COMEX: Deeply Observing Application Behavior on Real Devices

Zeya Umayya[1], Dhruv Malik[1], Arpit Nandi[1], Akshat Kumar[1], Sareena Karapoola[2], Sambuddho[1]
[1]IIIT Delhi, [2]IIT Madras India
CSET 2024

Outline

- Android Threat Landscape
- COMEX Modules
- APK Execution Time
- Variety of Data Collection
- Use-case of COMEX Data
- Previous Android Testbeds
- Challenges
- Conclusion

Android

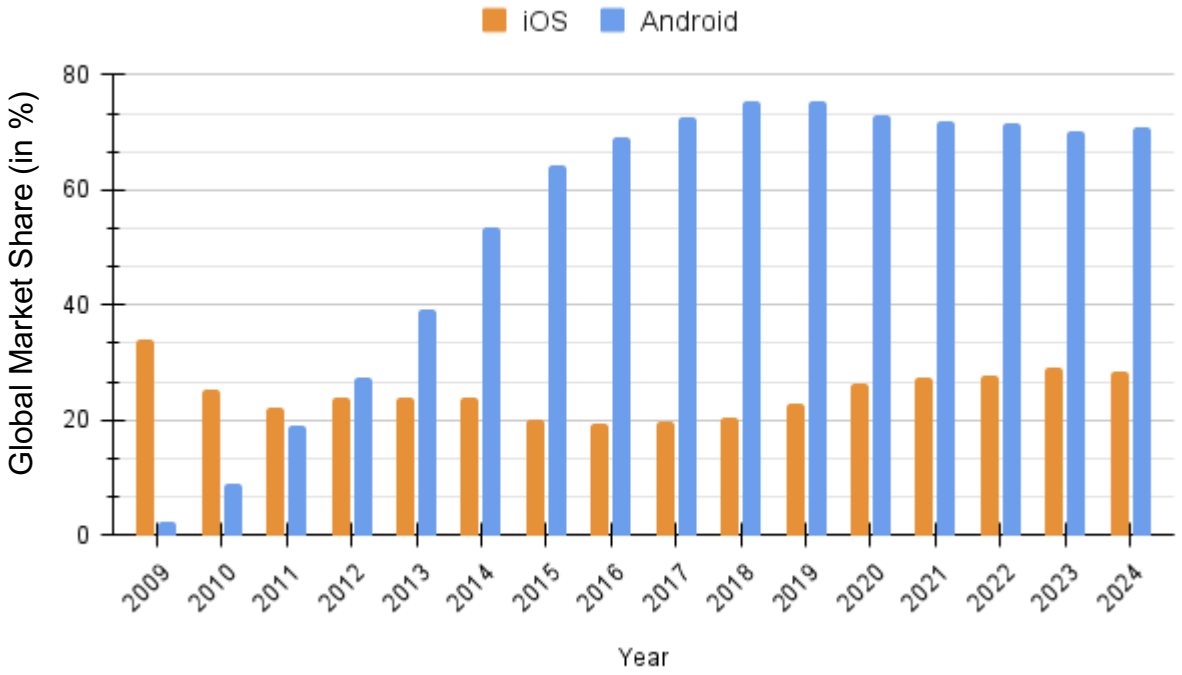


Image source -https://explodingtopics.com/blog/iphone-android-users

Increase in

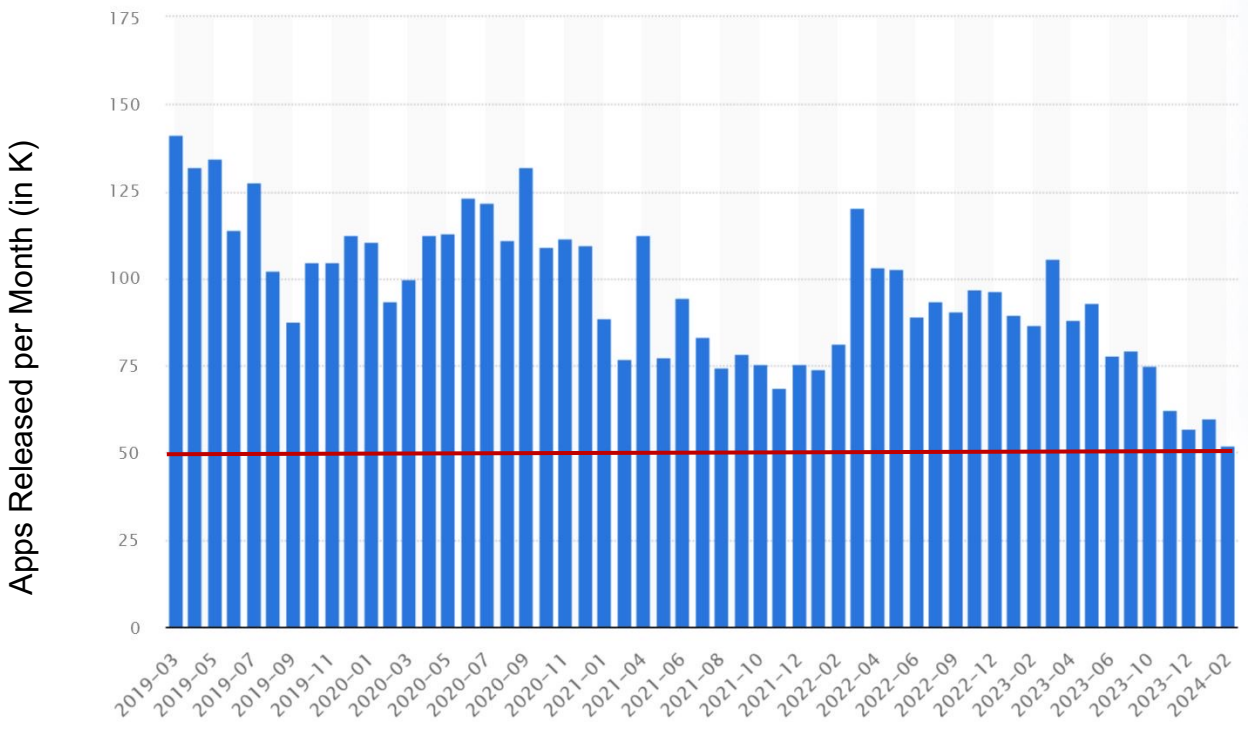


Image source - https://www.statista.com/statistics/1020956/android-app-releases-worldwide/

Resultant Increase –> 

There is a significant increase in the number of malicious activities by an application.

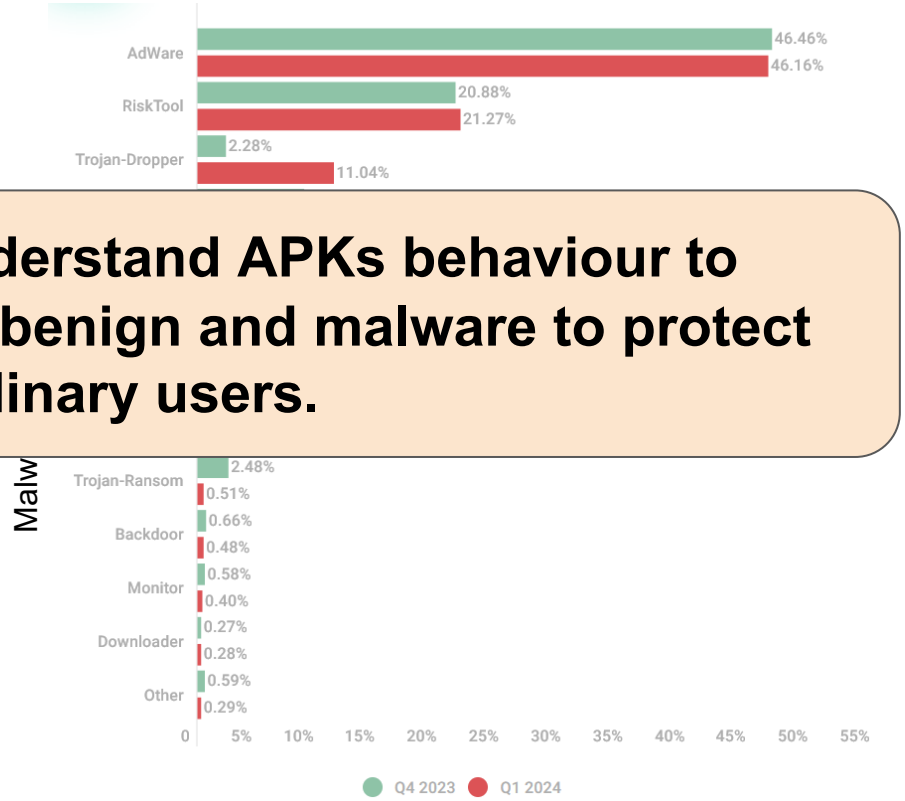
- **18% of clicked phishing emails** in 2022 came from a mobile device.
  (Verizon Mobile Security Index 2022)

- **46% organizations** that had suffered a mobile-related security breach in 2022 said that **app threats were a contributing factor**.
  (Verizon Mobile Security Index 2022)

- **9% of organizations** suffered a **mobile malware attack** in 2023.
  (Check Point 2023 Cyber Security Report)

Resultant Increase –>

- New malware variants in different categories
- D



Critical need to understand APKs behaviour to differentiate between benign and malware to protect ordinary users.

