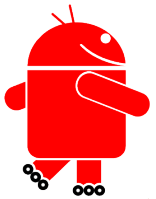
Android Malware Analysis: from technical difficulties to scientific challenges

Jean-François Lalande

Keynote – SecITC 2018

Bucharest, Romania
November 8th 2018



Inria INVENTEURS DU MONDE NUMÉRIQUE
CominLabs
CentraleSupélec

Context

Google Play Store: 3.5 million applications (2017)

- Malware are uploaded to the Play Store
- + Third party markets

Research efforts:

- Detection, classification
- Payload extraction, unpacking, reverse
- Execution, triggering

Difficulties for experimenting with Android malware samples?
What are the upcoming scientific challenges?

Researchers

Usually they do:

- You have an idea
- You develop
- You take a dataset
- You evaluate

We also do this :)

We = Valérie Viet Triem Tong, Guillaume Hiet, Mourad Leslous (PhD), Pierre Graux (PhD) and many other master students...

Challenges

About datasets of malware

- Is there any datasets?
- Can we build one?

About analysis of malware

- What are the difficulties?
- Is it reliable and reproducible?
- Are samples really malware?
- How much does it cost?
- Is it scalable?

Upcoming challenges

- What next?

Example from the state of the art

About papers that work on Android malware...

- CopperDroid [Tam et al. 2015]:
  - 1365 samples
  - 3% of payloads executed
- IntelliDroid [Wong et al. 2016]:
  - 10 samples
  - 90% of payloads executed
- GroddDroid [us ! 2015]:
  - 100 samples
  - 24% of payloads executed
- ...

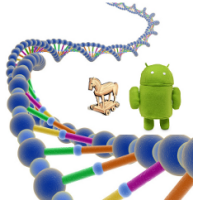
Is it easy to get these figures (and to reproduce them)?
Are these results relevant?

Why building a dataset ?

Papers with **Android malware experiments**:

- use **extracts of reference datasets**:
  - The Genome project (stopped !) [Zhou et al. 12]
  - Contagio mobile dataset [Mila Parkour]
  - Hand crafted malicious apps (DroidBench [Artz et al. 14])
  - Some Security Challenges' apps
- need to be **significant**:
  - Tons of apps (e.g. 1.3 million for PhaLibs [Chen et al. 16])
  - Some apps (e.g. 11 for TriggerScope [Fratantonio et al. 16])

- A **well documented** dataset does not exist !
- Online services give **poor information** !



Building a dataset

Collect malware

- from online sources, or researchers
- study the samples manually

Methodology:

- manual reverse of 7 samples
- manual triggering (not obvious)
- execution and information flow capture

By Con-struct + replicant community [CC BY-SA 3.0]

A collection of malware totally reversed

Kharon dataset: **7** malware[1]:

`http://kharon.gforge.inria.fr/dataset`

- DroidKungFu, BadNews (2011, 2013)
- WipeLocker (2014)
- MobiDash (2015)
- SaveMe, Cajino (2015)
- SimpleLocker (2014)

[1] Approved by **Inria's Operational Legal and Ethical Risk Assessment Committee**: We warn the readers that these samples have to be used for research purpose only. We also advise to carefully check the SHA256 hash of the studied malware samples and to manipulate them in a sandboxed environment. In particular, the manipulation of these malware impose to follow safety rules of your Institutional Review Boards.

Remote admin Tools

Install malicious apps:

- **Badnews**: Obeys to a remote server + delays attack
  Triggering: Patch the bytecode + Build a fake server
- **DroidKungFu1** (well known): Delays attack
  Triggering: Modify 'start' to 1 in `sstimestamp.xml` and reboot the device

Blocker / Eraser

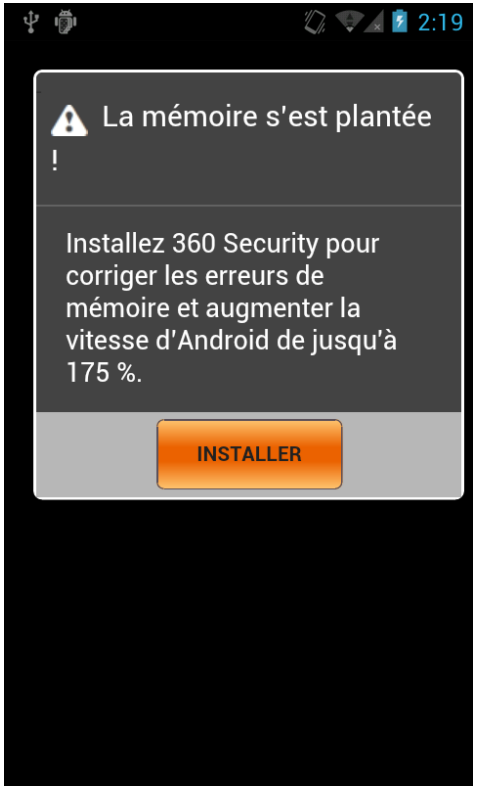
Wipes of the SD card and block social apps:

- **WipeLocker**: Delayed Attack
  Triggering: Launch the app and reboot the device

Adware

Displays adds after some days:

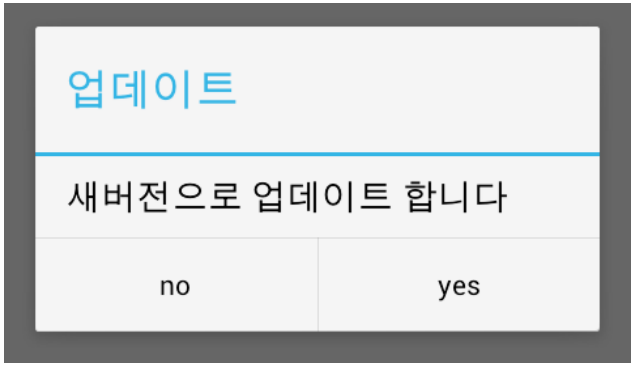
- **MobiDash**: Delayed Attack
  Triggering: Launch the application, reboot the device and modify `com.cardgame.durak_preferences.xml`



Spyware

Steals contacts, sms, IMEI, . . .

- **SaveMe**: Verifies the Internet access
  Triggering: Enable Internet access and lauch the app
- **Cajino**: Obeys a Baidu remote server
  Triggering: Simulate a server command with an Intent

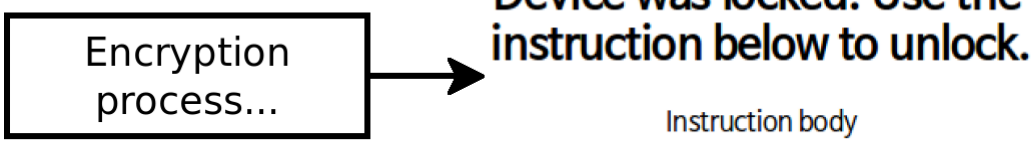


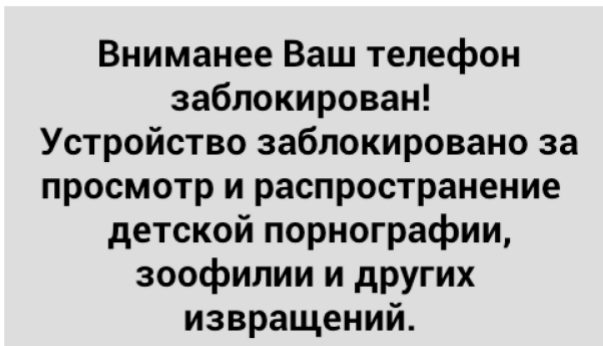
Ransomware

Encrypts user's files and asks for paying:

SimpleLocker

- Waits the reboot of the device
  Triggering: send a BOOT_COMPLETED intent





More details about SimpleLocker...

