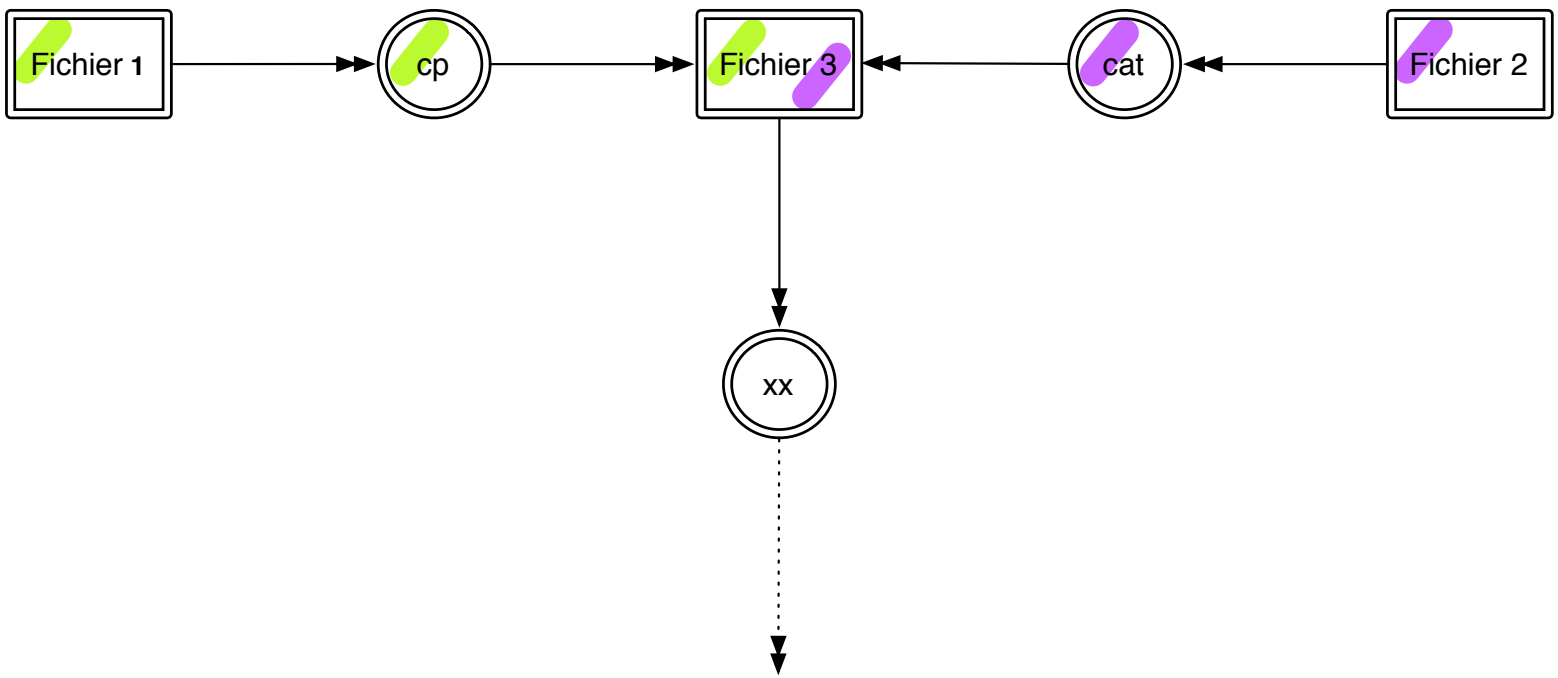
Principles

1. A mark is attached to each sensitive piece of information
2. Marks are propagated at each observation of a flow



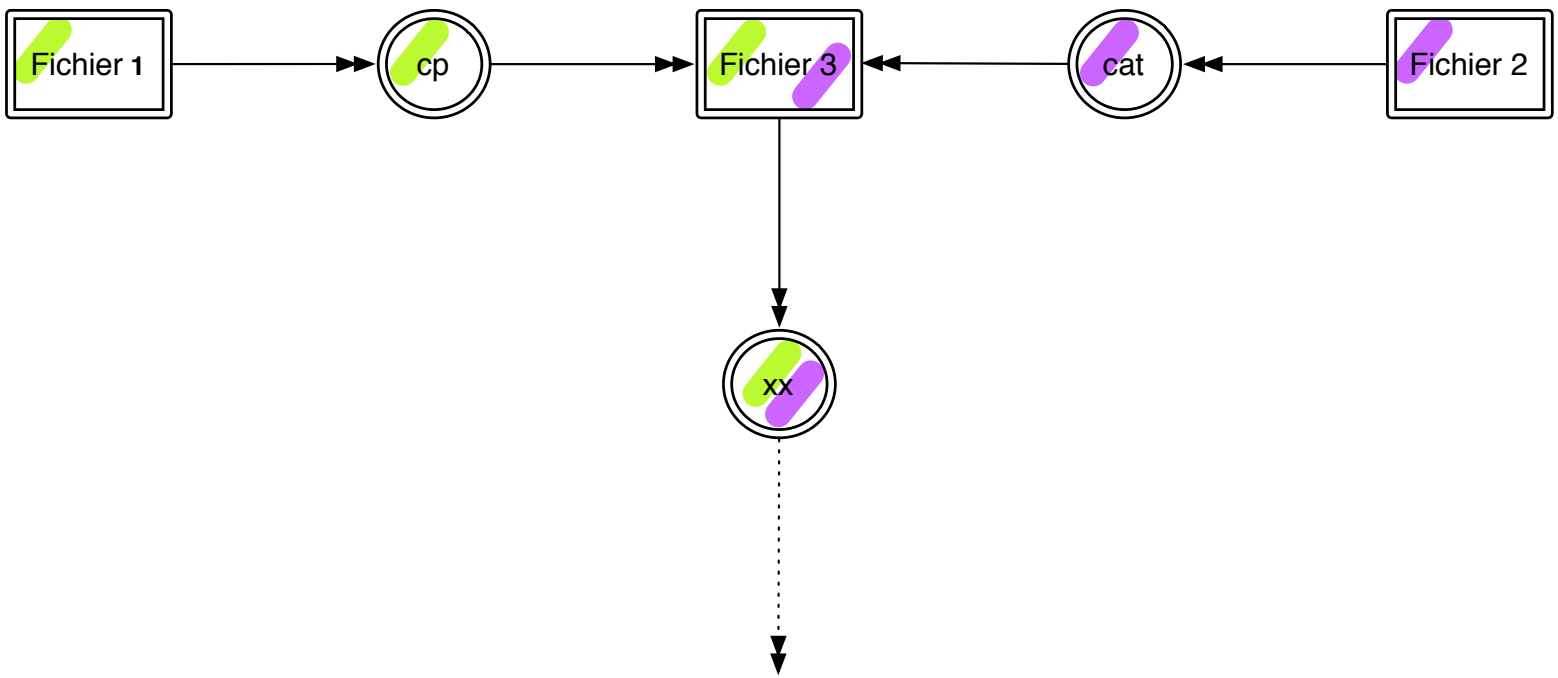
Principles

1. A mark is attached to each sensitive piece of information
2. Marks are propagated at each observation of a flow



Principles

1. A mark is attached to each sensitive piece of information
2. Marks are propagated at each observation of a flow



Marks initialization

In our model, origin of information is explicit

Each sensitive piece of information is uniquely identified

Information flows

are performed by the kernel during system calls.

(Andro)Blare : our information flow monitor `blare-ids.org`

(Andro)Blare

1. maintains marks in extended attributes of files
2. updates marks at each observation of an information flow
3. is implemented within an adaptation of LSM framework

(Andro)Blare observes

Each system call

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

that generates an information flow between files, sockets or processes.

`(file, socket, processus)`

Blare has a local but accurate view of explicit information flows

Blare logs

each observed flow

`[TIMESTAMPS] [ORIGIN] [DESTINATION] [ITAG]`

that generates a huge amount of log entries

Understanding data dissemination

To better understand how a malware impacts its environment

- Which files have been modified ?
- Which processes have been executed ?
- Which remote IP adresses have been contacted ?