Image source - https://securelist.com/it-threat-evolution-q1-2024-mobile-statistics/112750/

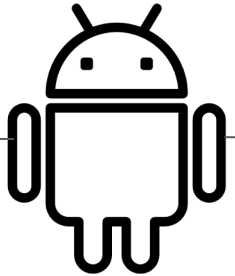
COMEX Testbed

We developed COMEX which is:

- Targeted for **real Android devices**
- **Does not** require any type of instrumentation
- Data from **all three sources** (i.e., OS+Network+Hardware)
- Considers **user input**
- Basic **analysis** of the raw data obtained
- Is **functional** and **available** for use

APK Analysis

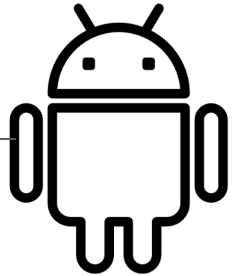
Static



Dynamic

APK Analysis

Static

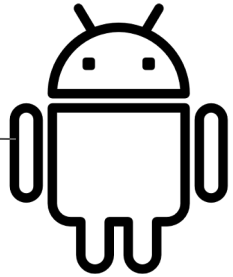


APK = Zip file

- Permissions
- Services
- Intent filters
- Packages, *etc.*

APK Analysis

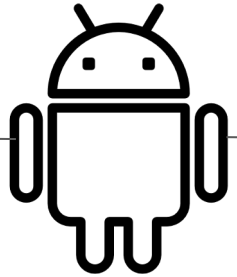
Static



- Fails in cases of
  - Obfuscated APKs
  - Encrypted APKs
  - Downloader type of APKs

- True behaviour only revealed at runtime for such APKs

APK Analysis



Static

- Fails in cases of
  - Obfuscated APKs
  - Encrypted APKs
  - Downloader type of APKs
- True behaviour only revealed at runtime for such APKs

Dynamic

- Run the APK in
  - **Emulator** or,
  - **Real-device**
- Observe APK specific events