Example: SimpleLocker

The main malicious functions:

```
org.simplelocker.MainService.onCreate()
org.simplelocker.MainService$4.run()
org.simplelocker.TorSender.sendCheck(final Context context)
org.simplelocker.FilesEncryptor.encrypt()
org.simplelocker.AesCrypt.AesCrypt(final String s)
```

The encryption loop:

```
final AesCrypt aesCrypt = new AesCrypt("jndlasf074hr");

for (final String s : this.filesToEncrypt) {
  aesCrypt.encrypt(s, String.valueOf(s) + ".enc");
  new File(s).delete();
}
```

Dataset overview

Type	**Name**	**Protection against dynamic Analysis** $\rightarrow$ *Remediation*
RAT	Badnews	Obeys to a remote server and delays the attack $\rightarrow$ *Modify the apk* $\rightarrow$ *Build a fake server*
Ransomware	SimpleLocker	Waits the reboot of the device $\rightarrow$ *send a BOOT_COMPLETED intent*
RAT	DroidKungFu	Delayed Attack $\rightarrow$ *Modify the value* `start` *to* `1` *in sstimestamp.xml*
Adware	MobiDash	Delayed Attack $\rightarrow$ *Launch the infected application, reboot the device and modify* `com.cardgame.durak_preferences.xml`
Spyware	SaveMe	Verifies the Internet access $\rightarrow$ *Enable Internet access and launch the application*
Eraser+LK	WipeLocker	Delayed Attack $\rightarrow$ *Press the icon launcher and reboot the device*
Spyware	Cajino	Obeys to a remote server $\rightarrow$ *Simulate the remote server by sending an intent*

New recent datasets

AndroZoo [Allix et al. 2016]

- 3 million apps
- With pairs of applications (repackaged ?)

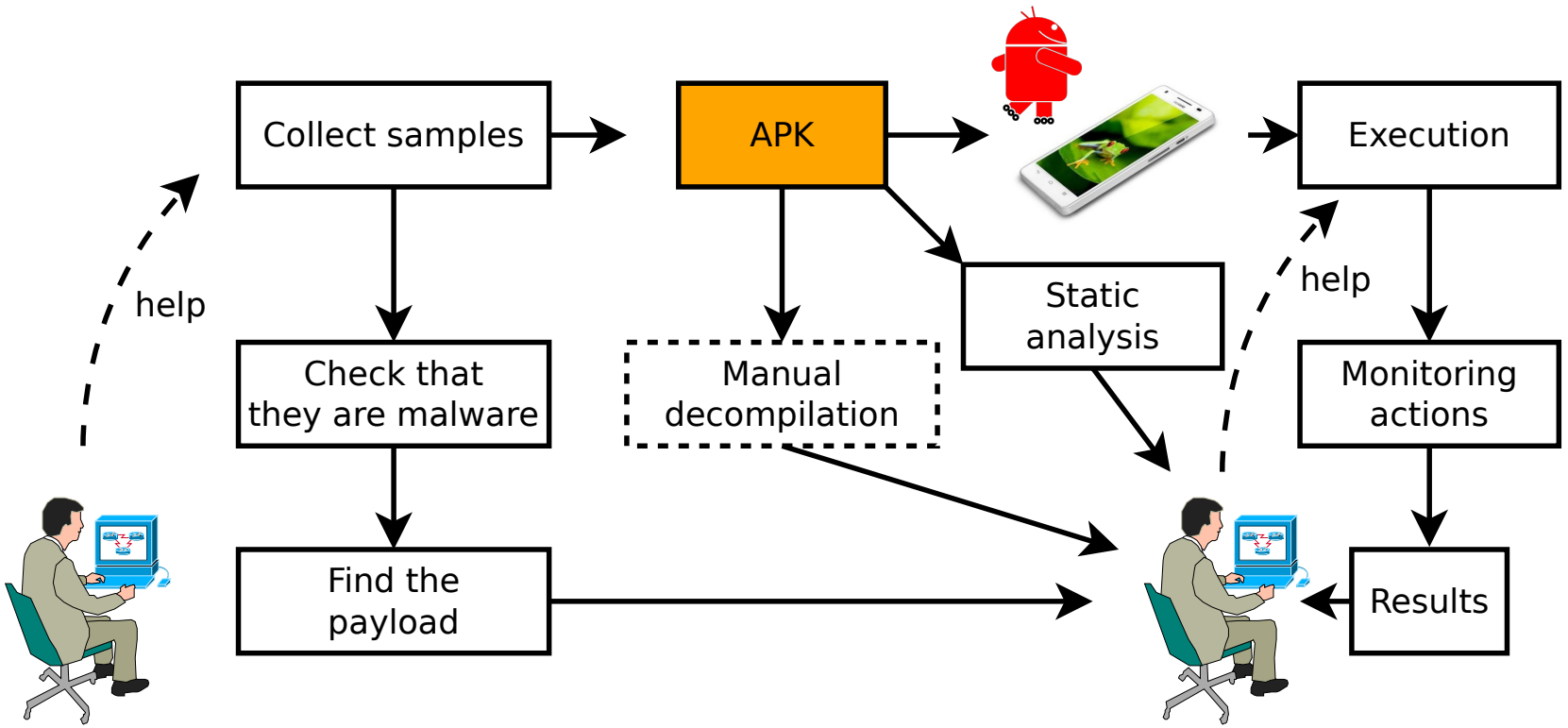
The AMD dataset [Wei et al. 2017]

- 24,650 samples
- With contextual informations (classes, actions, . . .)

We need more contextual information !

- Where is the payload ?
- How to trigger the payload ?
- Which device do I need ?

Designing an experiment from scratch



We have not time for these folks!
We want an *automatic* process...

Difficulties

1. Is this apk a malware?
2. Where is the payload?
   - locating the payload $\neq$ classifying a malware/goodware
   - what does the payload?
3. Is the static analysis possible?
   - What is the nature of the code?
   - Is there any countermeasure?
4. How to execute automatically the malware?
   - How to handle the GUI?
   - How to find entry points?
   - How to monitor the execution?

Difficulties

1. Is this apk a malware?
2. Where is the payload?
   - locating the payload $\neq$ classifying a malware/goodware
   - what does the payload?
3. Is the static analysis possible?
   - What is the nature of the code?
   - Is there any countermeasure?
4. How to execute automatically the malware?
   - How to handle the GUI?
   - How to find entry points?
   - How to monitor the execution?

Check that a sample is a malware?

Manually. . . for 10 samples ok, but for more ?

