Information flow monitoring

at the Operating System level

Valérie Viet Triem Tong

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

Journées Nationales 2018 Pré-GDR Sécurité Informatique

June 2018

From a general point of view

Information flows monitoring makes us aware of

How *a given piece of information* spreads in *a system*

Two requirements

1. An identification of pieces of information to track
2. A monitor aware of all information flows in the system

Observation level

The system can be

- an operating system
- a network
- a hardware device
- a program

Information	*Container*
integer	register
variable value	variable
file content	file
virtual machine	cloud architecture
knowledge	user
⋮	⋮

Existing information flows monitors

Name	Observation level	System	Since
CAMFlow	Cloud	Linux VMs	2014
Blare	**OS**	**Linux**, **Android**	**2003**
Laminar	OS	Linux	2009
Histar	OS	Unix like	2006
Weir	OS	Android	2016
TaintDroid	Application	Java Virtual Machine	2010
Flowcaml	Application	Ocaml compiler	2003
Jif	Application	Java compiler	2003
Raksha	CPU	Co-processor	2007
HardBlare	**CPU**	**Co-processor**	**2015**

Awareness about how information spreads can probably improve security

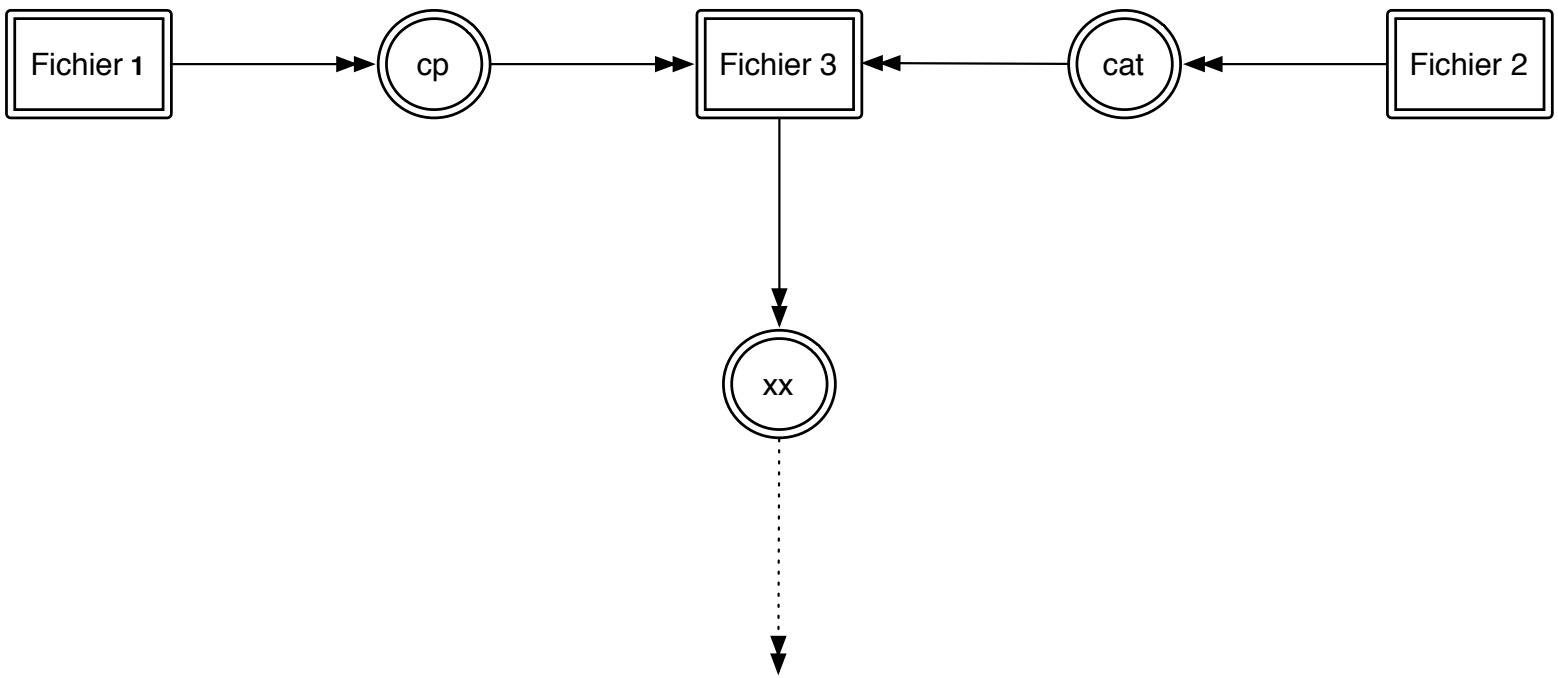
A lot of application domains but today, we focus on

- *Prevention* of unwanted information flows
- *Detection* of advanced persistant threats
- *Understanding* of malware attacks

Before addressing implementation and trust in the implementation

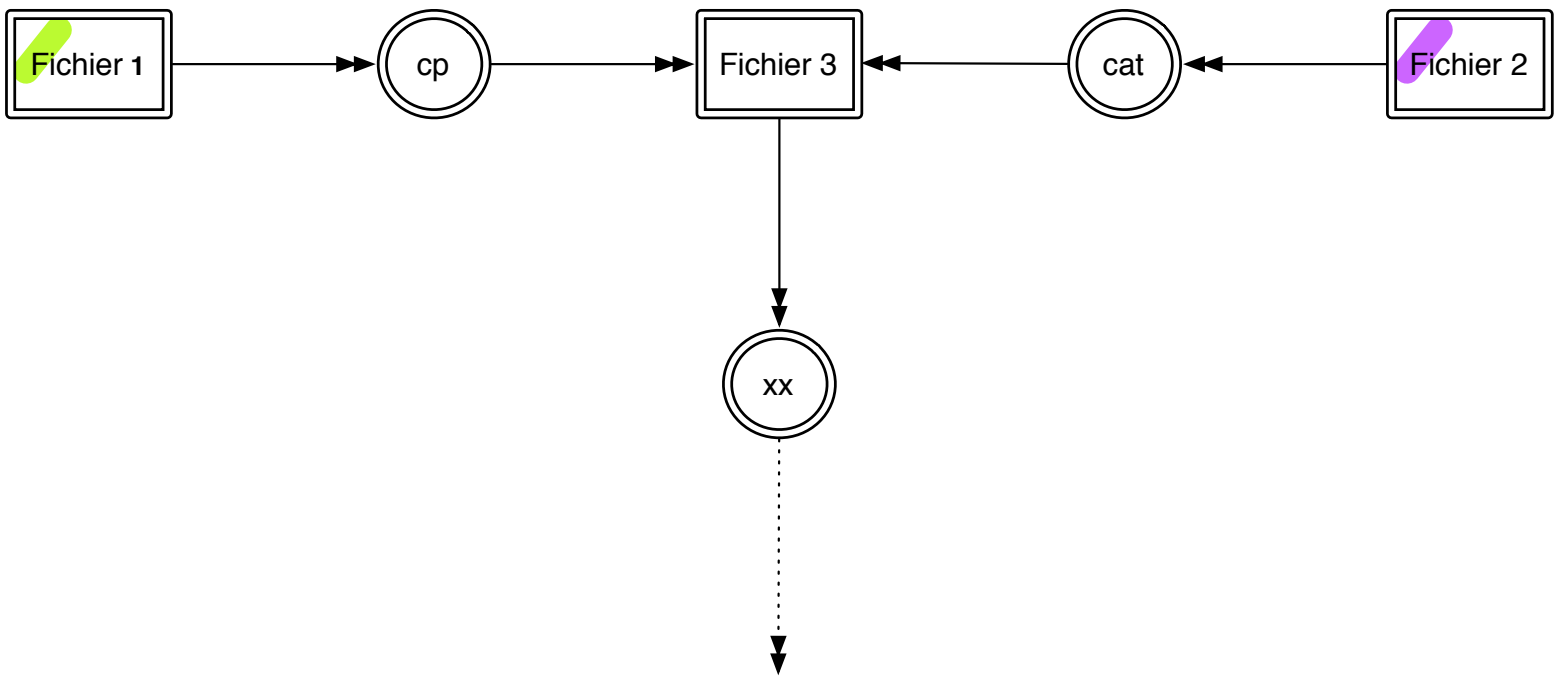
If the observation level is the operating system

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



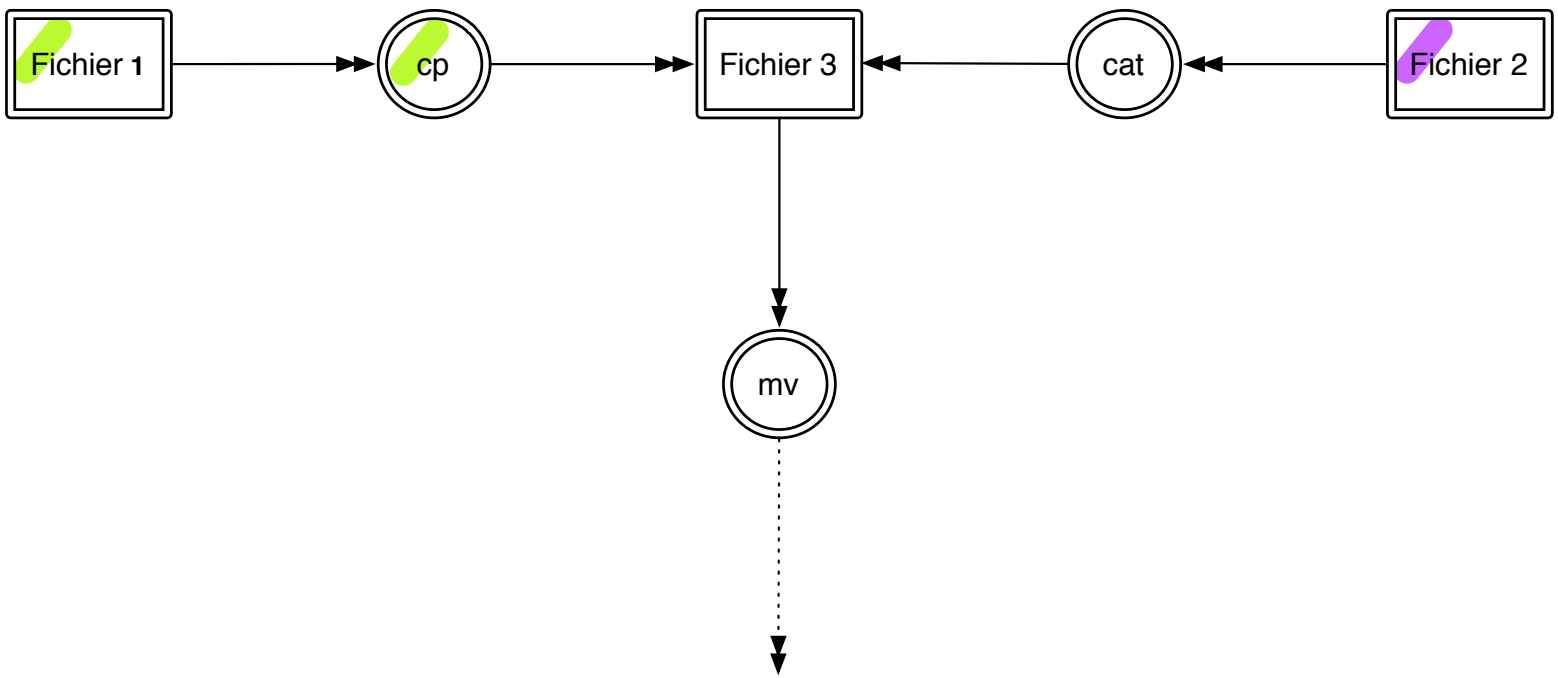
If the observation level is the operating system