COMEX Design

COMEX Design

AXMod

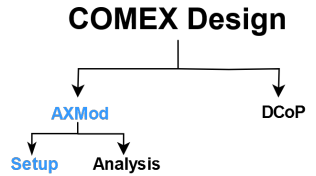
Analysis e**X**ecution **Mod**ule

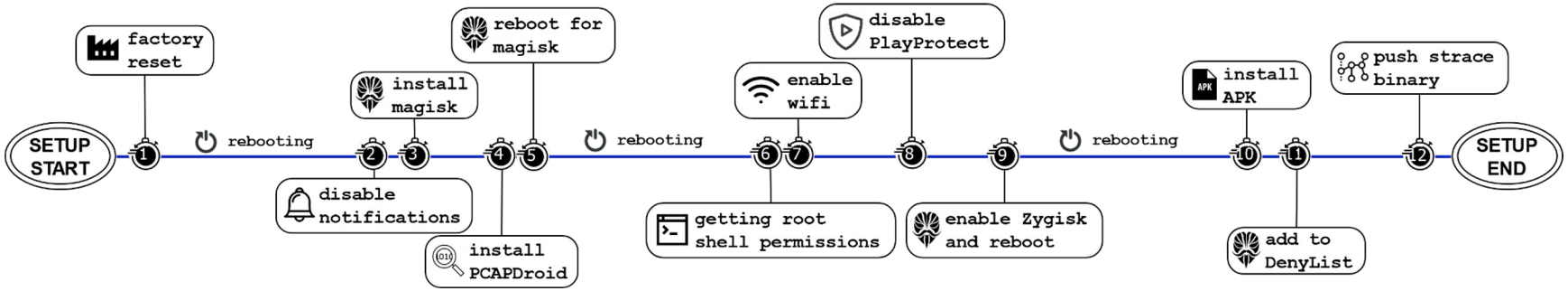
DCoP

Data **Co**llection **P**ipeline

COMEX Design

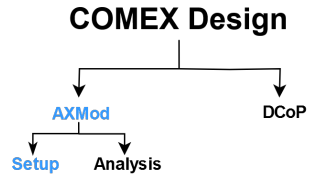
AXMod – Analysis Execution Module

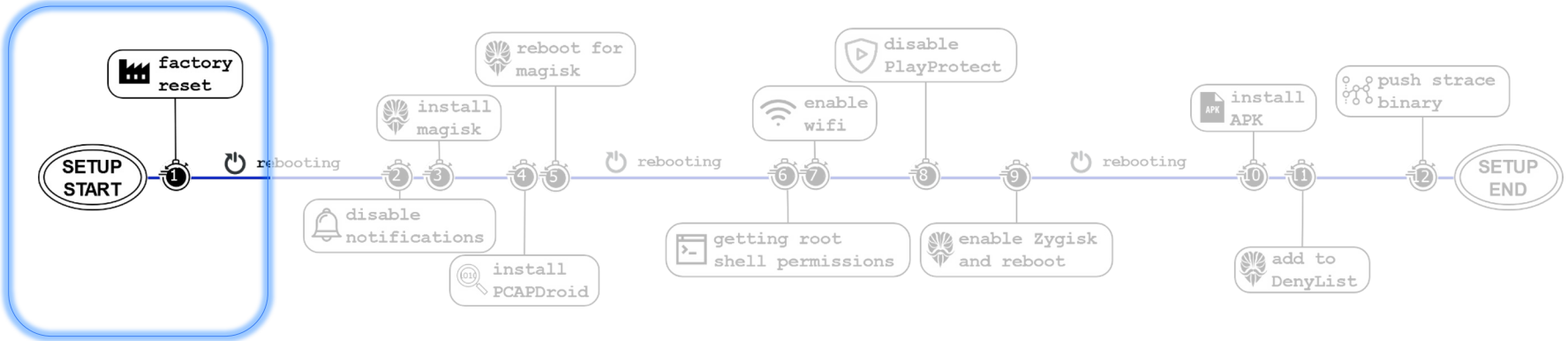




Setup Phase

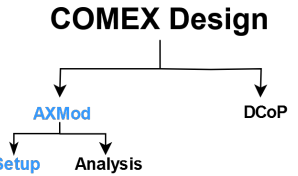
AXMod – Analysis Execution Module

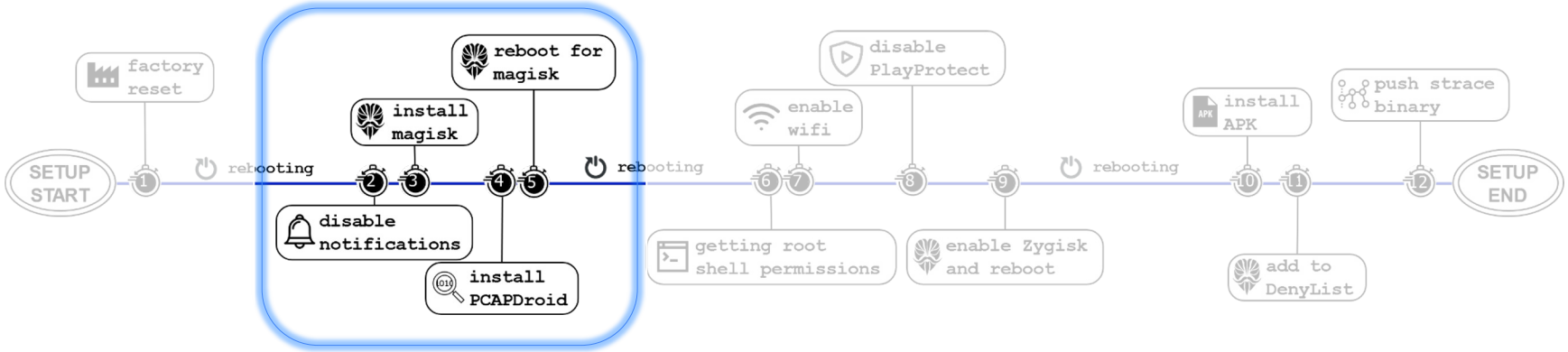




Setup Phase

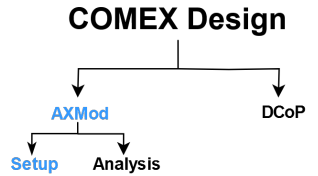
AXMod – Analysis Execution Module

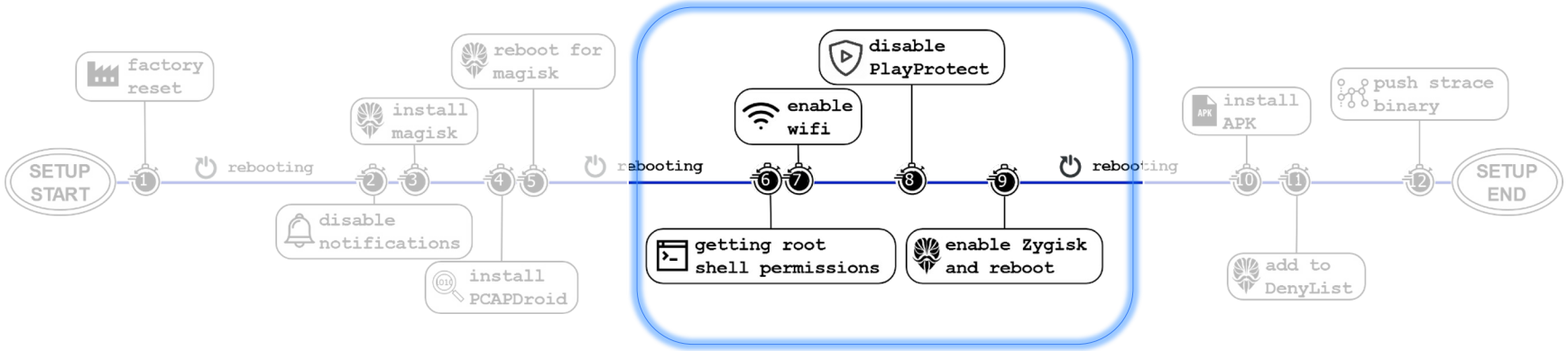




Setup Phase

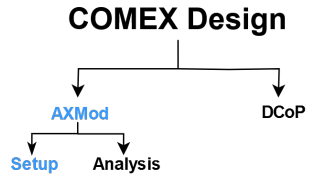
AXMod – Analysis Execution Module

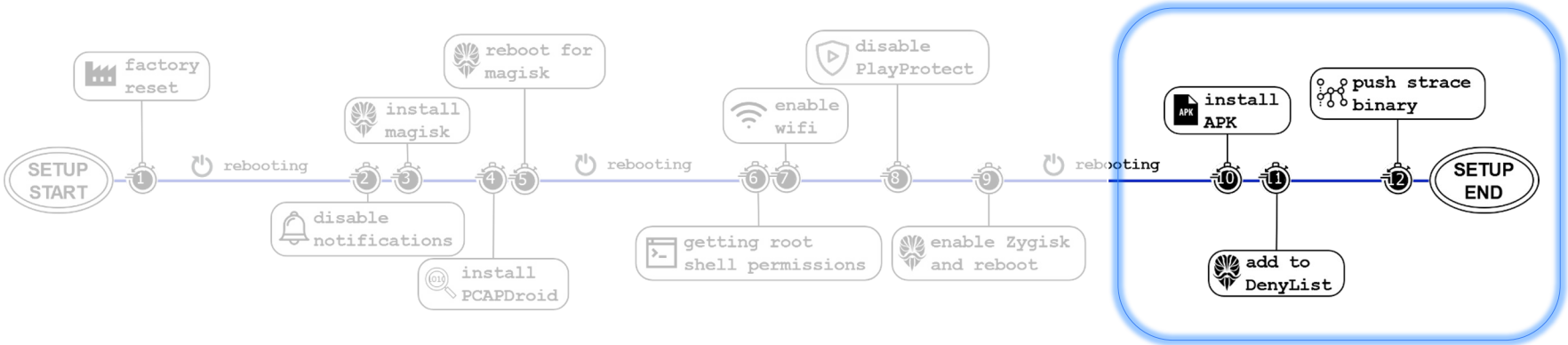




Setup Phase

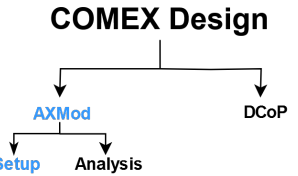
AXMod – Analysis Execution Module

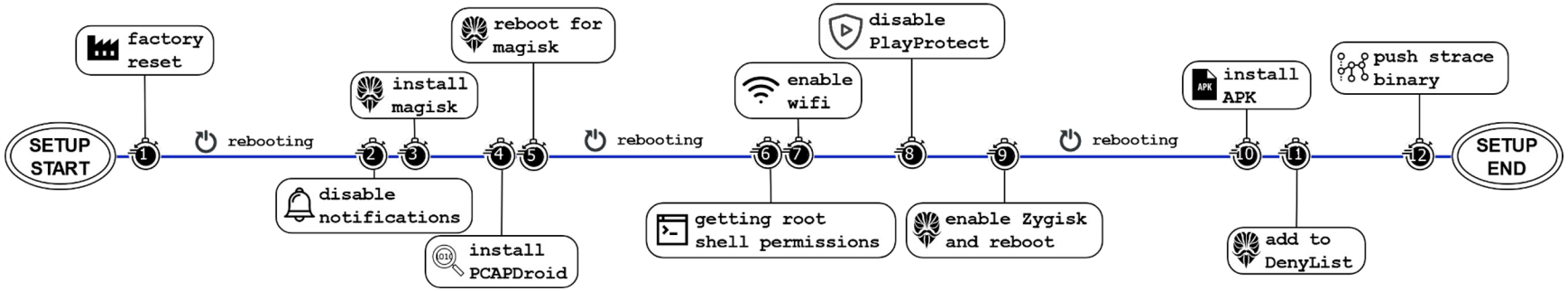




Setup Phase

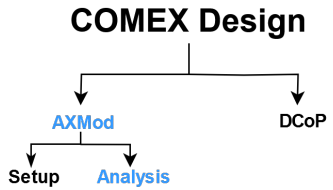
AXMod – Analysis Execution Module

