Matrioska: A User-Centric Defense Against Virtualization-Based Repackaging Malware on Android

Samuele Doria
samueledoria@gmail.com

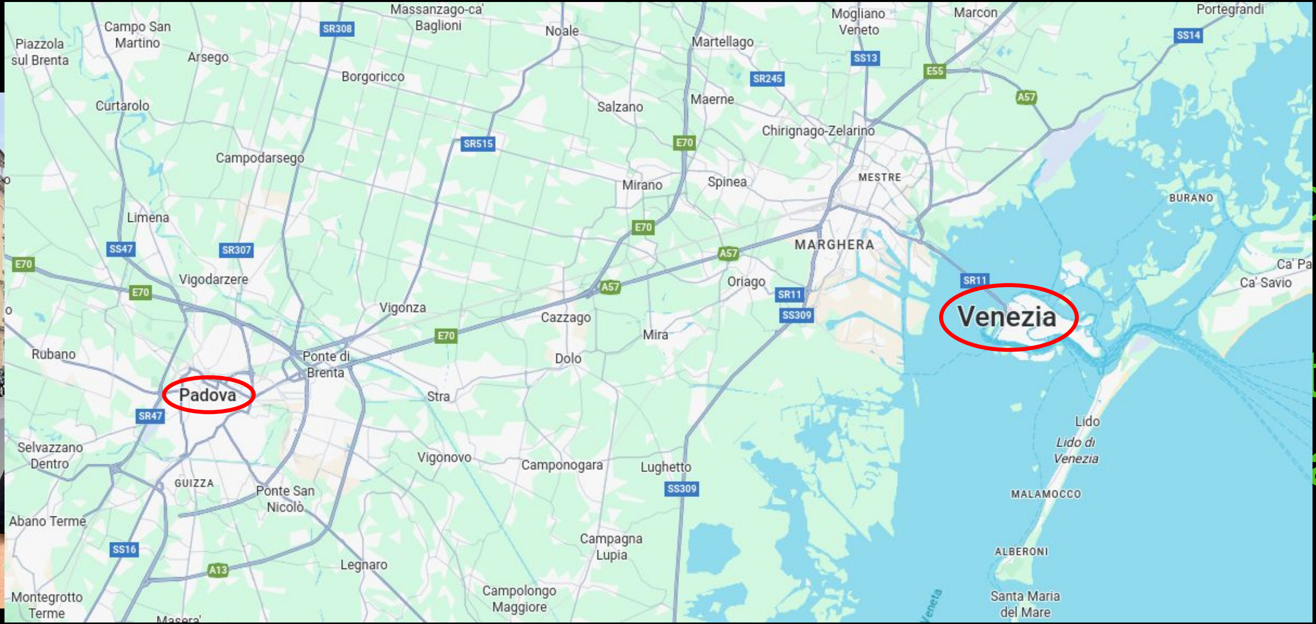
Source: Chat-GPT

MOBILE HACKING COMMUNITY @ DEF CON 33

$ whoami

- PhD Student @ University of Padova
- Researching Software Security, focusing on Android
  - Malware detection; static, dynamic and hybrid analysis; etc...

$ whoami



of Padova

curity,

tic,

alysis;

$ cat talk.txt

R+R: Matrioska: A User-Centric Defense Against Virtualization-Based Repackaging Malware on Android

Simone Zerbini
Department of Mathematics
University of Padua
Padua, Italy
simone.zerbini@unipd.it

Samuele Doria
Department of Mathematics
University of Padua
Padua, Italy
sdoria@math.unipd.it

Primal Wijesekera
International Computer Science Institute
Berkeley, USA
primalw@icsi.berkeley.edu

Serge Egelman
International Computer Science Institute /
University of California Berkeley
Berkeley, USA
egelman@cs.berkeley.edu

Eleonora Losiouk
Department of Mathematics
University of Padua
Padua, Italy
eleonora.losiouk@unipd.it

Full paper

Why care about this talk

- Listen to some cool **academic research** (unbiased opinion :))
- You like Android, malware, and dynamic analysis
- You want to know what **virtualization** on Android is
  - spoiler: it’s different from traditional virtualization

Why it's important

- Recently, new malware samples using virtualization have been found [1]
- Zimperium discovered **29** new malicious apps using virtualization containers
  - **Hundreds of thousands** of installs across third-party markets

[1] https://zimperium.com/blog/your-mobile-app-their-playground-the-dark-side-of-the-virtualization

Why it’s important

- The malware includes:
  - A **user-facing app** (e.g., fake banking/utility app)
  - An embedded plugin APK that’s **silently loaded**
- Malware behavior happens inside the plugin, **out of reach** of most defenses

Why it’s important

- Plugin apps **downloaded at runtime** from C2 servers
- Threats: Credential theft, overlay attacks, ad fraud, spyware, and more
- Detection Challenges: No APK installation, **hides inside a Virtual Environment**

But, what is a Virtualization-Based Repackaging malware?

- Repackaging malware is a **modified version** of a legitimate app where malicious code is injected.

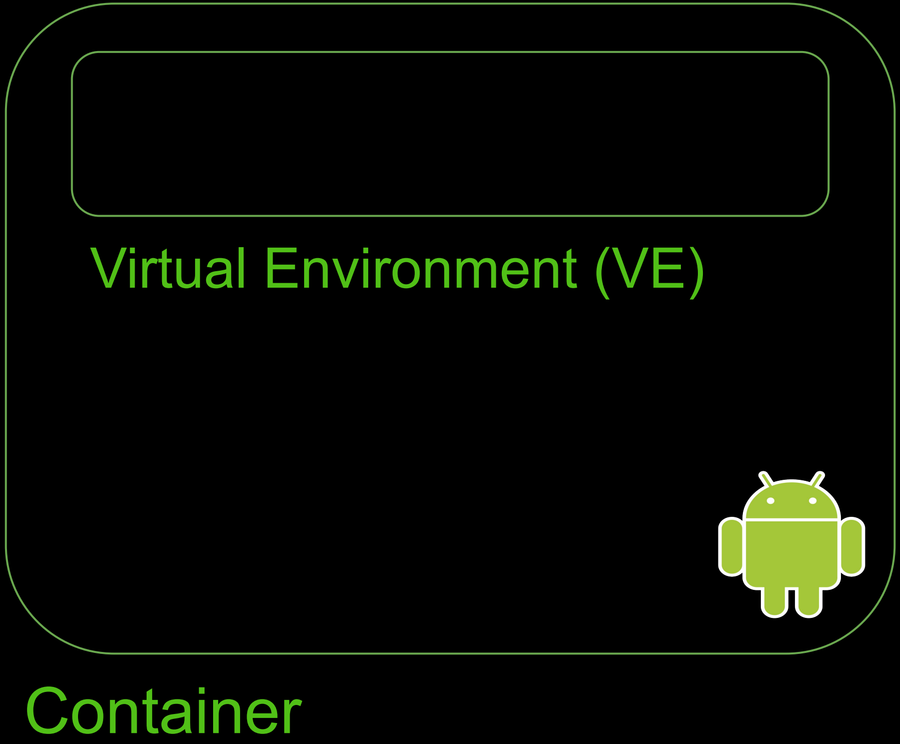
- Virtualization-Based Repackaging (VBR) malware takes advantage of **virtualization** to deploy the attacks.

But, what is Android’s Virtualization?