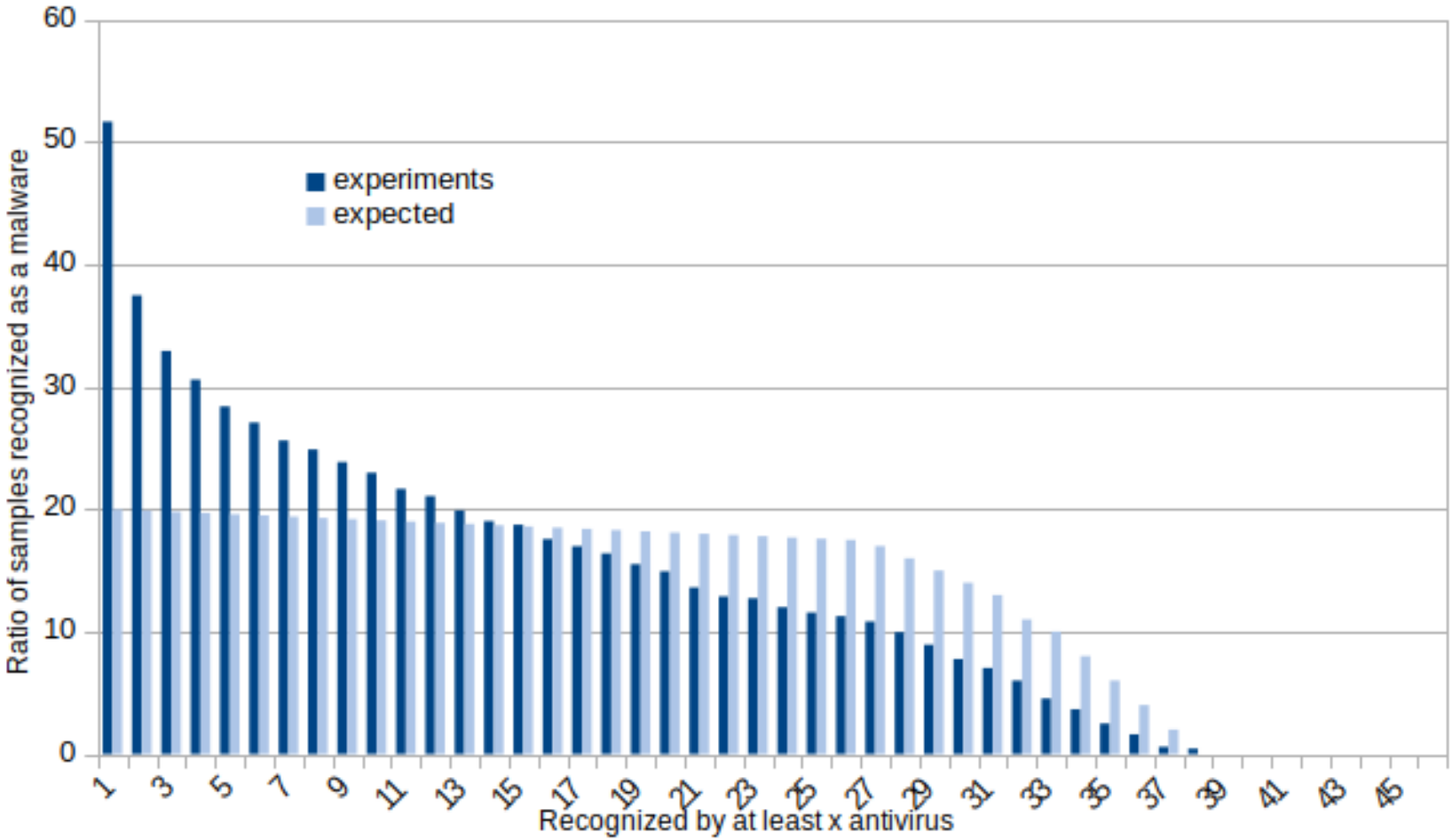
Ask VirusTotal!

- ∼45 antiviruses software
- Use a threshold to decide (e.g. 20 antiviruses)
- Free upload API (few samples / day)
- Used by others in papers

Is it a good idea?

An experiment with 683 fresh samples

Threshold of *x* antiviruses recognizing a sample?



Check that a sample is a malware?

Not solved:

- using VirusTotal
- for fresh new samples

Solved:

- for old well-known samples
- by many learning papers (detection rate $\geq 90\%$)
  e.g. Milosevic et al.: precision of 87% with Random Forests
  e.g. Zhu et al.: precision of 88% with Rotation Forests

Difficulties

1. Is this apk a malware?
2. Where is the payload?
   - locating the payload $\neq$ classifying a malware/goodware
   - what does the payload?
3. Is the static analysis possible?
   - What is the nature of the code?
   - Is there any countermeasure?
4. How to execute automatically the malware?
   - How to handle the GUI?
   - How to find entry points?
   - How to monitor the execution?

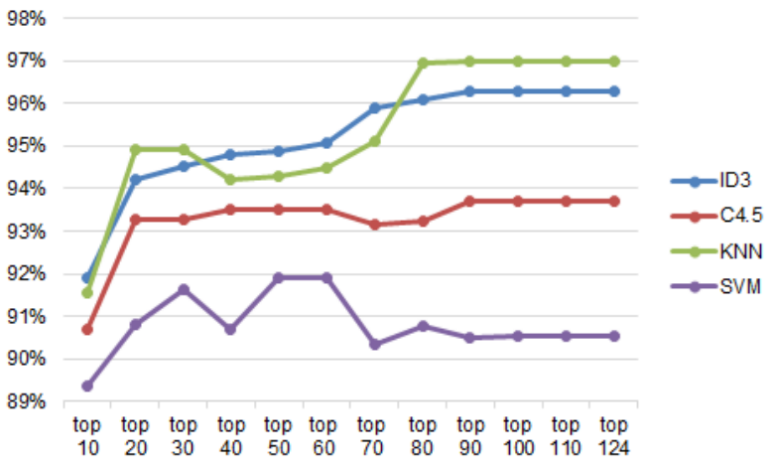
Where is the payload?

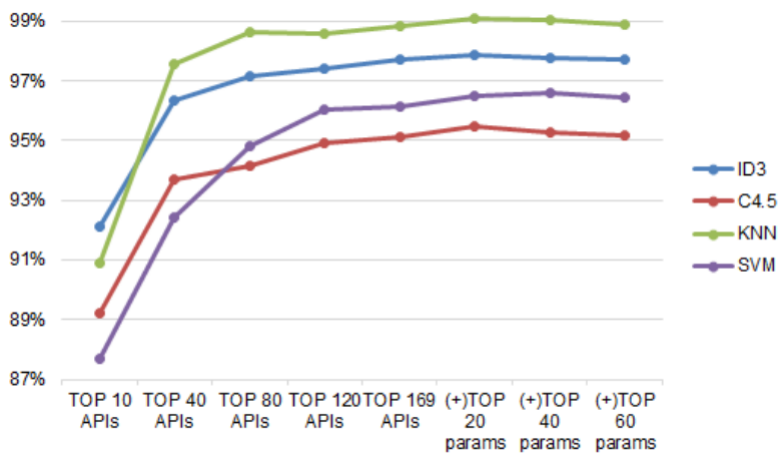
Seminal paper: “DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android”’ Aafer et al. (2013)
$\Rightarrow$ Extract relevant features from API analysis.
Enables to:

- gives more meaning to the payload
- classifies apps with more accuracy

Results from Aafer et al. (2013):