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



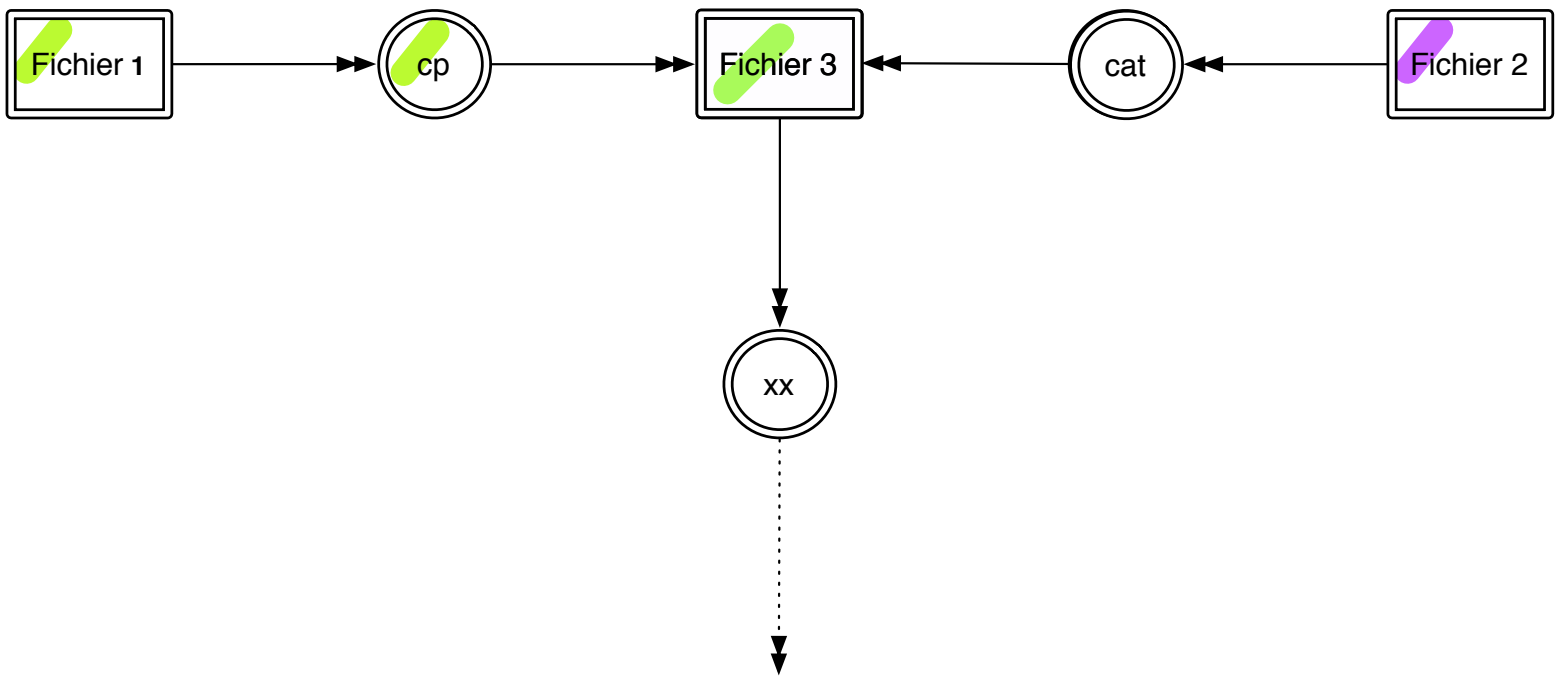
If the observation level is the operating system

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



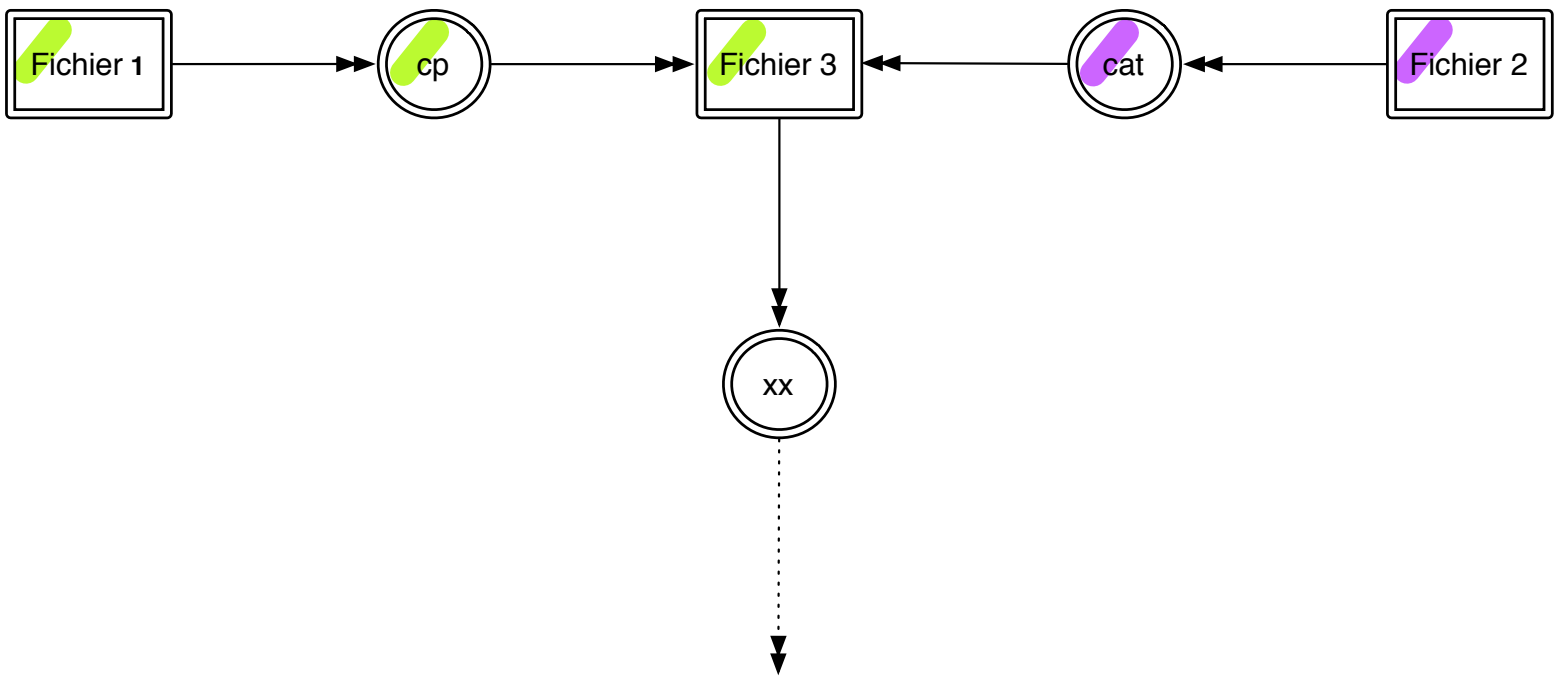
If the observation level is the operating system

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



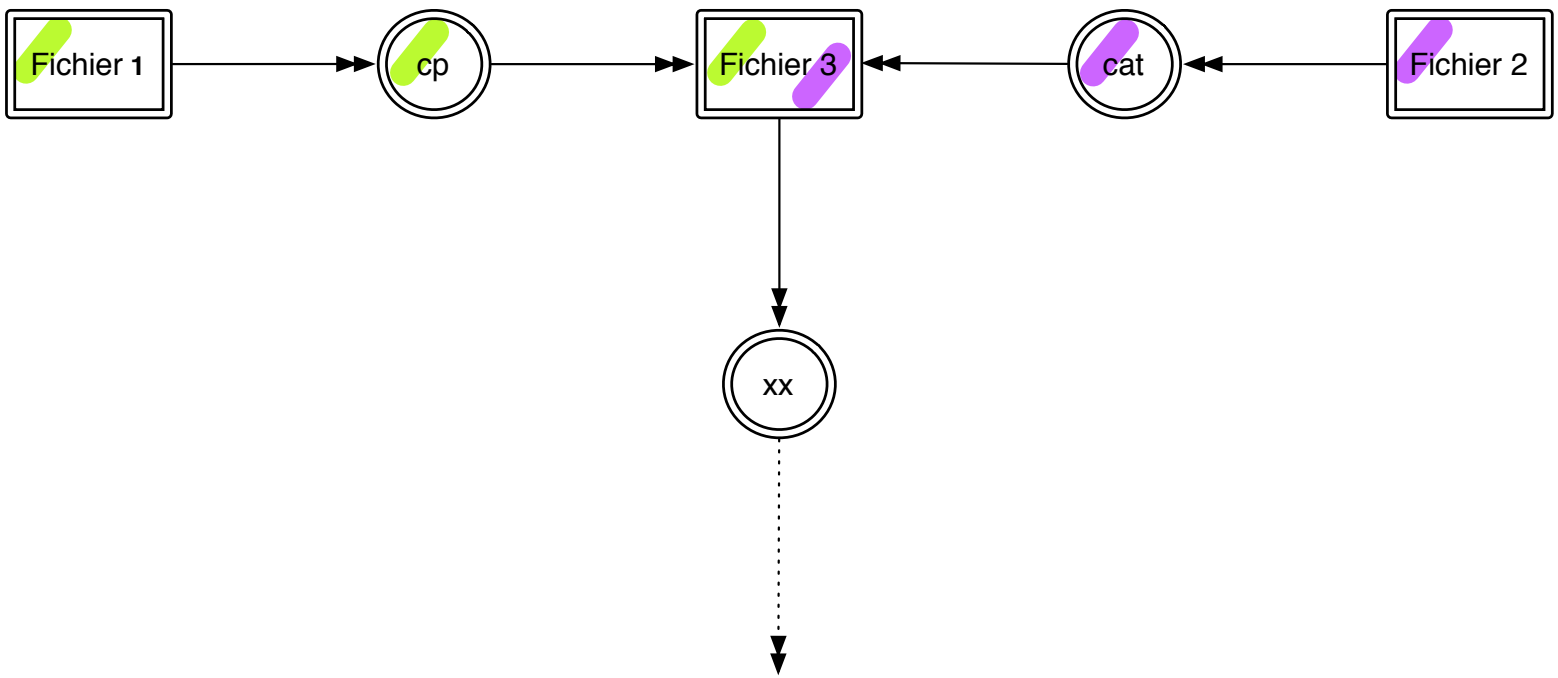
If the observation level is the operating system

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



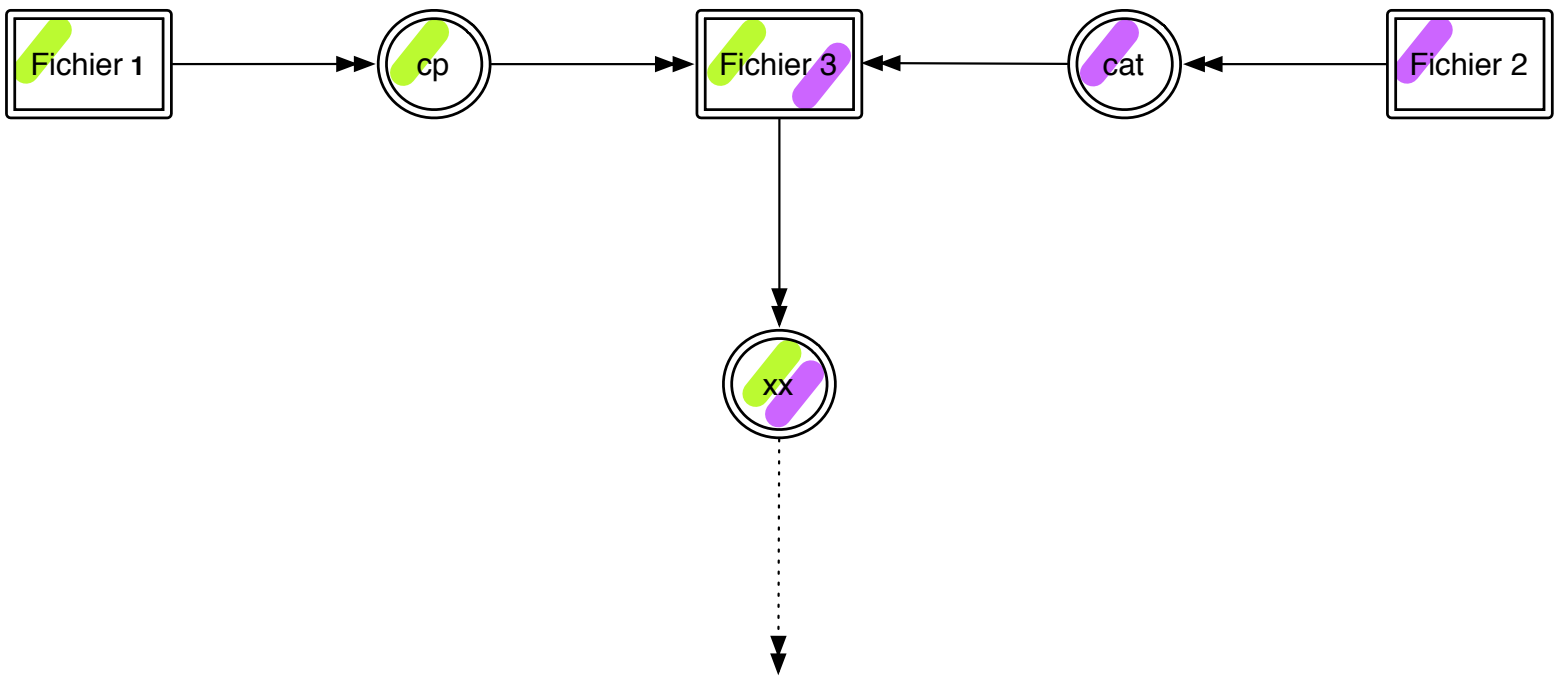
If the observation level is the operating system