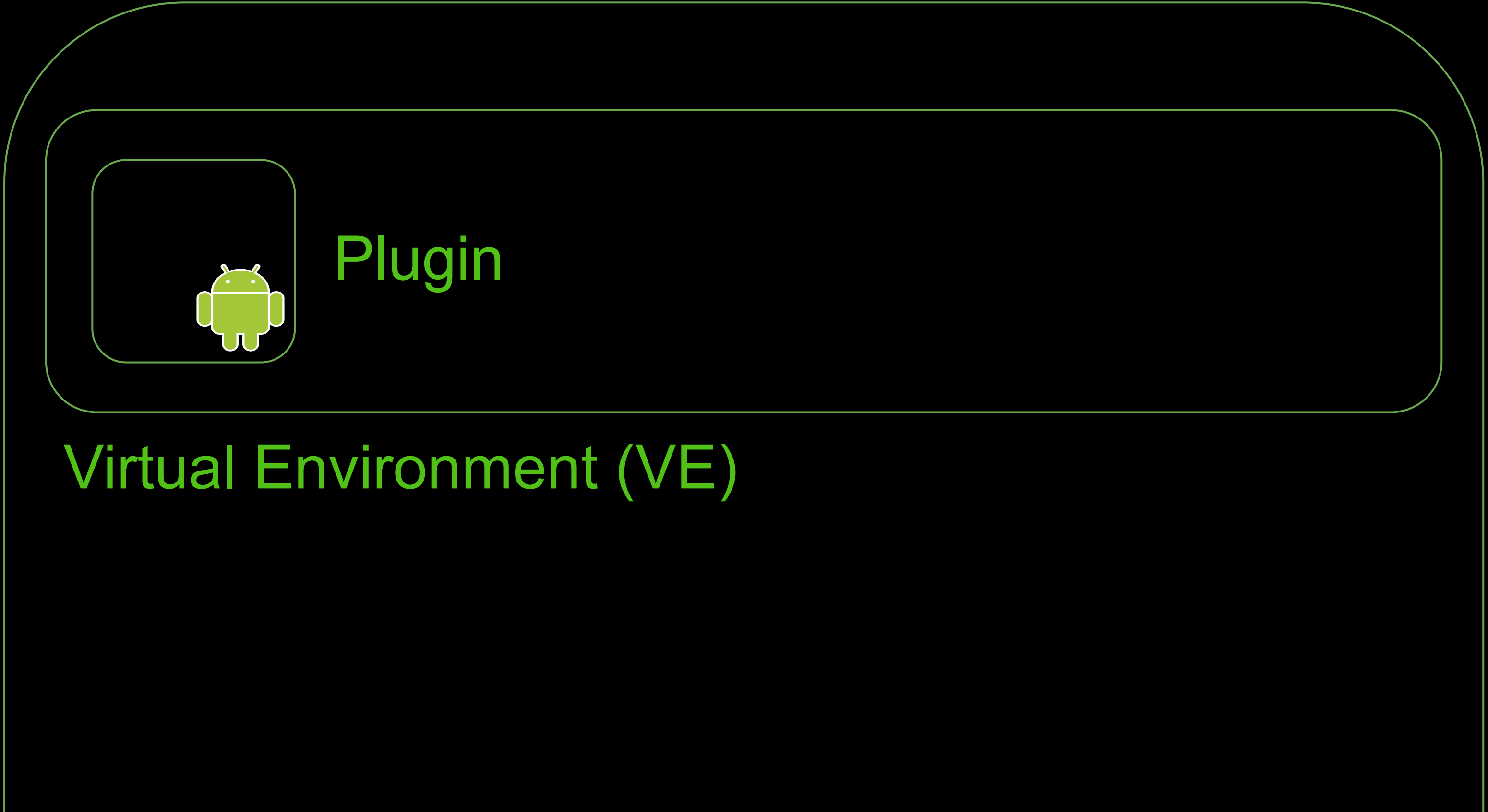
The Android Virtualization Technique

Container

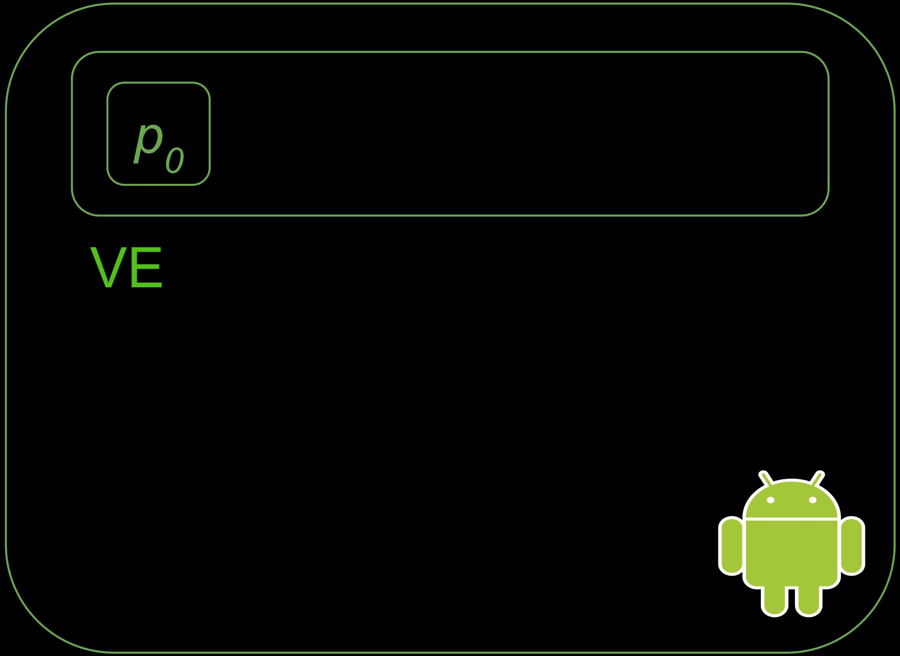
The Android Virtualization Technique



The Android Virtualization Technique



The Android Virtualization Technique



Container

The Android Virtualization Technique

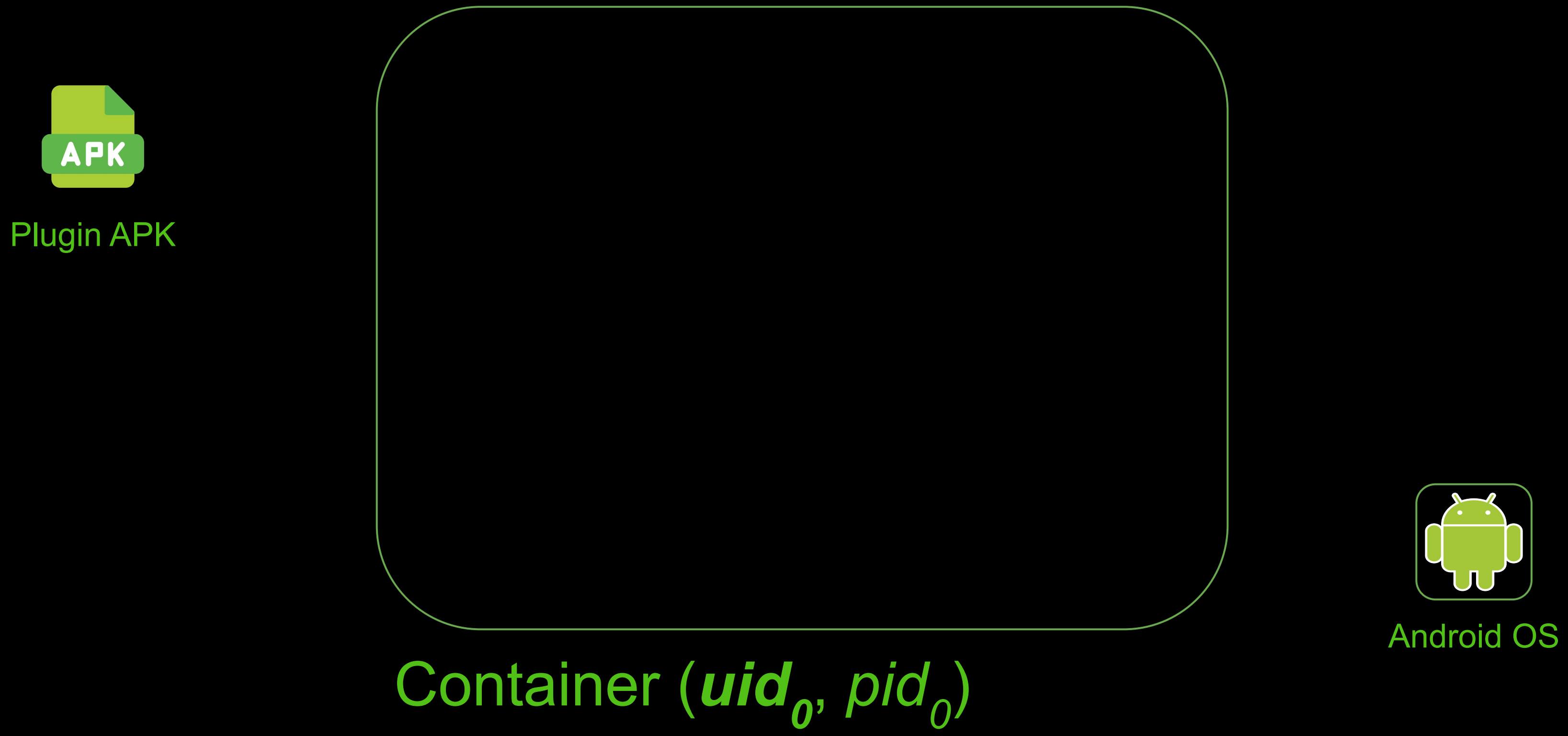
Container

Characteristics:

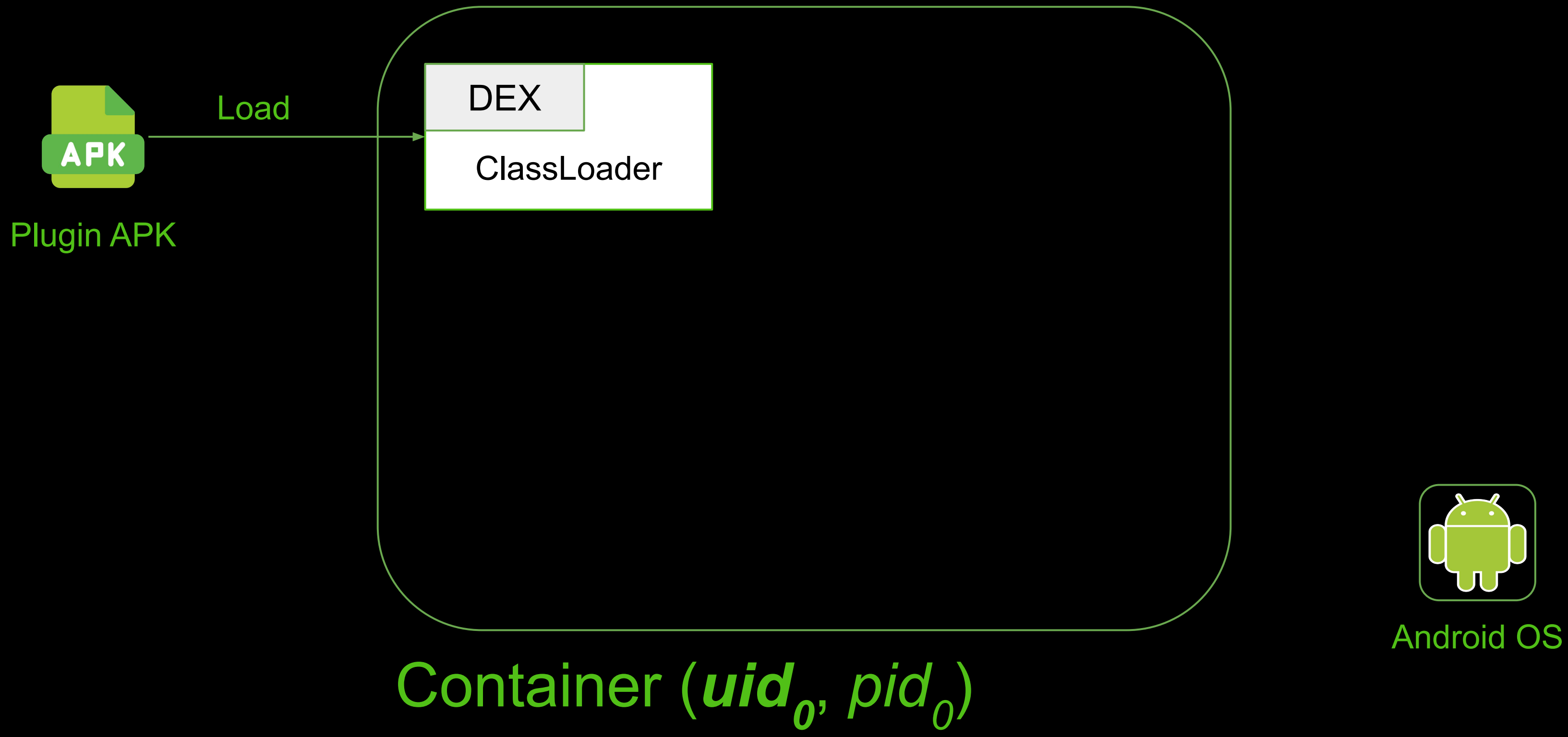
- **sharing** the UID and the permissions
- breaking app **isolation**

- p_i = plugin(s)