- detection accuracy permission based / api based

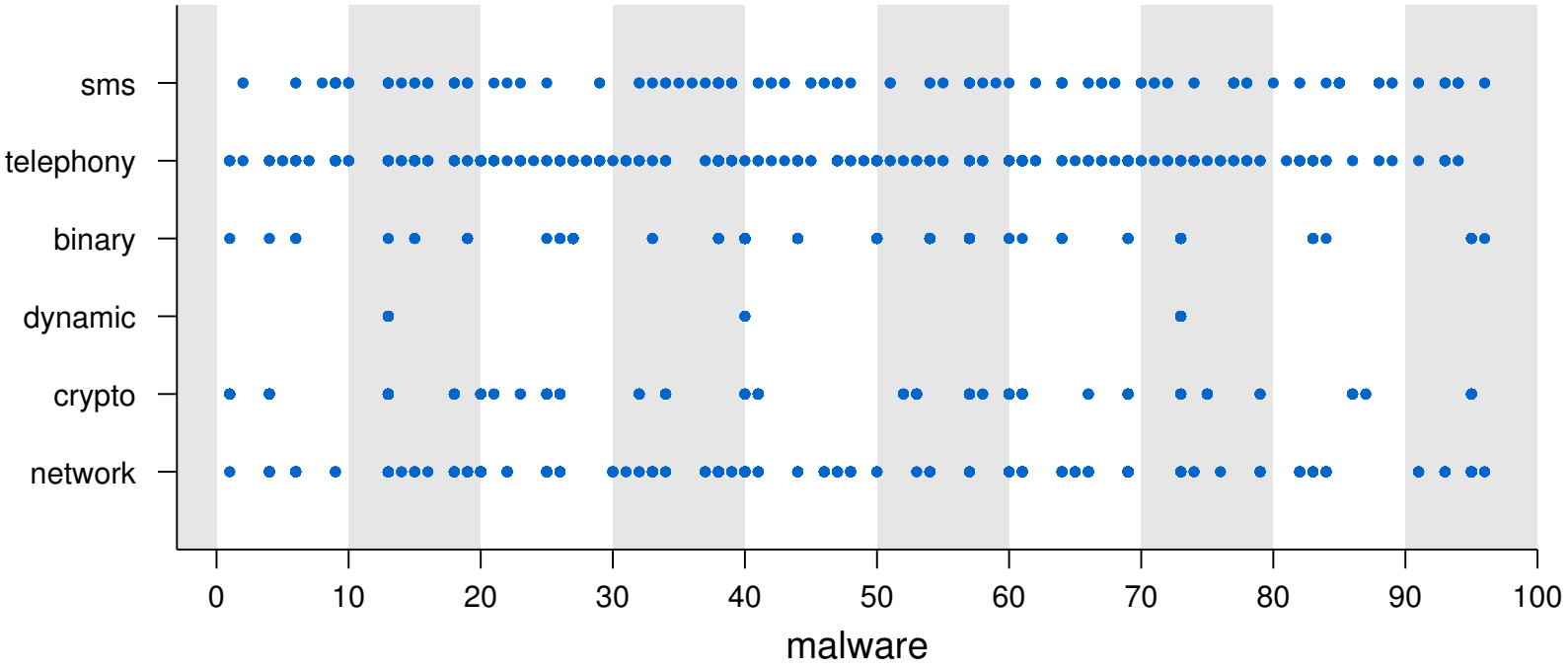




Extracted from DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android, Aafer et al.

Giving meaning to the payload

Graphical representation of malware features. . .



. . . with the limit that malware can be piggybacked apps!
(Li li et al. 2017)

Difficulties

1. Is this apk a malware?
2. Where is the payload?
   - locating the payload $\neq$ classifying a malware/goodware
   - what does the payload?
3. Is the static analysis possible?
   - What is the nature of the code?
   - Is there any countermeasure?
4. How to execute automatically the malware?
   - How to handle the GUI?
   - How to find entry points?
   - How to monitor the execution?

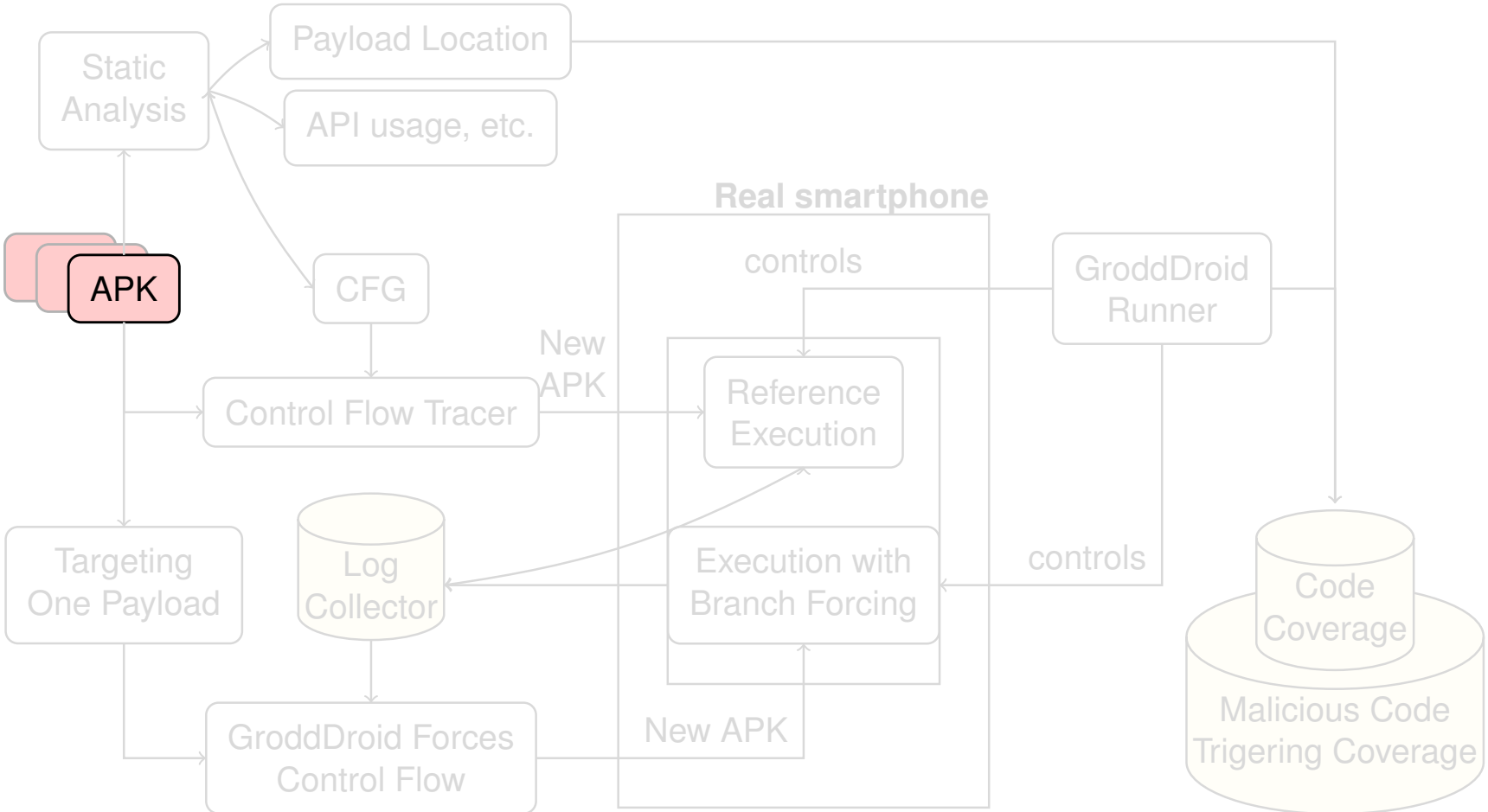
Analyzing malware

Main analysis methods are:

- **static analysis**:
  $\Rightarrow$ try to recognize known characteristics of malware in the code/resources of studied applications