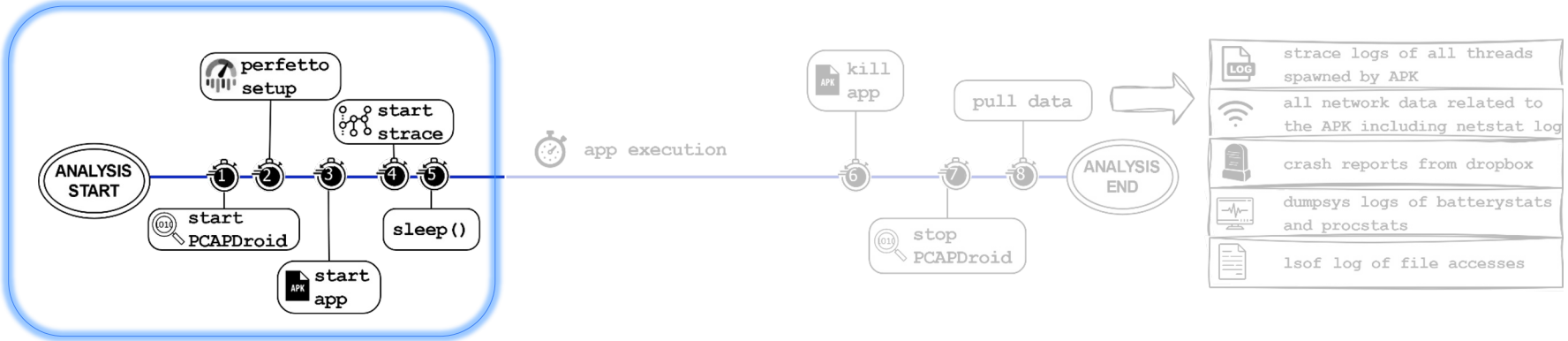




Setup Phase

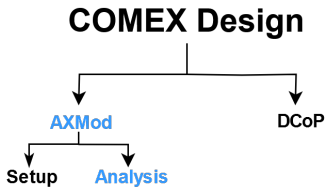
AXMod – Analysis Execution Module

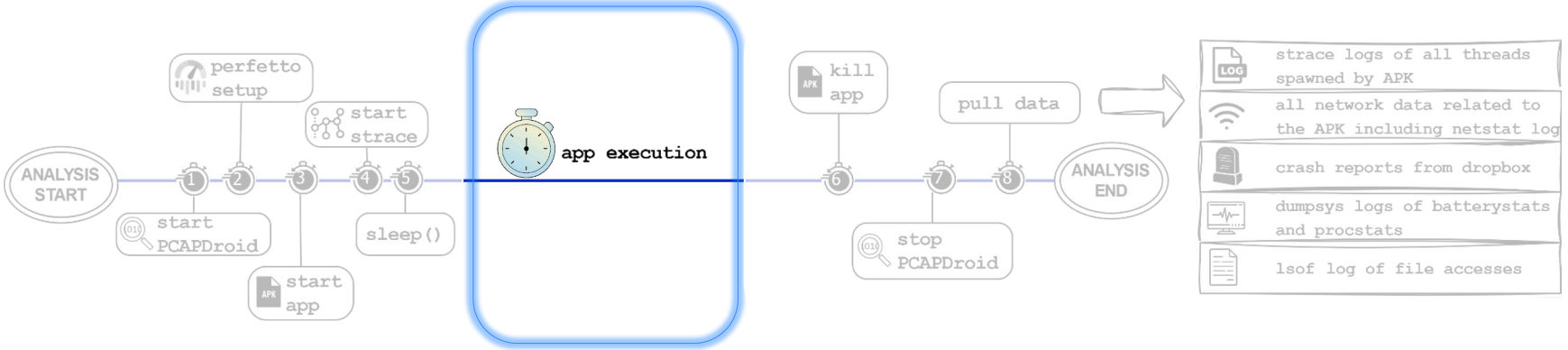




Analysis Phase

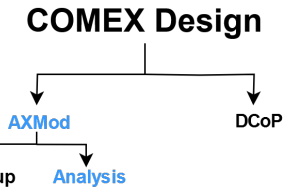
AXMod – Analysis Execution Module

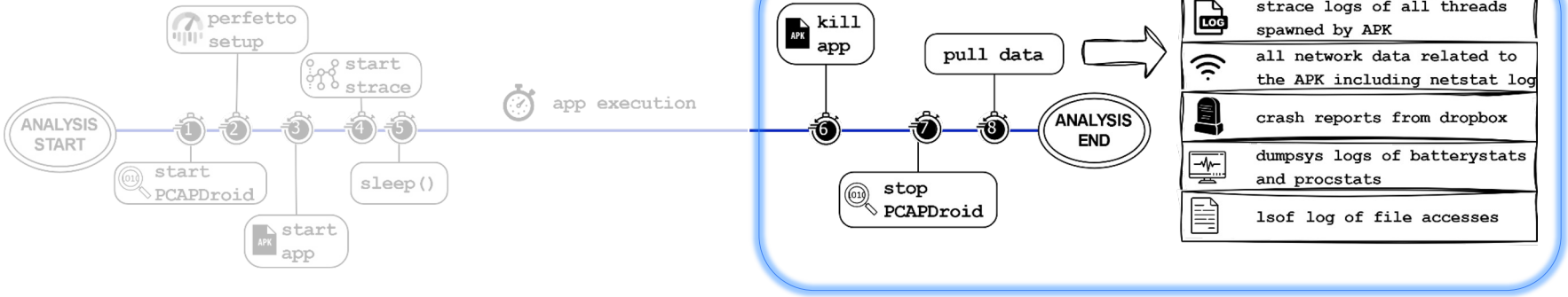




Analysis Phase

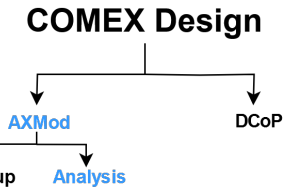
AXMod – Analysis Execution Module

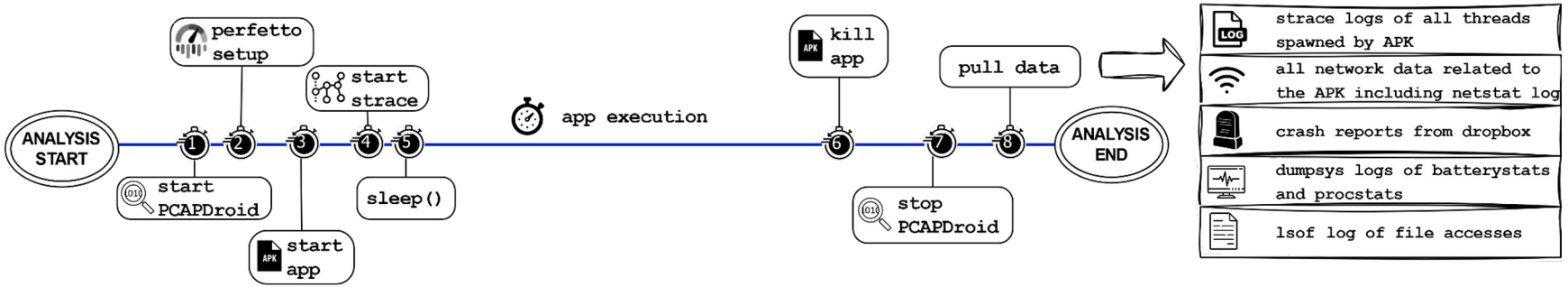




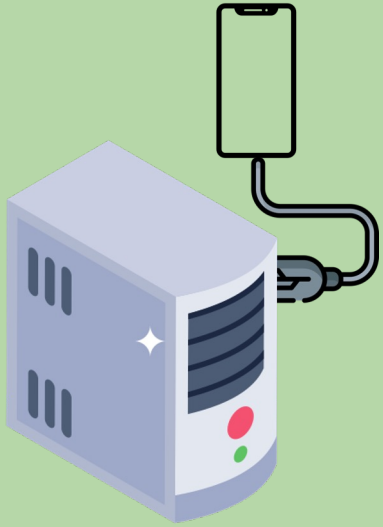
Analysis Phase

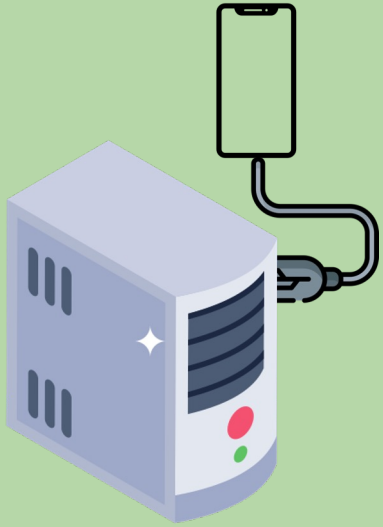
AXMod – Analysis Execution Module



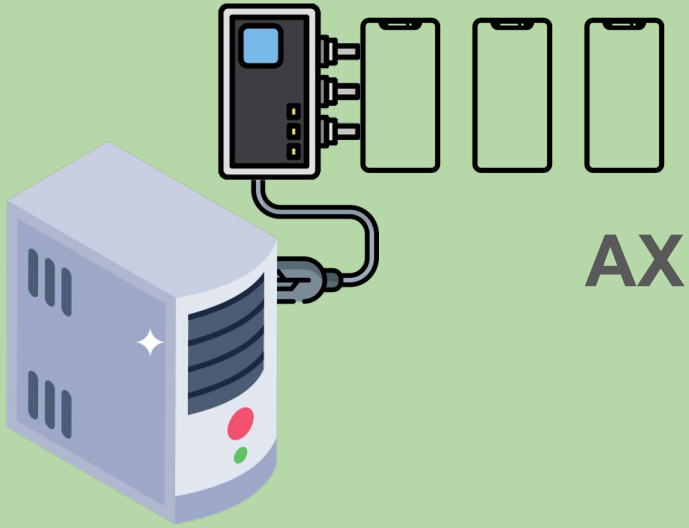


Analysis Phase

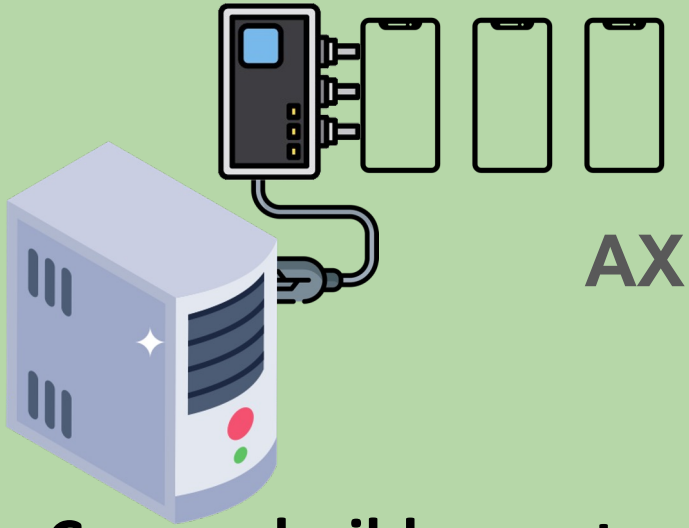




AXMod ✔



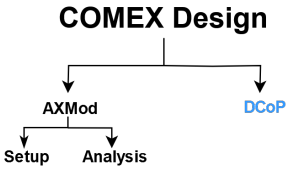
AXMod + ?



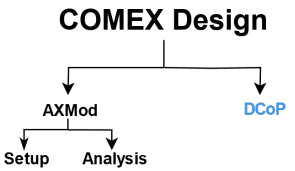
AXMod + ?

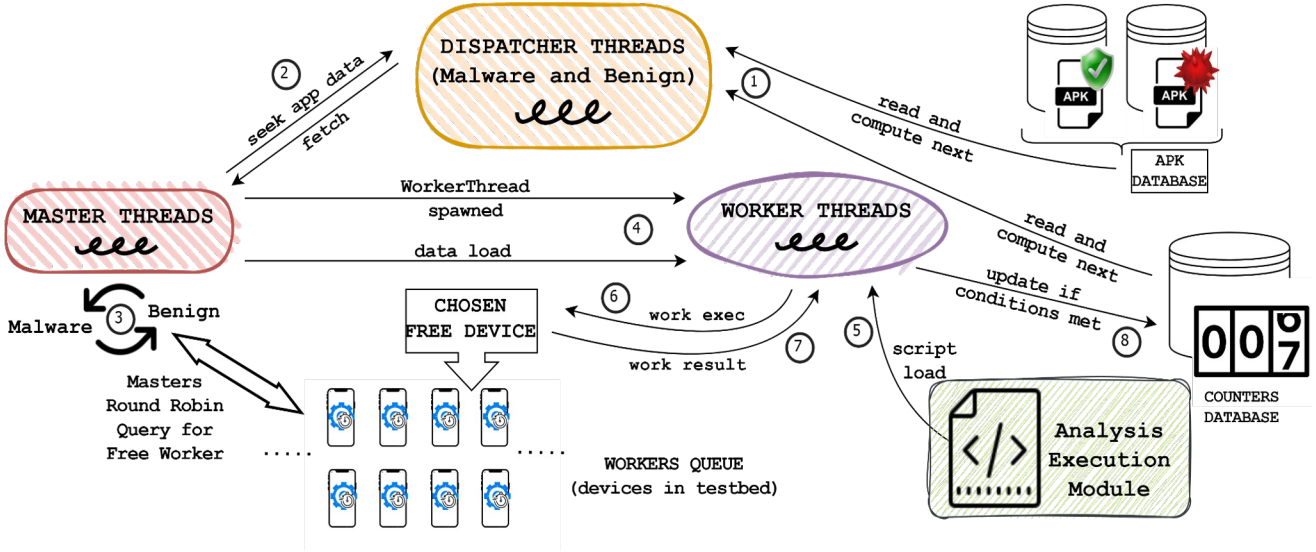
Can we build an automated pipeline for parallel raw data collection of a large dataset?