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



If the observation level is the operating system

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



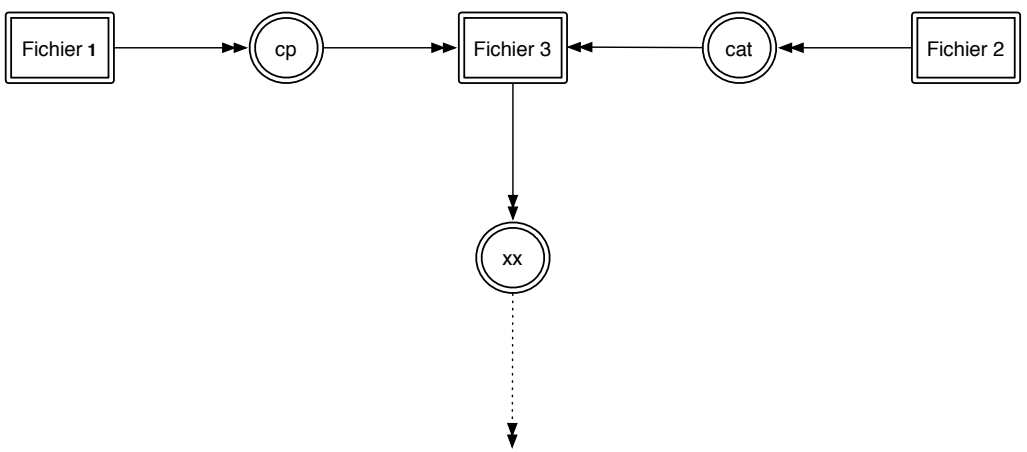
Prevention of unwanted information flows

Prevention of unwanted information flows

Information flows monitoring enables fine-grained security policies

Prevention of unwanted information flows

Information flows monitoring enables fine-grained security policies

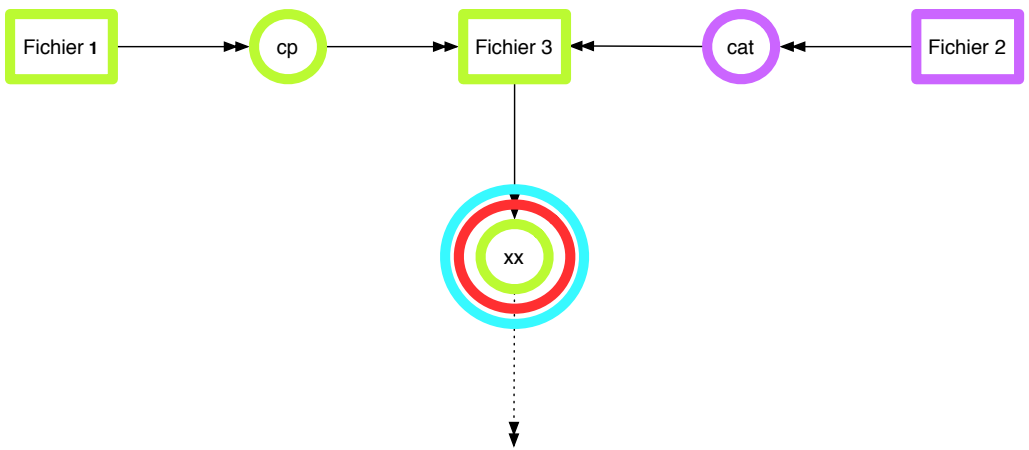


Step 1 :

Set security policy

Prevention of unwanted information flows

Information flows monitoring enables fine-grained security policies

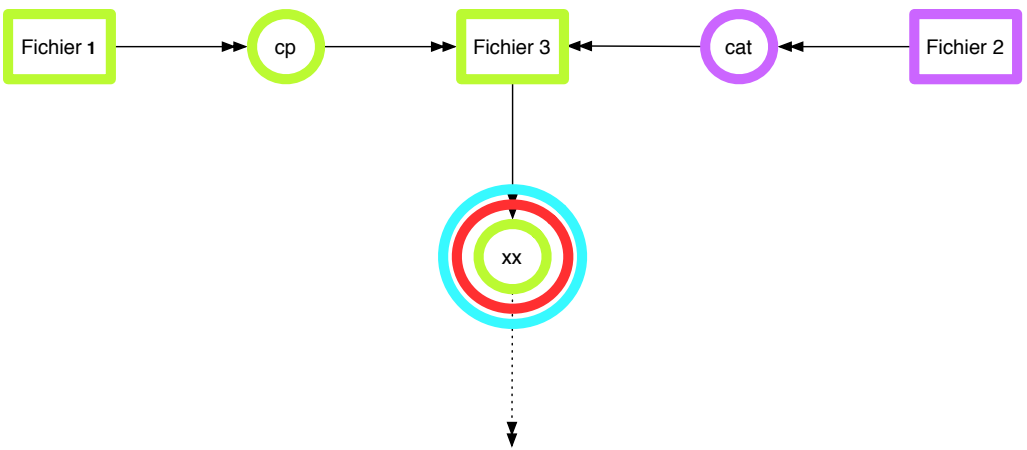


Step 1 :

Set security policy

Prevention of unwanted information flows

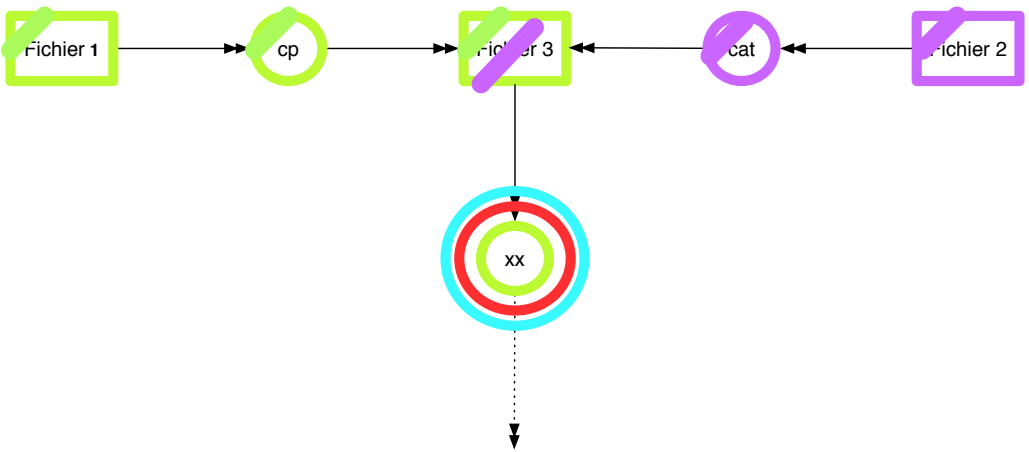
Information flows monitoring enables fine-grained security policies



Step 2 :
Observe dissemination

Prevention of unwanted information flows

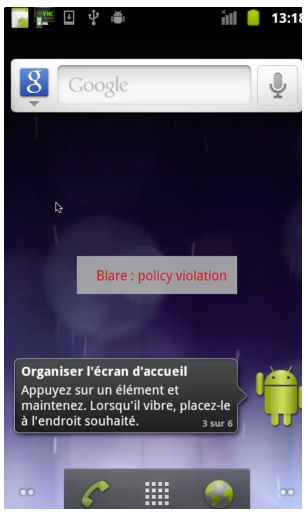
Information flows monitoring enables fine-grained security policies

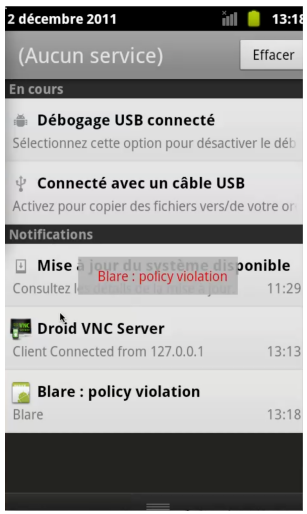


Step 3 :

Alert or block

Prevention of unwanted information flows





Conclusive approach

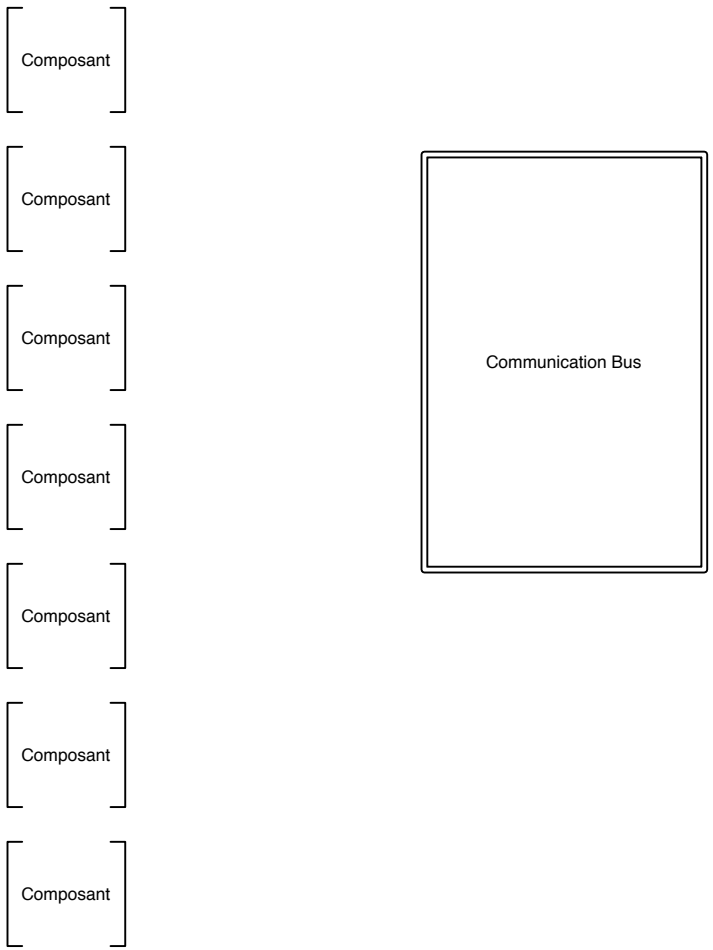
Information flow monitoring at system level allows the enforcement of fine-grained policies

Open questions

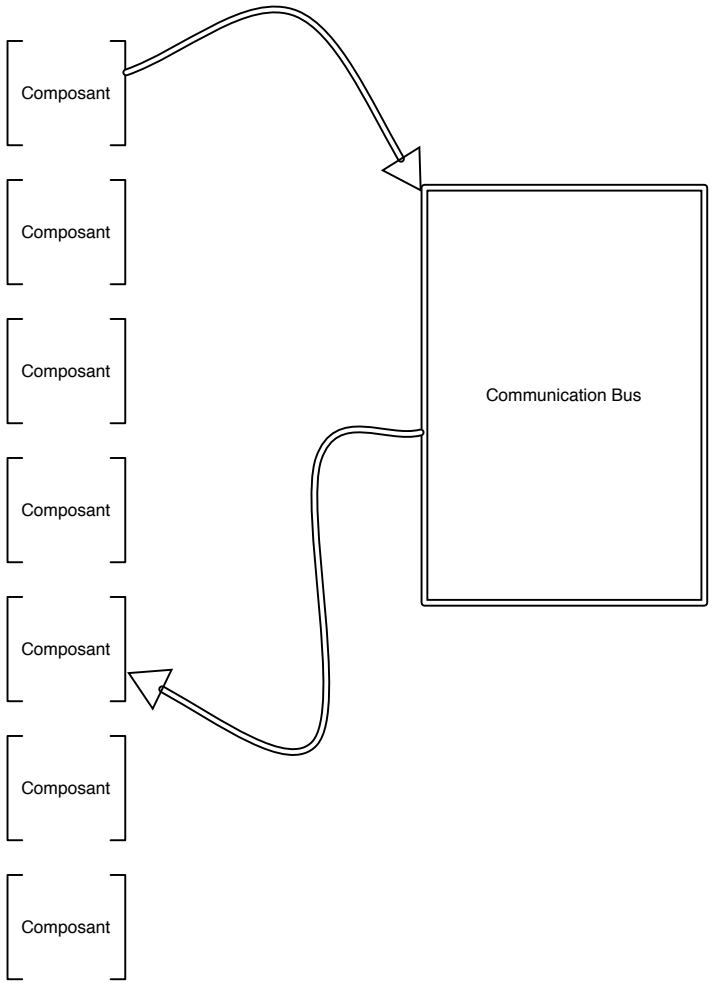
How can we cope with

- Policies specifications
- Declassification
- Taint explosion
- Bus communication