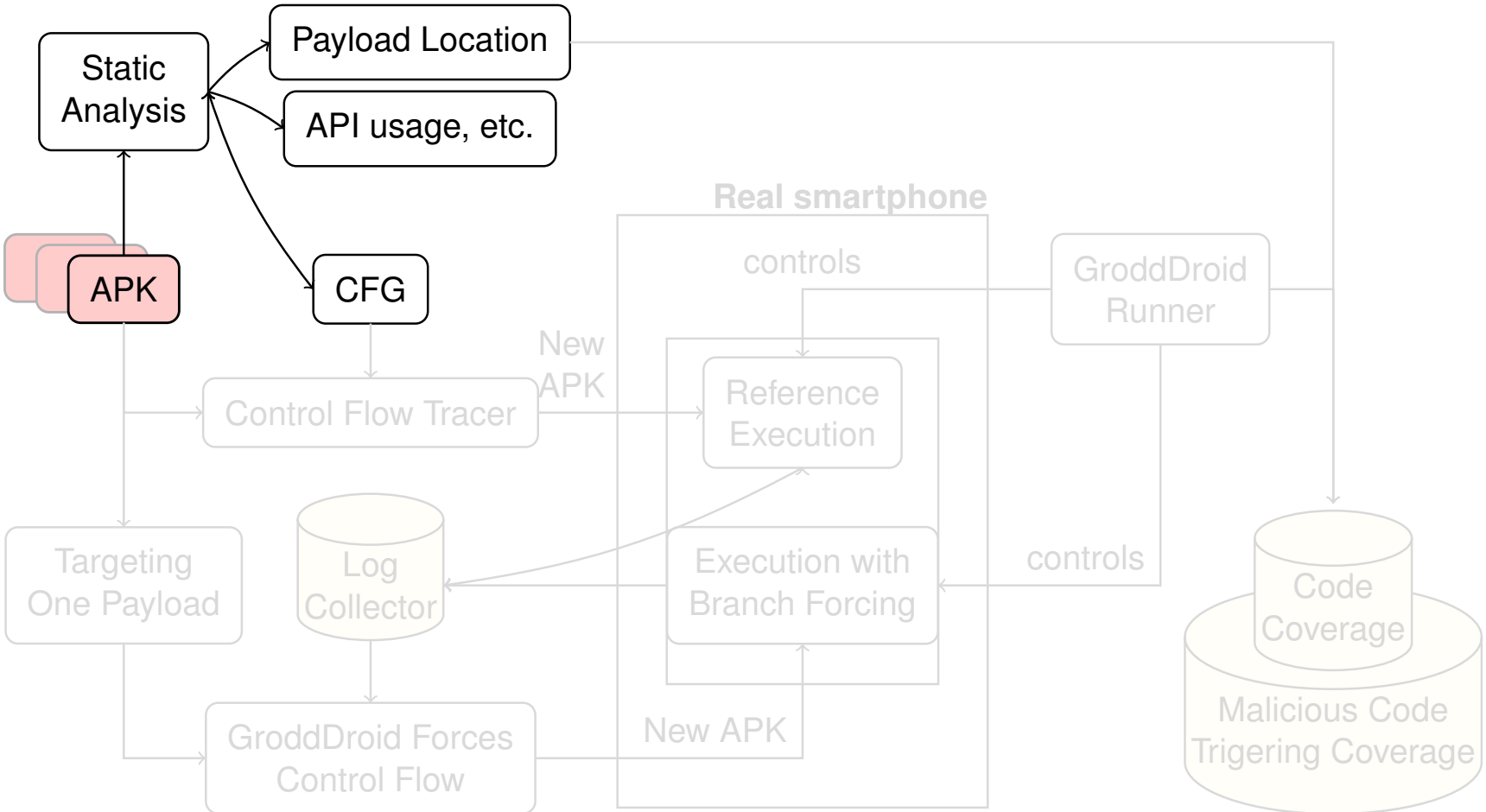
- **dynamic analysis**:
  $\Rightarrow$ try to execute the malware

Our analysis framework: GroddDroid[2]



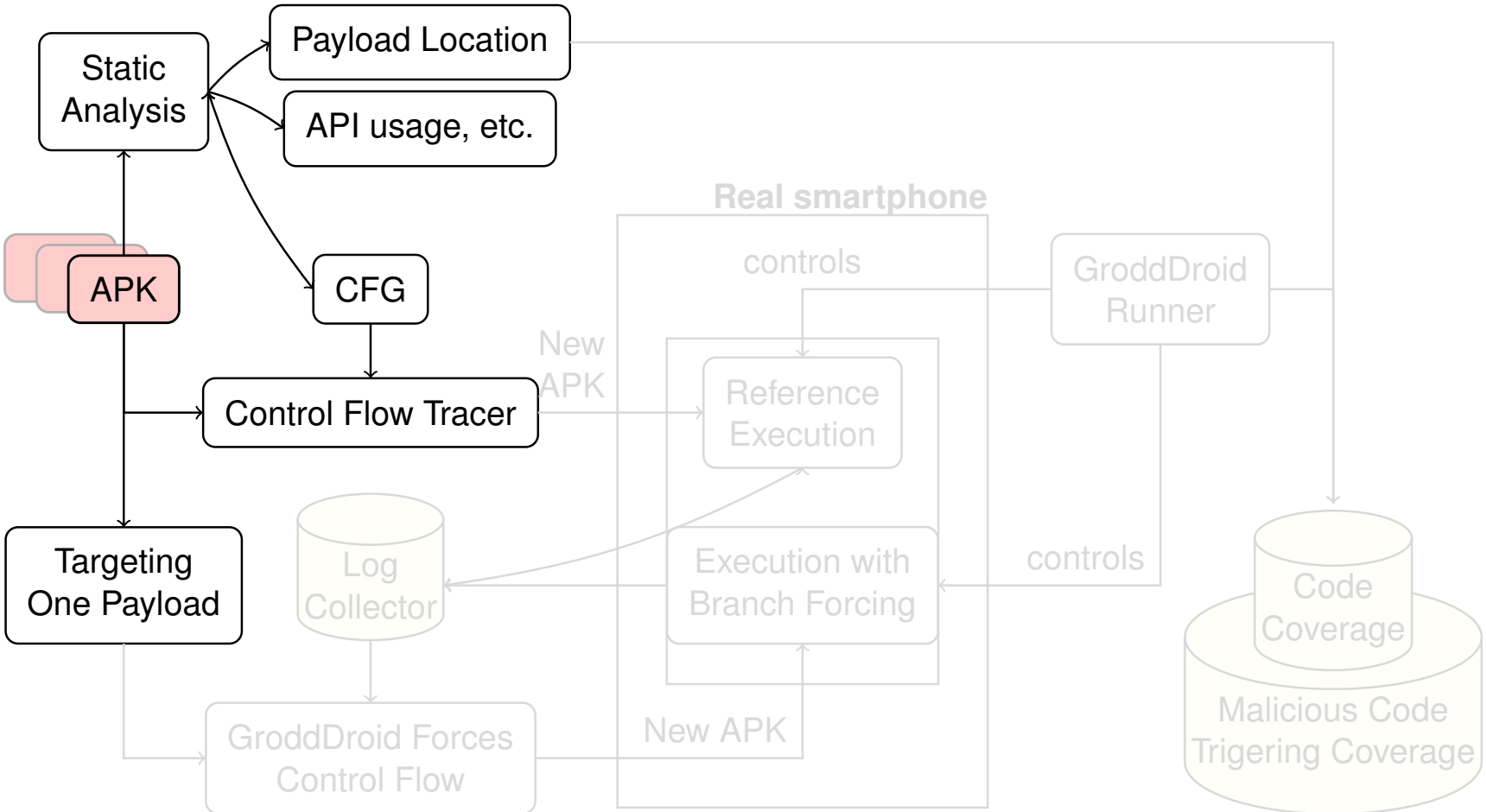
[2] Abraham et al. 2015, Leslous et al. 2017

Our analysis framework: GroddDroid[2]