DCoP – Data Collection Pipeline

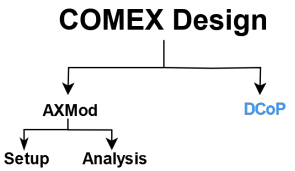


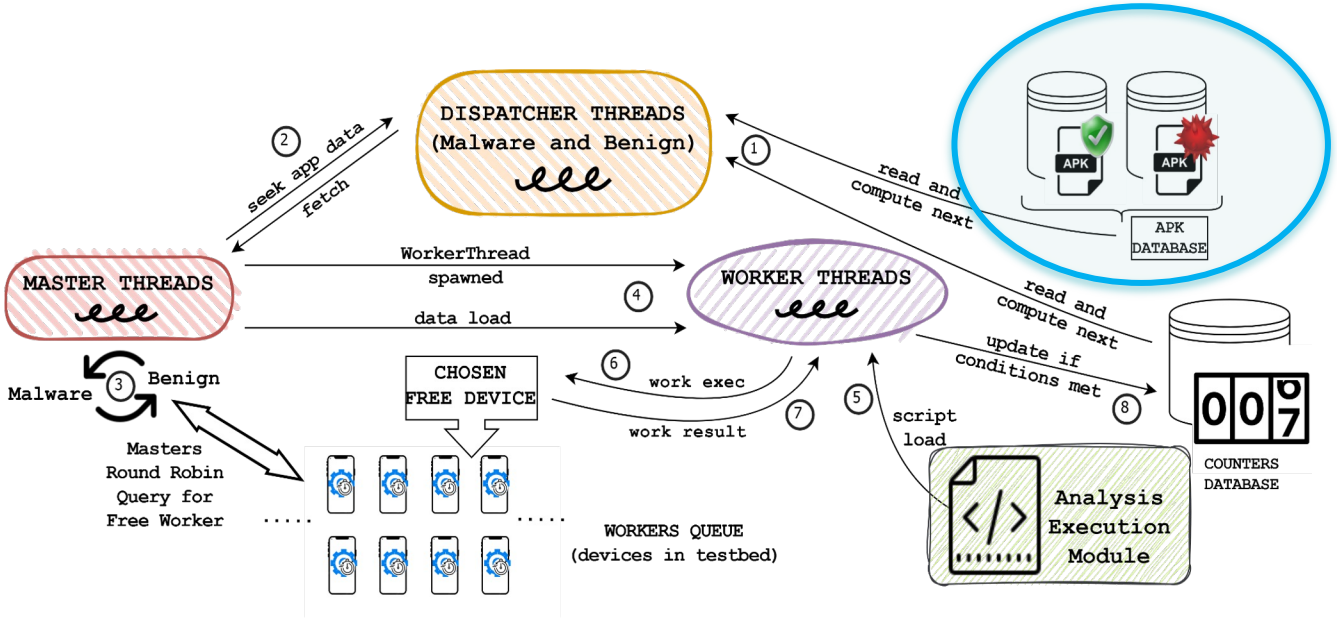
DCoP – Data Collection Pipeline



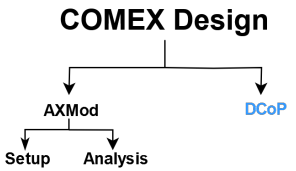


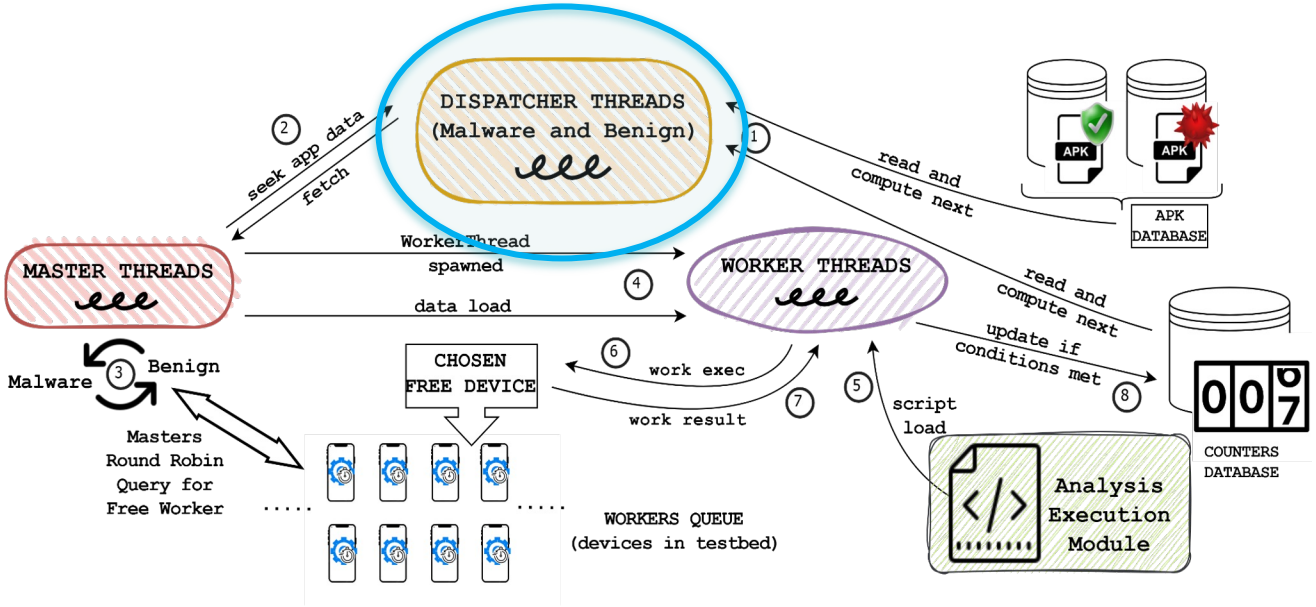
DCoP – Data Collection Pipeline



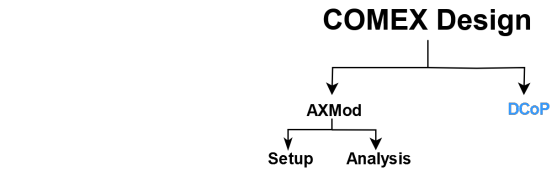


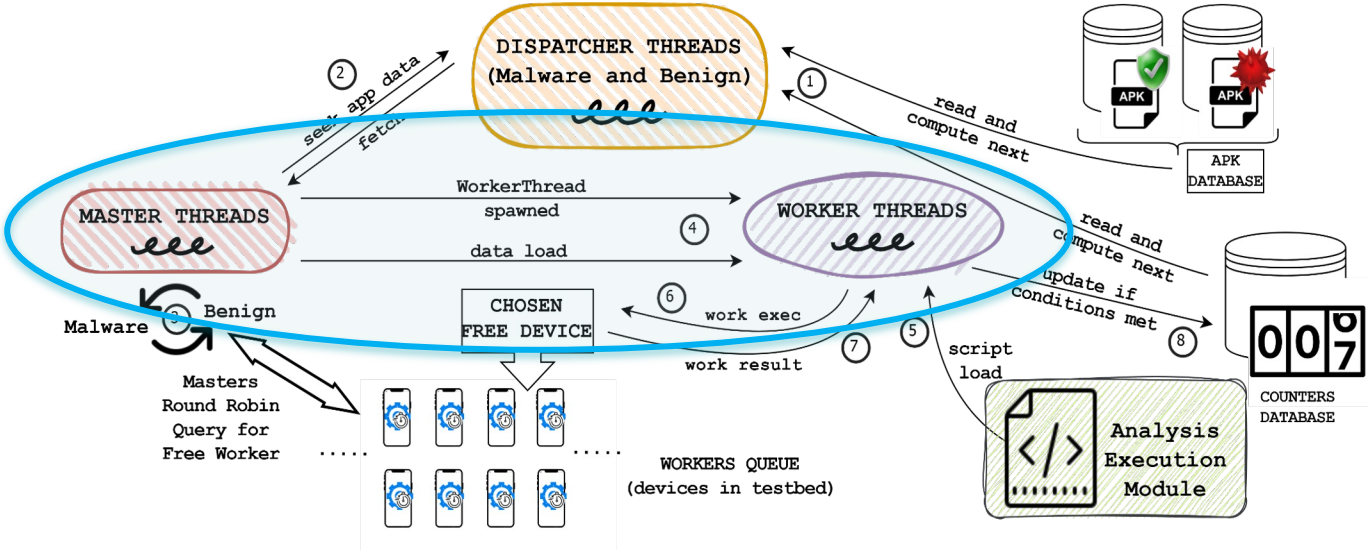
DCoP – Data Collection Pipeline



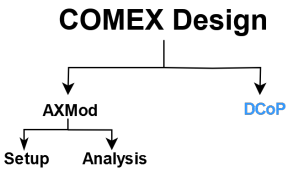


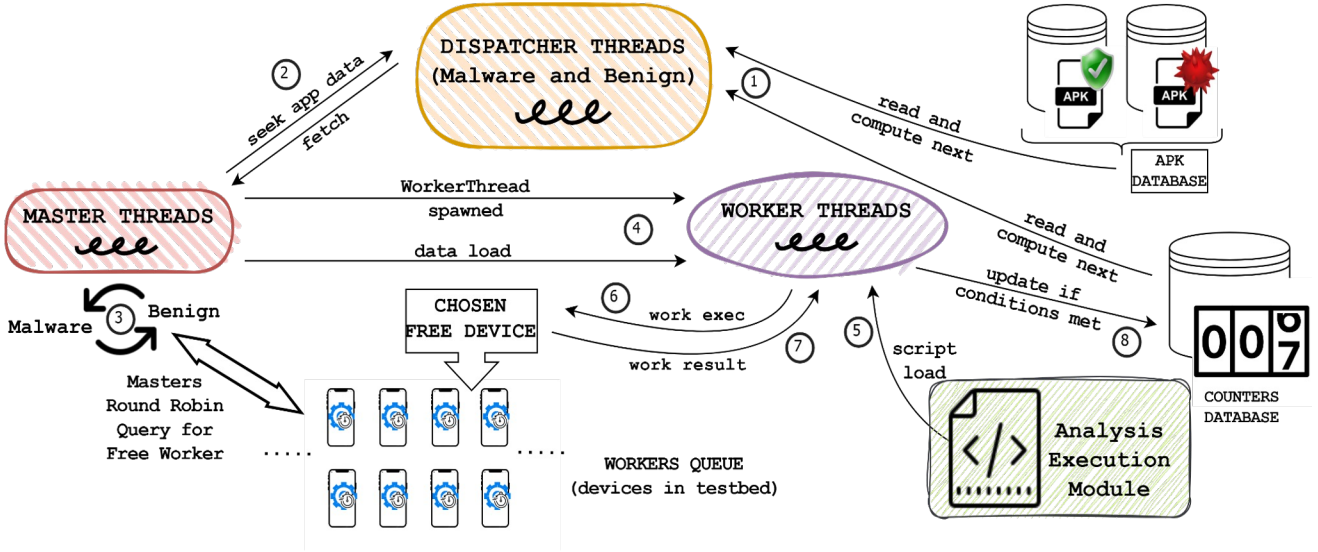
DCoP – Data Collection Pipeline





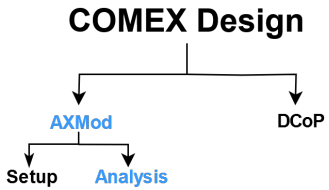
DCoP – Data Collection Pipeline

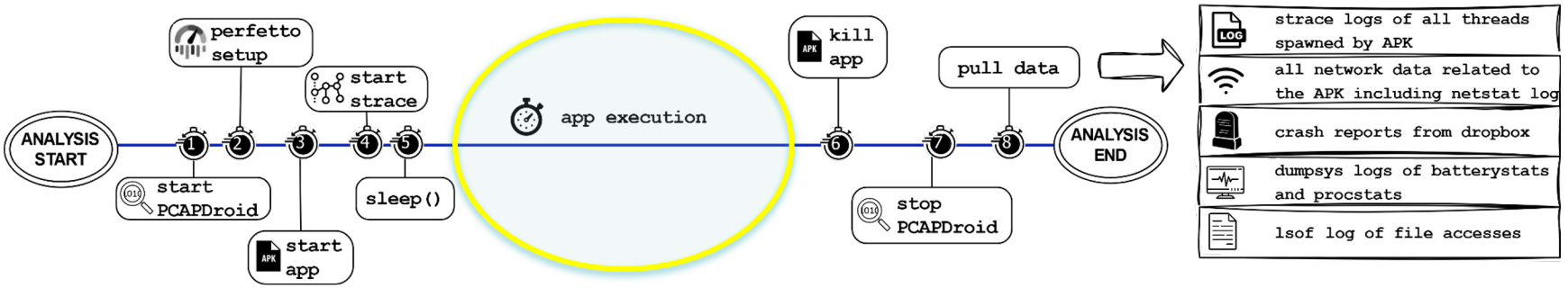




COMEX Module I - AXMod

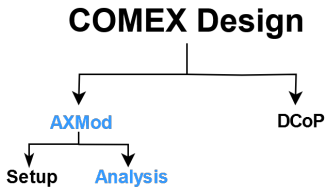