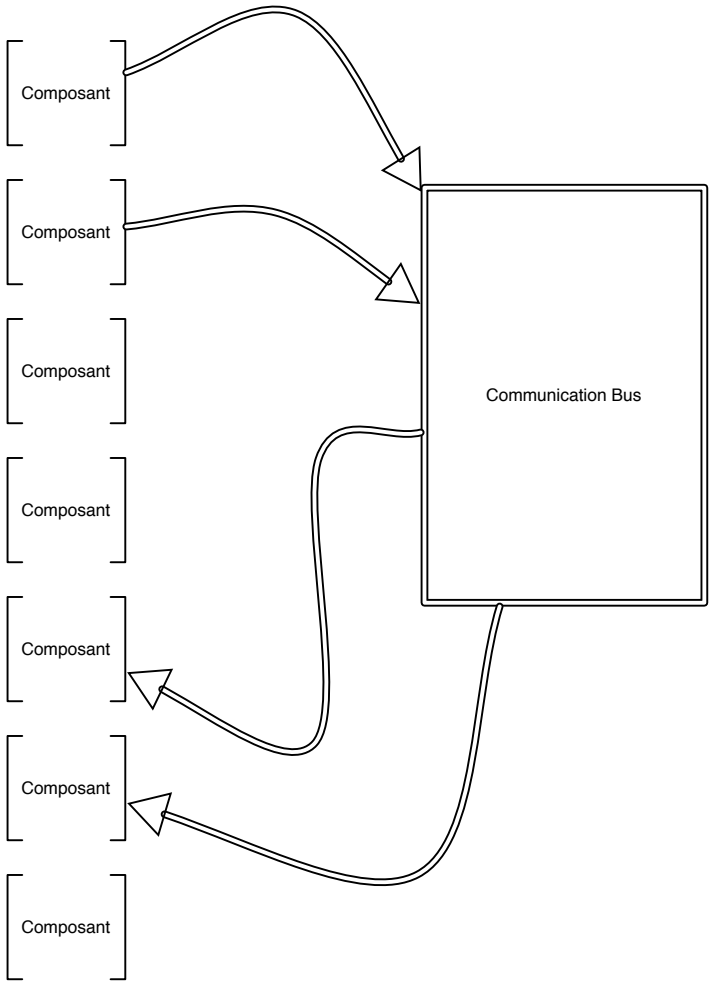
The specific problem of bus communication



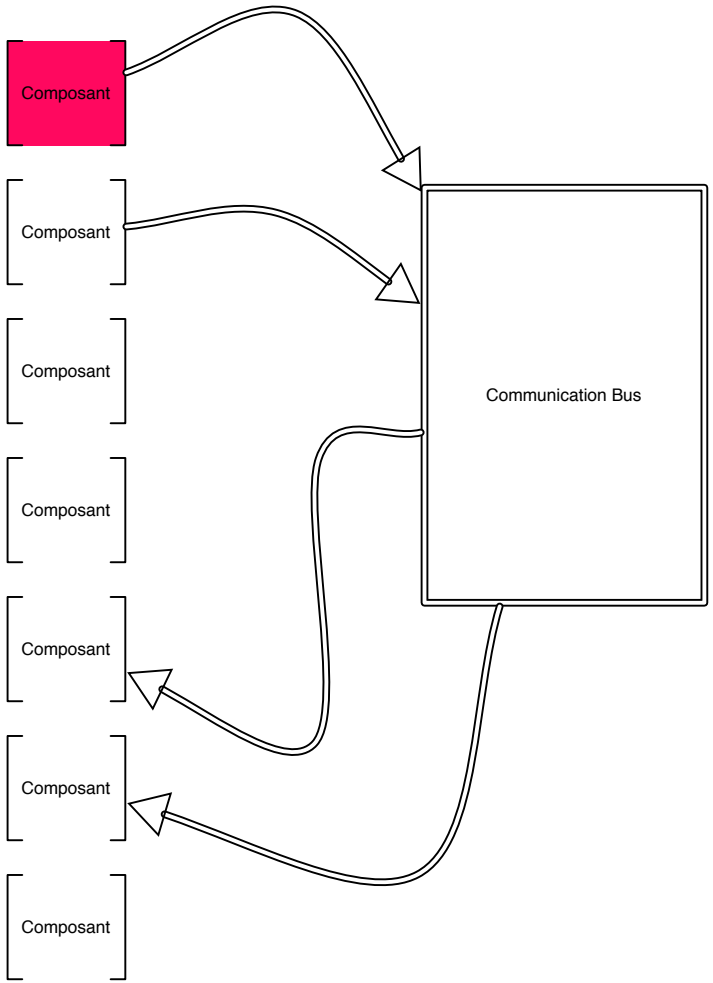
The specific problem of bus communication



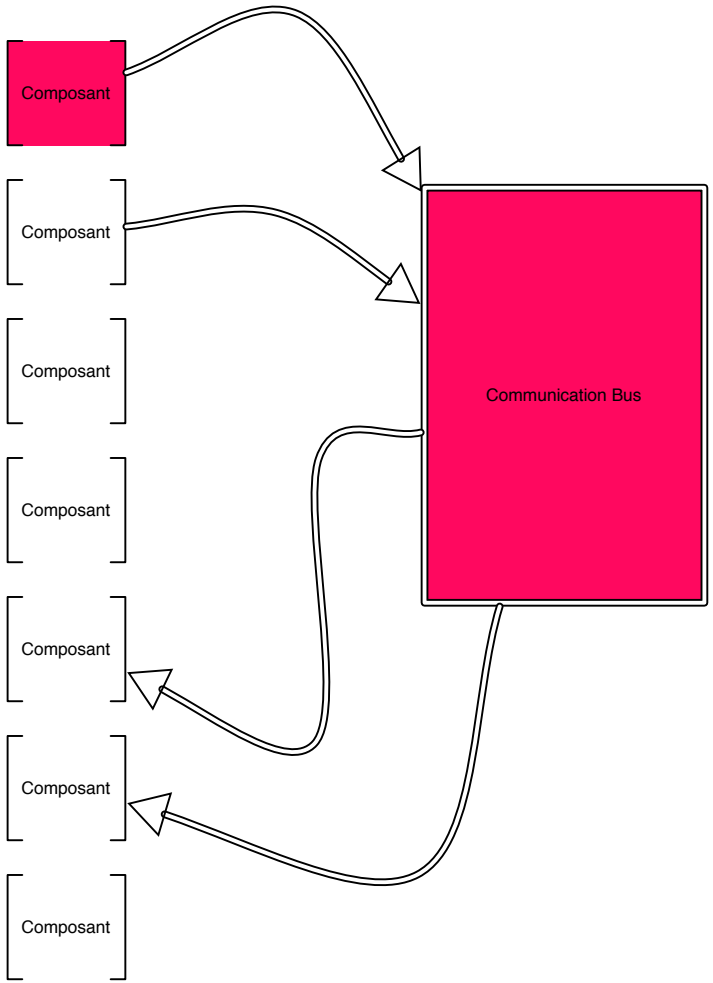
The specific problem of bus communication



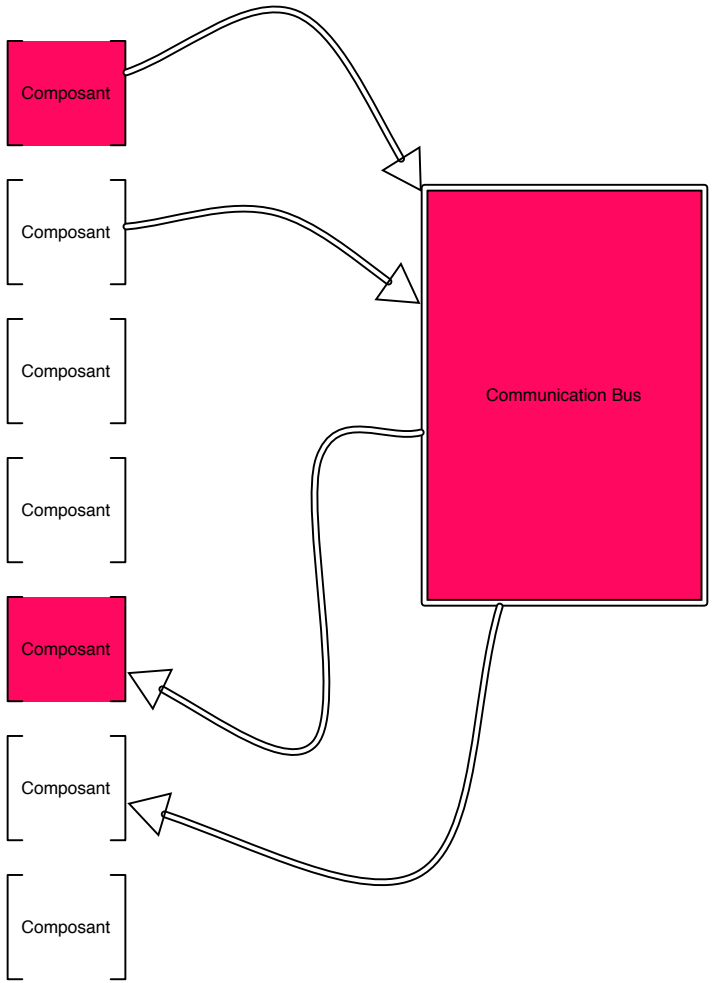
The specific problem of bus communication



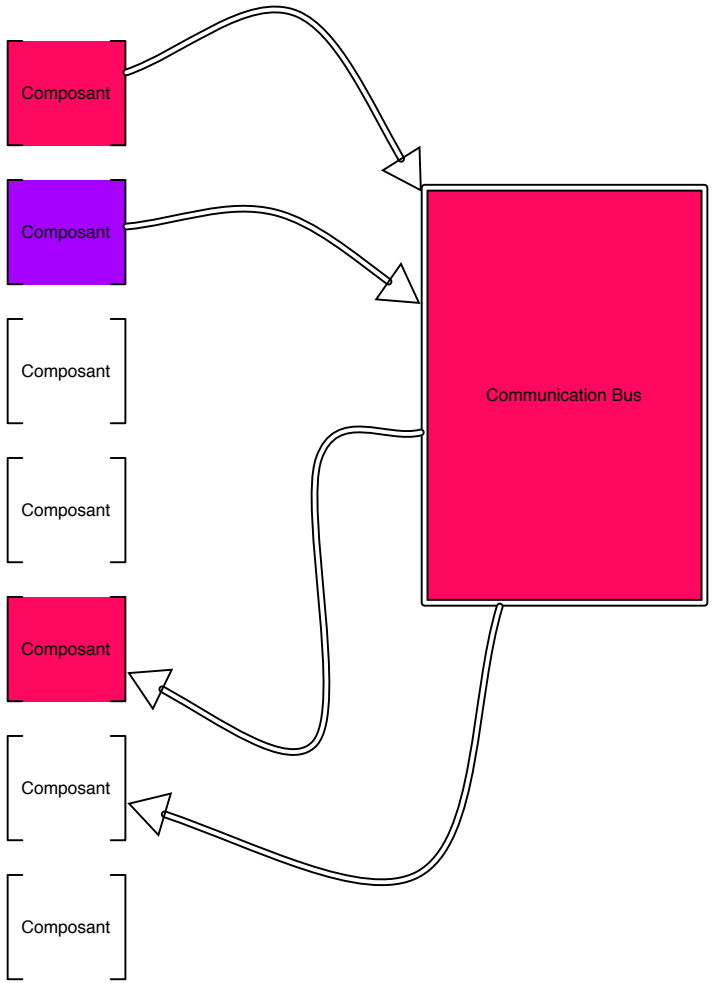
The specific problem of bus communication



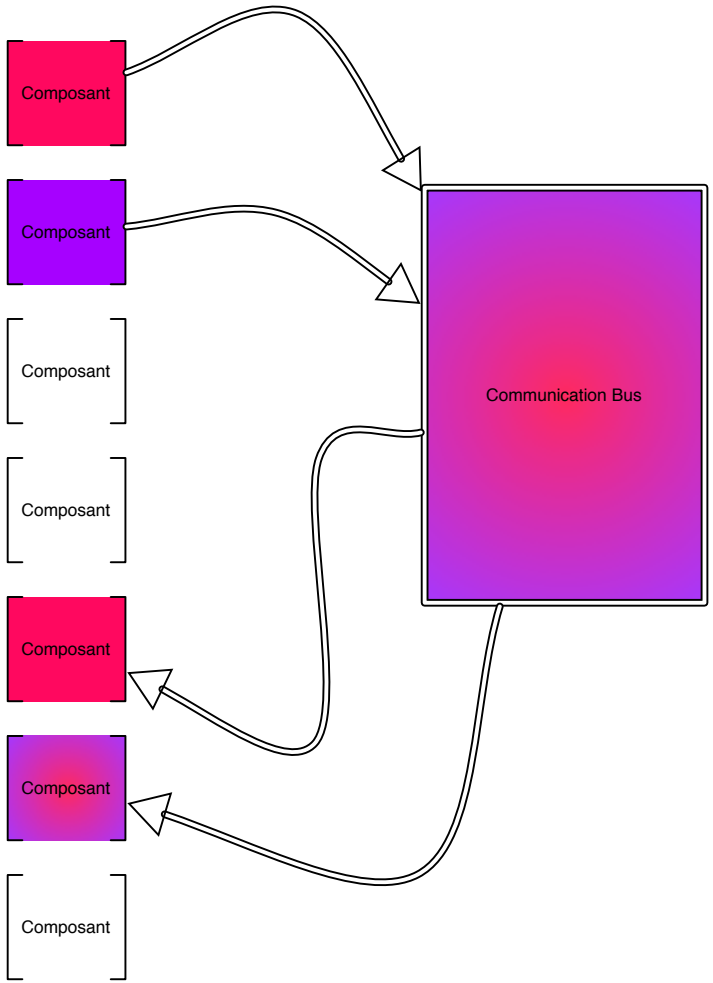
The specific problem of bus communication