And how these objects interact ?

At the end of the execution where are data originated from the application ?

The answers are in the log of the information flow monitor.

Extracting knowledge of logs

A System Flow Graph (SFG) is a directed labelled graph

Nodes are labelled by

- type ;
- name ;
- system id

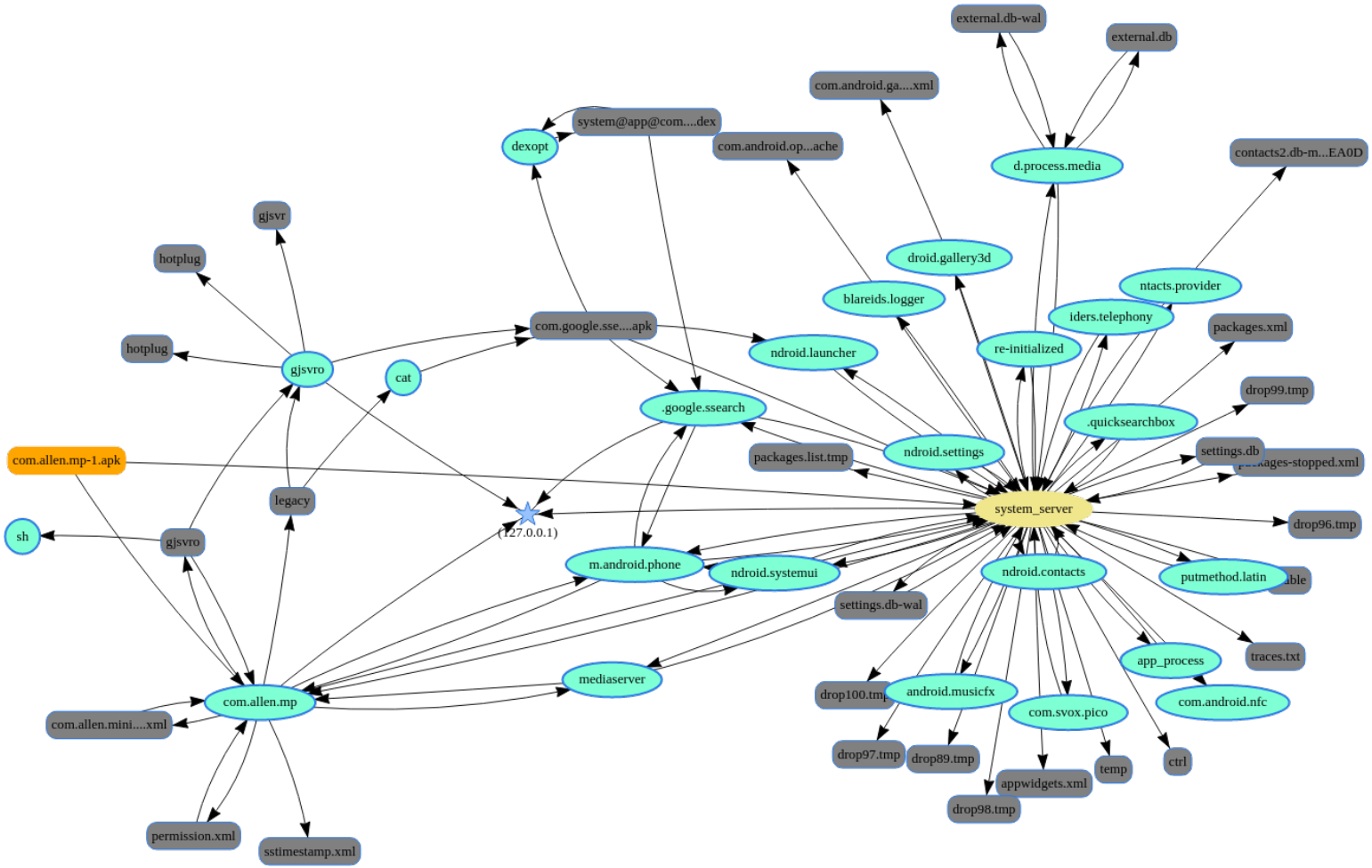
and model containers of information

Edges are labelled by

- timestamps ;
- itags of involved contents

and model information flows

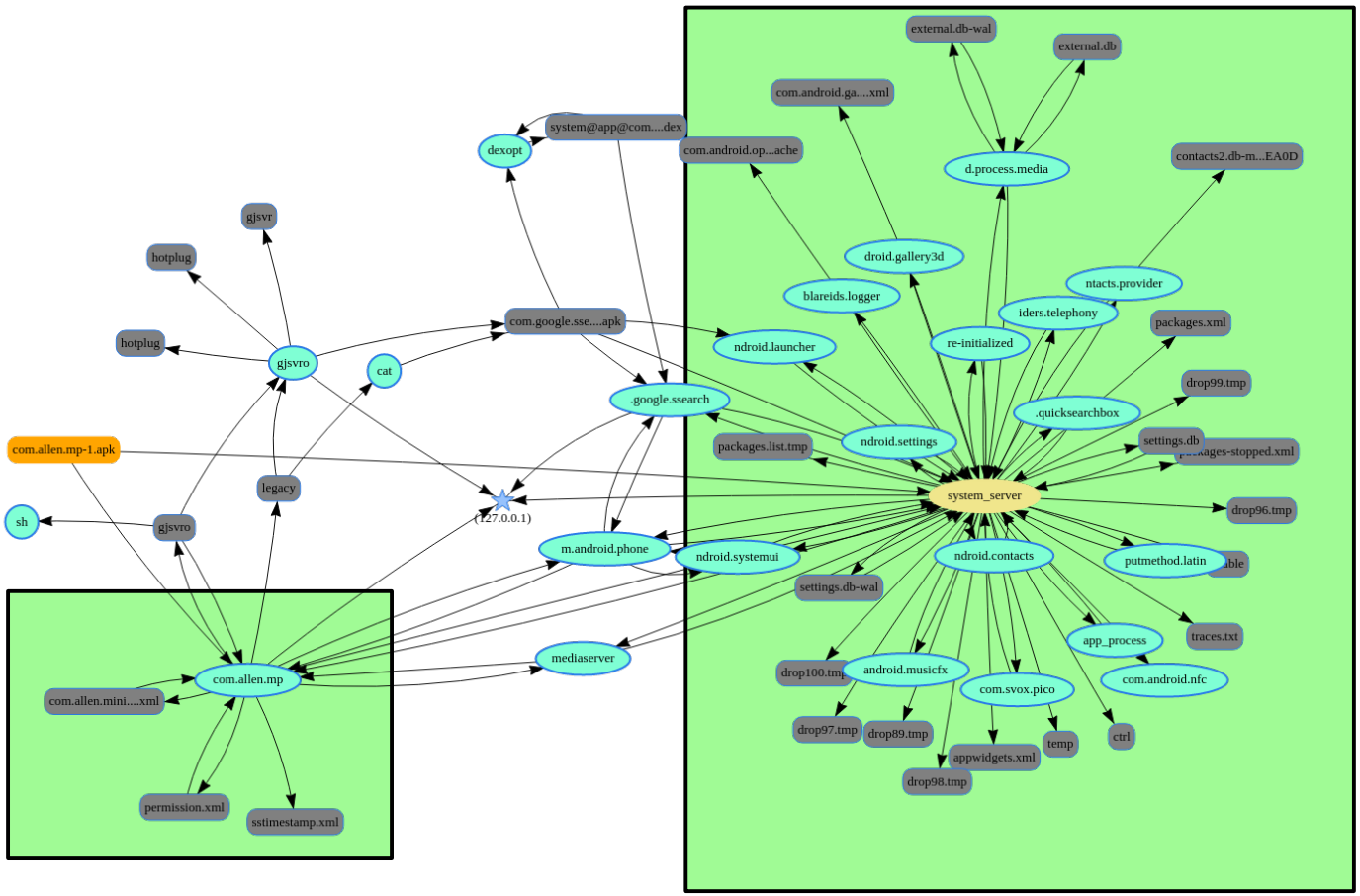
SFG due to the monitoring of an Android application



