Triggering and monitoring suspicious execution in Android applications

T. Genet, J.F. Lalande, M. Leslous, Valérie Viet Triem Tong

CentraleSupelec
Cidre project INRIA/CentraleSupelec/CNRS/University of Rennes 1

February 4th, 2016
This talk focuses on **Security of Android devices**

- 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
How to get rid of these malware?

We want to detect / classify / recognize Android malware
- using dynamic analysis (on a real device)
- with a short a precise representation of their behavior
What applications do instead of What application are

We use dynamic information flow tracking at system level
we observe how an app disseminates data in the operating system.

We compute a *System Flow Graph* ...
that represent the external behavior of an application.

We recognize
if a given application presents a known malicious behavior

We trigger
suspicious executions
”if applications embed the same repacked malicious piece of code then they should present similar behaviors”

How capturing a behavioral signature of Android malware?
How triggering an Android malware?
Outline

1. Information flows monitoring at system level
2. Capturing malware behavior
3. Suspicious code triggering
4. Conclusion
Monitoring information flows at system level

foundations
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
by an identifier $i_1, i_2, \ldots, i_n$ called itag

Example

That was the sense of the previous green and violet marks.
Propagation of the marks

Each observation of an information flow
An information flow $A \rightarrow B$ modifies the content of $B$,

- either the content coming from $A$ overwrites the previous content
  
  \[ B.\text{itag} = A.\text{itag} \]

- or the content is added
  
  \[ B.\text{itag} = B.\text{itag} \cup A.\text{itag} \]
Propagation of the marks

Each observation of an information flow

An information flow $A \rightarrow B$ modifies the content of $B$,
- either the content coming from $A$ overwrites the previous content
  \[ B.\text{itag} = A.\text{itag} \]
  \[
  \text{cp} \rightarrow \text{Fichier 3}
  \]
- or the content is added
  \[ B.\text{itag} = B.\text{itag} \cup A.\text{itag} \]
Propagation of the marks

Each observation of an information flow

An information flow $A \rightarrow B$ modifies the content of $B$,

- either the content coming from $A$ overwrites the previous content
  
  $$B.\text{itag} = A.\text{itag}$$

- or the content is added
  
  $$B.\text{itag} = B.\text{itag} \cup A.\text{itag}$$
Propagation of the marks

Each observation of an information flow

An information flow $A \rightarrow B$ modifies the content of $B$,

- either the content coming from $A$ overwrites the previous content

\[ B\cdot itag = A\cdot itag \]

- or the content is added

\[ B\cdot itag = B\cdot itag \cup A\cdot itag \]
(Andro)Blare: our information flow monitor

(Andro)Blare

1. Maintains \textit{itag} in extended attributes of files
2. Updates \textit{itag} values at each observation of an information flow
3. Is implemented within an adaptation of LSM framework
(Andro)Blare observes

Each system call

\((\text{read, write, fork, execve, ...})\)

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

\((\text{file, socket, processus})\)

Blare has a local but accurate view of explicit information flows

Blare logs

each observed flow

\([\text{TIMESTAMPs}][\text{ORIGIN}][\text{DESTINATION}][\text{ITAG}]\)

that generates a huge amount of log entries
Understanding data dissemination

To better understand how an application impacts its environment:

- Which files have been modified?
- Which processes have been executed?
- Which remote IP addresses 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.
Understanding malware behavior

*Using logs of an 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
Representing behaviors

A SFG describes all the information flow involving a marked data or contaminated by a marked data.

If we have only marked the code of an app, we learn how this app contaminates the operating system.
SFG due to the monitoring of an Android application
Usual Subgraphs
Malicious behavior

The attack is described in the remaining subgraph
System Flow Graph to understand malware behaviors
Labex CominLabs Kharon Project (2015-2018)

Aims

Being able to

- 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.
- ElseGazebo an analysis platform
- Kharon dataset a collection of reversed malware
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

[Malcon’15] *Best Paper*

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*

---

1. together with JF Lalande
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

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

Experiments

GroddDroid succeed to trigger 28 of the most scored units of code appearing on 100 malware
GroddDroid is included in ElseGazebo a platform for malware analysis
Future works
Enhancing GroddDroid
Mourad Leslous Phd thesis

More tests on more applications
to highlight normal behaviors, to compute more behavioral signatures, to discover unknown malware

Evade the detection
in studying behavioral obfuscation

http://kharon.gforge.inria.fr