To capture **maximum APK behavior** and, at the same time, **maximize the parallel APK executions**, we need an empirically deduced *approximate* time duration.

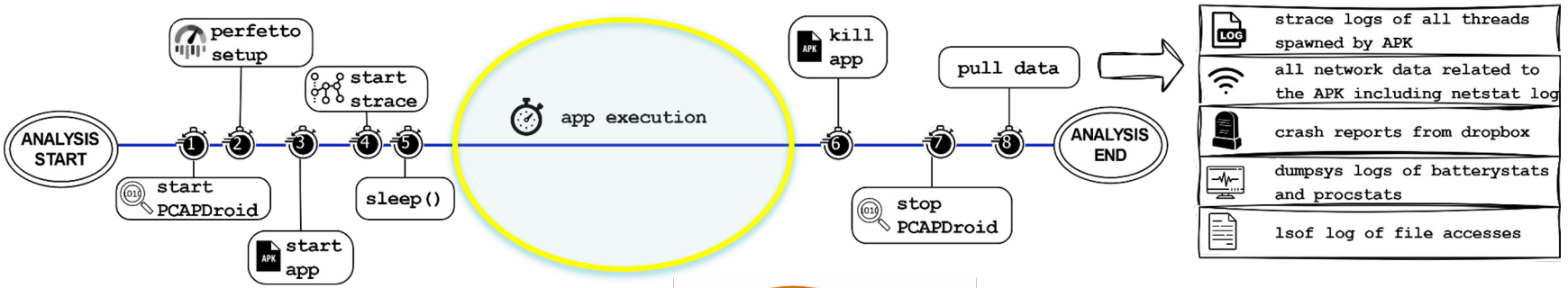




COMEX Module I - AXMod

To capture **maximum APK behavior** and, at the same time, **maximize the parallel APK executions**, we need an empirically deduced *approximate* time duration.





APK Execution Time?

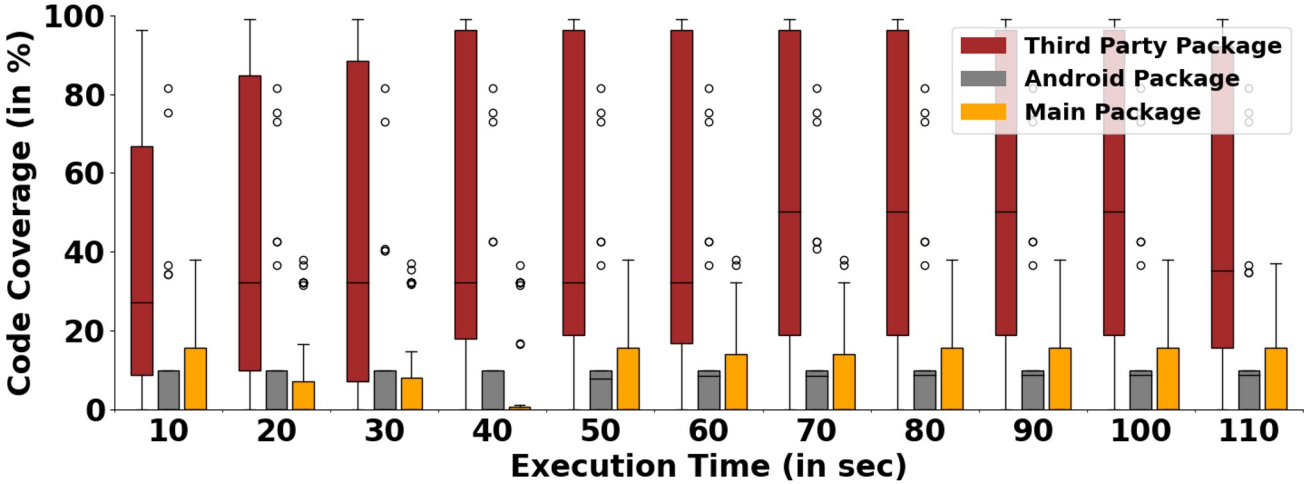
Code Coverage
The percentage of code traversed during APK execution.

ACVTool
Percentage code covered in each *package* of the APK

APK Package Types
- Android official packages
- Main APK package
- Third-party packages

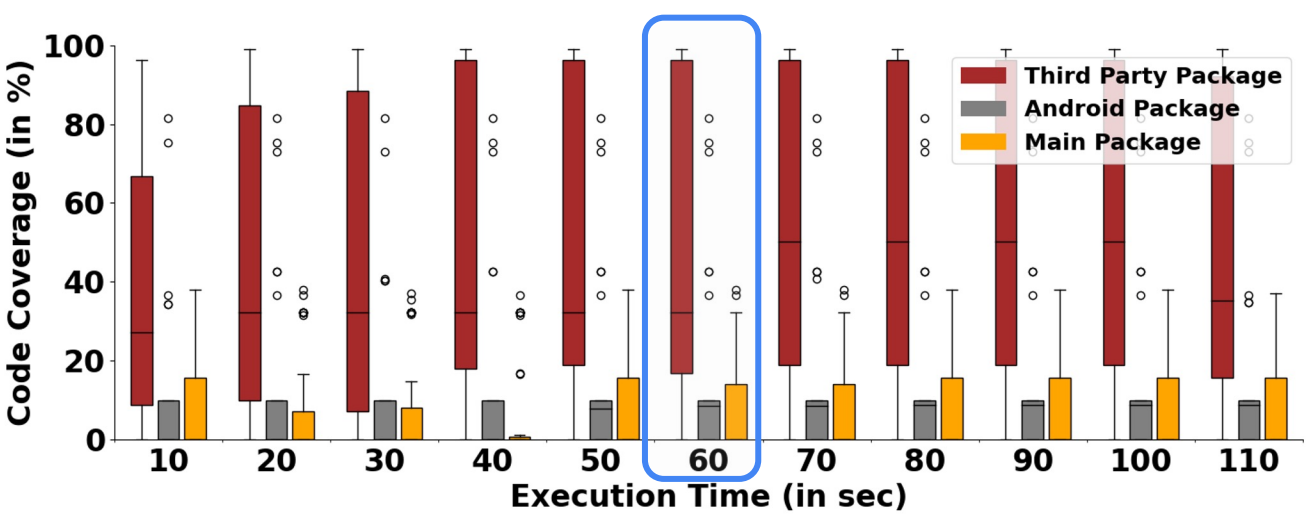
APK Execution Time?