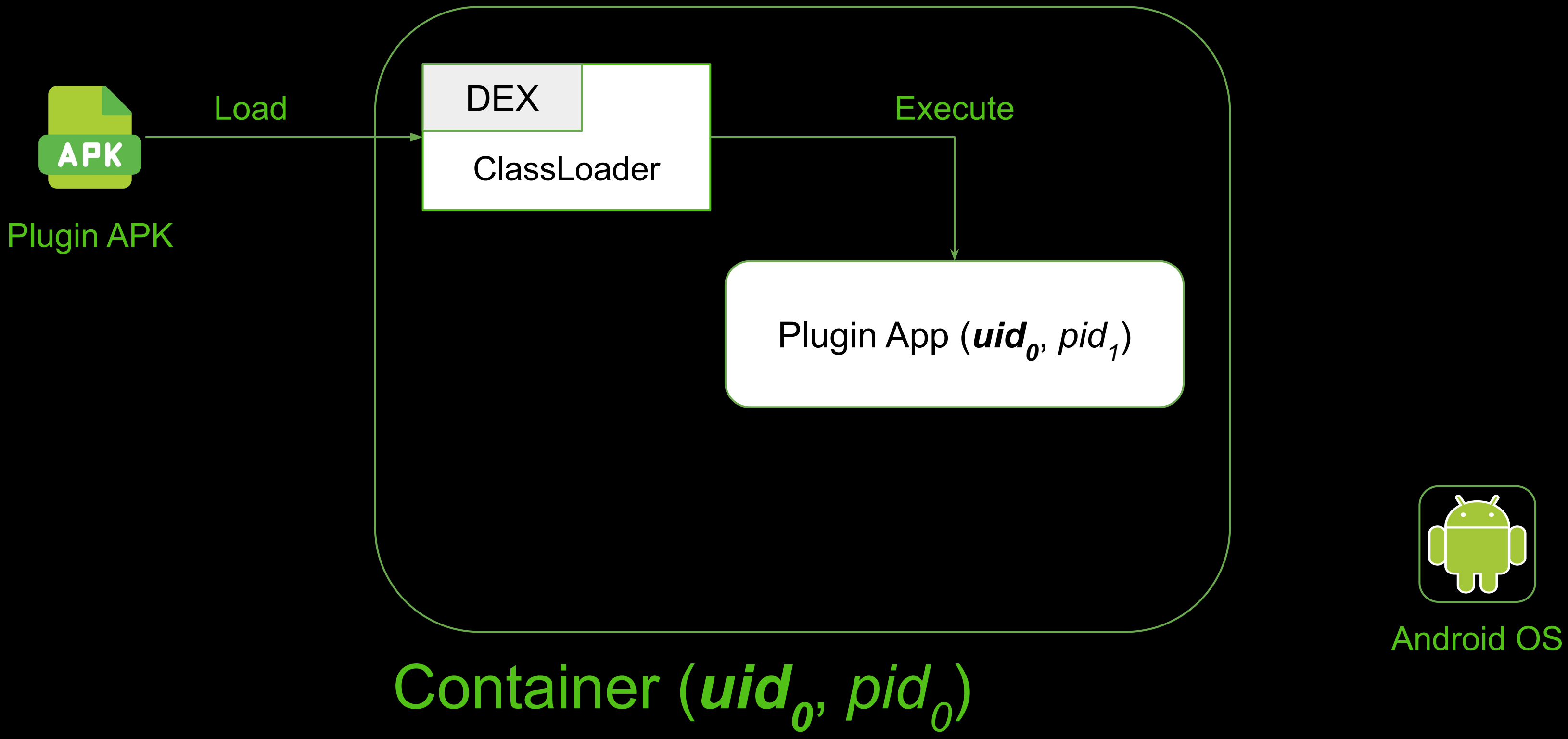
The Android Virtualization Technique



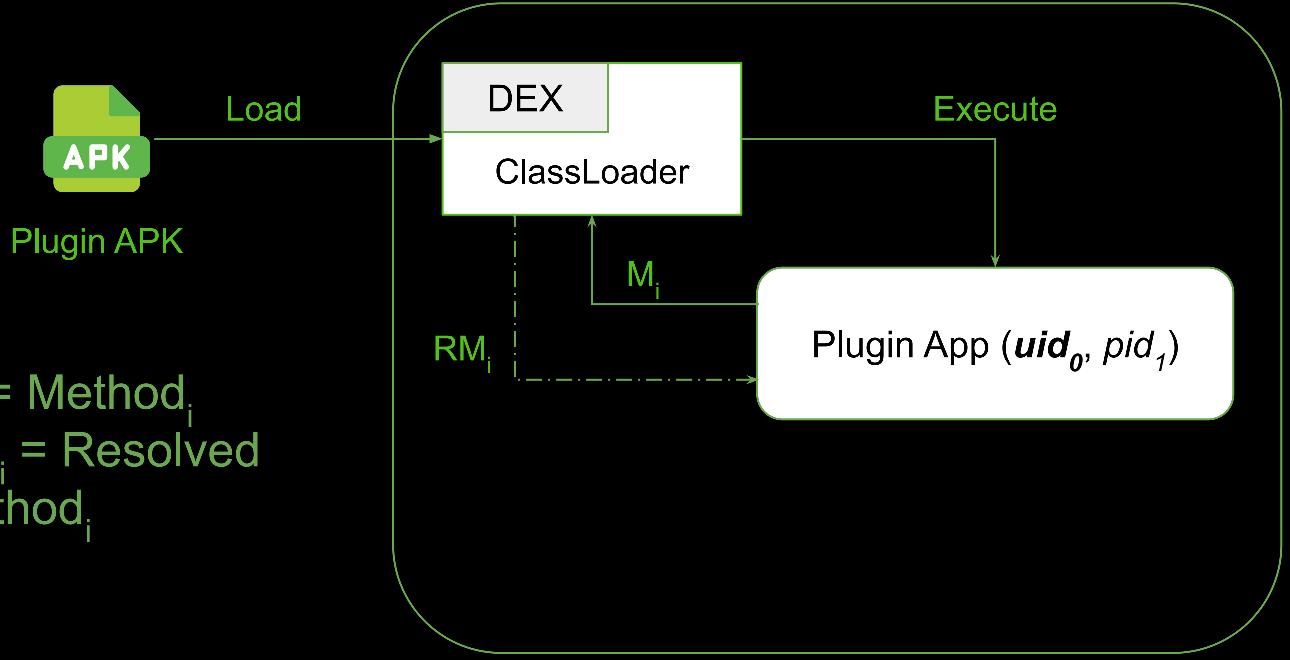
The Android Virtualization Technique



The Android Virtualization Technique



The Android Virtualization Technique

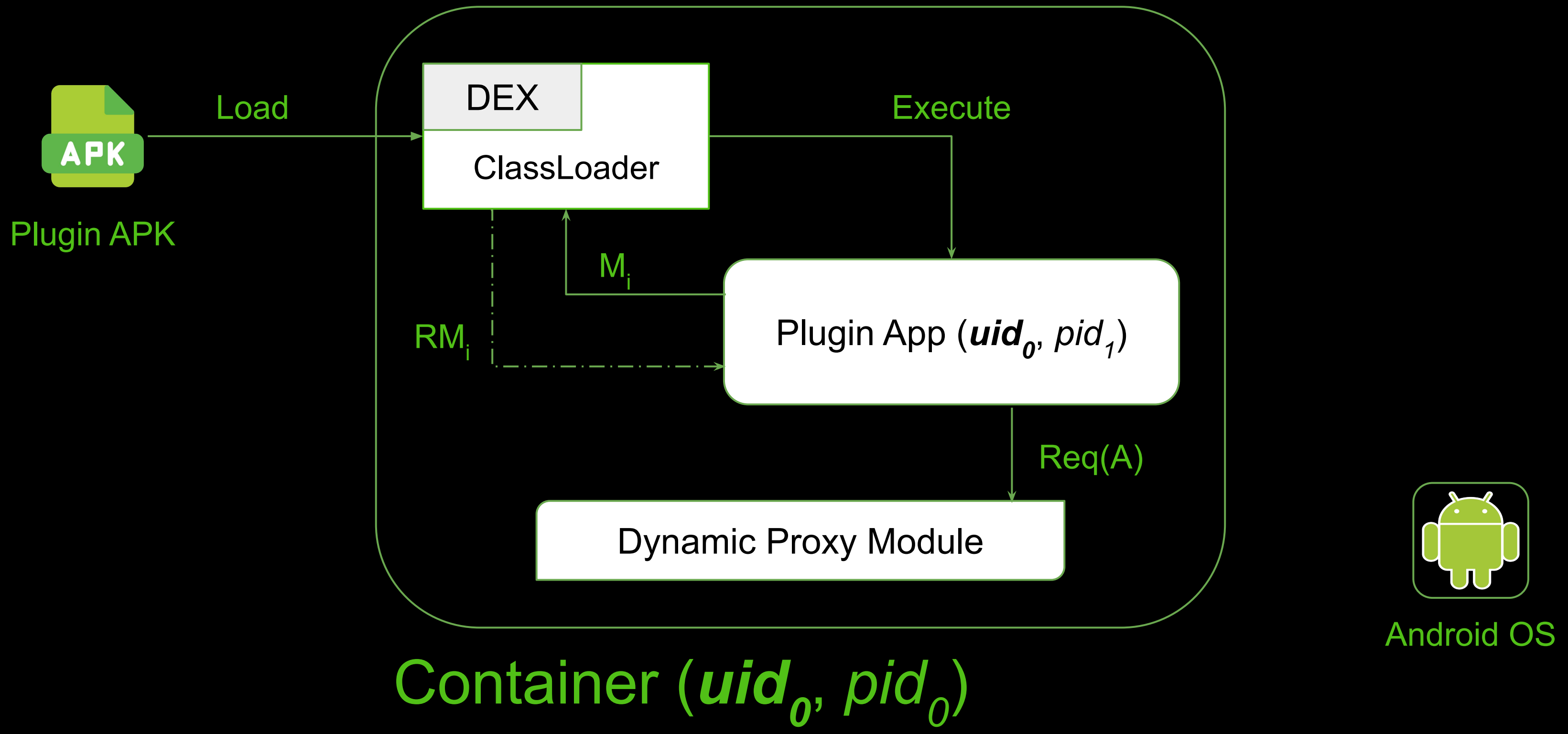


Container ($\boldsymbol{uid_0}$, pid_0)

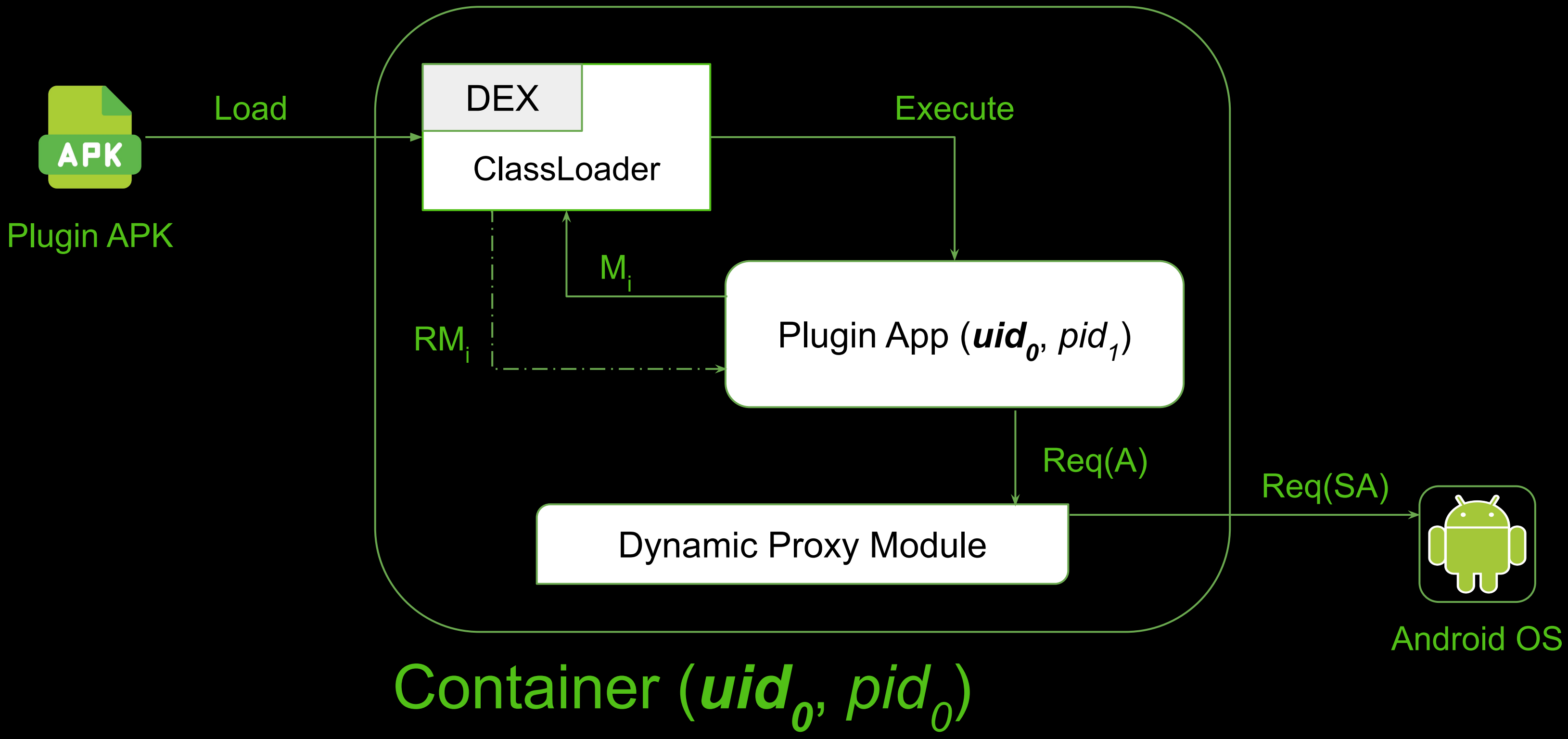
M_i = Method_i
RM_i = Resolved Method_i

Android OS

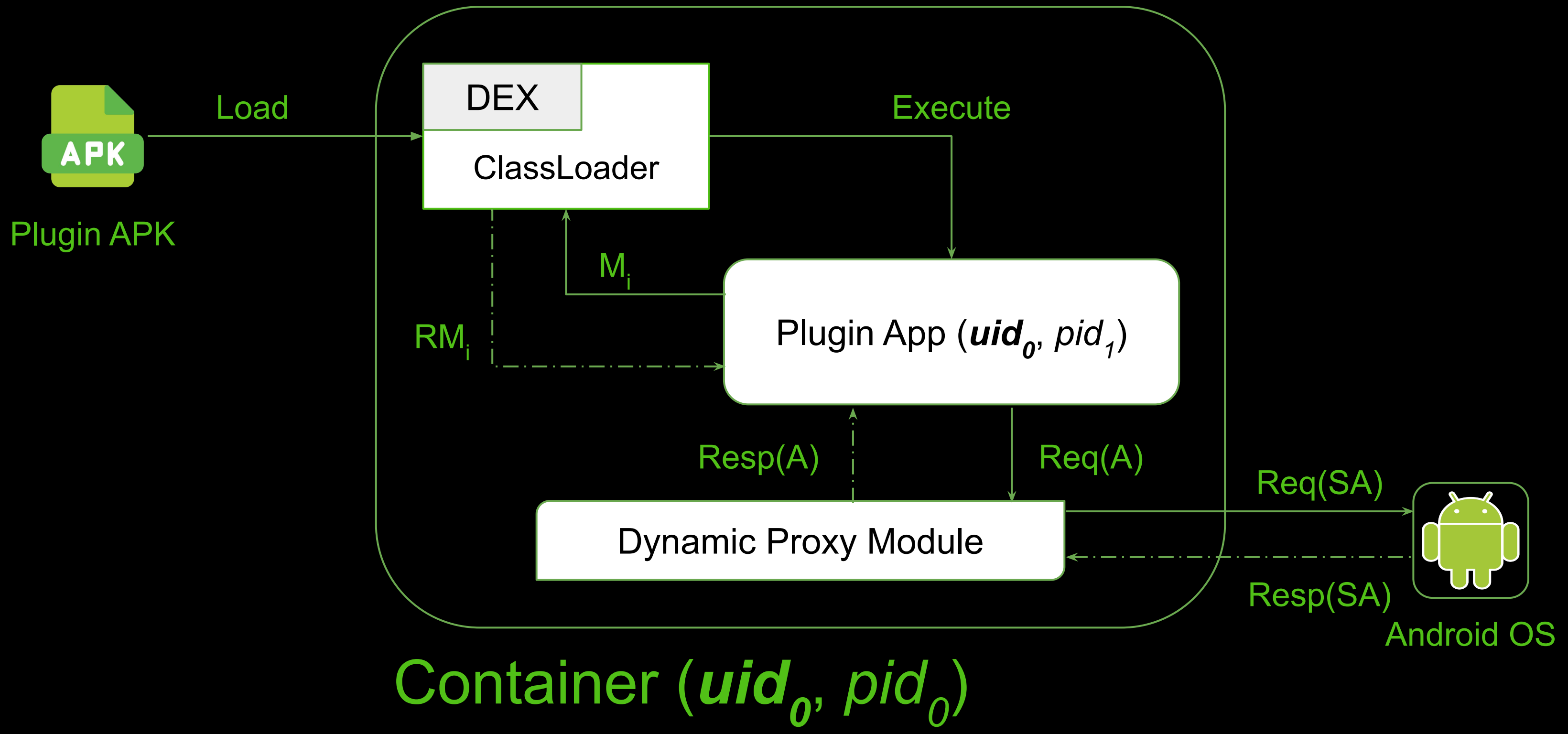
The Android Virtualization Technique



The Android Virtualization Technique



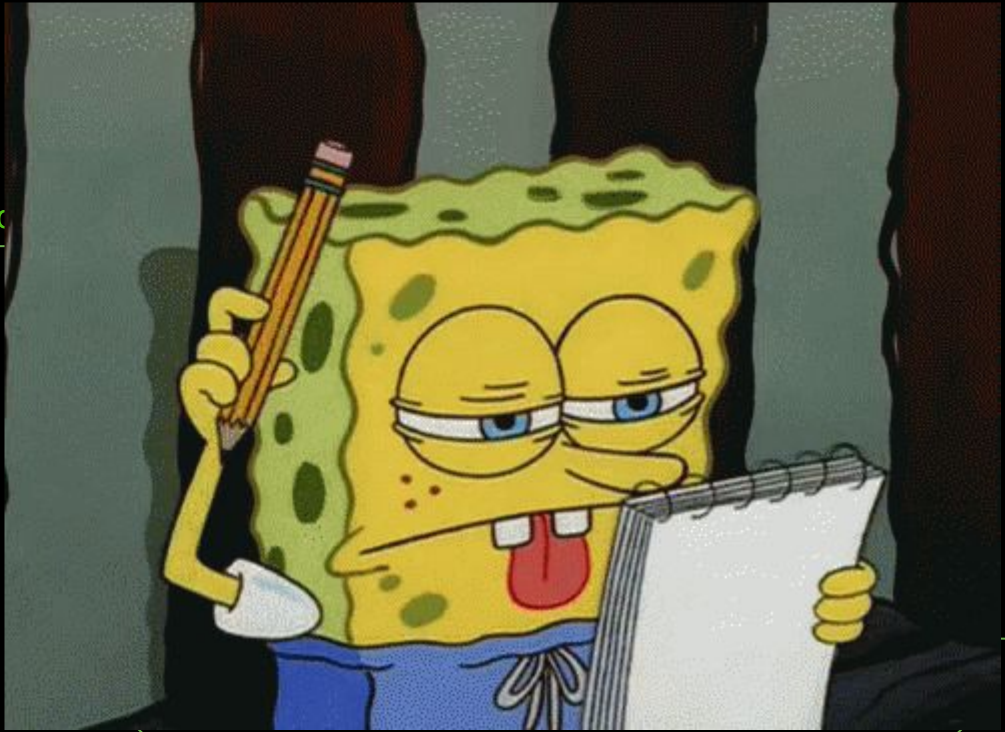
The Android Virtualization Technique



The Android Virtualization Technique

Plugin APK

Loa



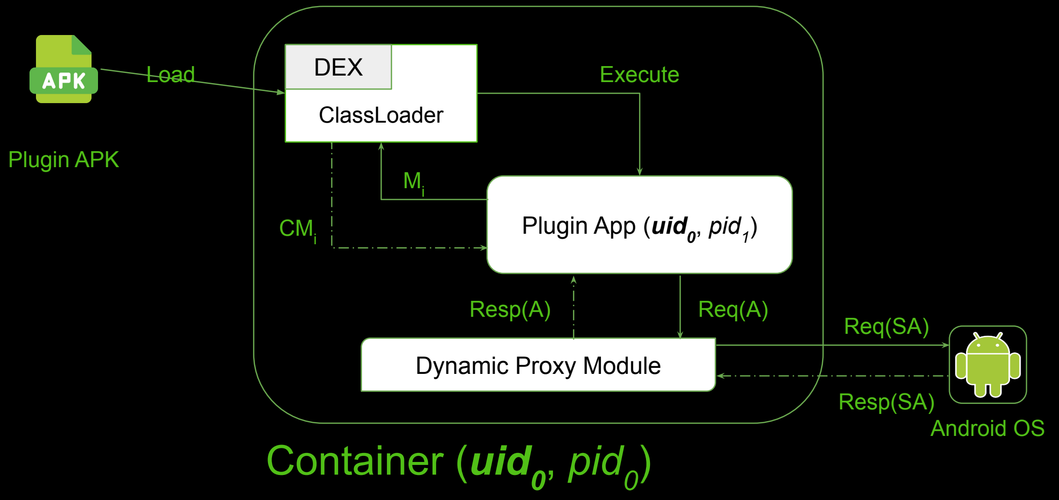
Req(SA)