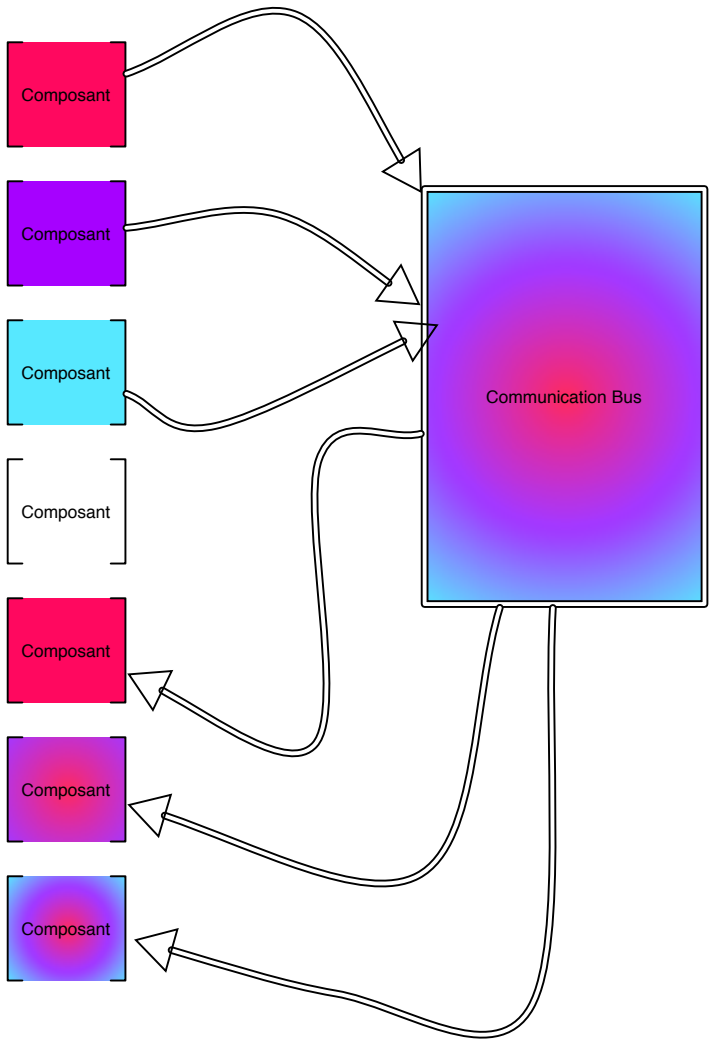
The specific problem of bus communication



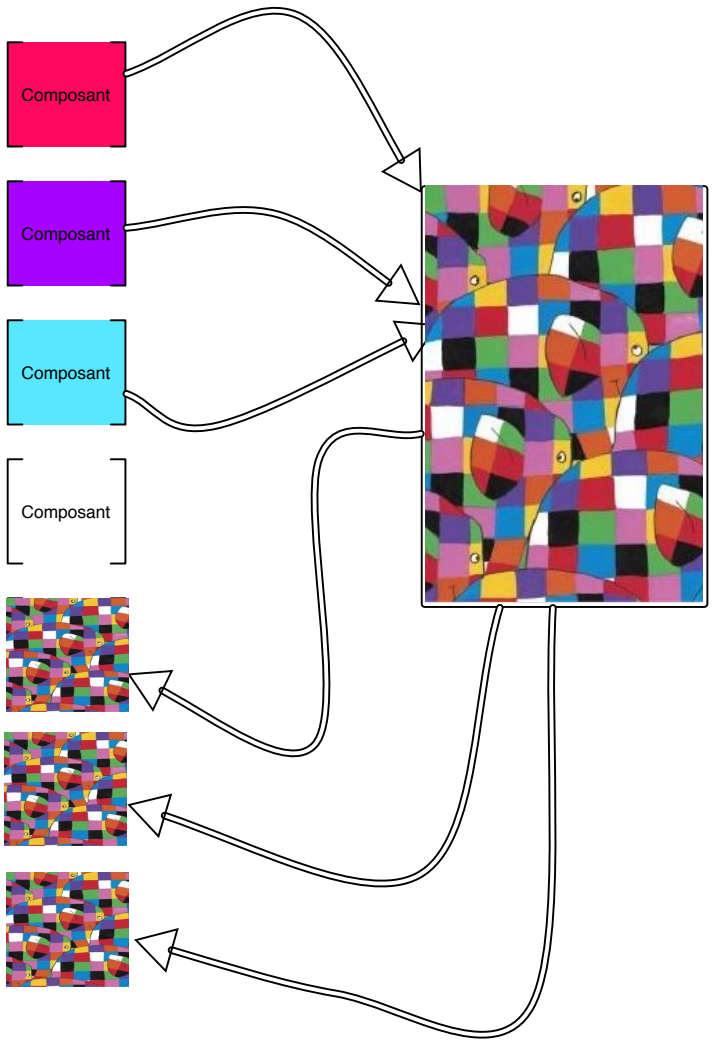
The specific problem of bus communication



The specific problem of bus communication



The specific problem of bus communication



Detection of Advanced Persistant Threats

work explored with G.Brogi during its Phd

Looking for a needle in a haystack

Advanced persistant threats (APT)

APT are attacks campaigns

- customed for the target,
- long term (months or years),
- that can rely on zero-day attacks.

Information flows highlight links between singles attacks

Use case : a network under supervision of IDS

Starting observation

Example of an APT against a workstation

1. In August, the user opens an infected PDF (spearphishing), which launches a very simple RAT
2. The attacker immediately uses this simple RAT to install a full-featured RAT and start looking for a vulnerability
3. In November, the attacker uses the second RAT to exploit the vulnerability they found and elevate their privileges
4. Attacker installs a third RAT with elevated privileges and persistence

Starting observation

Each attack sets up the next

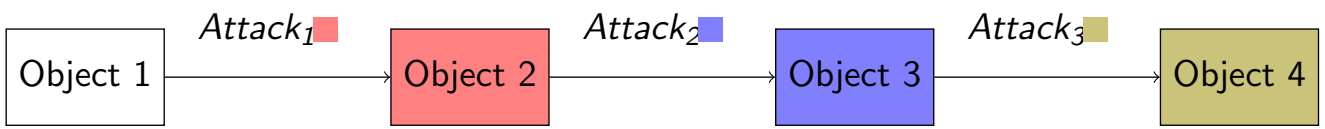
The attacker uses the tools set up during one attack to execute the next one

Information flows between theses tools betray the attacker

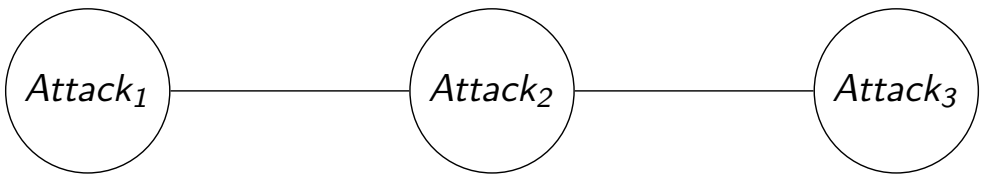
Information flows permits to reconstruct attacks chain

Highlighting attacks chains ?

Object-level view of links between attacks



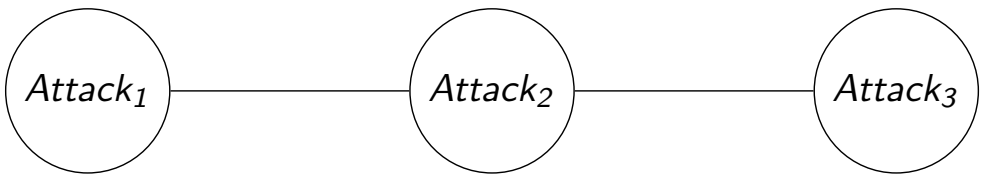
Results may differ from



Highlighting attacks chains ?

Object-level view of links between attacks

Results may differ from



Starting observation

A promising research track

- good first results but hard to evaluate
- subject to false positive

Understanding malware attacks

[The Cominlabs Kharon project] joint work with Jean François Lalande and Thomas Genet

Understanding malware attacks

Information flows are valuable clues in malware investigation

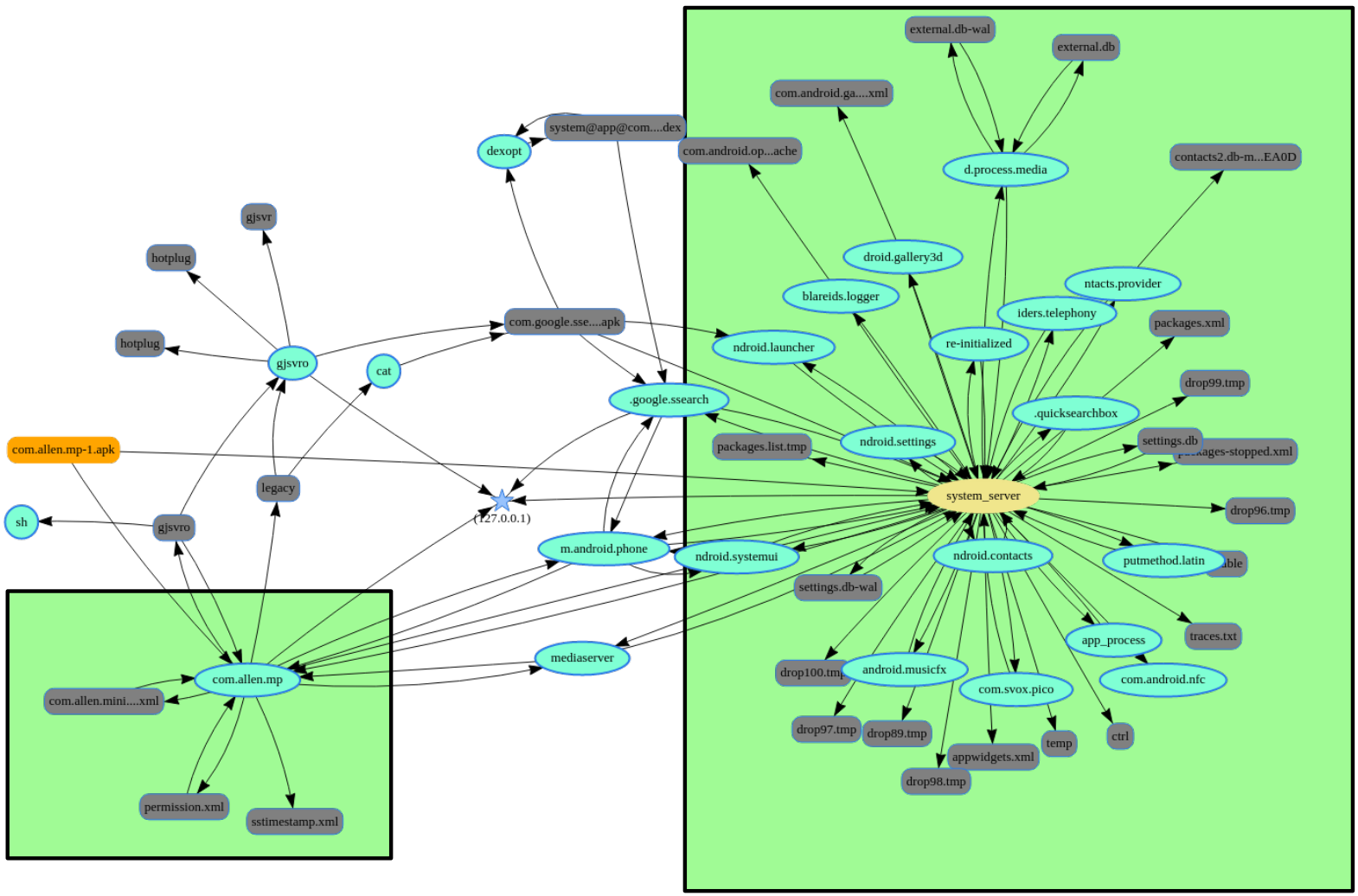
Overview

- The observation level is the Android operating system
- Only the `.apk` file is marked
- No security policy
- **Requires the execution of the malware**