Understanding malware behavior

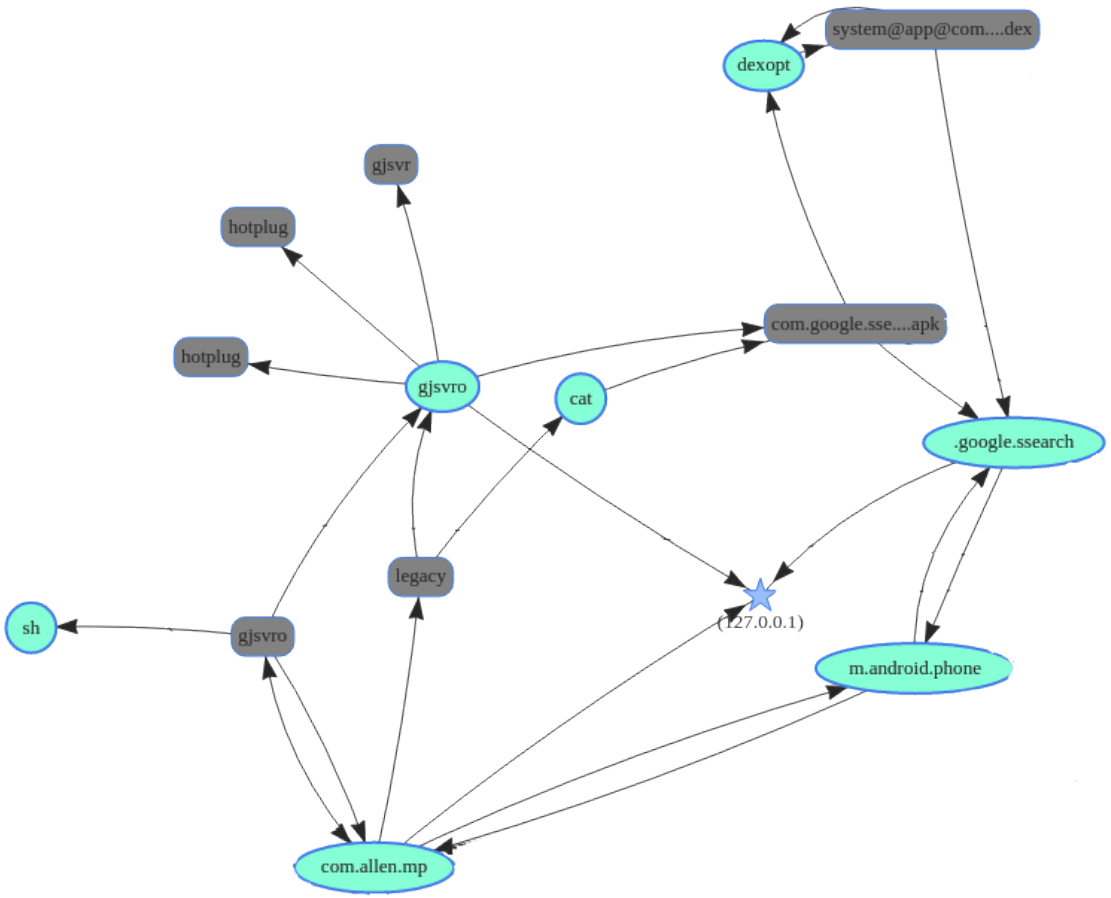
Using logs of an information flow monitor