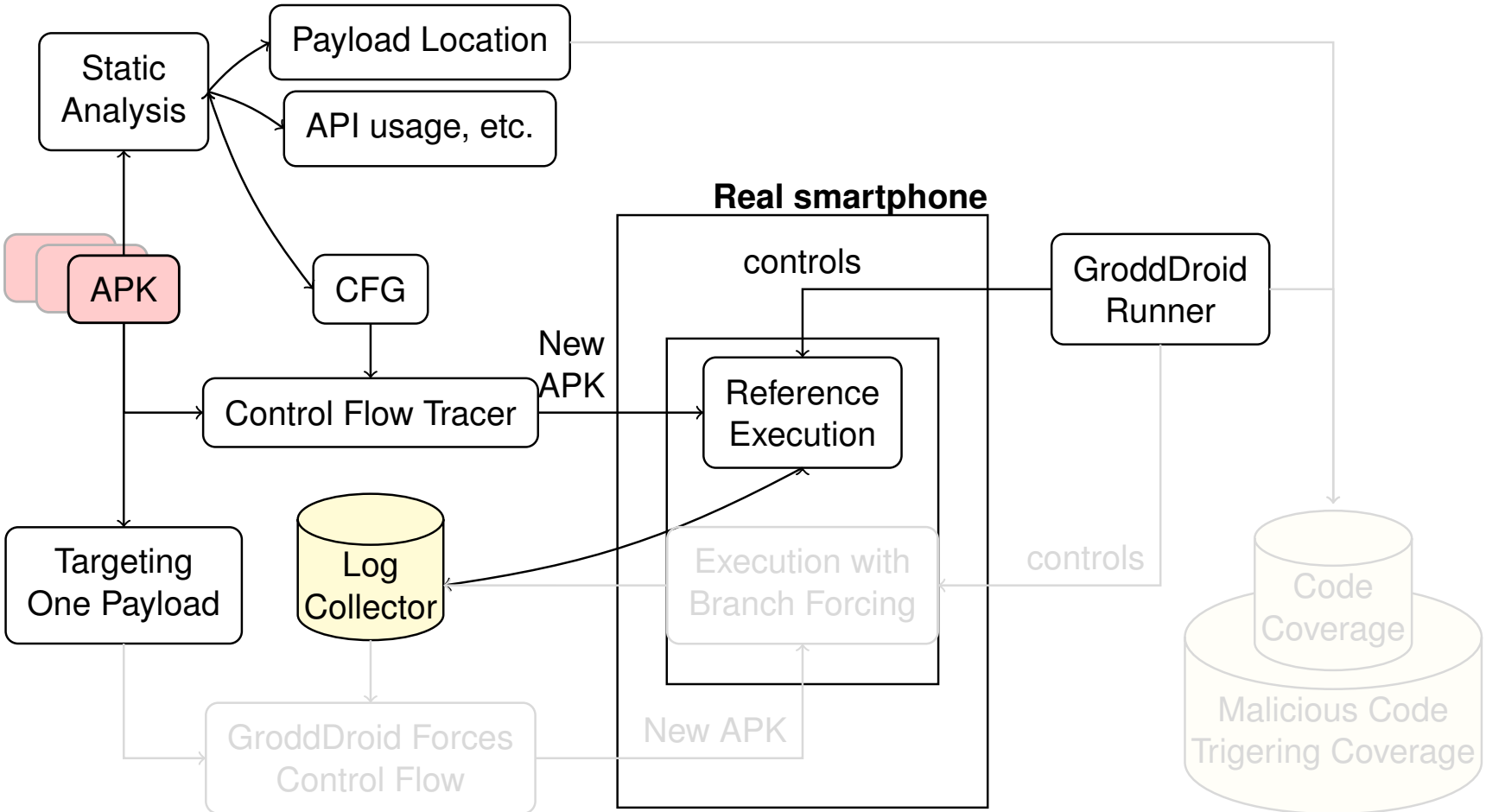
[2] Abraham et al. 2015, Leslous et al. 2017

Our analysis framework: GroddDroid[2]



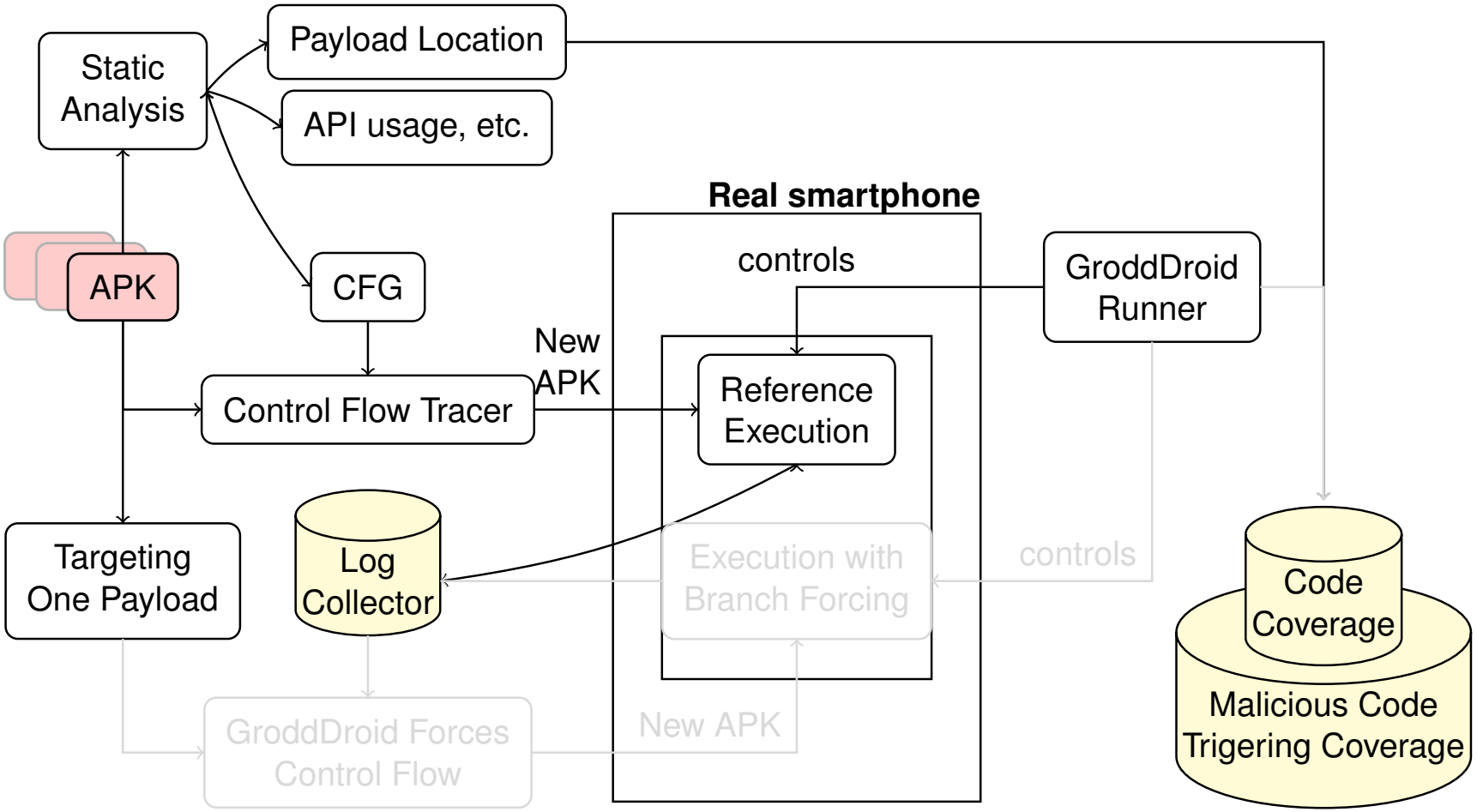
[2] Abraham et al. 2015, Leslous et al. 2017

Our analysis framework: GroddDroid[2]



[2] Abraham et al. 2015, Leslous et al. 2017

Our analysis framework: GroddDroid[2]