- Code coverage of 500 APKs



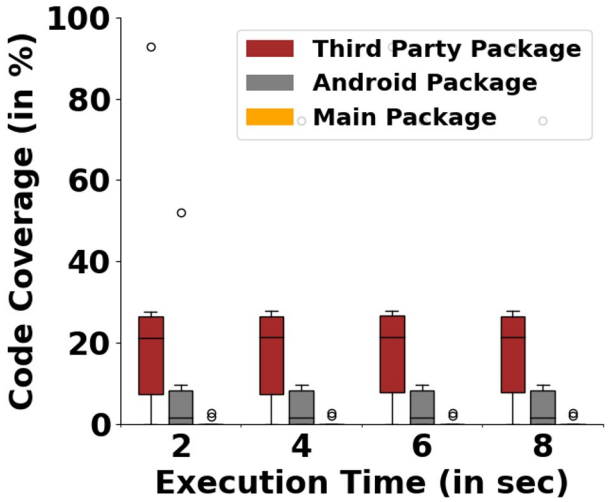
APK Execution Time?

- Code coverage of 500 APKs



APK Execution Time?

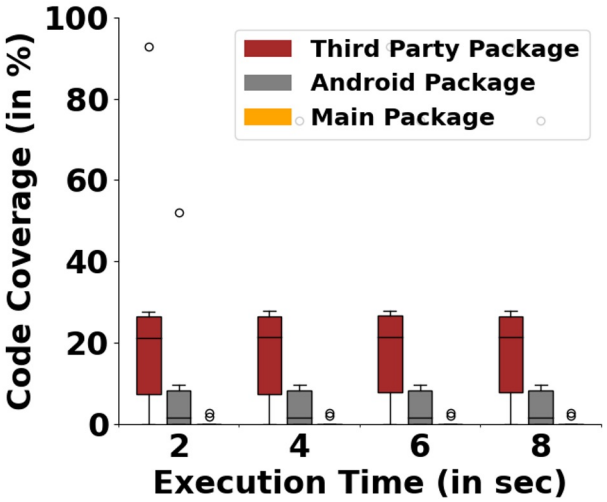
- With varying rates of user-inputs: 5/10/15/20 per sec



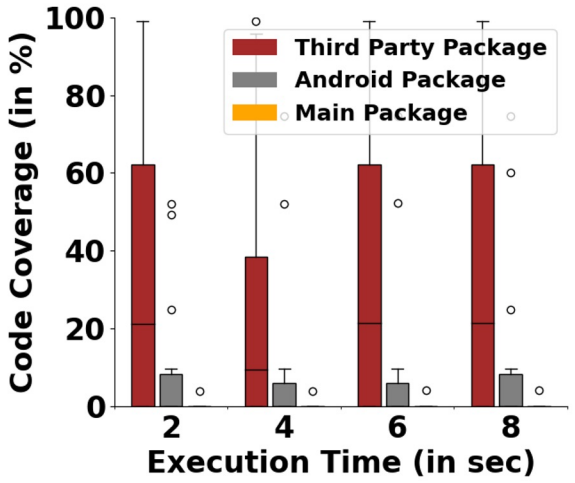
5 user-inputs/sec

APK Execution Time?

- With varying rates of user-inputs: 5/10/15/20 per sec



5 user-inputs/sec



15 user-inputs/sec

APK Execution Time?

Execution time ~60 sec with 15 user inputs/sec

Summary of Data Collection

Types of Raw Data

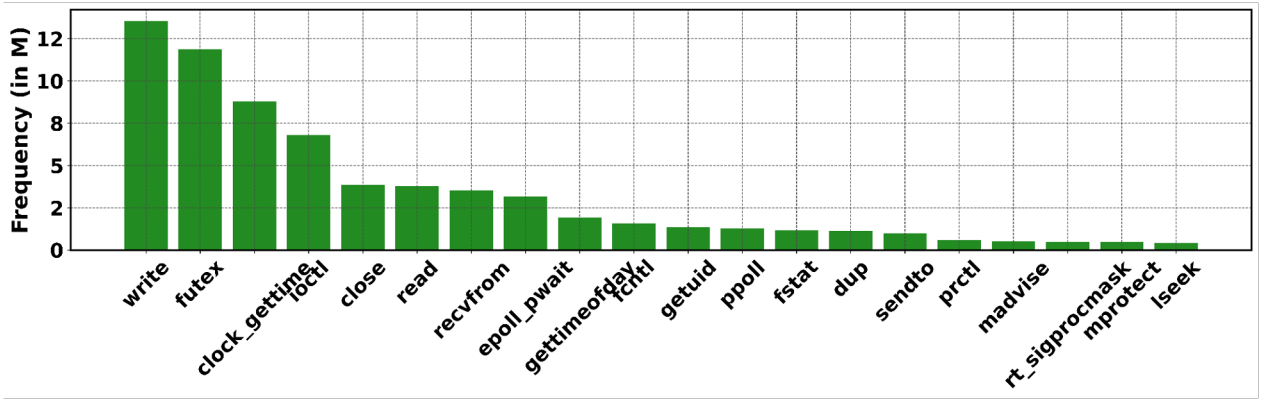
- OS data
  - System calls
  - Binder Transactions
  - List of file descriptors
  - Failed applications
- Network data
  - Traffic related features
  - Network statistics
- Hardware data
  - Battery status
  - Procstats

Summary of 1000 APK analysis

- Pipeline details
  - 6 Motorola G40 devices
  - Android v12
- Amount of raw data collected
  - 400 GBs

Variety of Data Collection

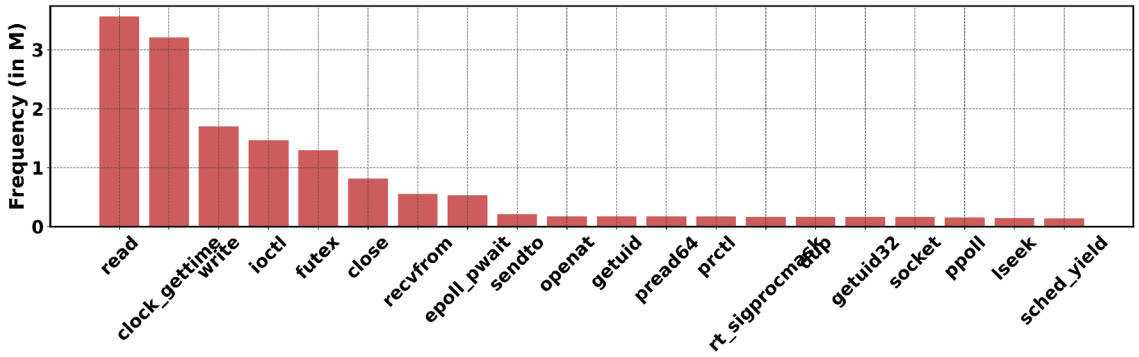
OS Level Data - system calls for Benign APKs



Top 20 System Calls

Variety of Data Collection

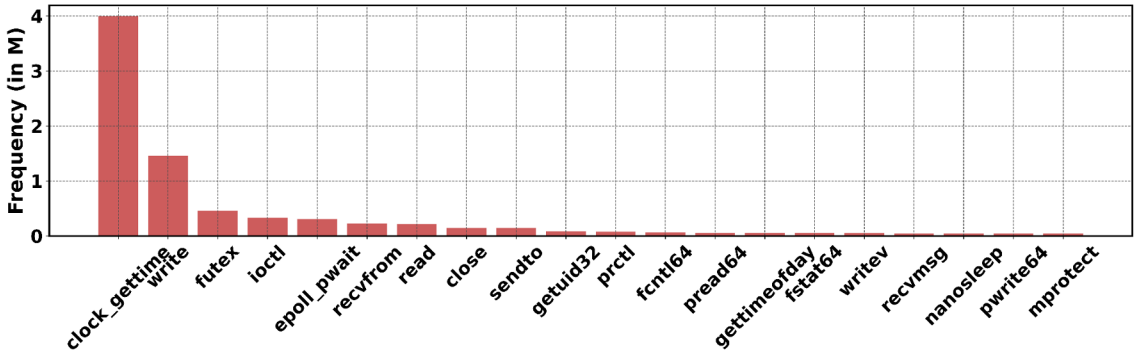
OS Level Data - system calls for Malware APKs



Family - **jiagu**

Variety of Data Collection

OS Level Data - system calls for Malware APKs



Family - **smsreg**

Variety of Data Collection

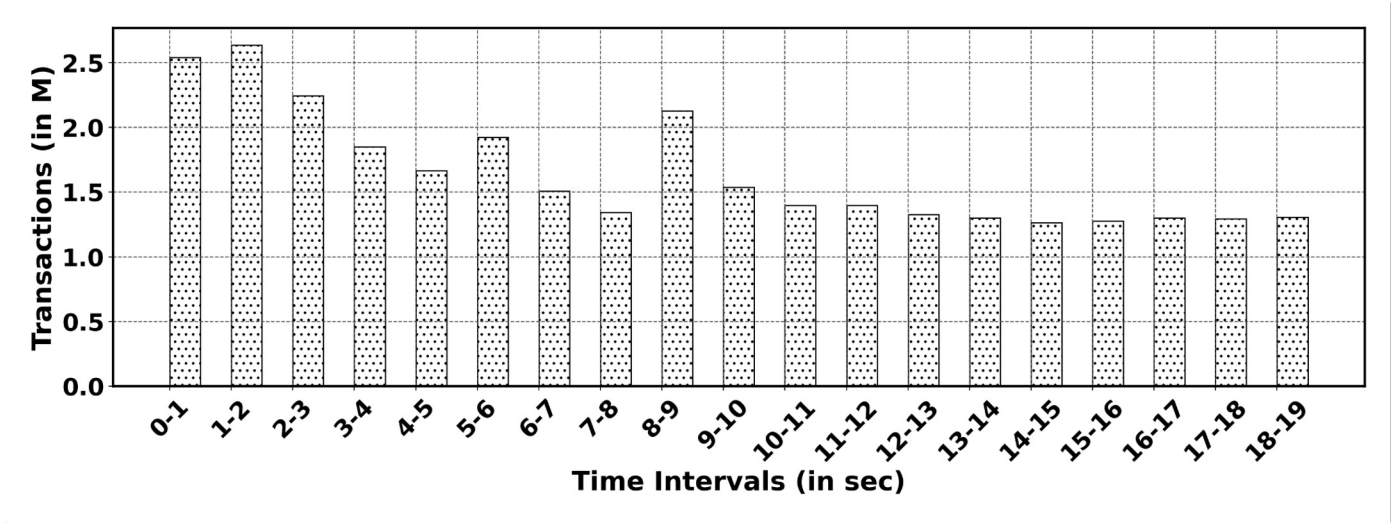
OS Level Data - Binder Transactions

Binder is an inter-process communication (IPC) mechanism in Android OS.