Understanding malware attacks

```
6>[  125.155895] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.156034] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.157058] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.157214] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.157360] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.157694] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.157840] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.157979] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.158158] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.158313] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.158452] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.158632] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.158790] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.158963] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.159110] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.159248] [BLARE_POLICY_VIOLATION] file /data/data/com.android.providers.settings/databases/settings.db-journal 8153
6>[  125.163374] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.164408] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.164598] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.164822] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.164999] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.165177] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.168191] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.170381] [BLARE_POLICY_VIOLATION] file /data/data/com.android.providers.settings/databases/settings.db-journal 8153
6>[  125.170520] [BLARE_POLICY_VIOLATION] file /data/data/com.android.providers.settings/databases/settings.db 81530 > proc
6>[  125.170741] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.170918] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.171060] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.171241] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.171389] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.171530] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.171713] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.171857] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.171996] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.172176] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.172332] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.172471] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.172652] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.172809] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
6>[  125.172990] [BLARE_POLICY_VIOLATION] process system_server:android.bg 767 > file /data/data/com.android.providers.sett
```

Understanding malware attacks



Understanding malware attacks

Understanding malware attacks

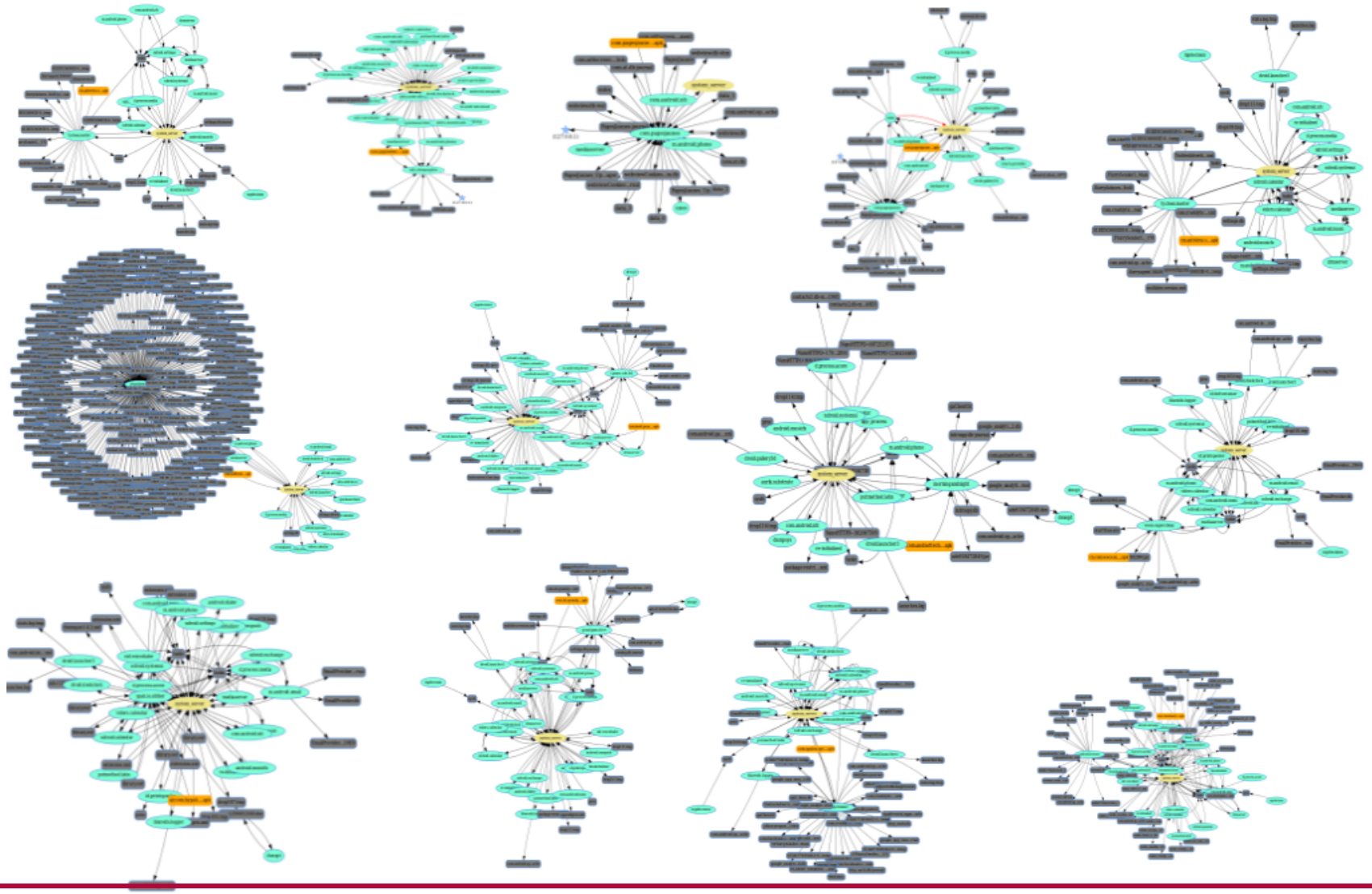
Information flows easily unmask malware behaviors

Challenges explored in the Kharon CominLabs project

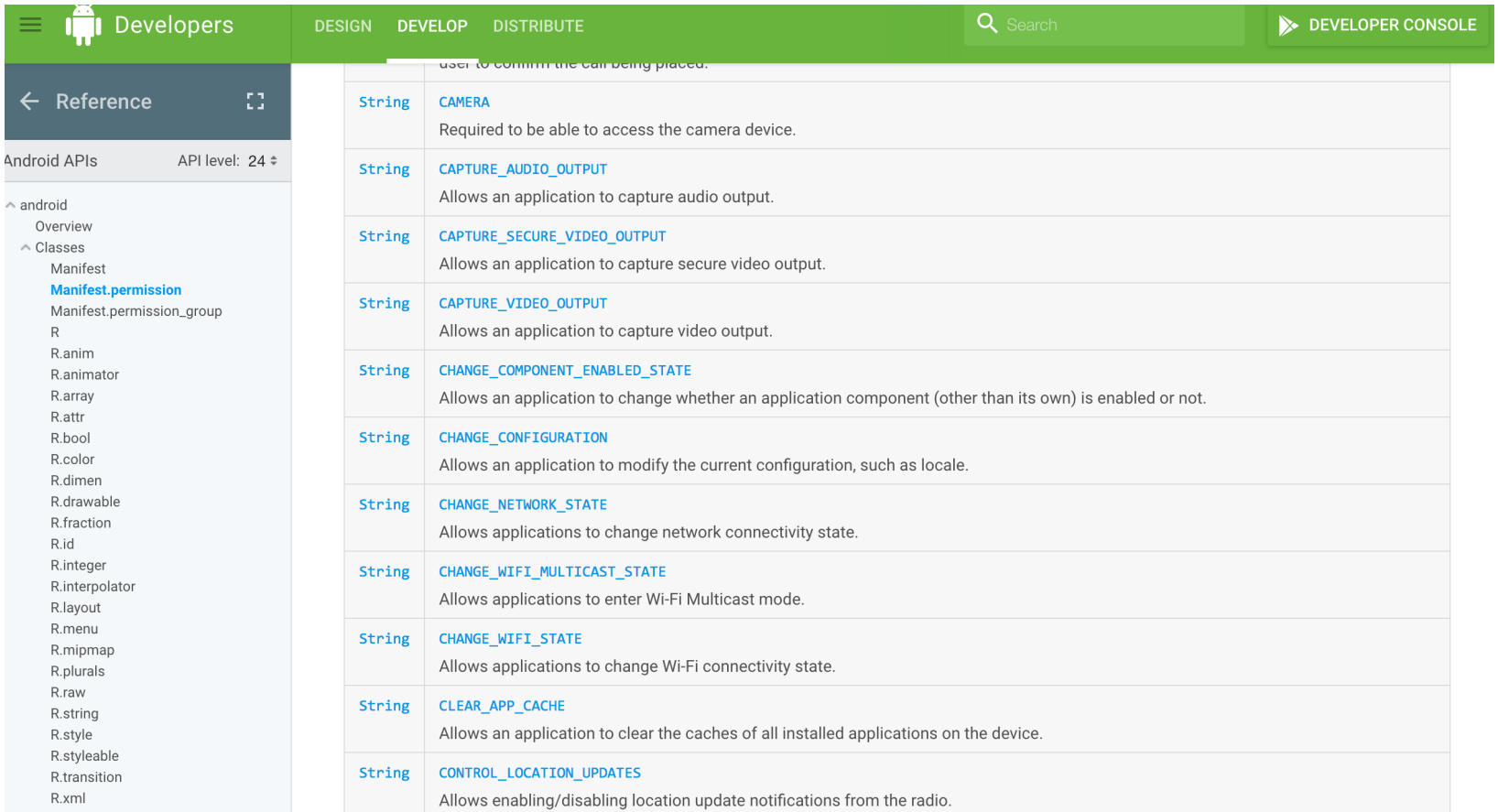
1. Automatic triggering with GroddDroid
   - prior static analysis
   - light malware alteration
   - systematic UI exploration
2. Model of normal information flows
3. Depacking

kharon.gforge.inria.fr *website project*
kharon.irisa.fr *online platform*
hosted by the High Security Laboratory (LHS)

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



We study the possible Android permissions (> 140)



A model in 4 parts

An Android application is allowed to

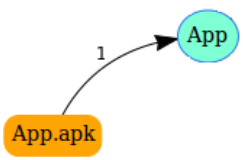
1. be executed
2. read/write files and sockets
3. install package
4. request services

App.apk

A model in 4 parts

An Android application is allowed to

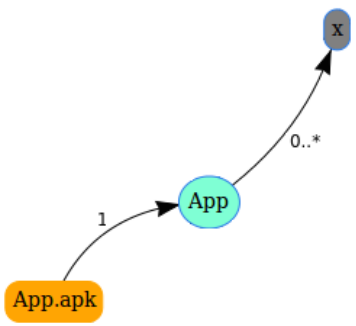
1. be executed
2. read/write files and sockets
3. install package
4. request services



A model in 4 parts

An Android application is allowed to

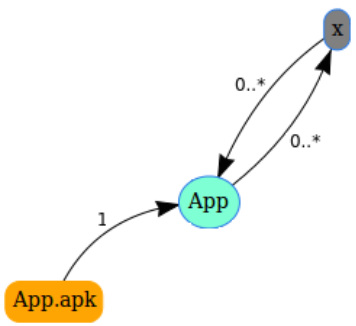
1. be executed
2. read/write files and sockets
3. install package
4. request services



A model in 4 parts

An Android application is allowed to

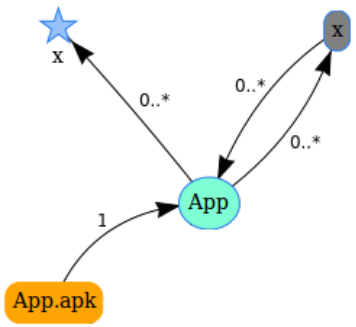
1. be executed
2. read/write files and sockets
3. install package
4. request services



A model in 4 parts

An Android application is allowed to

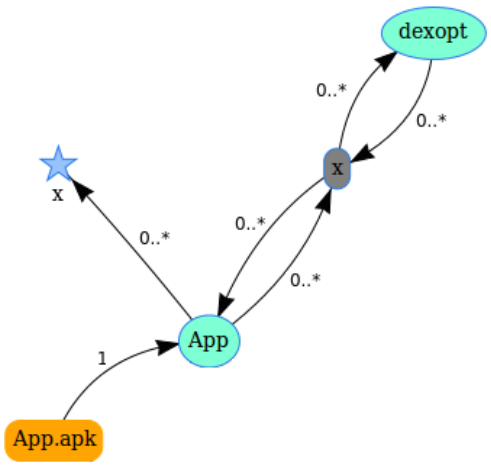
1. be executed
2. read/write files and sockets
3. install package
4. request services



A model in 4 parts

An Android application is allowed to

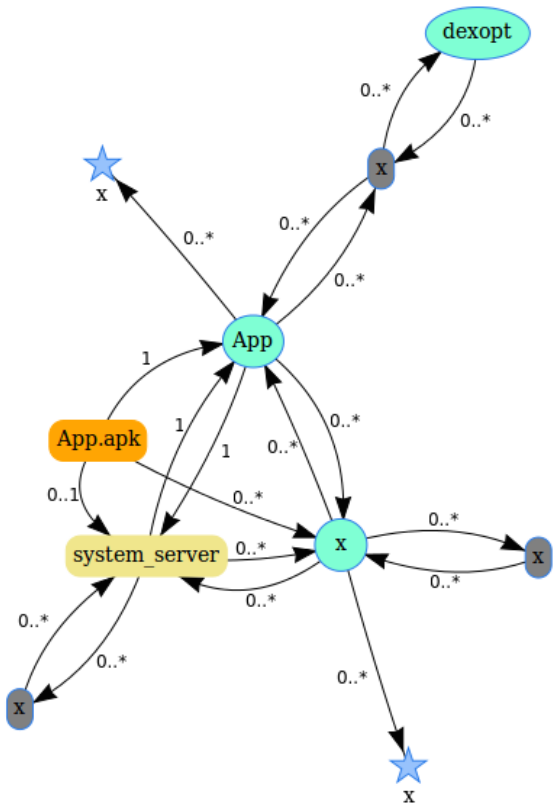
1. be executed
2. read/write files and sockets
3. install package
4. request services