[2] Abraham et al. 2015, Leslous et al. 2017

Demo

GroddDroid output

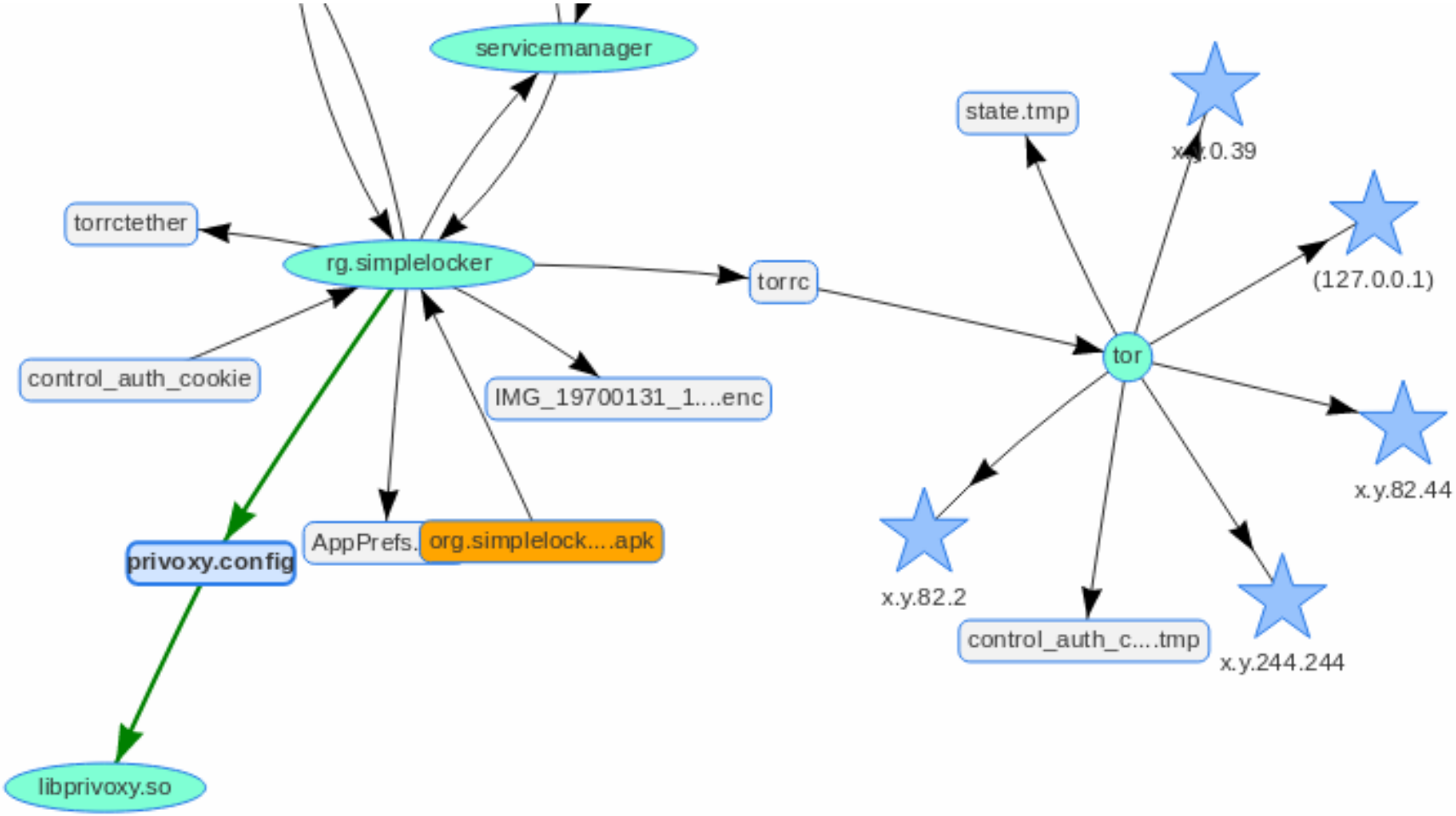
From logs:

- CFG: static Control Flow Graph
- payload location
- payload coverage (executed)
- screens

and with Blare (www.blare-ids.org):

- IFG: Information Flow Graph (at OS level)
- Spawned process
- Corruption attempts of the system
- Modifications of user files
- Internet connections

GroddDroid output example: simplelocker



Analyzing malware

Main analysis methods are:

- **static analysis**:
  $\Rightarrow$ try to recognize known characteristics of malware in the code/resources of studied applications

- **dynamic analysis**:
  $\Rightarrow$ try to execute the malware

Analyzing malware

Main analysis methods are:

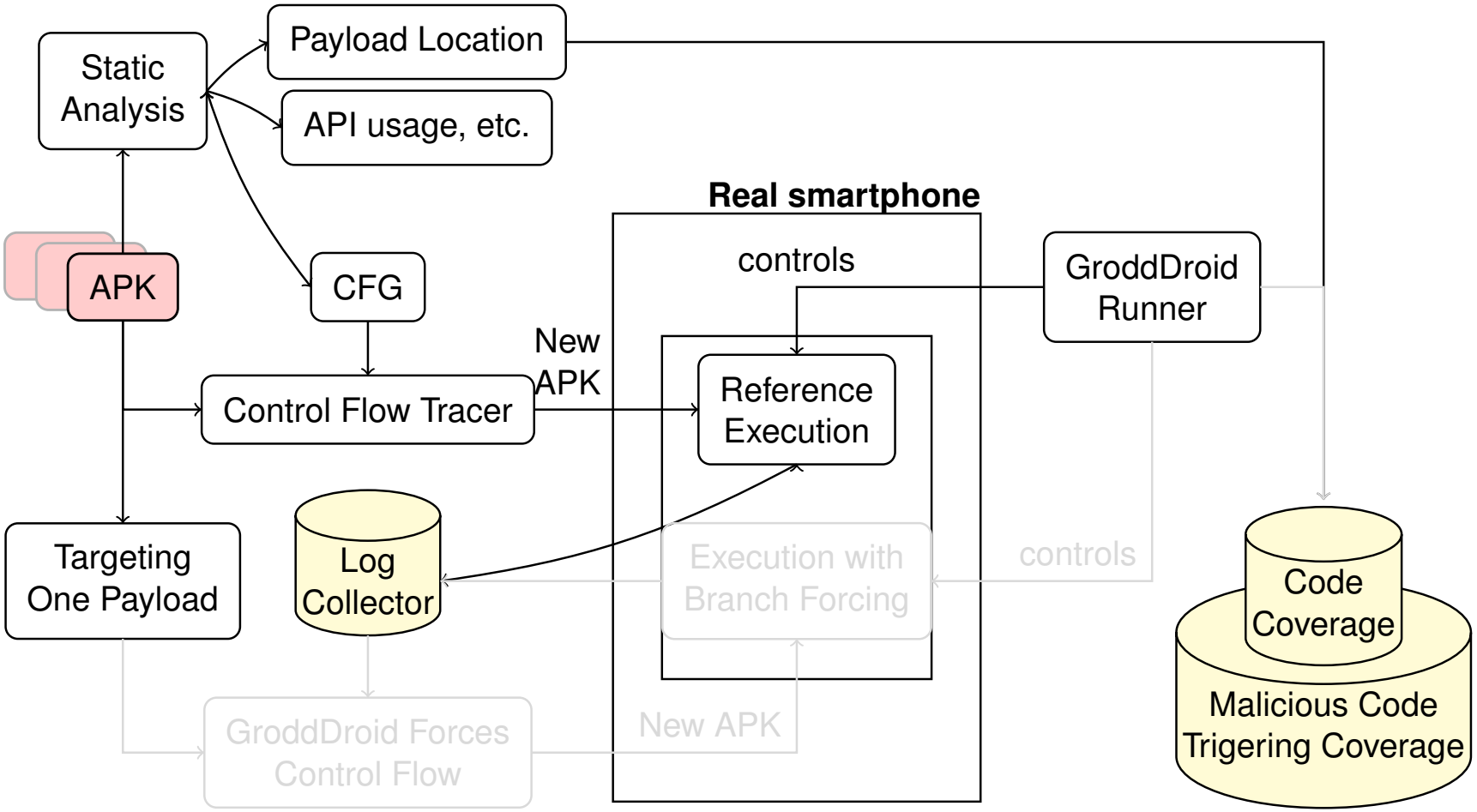
- **static analysis**:
  ⇒ try to recognize k… characteristics of… code/resourc…