Resp(SA)

Android OS

Container (***uid**$_0$*, *pid$_0$*)

The Android Virtualization Technique



- Container and plugin have the same **UID**
- The container has to declare **all permissions**
- The container has to **manage** the life cycle of all the plugin's components

Virtualization Frameworks

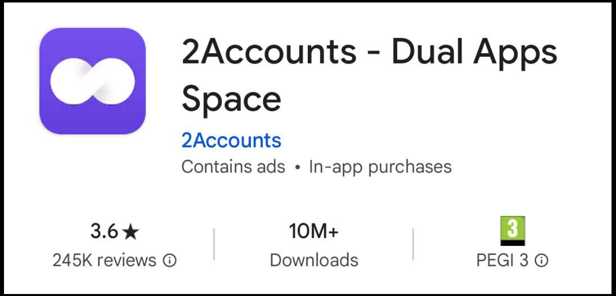
- Many virtualization frameworks are **open-source** (e.g., VirtualApp [2], DroidPlugin [3])
- Developers quickly found out they could also add **hooking** capabilities without needing root privileges
  - YAHFA [4]

[2] https://github.com/asLody/VirtualApp
[3] https://github.com/DroidPluginTeam/DroidPlugin
[4] https://github.com/PAGalaxyLab/YAHFA

The Android Virtualization Technique

Multiple purposes:

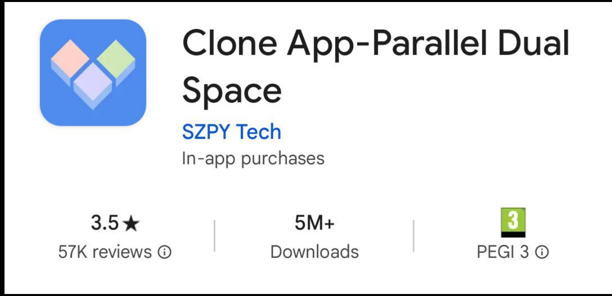
- multiple instances of the same app
- sandboxing malware



Source: Google Play Store

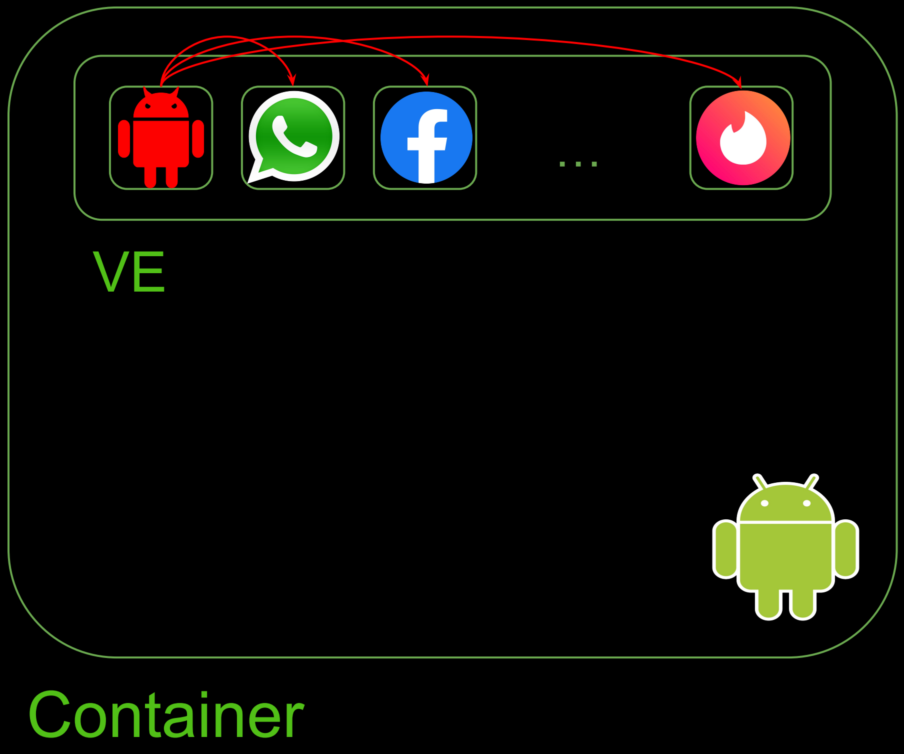
but also…

- **deploying** malware



Source: Google Play Store

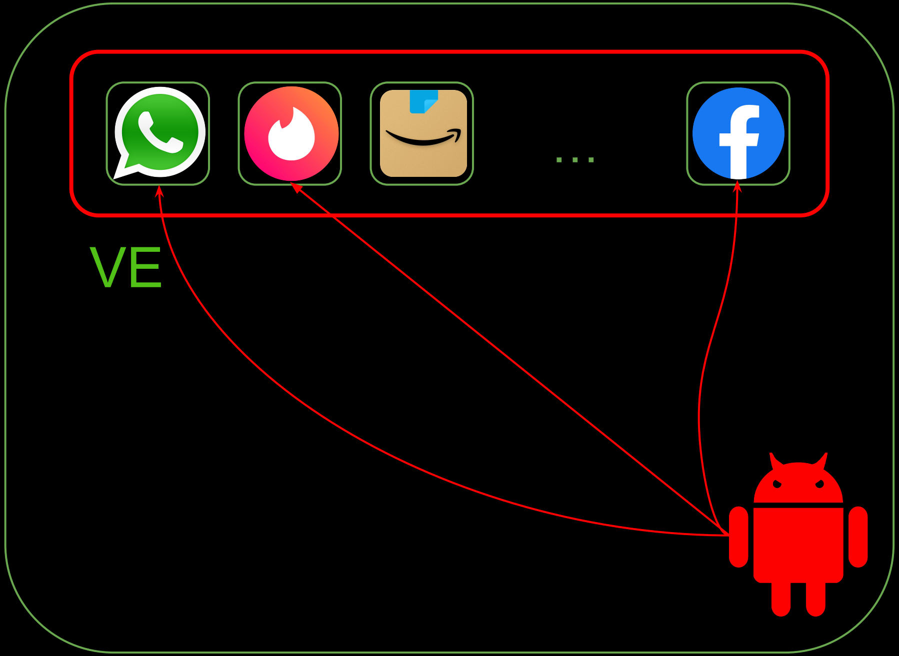
Malicious plugin



Malicious plugin:

- abuses the inherited **permissions** from the container
- **steals** data
- **injects** code

Virtualization-Based Repackaging (VBR) malware

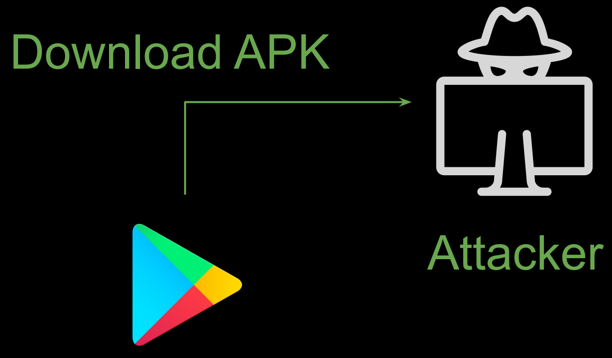


Container

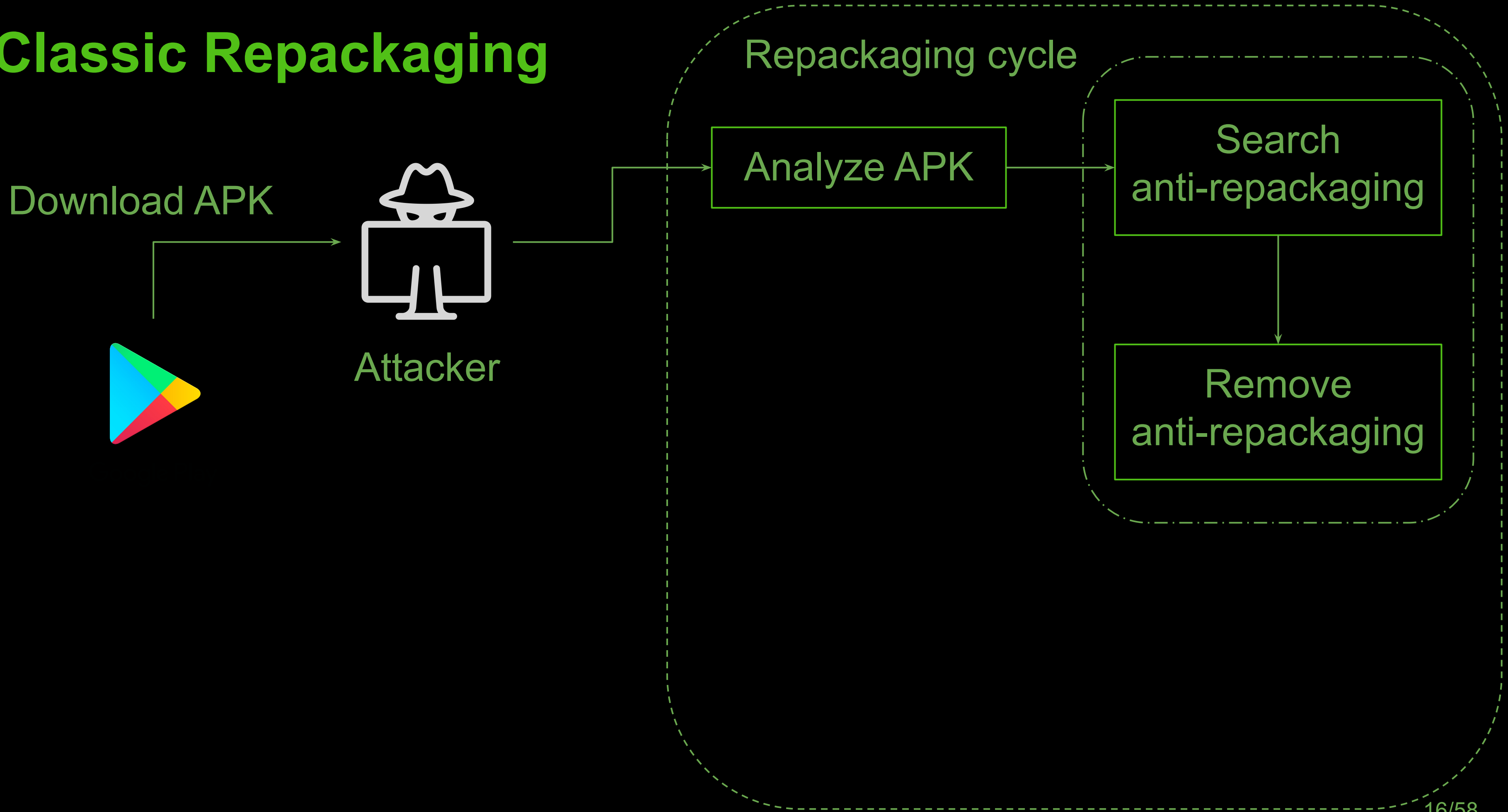
Malicious Container:

- **controls** the plugins' execution
- **steals** credentials and private information
- **injects** advertisement