Usual Subgraphs

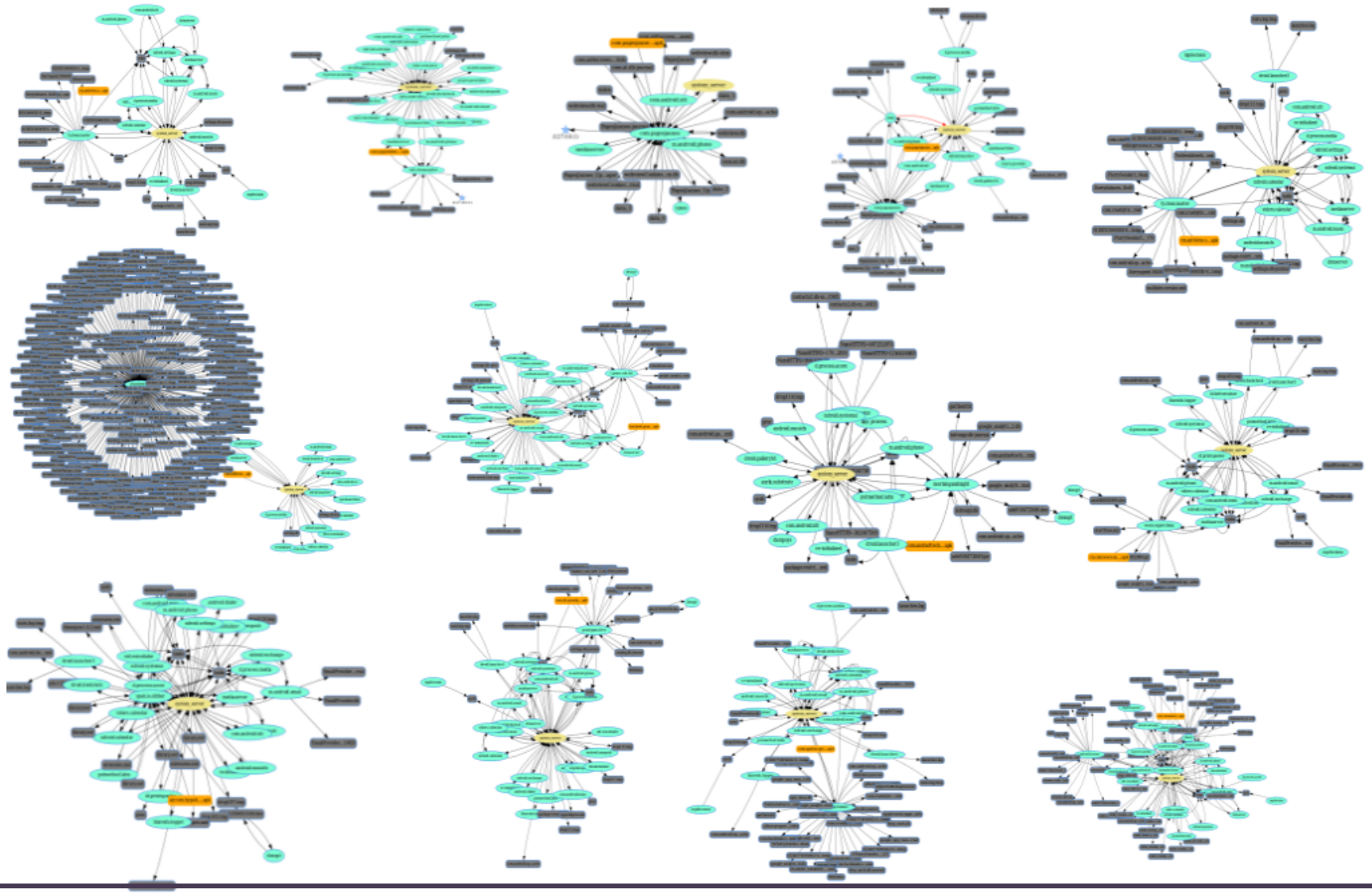


Malicious behavior

The attack is described in the remaining subgraph



What can we do on big collection of malware ...



Conclusion

Labex CominLabs Kharon Project (2015-2018)

Main objectives of the project

- Decide if an application is a malware by studying its SFG.
- Give a simple representation of malware behavior.

Already realized

- GroddDroid a framework to triggering suspicious behaviors.
- The Kharon Platform an analysis platform located at the LSH-Rennes
- Kharon dataset a collection of reversed malware

Together with **J.F. Lalande (Insa CVL) & T. Genet (U. Rennes 1)** & R. Andriatsimandefitra (Phd)& M. Leslous (Phd) & A. Trulla (Phd) and more than 15 students from CentraleSupelec and Insa CVL.

`kharon.gforge.inria.fr`