Countermeasures: reflection, obfuscation, dynamic loading, encryption, native

- **dynamic analysis**:
  ⇒ try to execute the malware

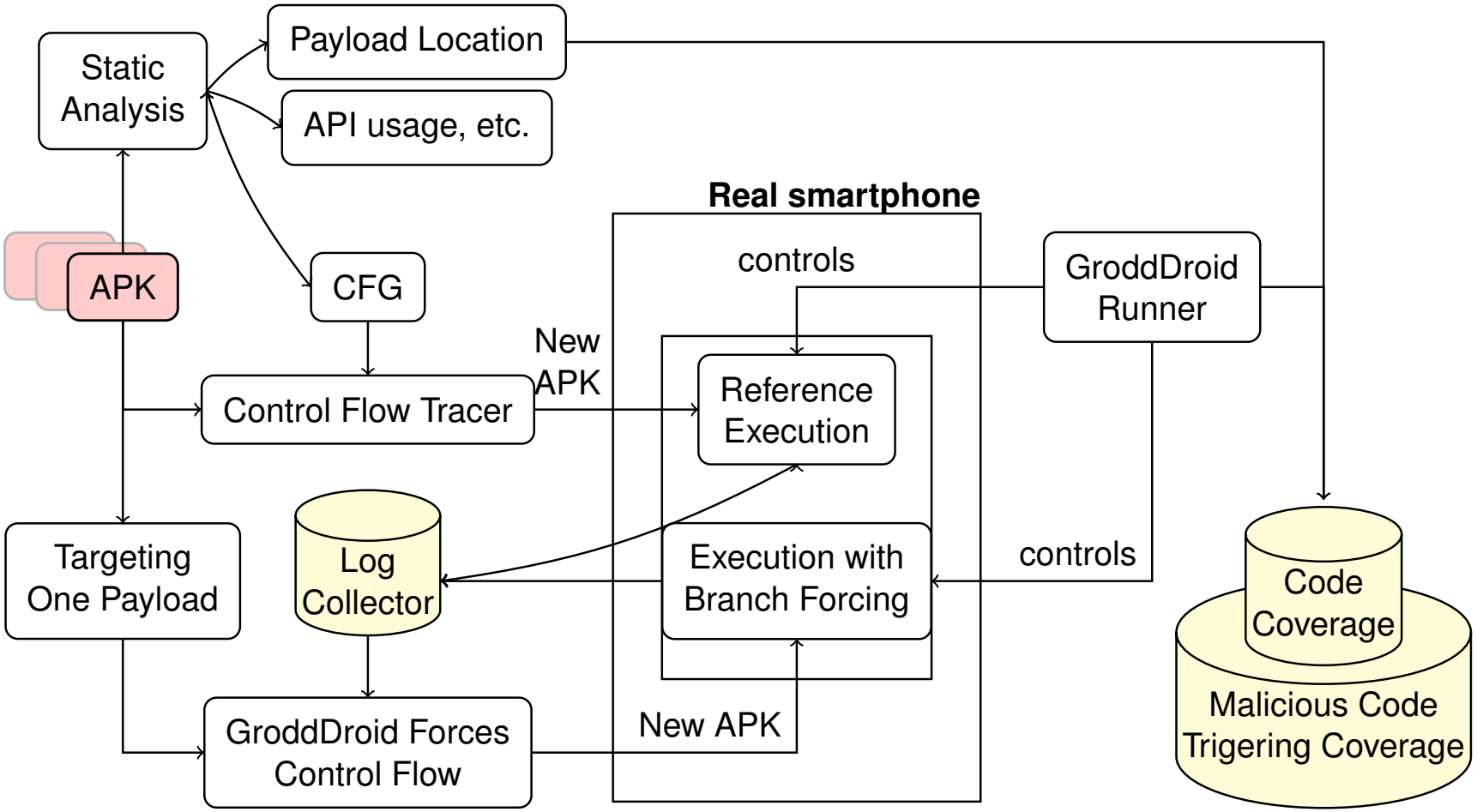
Countermeasures: logic bomb, time bomb, remote server

Our analysis framework: GroddDroid[2]



[2] Abraham et al. 2015, Leslous et al. 2017

Our analysis framework: GroddDroid[2]



[2] Abraham et al. 2015, Leslous et al. 2017

Solving attacker's countermeasures

Implemented / Possible solutions against attacker's countermeasures:

Problem	Solution
malformed files	ignore it if possible
reflection	execute it
dynamic loading	execute it
logic/time bomb	force conditions
native code	watch it from the kernel
packing	???
(dead) remote server	???

Malware analysis

We have developed software for:

- Static, dynamic analysis
- Smartphone flashing with custom kernel
- More info: `http://kharon.gforge.inria.fr`

Dynamic analysis requires a lot of efforts to be automatized.

- Is it working all the time for all malware?
- Is it efficient?

Reliability

Some malware crash (and people don't care. . .)

Crash ratio (at launch time):

- AMD dataset [Yang et al. 2017]: 5%
- Our native dataset: 20%

We need to know the reasons behind the crash

Performances

Time evaluation (average):

For one app and one payload:

- Flashing device: 60 s
- Static analysis: 7 s
- Dynamic analysis (execution): 4 m
- Total: 5 m

- From the AMD dataset: 135 samples
- 100 payloads per app
- All Android OS: 8 versions
- Total: **1 year** of experiments (with 1 device)

Performances

Time evaluation (average):

For one app and one payload:

- Flashing device: 60 s
- Static analysis: 7 s
- Dynamic analysis (execution): 4 m
- Total: 5 m

- From the AMD dataset: 135 samples
- 100 payloads per app
- All Android OS: 8 versions
- Total: **1 year** of experiments (with 1 device)

Scalability

We need a **real** smartphone.

At this time we use:

- A server running our software
- A pool of 1 to 5 smartphones (USB limitations ?)

Datasets

We need better datasets

- Up-to-date with fresh and old malware
- Labelled samples
  - Payload location
  - Formal description of the payload
  - already some initiatives: AndroZoo [Allix et al.]

Scalability

We need a more scalable running platform

- Real devices have limited ressources
- Emulators are easy to detect

It remains an open problem. . .

Countermeasures

Attackers now include countermeasures

- Logic bombs => done :)
- Native code => working on it!
- Packers => working on it!
- Analysis detection code
- Variations of malware

e.g. Yang et al. 2017 proposed a semantic analysis for building variations of malware

Android's Future

Evolution of the platform:

- Apps can be developed in Kotlin
- Fuschia can become the new underlying OS

Android is everywhere:

- Wear 2+
- Android Automotive
- Android Things

Will malware exist?

Conclusion

Designing experiments on Android malware is a difficult challenge!

Upcoming challenges are great!

http://kharon.gforge.inria.fr
http://kharon.gforge.inria.fr/dataset


Questions ?
©Inria / C. Morel

References

- H. J. Zhu, Z. H. You, Z. X. Zhu, W. L. Shi, X. Chen, and L. Cheng, “DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model,” Neurocomputing, vol. 272, pp. 638–646, 2018.
- N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, “Machine learning aided Android malware classification,” Comput. Electr. Eng., vol. 61, pp. 266–274, Jul. 2017.
- Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android,” Secur. Priv. Commun. Networks, vol. 127, pp. 86–103, 2013.
- L. Li et al., “Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting,” IEEE Trans. Inf. Forensics Secur., vol. 12, no. 6, pp. 1269–1284, Jun. 2017.
- W. Yang, D. Kong, T. Xie, and C. A. Gunter, “Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps,” 2017, pp. 288–302.
- A. Abraham, R. Andriatsimandefitra, A. Brunelat, J. F. Lalande, and V. Viet Triem Tong, “GroddDroid: A gorilla for triggering malicious behaviors,” in 2015 10th International Conference on Malicious and Unwanted Software, MALWARE 2015, 2016, pp. 119–127.
- M. Leslous, V. Viet Triem Tong, J.-F. Lalande, and T. Genet, “GPFinder: Tracking the Invisible in Android Malware,” in 12th International Conference on Malicious and Unwanted Software, 2017, pp. 39–46.
- K. Allix, T. F. Bissyandeé, J. Klein, and Y. Le Traon. AndroZoo: Collecting Millions of Android Apps for the Research Community. Mining Software Repositories (MSR) 2016.