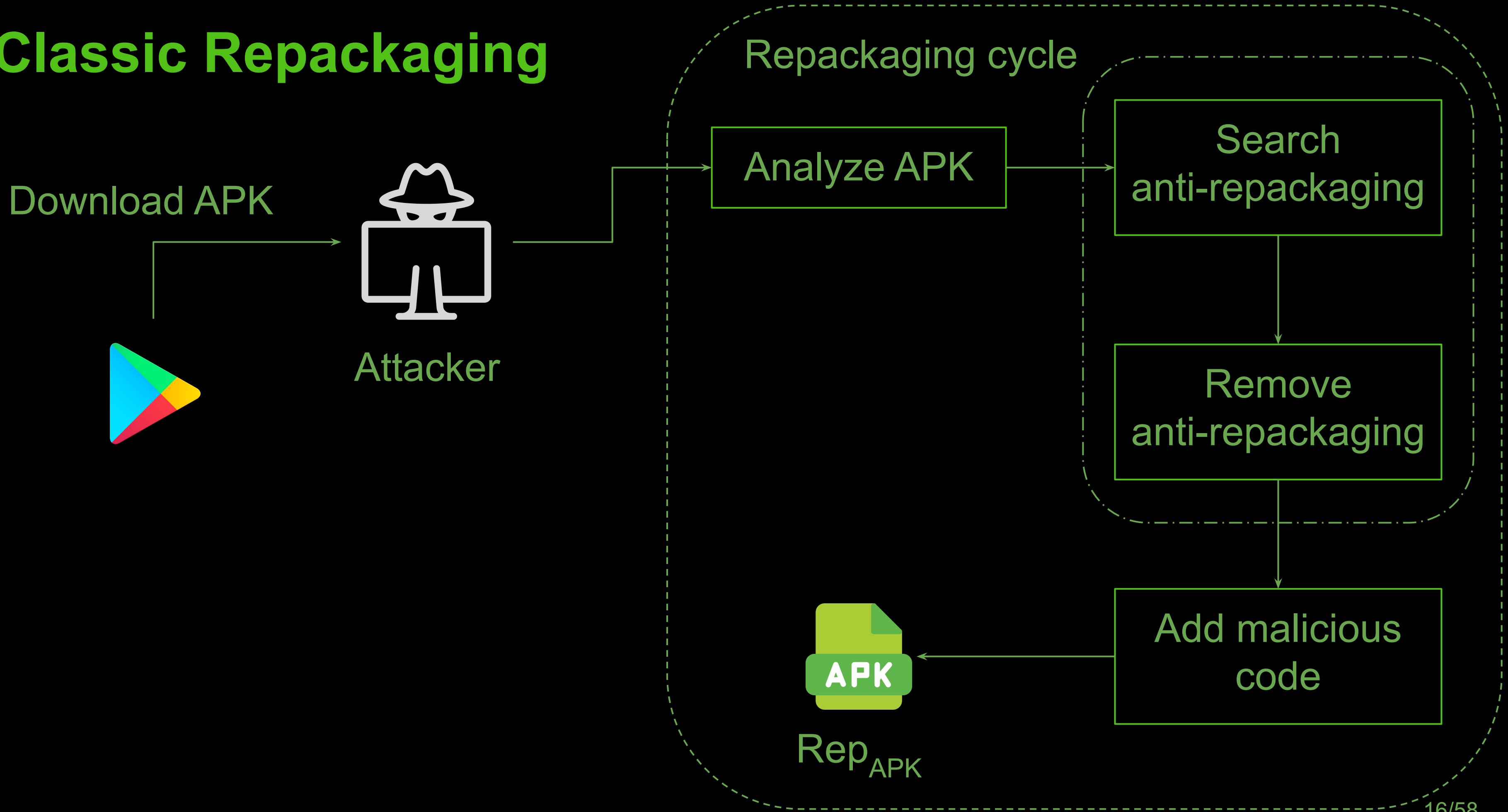
Classic Repackaging



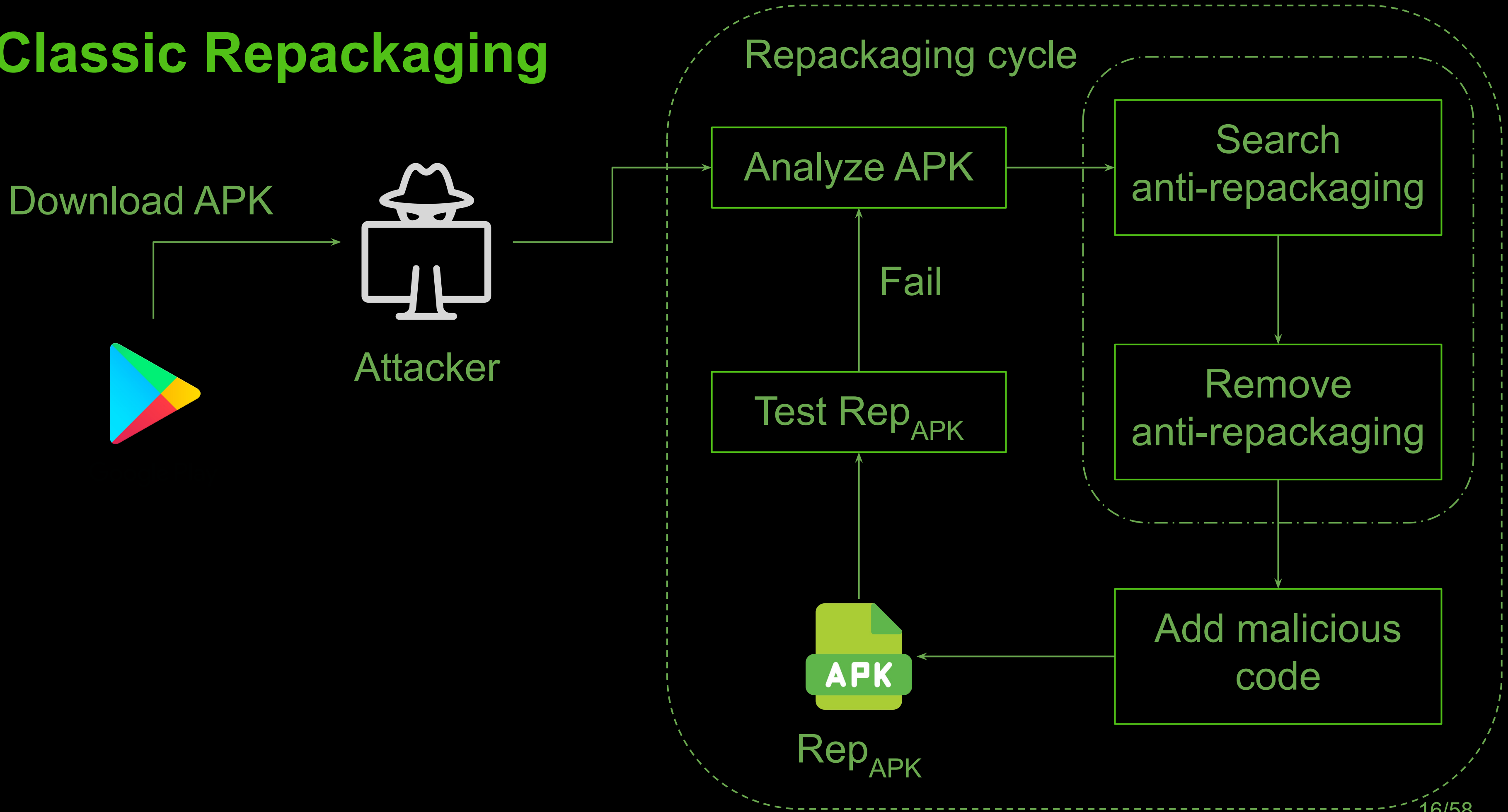
Classic Repackaging



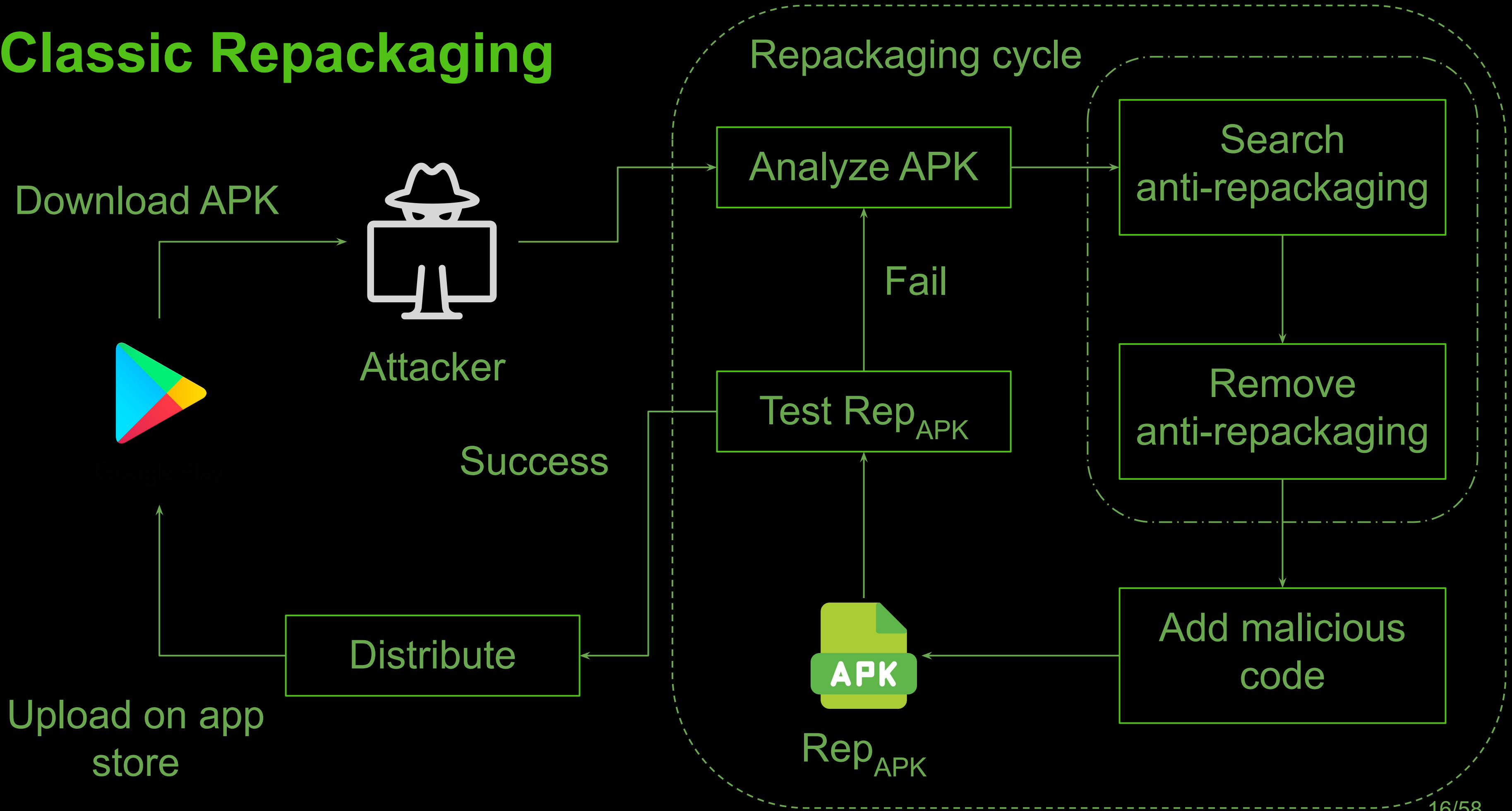
Classic Repackaging



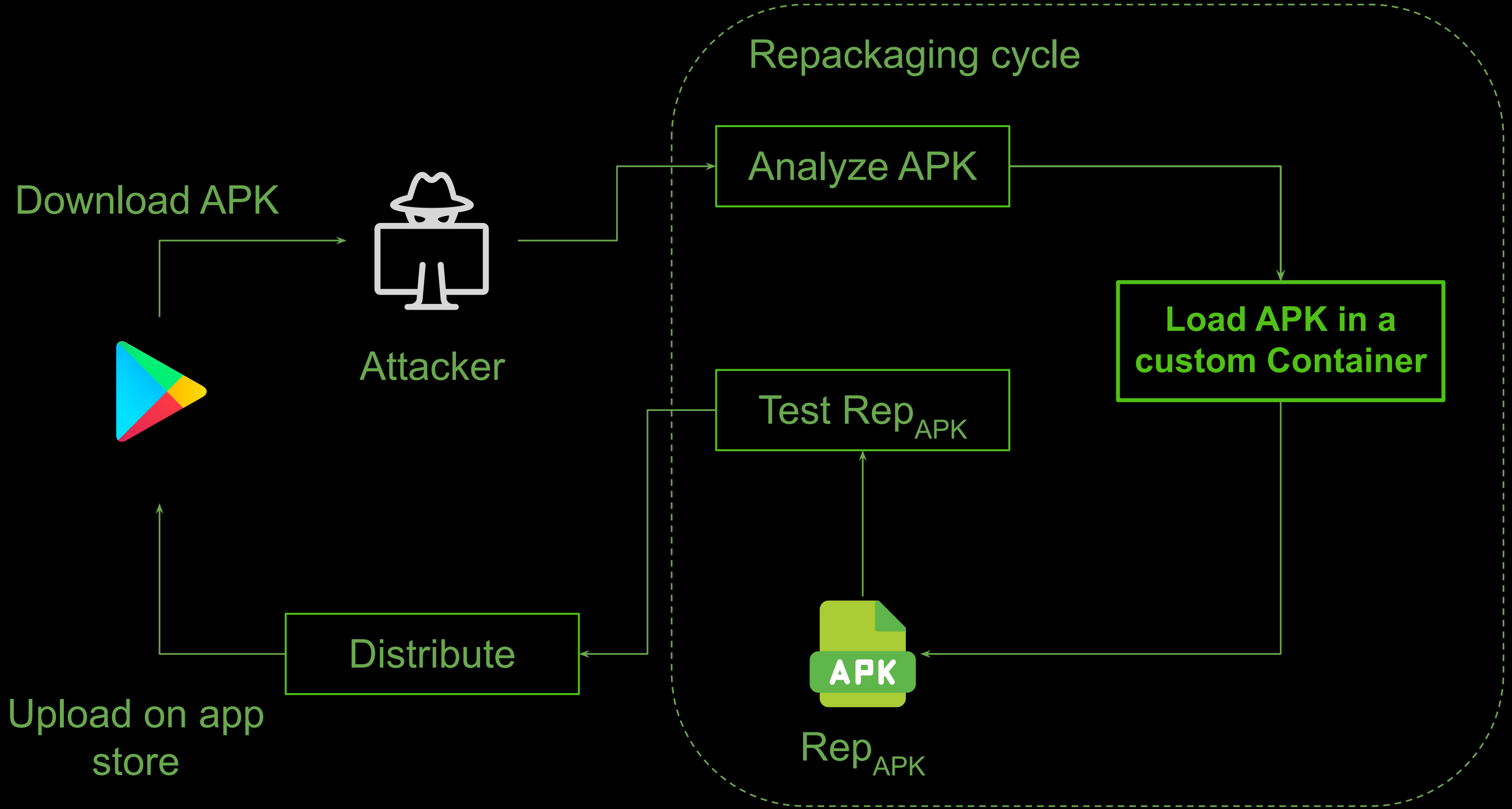
Classic Repackaging



Classic Repackaging



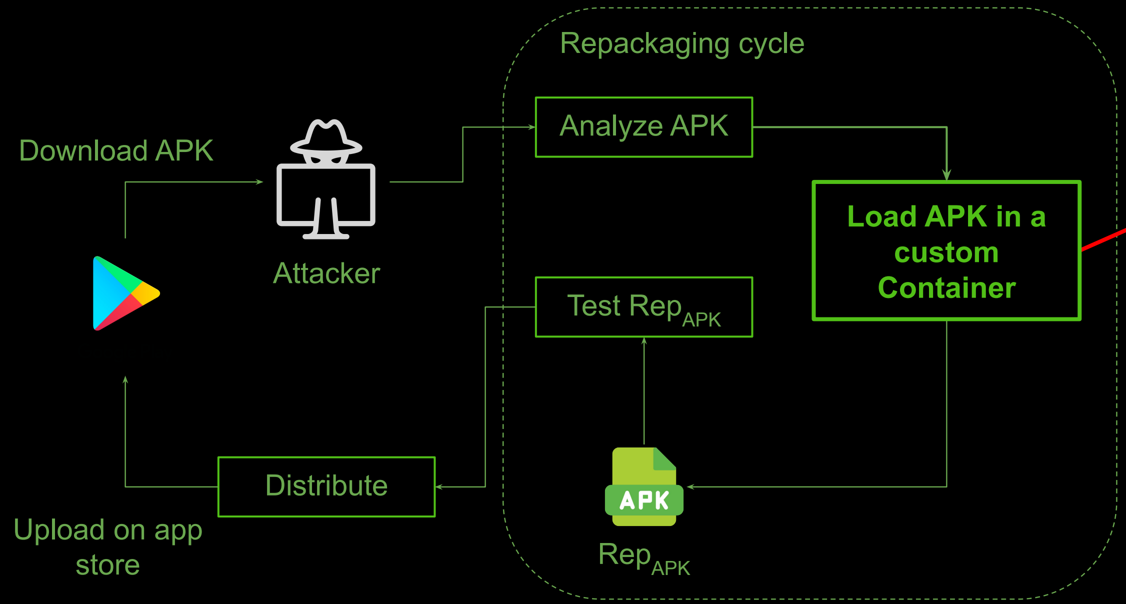
Virtualization-Based Repackaging



Repackaging Compared

	Classic Repackaging	Virtualization-Based Repackaging
Malicious Payload Location	Injected directly into the APK	Part of the custom Container
Signature Checks	App's signature is modified	Plugin remain unmodified
Visibility	Visible as an installed app	Plugin can remain hidden
Legitimate use	Rare	Common in dual-app tools

Virtualization-Based Repackaging



Can virtualization be prevented?

Can Virtualization-Based Repackaging be detected?

State-of-the-art

- **VAHunt** [5] is the state-of-the-art static analysis tool for VBR malware
- Authors reported to have found **71,303** VBR malware in a dataset of 139K virtualization-based apps

[5] L. Shi, J. Ming, J. Fu, G. Peng, D. Xu, K. Gao, and X. Pan, “Vahunt: Warding off new repackaged android malware in app-virtualization’s clothing,” in ACM SIGSAC Conference on Computer and Communications Security (2020)

State-of-the-art

- Four **anti-virtualization libraries** have been previously proposed
- They can be embedded in the apps and recognize the presence of virtualization

Are they effective and adopted among the most downloaded apps?

State-of-the-art

- **Marvel** [6] prevents repackaging attacks by forcing the protected apps to run only in a **trusted container**
- The apps are protected by removing portions of code, that are re-introduced by the trusted container

Is Marvel enough to protect against VBR malware?