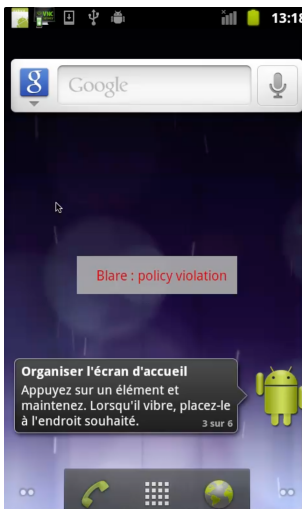
A model in 4 parts

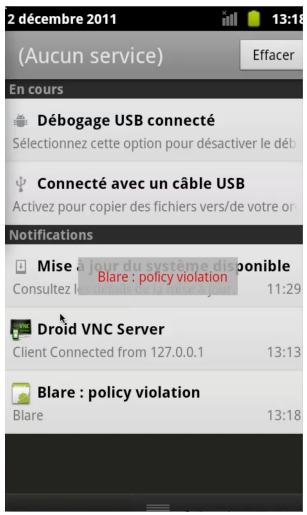
An Android application is allowed to

1. be executed
2. read/write files and sockets
3. install package
4. request services



Understanding malware attacks





Conclusive approach

Information flow monitoring allows to quickly understand an attack

Open questions

- Automatic triggering
- Packers
- Behavioral obfuscation

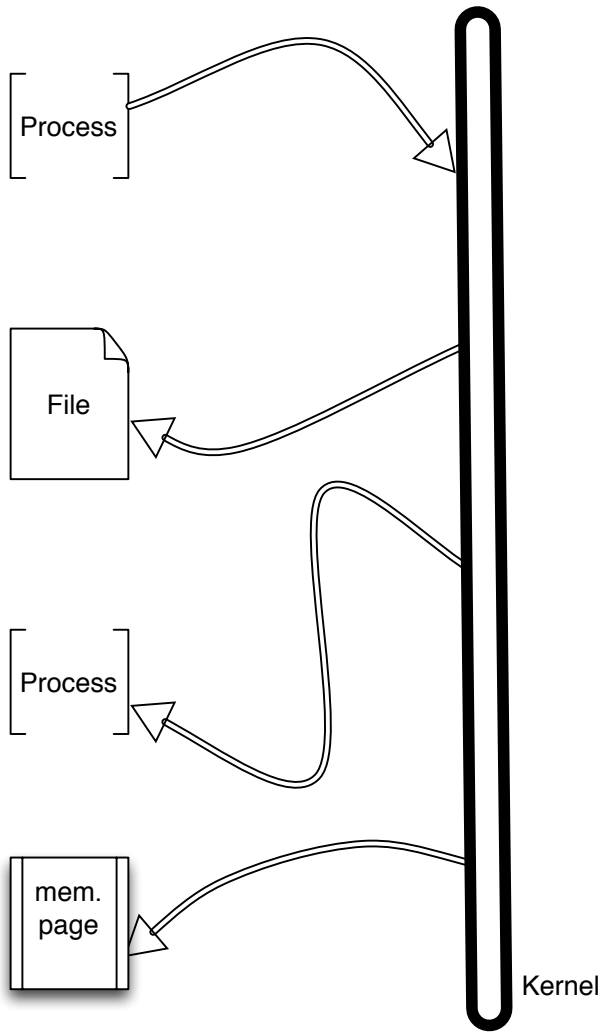
Implementation of information flow monitors

[Phd Laurent Georget 2018] joint work with M.Jaume, F.Tronel, G.Piolle

Previous works make sense if ..

You trust your information flow monitor

Previous works make sense if ..



At the OS level, flows are performed by the kernel

Existing information flow monitors rely on LSM

The Linux Security Modules kernel subsystem

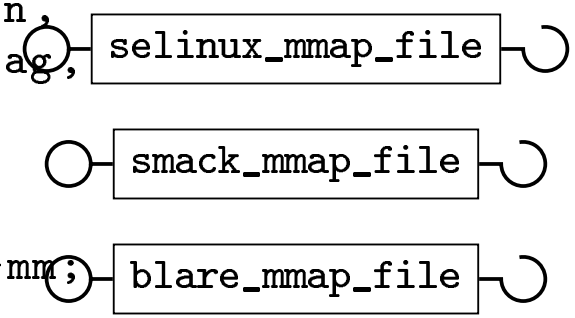
- adds security fields to kernel data structures
- inserts calls to hook functions to manage the security fields

Linux Security Module

```
1  unsigned long vm_mmap_pgoff(struct file *file,
   unsigned long addr, unsigned long len,
   unsigned long prot, unsigned long flag,
   unsigned long pgoff)
5  {
       unsigned long ret;
       struct mm_struct *mm = current->mm;
       unsigned long populate;
       ret = security_mmap_file(file, prot, flag);
10     if (!ret) {
           if (down_write_killable(&mm->mmap_sem))
               return -EINTR;
           ret = do_mmap_pgoff(file, addr, len, prot,
                   flag, pgoff, &populate);
15         up_write(&mm->mmap_sem);
           if (populate)
               mm_populate(ret, populate);
       }
       return ret;
20 }
```

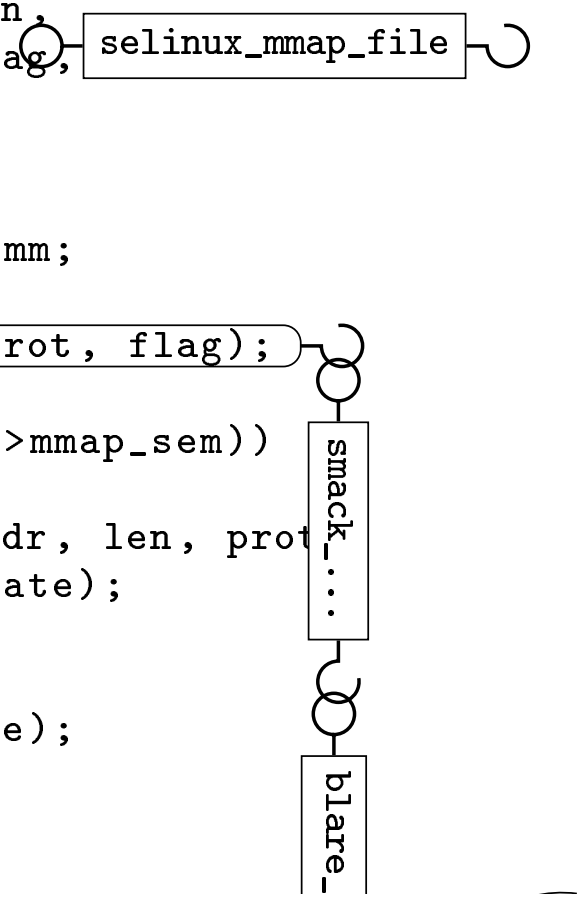
Linux Security Module

```
1  unsigned long vm_mmap_pgoff(struct file *file,
   unsigned long addr, unsigned long len,
   unsigned long prot, unsigned long flag,
   unsigned long pgoff)
5  {
       unsigned long ret;
       struct mm_struct *mm = current->mm;
       unsigned long populate;
       ret = security_mmap_file(file, prot, flag);
10     if (!ret) {
           if (down_write_killable(&mm->mmap_sem))
               return -EINTR;
           ret = do_mmap_pgoff(file, addr, len, prot,
                   flag, pgoff, &populate);
15         up_write(&mm->mmap_sem);
           if (populate)
               mm_populate(ret, populate);
       }
       return ret;
20 }
```

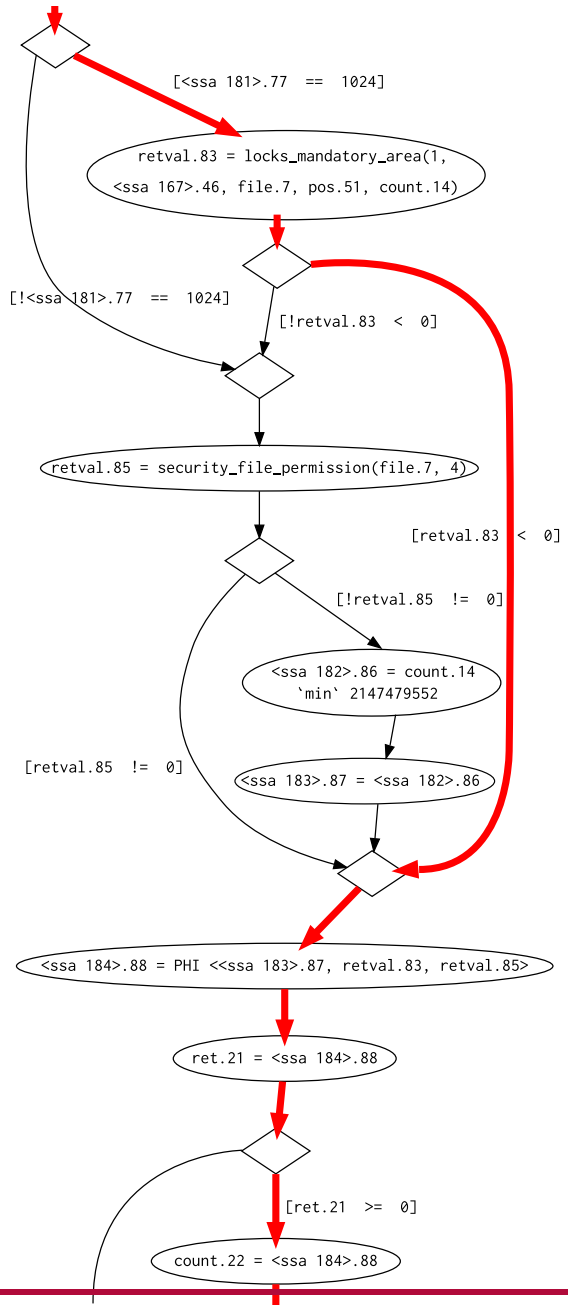


Linux Security Module

```
1  unsigned long vm_mmap_pgoff(struct file *file,
   unsigned long addr, unsigned long len,
   unsigned long prot, unsigned long flag,
   unsigned long pgoff)
5  {
       unsigned long ret;
       struct mm_struct *mm = current->mm;
       unsigned long populate;
       ret = security_mmap_file(file, prot, flag);
10     if (!ret) {
           if (down_write_killable(&mm->mmap_sem))
               return -EINTR;
           ret = do_mmap_pgoff(file, addr, len, prot,
                     flag, pgoff, &populate);
15         up_write(&mm->mmap_sem);
           if (populate)
               mm_populate(ret, populate);
       }
       return ret;
20 }
```



An information flow monitor is only aware of flows captured by a LSM hook, and LSM has been developed for access control.



Can we trust Blare ? [Phd Georget'17]

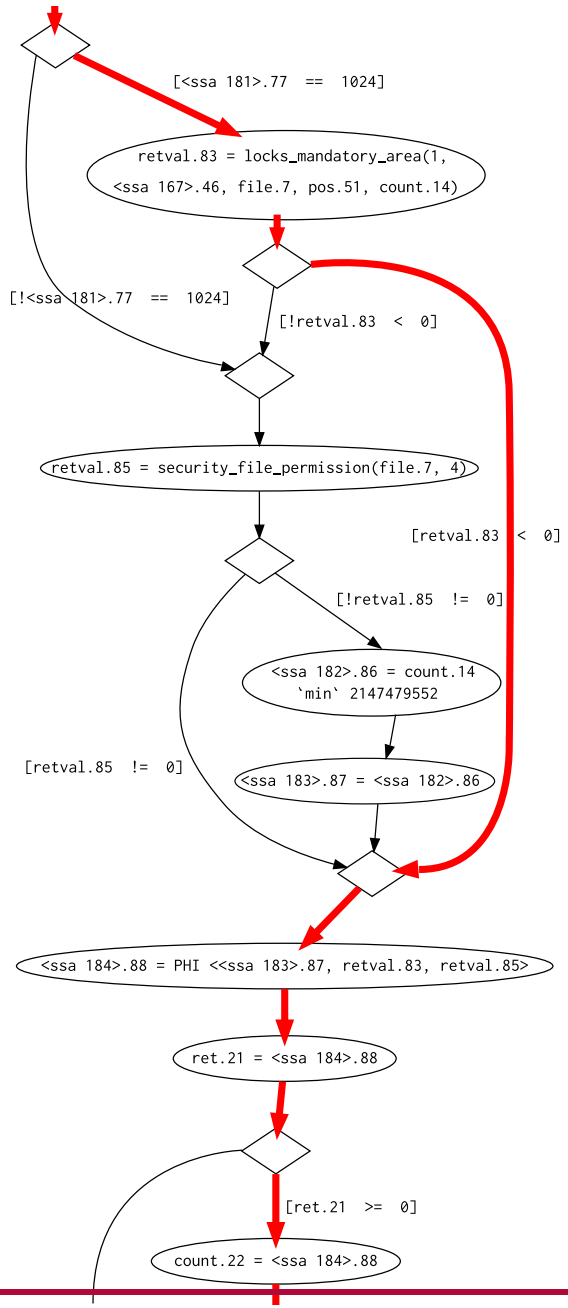
1. Can LSM hook be evaded ?