Variety of Data Collection

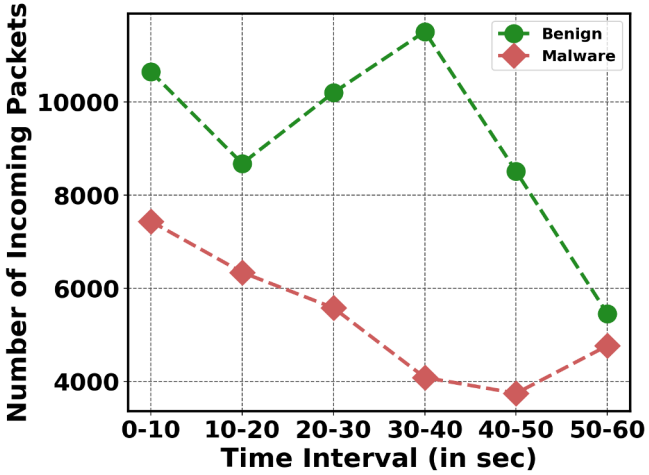
OS Level Data - Binder Transactions

Binder is an inter-process communication (IPC) mechanism in Android OS.



Variety of Data Collection

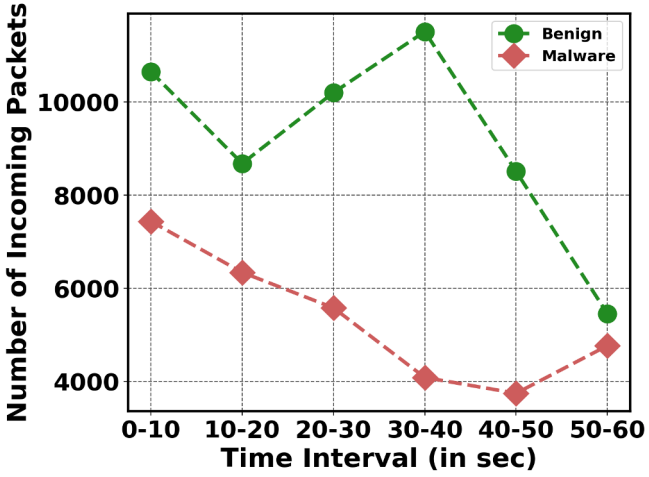
Network Level Data - cumulative packets of 500 Benign + 500 Malware APKs



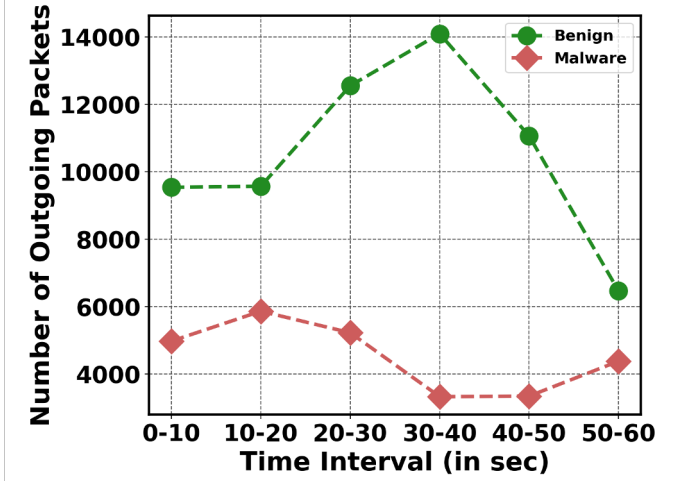
Incoming Network Packets

Variety of Data Collection

Network Level Data - cumulative packets of 500 Benign + 500 Malware APKs



Incoming Network Packets



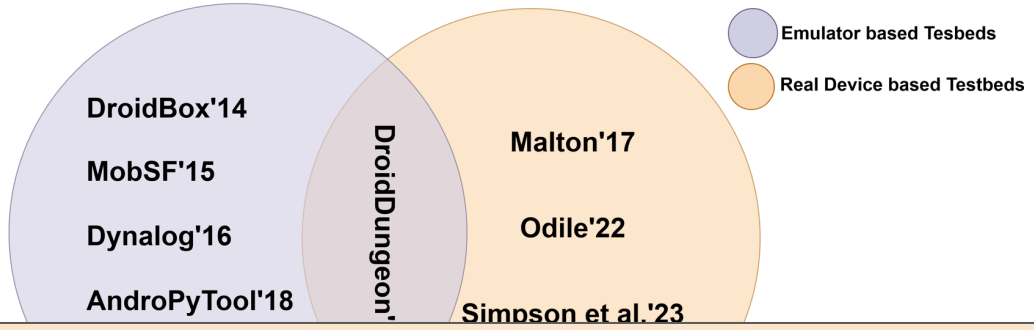
Outgoing Network Packets

Use-case of COMEX Data

- Scale-up data collection for comprehensive analysis
- Enhance detection mechanisms by correlating different data sources

Data Source	Possible extraction of raw data	Enhancement to diff works
System	System calls, binder transactions, files accessed, crashes *etc.*	Zhang *et al.* [87] Mercaldo *et al.* [59] DroidScribe [33], Andromaly [73], Chimera [34], EavesDroid [83]
Network	SrcPkts, DstPkts, SrcBytes, DstBytes, SAppBytes, DAppBytes, *etc.*	Fu *et al.* [42], Heldroid [19] DroidScribe [33] Dine *et al.* [35], Andromaly [73], Hybroid [65]
Hardware	CPU utilization, battery status, *etc.*	PowerSpy [61], Andromaly [73], Cabral *et al.* [29]

Previous Android Testbeds



None of them provide data from all three sources.

- All require at least one type of **Instrumentation** *except Simpson et al.*
- Although **mobSF** and **AndroPyTool** are publicly available and functional but these are emulator based solutions and does not provide data from all three sources.
- Rest are either non-functional or do not provide their codebases.

Challenges

- Providing consistent power to devices
- Non-parallelism in `monkeyrunner`
- Data Storage

Conclusion

- We present COMEX, a testbed for performing dynamic analysis on **real Android devices**.
- It does not require instrumentation.
- It is **modular** in nature -
  - AXMod
  - DCoP
- It provides maximum possible **raw data** as well as **processed data** from all sources - OS + Network + Hardware.
- To promote reproducibility we have made the source code and analysis scripts public.

https://github.com/zeya2u9/COMEX

Extras Ahead

AXMod Time Analysis

Setup Time (in sec)			Analysis Time (in sec)		
Steps	**10MB**	**24MB**	**Steps**	**10MB**	**24MB**
1	122.16	120.17	1	14.41	15.10
2	1.92	1.69	2	0.002	0.003
3 & 4	14.08	15.33	3	4.76	3.29
5	64.33	64.01	4	0.90	0.68
6	3.39	3.27	5	6.21	6.15
7	3.25	3.13	6	65.43	64.68
8	14.27	14.21	7	74.31	79.05
9	65.95	66.40	8	3.16	3.98
10	6.06	7.12	9	6.30	6.23
11	19.44	19.48	10	2.30	3.95
12	0.15	0.17	11	1.00	1.21
Total	315.01	314.98	Total	178.79	184.32

Here "Steps" refer to the steps of setup and analysis phase.

Code Coverage Analysis

ACVTool Workflow

Steps:

1. Instrument the original APK with ACVTool [instrument --wd <working_dir>]
2. Install the instrumented APK in the Android emulator or device. [install]
3. Activate the app for coverage measurement [activate <package_name>] (alternatively, [start <package_name>])
4. Test the application (launch it!)
5. Make a snap [snap <package_name>]
6. Apply the extracted coverage data onto the smali code tree [cover-pickles <package_name> --wd <working_dir>]
7. Generate the code coverage report [report <package_name> --wd <working_dir>]

Static V/S Dynamic

Example

Trojan Dropper 2018 sample

`09575e22c395f5e538b2987c69d47722bcbe69de969bdd8f7bc2dfa7d979f88a`

- Static features:
  - Permissions: ACCESS_FINE_LOCATION, ACCESS_NETWORK_STATE, *etc.*

- Dynamic Features:
  - Downloaded files
    - ELF library libcom.art.roct.so
    - 2 APKs (an adware and a benign file)