[6] A. Ruggia, E. Losiouk, L. Verderame, M. Conti, and A. Merlo, “Repack me if you can: An anti-repackaging solution based on android virtualization,” in ACSAC (2021)

Motivation

Do we have to be worried about VBR malware?

- VAHunt found **71,303** VBR malware over a dataset of 139K virtualization malware

Driving Research Questions

RQ1: *How effective are existing defence mechanisms against VBR malware?*

RQ2: *How prevalent are the current detection and defence mechanisms against VBR malwares?*

RQ1: Existing Defences Effectiveness

- We tested the four anti-virtualization libraries, comprising a total of 19 checks
- These are libraries embedded in apps to detect if they are running inside a Virtual Environment (VE).

RQ1: Existing Defences Effectiveness

- We crafted a container and used YAHFA to hook the checks and **alter** their results

Our crafted container is able to **bypass all of them**

RQ1: Existing Defences Effectiveness

Some checks are:

- Being able to **access more permissions** than the ones declared
    - We customized the container to conservatively request the plugin's permissions

RQ1: Existing Defences Effectiveness

Some checks are:

- Querying **package name and components** through the `PackageManager`
  - Hook the APIs and modify the response

RQ1: Existing Defences Effectiveness

Some checks are:

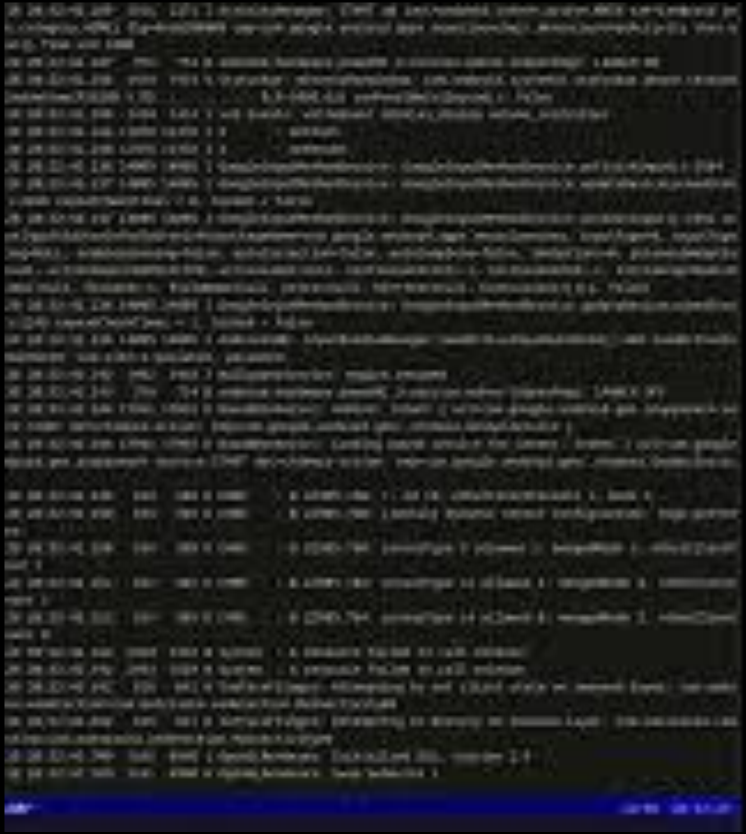
- Checking the **UID** and other running **processes** with the same one
    - Hook the `getRunningServices` API and remove the container listing

RQ1: Existing Defences Effectiveness

Some checks are:

- Throw an **exception** and analyzing the **stack trace**
  - Hook the `getStackTrace` API and remove references to the virtualization framework

Video Demo



RQ1: Existing Defences Effectiveness

- We used **Marvel** to protect the 10 lightest apps among the top 5,000 most downloaded in the Google Play store