   Yes . . . Answer obtained by static analysis of the code of syscalls
   $\longrightarrow$ minor modifications of LSM

2. Can such monitors cope with concurrent flows ?

   Attacks exist on monitors as Blare, Weir, Laminar
   $\longrightarrow$ New propagation algorithm proved using Coq
   implemented in RfBlare

   [Best Paper SEFM'17]



Can we trust Blare ? [Phd Georget'17]

1. Can LSM hook be evaded ?

   Yes . . . Answer obtained by static analysis of the code of syscalls

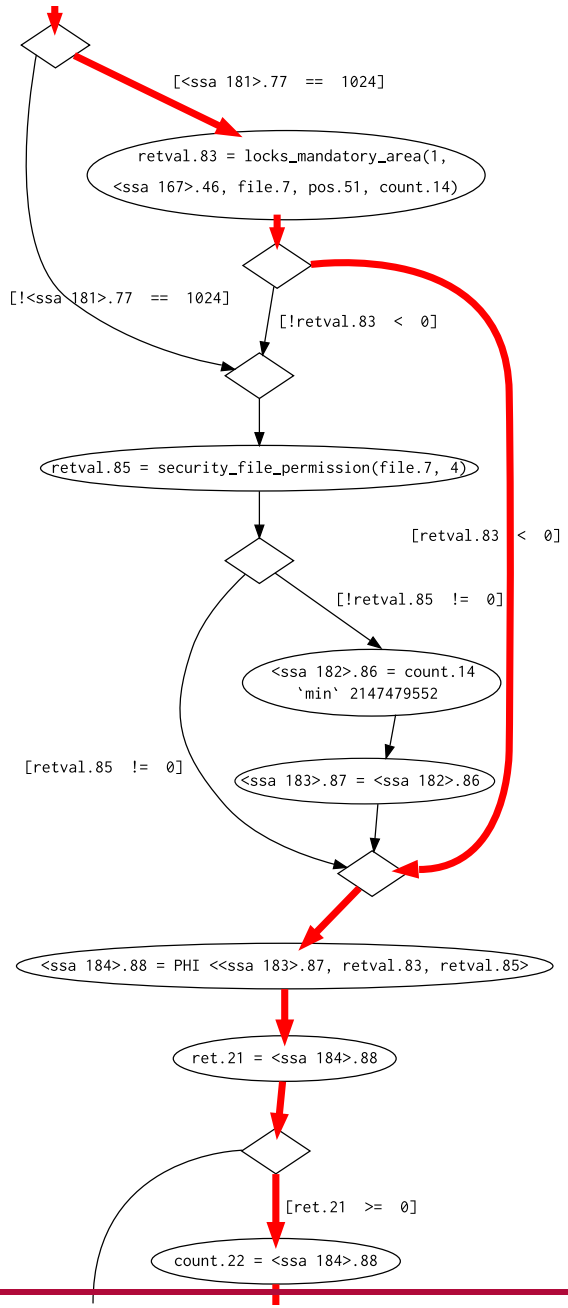
   ⟶ minor modifications of LSM

2. Can such monitors cope with concurrent flows ?

   Attacks exist on monitors as Blare, Weir, Laminar

   ⟶ New propagation algorithm proved using Coq implemented in RfBlare

   [Best Paper SEFM'17]



Can we trust Blare ? [Phd Georget'17]

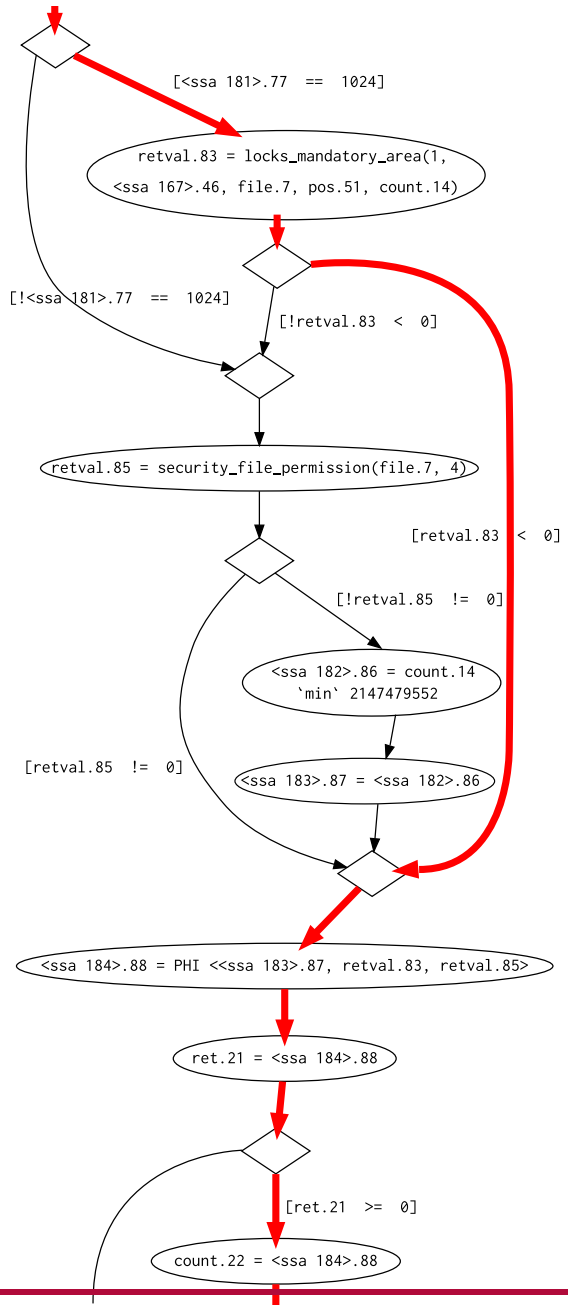
1. Can LSM hook be evaded ?

   Yes . . . Answer obtained by static analysis of the code of syscalls
   $\longrightarrow$ minor modifications of LSM

2. Can such monitors cope with concurrent flows ?

   Attacks exist on monitors as Blare, Weir, Laminar
   $\longrightarrow$ New propagation algorithm proved using Coq implemented in RfBlare

   [Best Paper SEFM'17]



Can we trust Blare ? [Phd Georget'17]

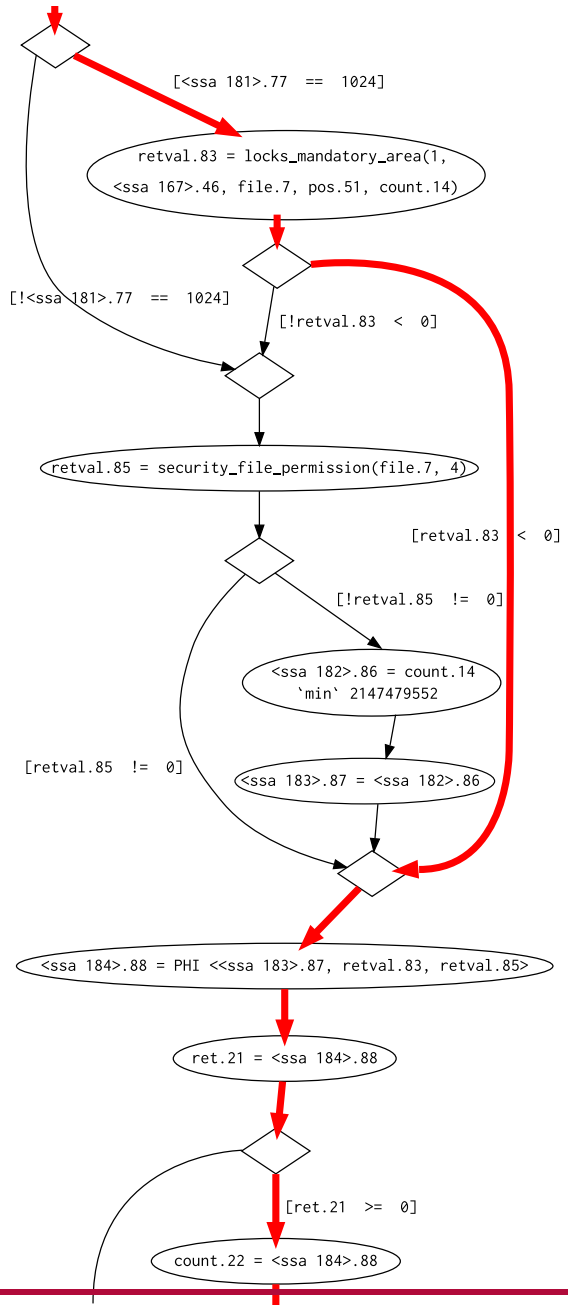
1. Can LSM hook be evaded ?

   Yes . . . Answer obtained by static analysis of the code of syscalls
   $\longrightarrow$ minor modifications of LSM

2. Can such monitors cope with concurrent flows ?

   Attacks exist on monitors as Blare, Weir, Laminar
   $\longrightarrow$ New propagation algorithm proved using Coq implemented in RfBlare

   [Best Paper SEFM'17]



Can we trust Blare ? [Phd Georget'17]

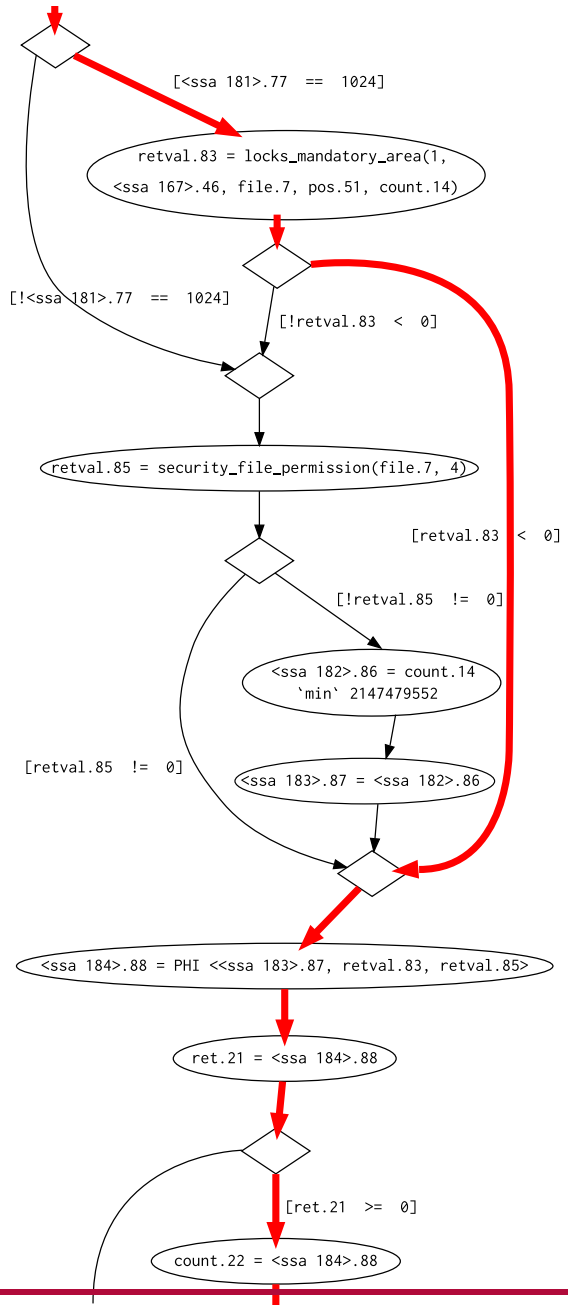
1. Can LSM hook be evaded ?

   Yes . . . Answer obtained by static analysis of the code of syscalls
   $\longrightarrow$ minor modifications of LSM

2. Can such monitors cope with concurrent flows ?

   Attacks exist on monitors as Blare, Weir, Laminar
   $\longrightarrow$ New propagation algorithm proved using Coq implemented in RfBlare

   [Best Paper SEFM'17]



Can we trust Blare ? [Phd Georget'17]

1. Can LSM hook be evaded ?

   Yes . . . Answer obtained by static analysis of the code of syscalls
   $\longrightarrow$ minor modifications of LSM

2. Can such monitors cope with concurrent flows ?

   Attacks exist on monitors as Blare, Weir, Laminar
   $\longrightarrow$ New propagation algorithm proved using Coq implemented in RfBlare

   [Best Paper SEFM'17]

Conclusion

Information flow monitoring for security

Pros

Information flow monitoring offers multiple ways to secure a system : detect, prevent and understand unwanted behaviors.

Cons

- accurate but tedious to configure
- accurate but need visualization tools
- requires changes in the core of the systems

Future works

- Exploration of new observation levels
- Guidelines for declassification
- Arithmetic of taints