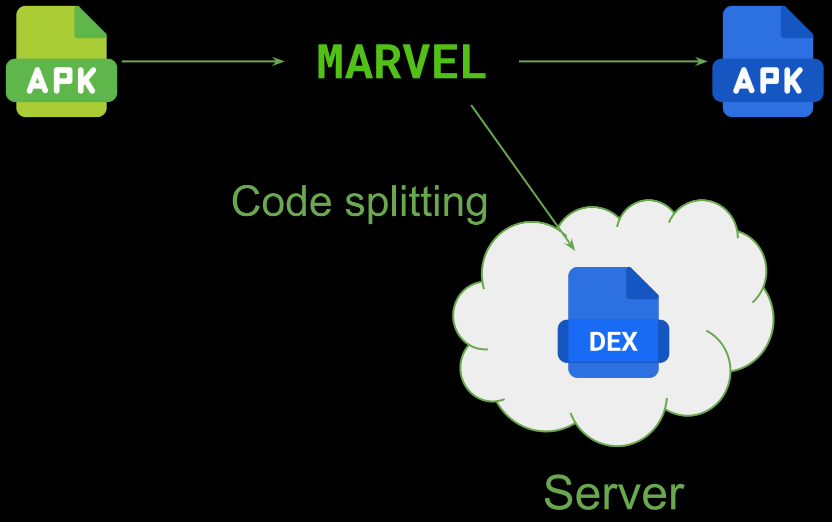
RQ1: Existing Defences Effectiveness

RQ1: Existing Defences Effectiveness

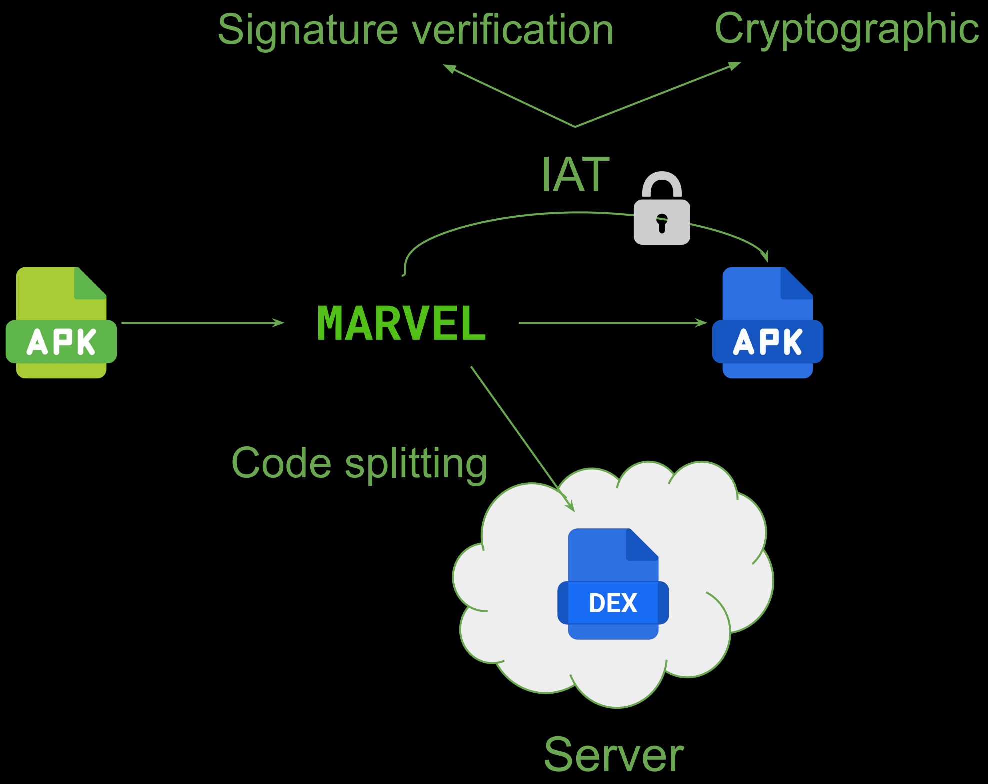
MARVEL

RQ1: Existing Defences Effectiveness

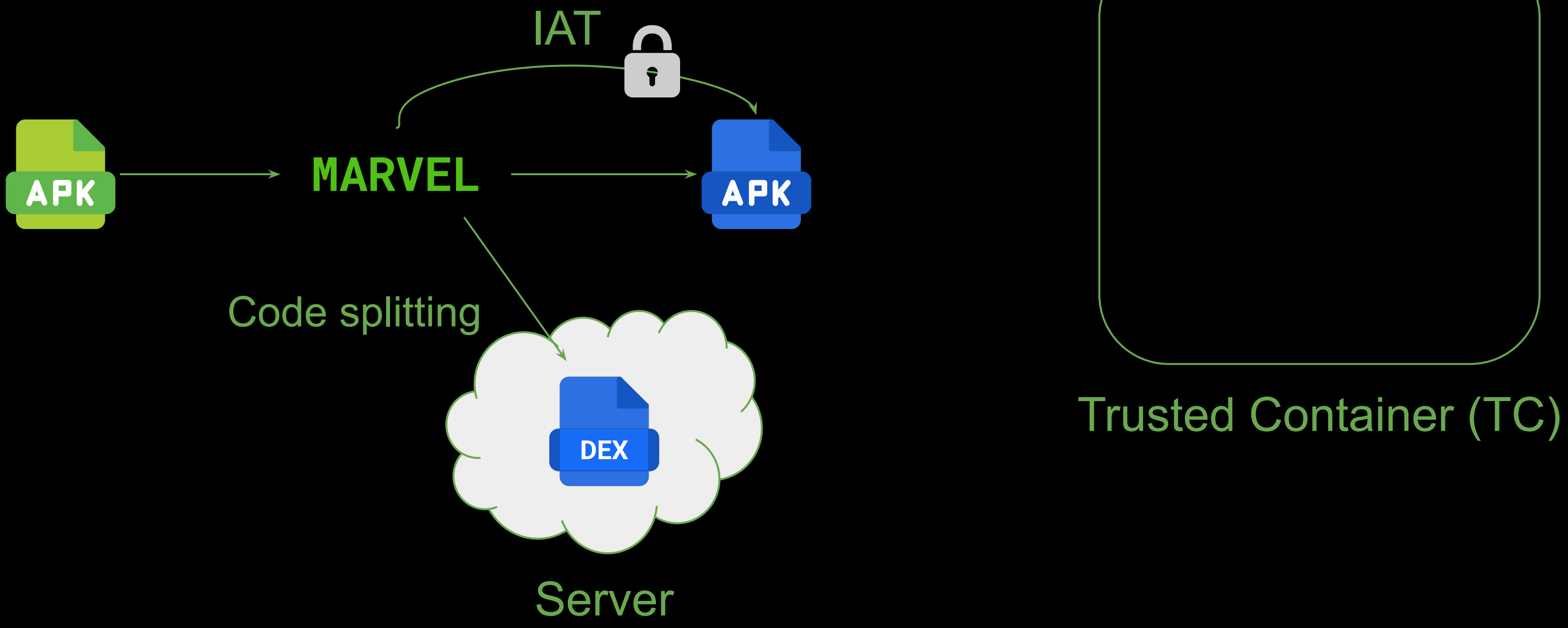
RQ1: Existing Defences Effectiveness



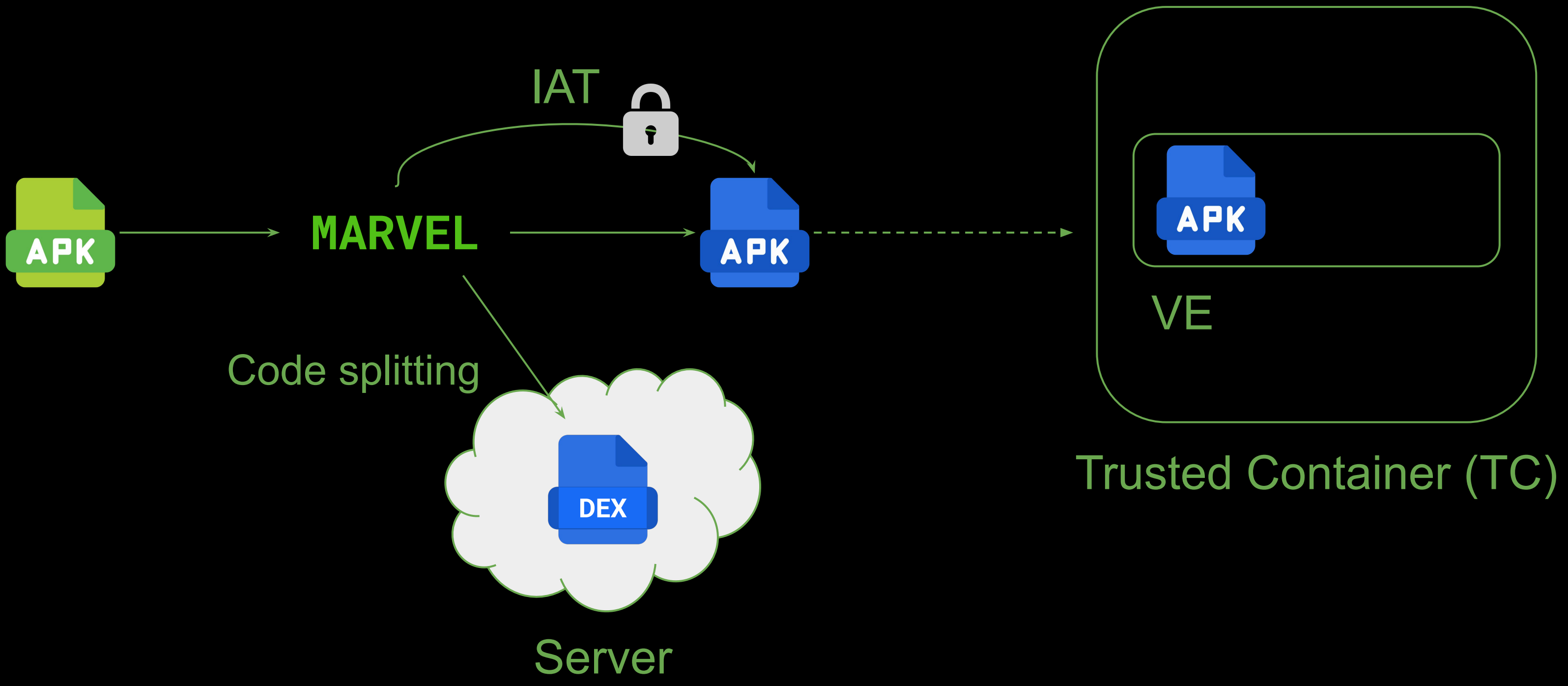
RQ1: Existing Defences Effectiveness



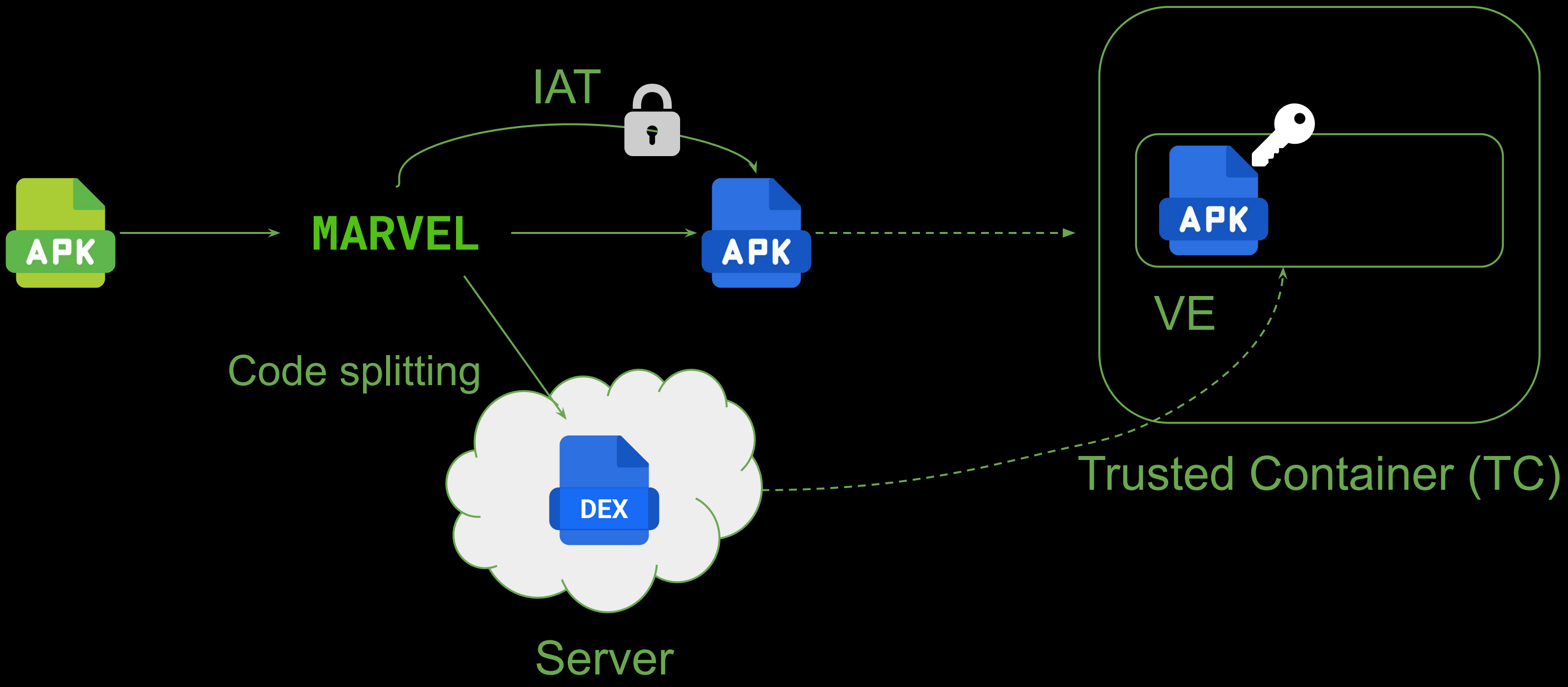
RQ1: Existing Defences Effectiveness



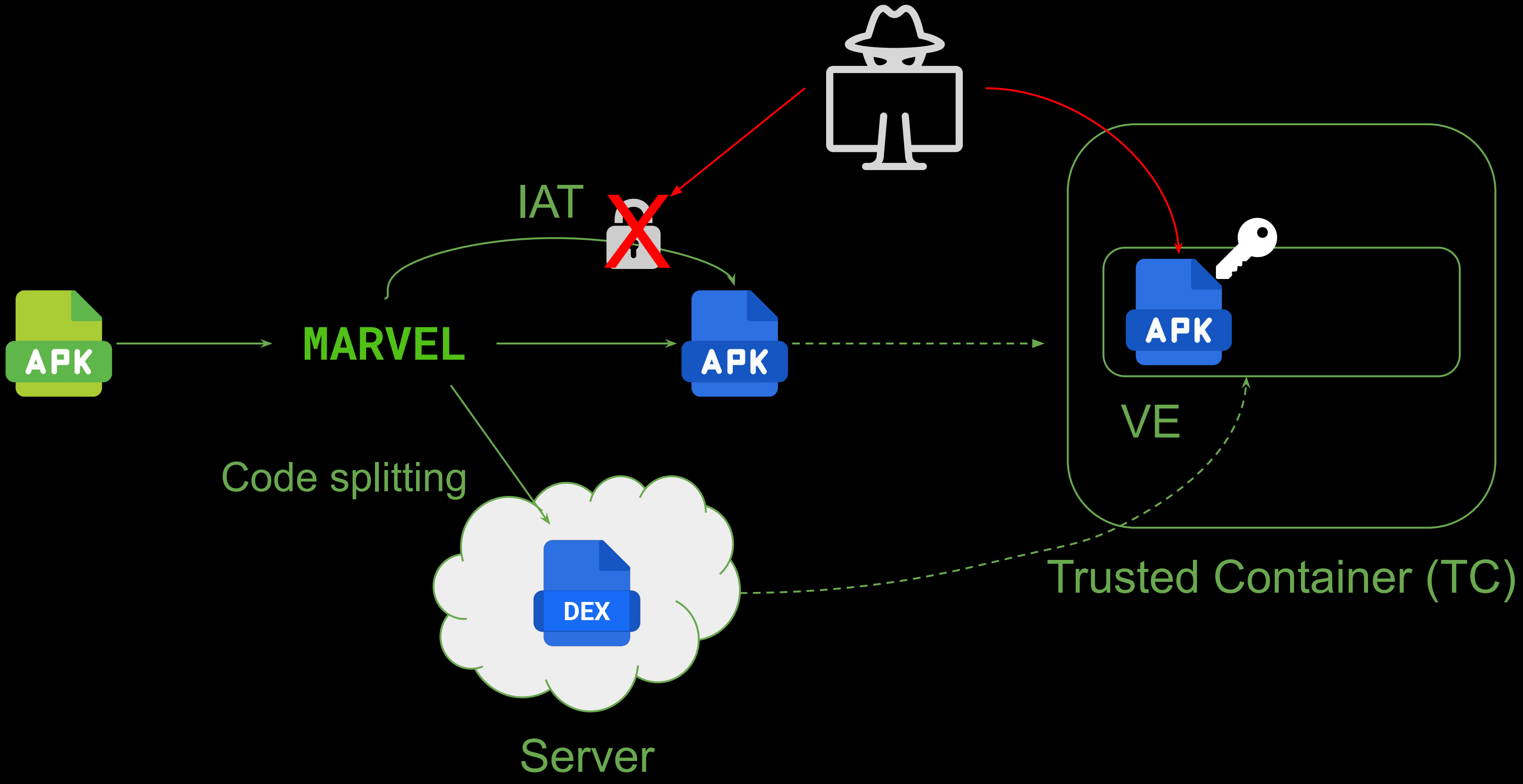
RQ1: Existing Defences Effectiveness



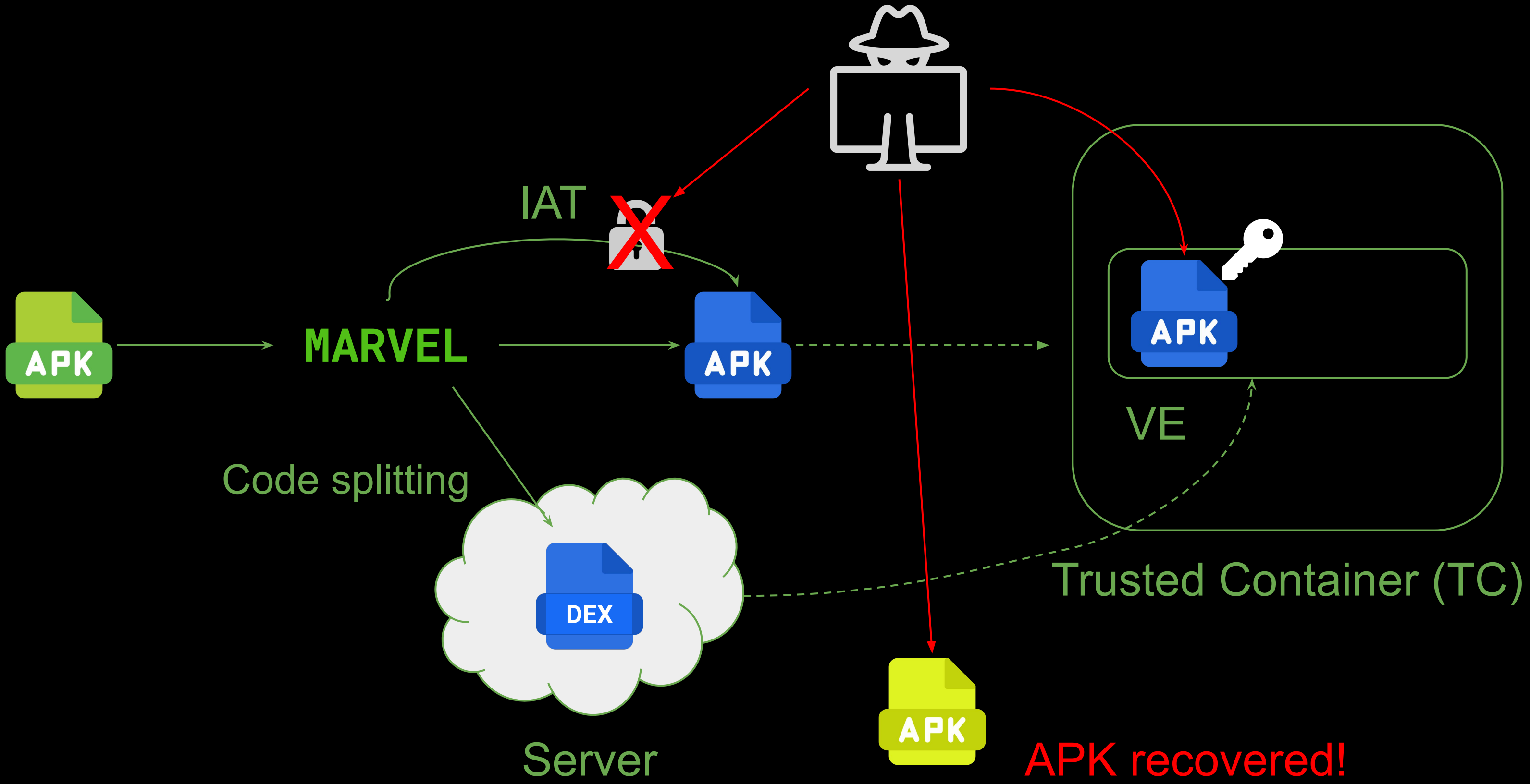
RQ1: Existing Defences Effectiveness



RQ1: Existing Defences Effectiveness



RQ1: Existing Defences Effectiveness



RQ1: Existing Defences Effectiveness

APK

M

Code s

ontainer (TC)

Server

APK

APK recovered!

RQ1: Existing Defences Effectiveness

- Using Frida [7], we were able to **intercept** the code loaded and **reconstruct** the original applications

[7] https://frida.re/

RQ1: Existing Defences Effectiveness

- The TC copies the DEX files with the `writeFileOnInternalStorage` API
  - We hooked it to intercept them and locally dump them
- For IAT, we simply deleted the local class that performs the checks ;)

We were able to **bypass** Marvel for **all of them** and allow execution outside the trusted container

RQ2: Defences adoption

- We executed the **top 5,000** most downloaded apps in "2Account" [8], one of the most popular virtualization apps
- We collected the **logs** of crashing apps and **reverse engineered** apps that failed to execute

[8] "2Account," https://play.google.com/store/apps/details?id=com.excelliance.multiaccount

RQ2: Defences adoption

- We executed the **top 5,000** most downloaded apps in "2Account" [8], one of the most popular virtualization apps
- We collected the **logs** of crashing apps and **reverse engineered** apps that failed to execute

None of them embedded any anti-virtualization measures or used Marvel

[8] "2Account," https://play.google.com/store/apps/details?id=com.excelliance.multiaccount

RQ2: Defences adoption

- 38 apps crashed in “2Accounts” for the following reasons:
  - Native library incompatibilities
  - Missing graphical resources
  - Incorrect file paths
  - 2Accounts bugs
- But **no signs of virtualization detections**

Matrioska's Motivation

- We found **several limitation** in all existing defences

- The anti-virtualization libraries and Marvel are **absent** among the top 5k most downloaded apps
    - They require **high involvement** from developers
    - VAHunt is a **static** solution and cannot be directly deployed on user's **devices**

Matrioska's Design

We propose **Matrioska**, a dynamic defence tool that:

- Detects **virtualization** usage and **repackaging** attacks
- Avoids malware evasion and anti-debugging techniques
- Requires no modifications to the Android OS or root privileges
- Can be installed as a third-party app

Matrioska's Design

- Matrioska dynamically inspects an application against **six signatures**, then computes a score
- A high-score indicates the presence of **virtualization** and/or **repackaging** behaviour

Matrioska's Signatures

The first two are checked statically:

- **S1**: An APK is found in the “assets” directory
- **S2**: One or more “stub components” can be found in the app
  - Stub components serve as place holders for the plugins’ components

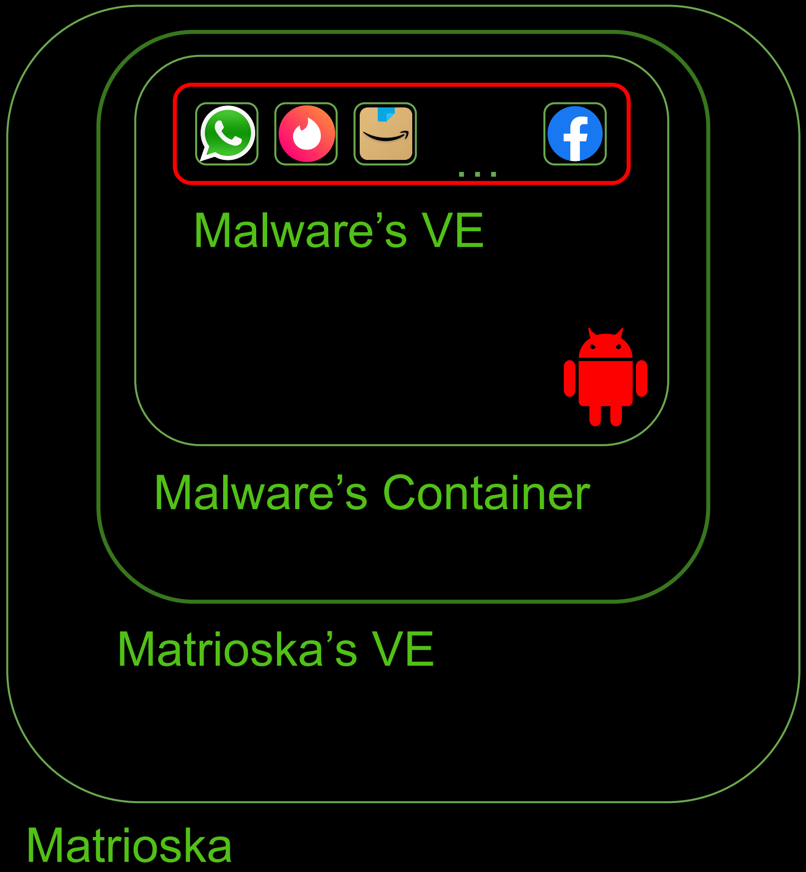
Matrioska's Signatures

The rest are measured during execution:

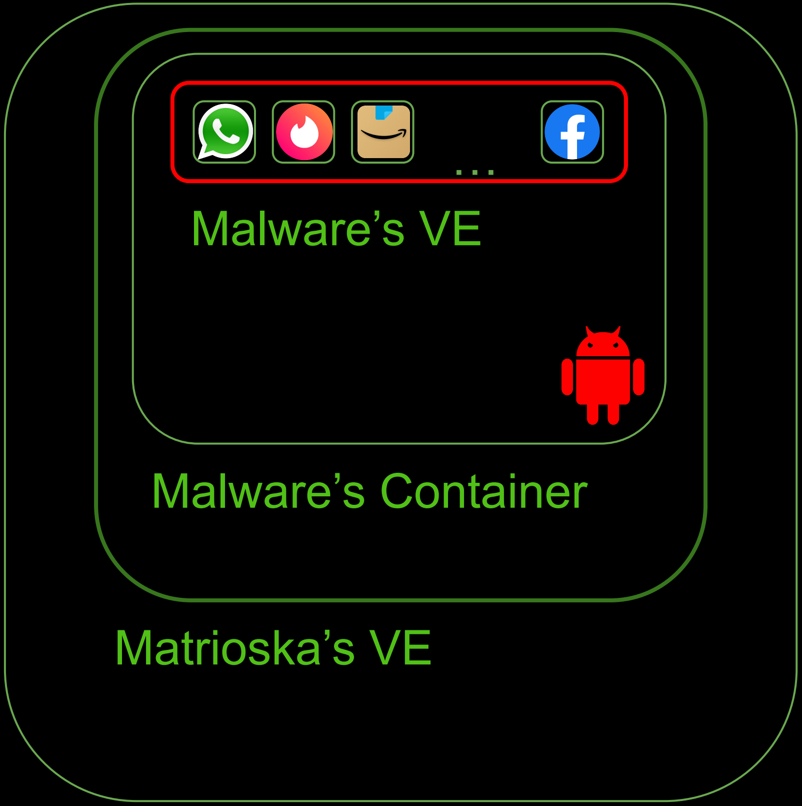
- **S3**: The app starts another app that is not installed in the devices (i.e., a plugin)
- **S4**: The app spawn processes sharing the same UID
- **S5**: An APK is downloaded or decrypted and saved in the storage
- **S6**: Container’s signature differs from the plugin’s

Matrioska's Design

Matrioska's Design



Matrioska's Design



Matrioska

Source: Microsoft Copilot

RQ3: Matrioska's Evaluation

- *How well does our solution perform in detecting VBR malwares with respect to the state-of-art?*
- We compared Matrioska against **VAHunt**, the state-of-the-art in detecting VBR malware

VAHunt

- VAHunt is a **static analysis** tool to detect VBR malware

1. Decompiles the APK into **smali code**
2. Detects **virtualization engine** presence with hardcoded strings
3. Analyzes behavior for signs of **silent loading**
   a. API calls, call chains
4. Checks the Manifest, APK files in the “assets” directory

RQ3: Matrioska's Evaluation

We used two different datasets:

- **Dataset 1**: 100 applications from the Google Play Store (50 with and 50 without virtualization) to measure virtualization detection

RQ3: Matrioska's Evaluation

Virtualization Detection

Dataset 1	TP	TN	FP	FN	Accuracy
Matrioska	50	49	1	0	**0.99**
VAHunt	21	50	0	29	0.71

VAHunt fails to generalize because of the several **hard-coded checks** in its logic

RQ3: Matrioska's Evaluation

- **Dataset 2**: VAHunt's authors refused to shared their dataset
- We collected **152,602** samples from AndroZoo [9] having a VirusTotal score equal to or greater than 10/66
- From 152,602 samples, we found a total of **1765** apps using virtualization

[9] AndroZoo, https://androzoo.uni.lu/

RQ3: Matrioska's Evaluation

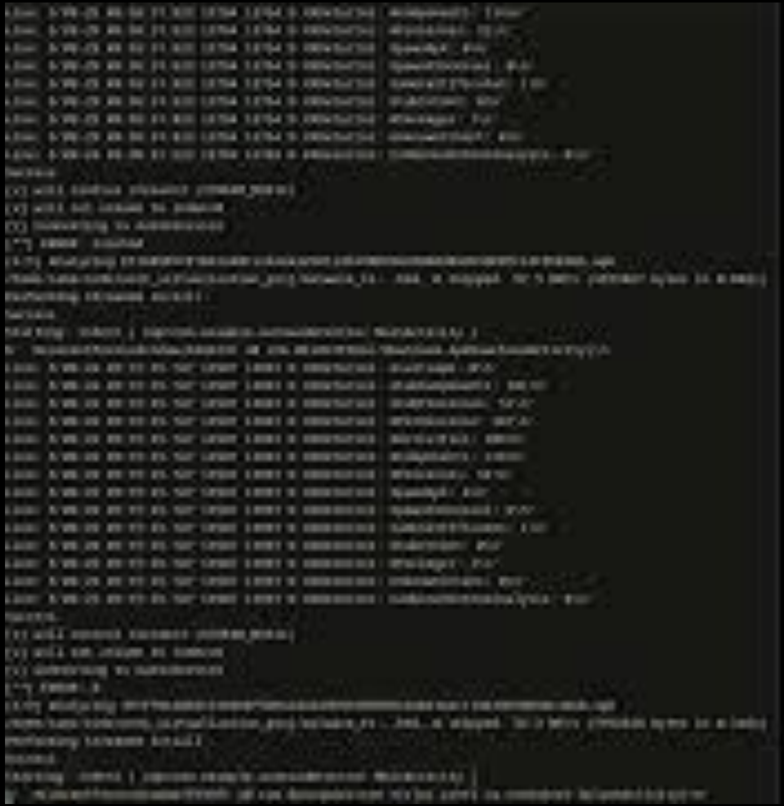
- We found **187** possible **VBR malware**, which we manually reverse engineered

VBR malware	FP	FN
Matrioska	**10**	**14**
VAHunt	23	39

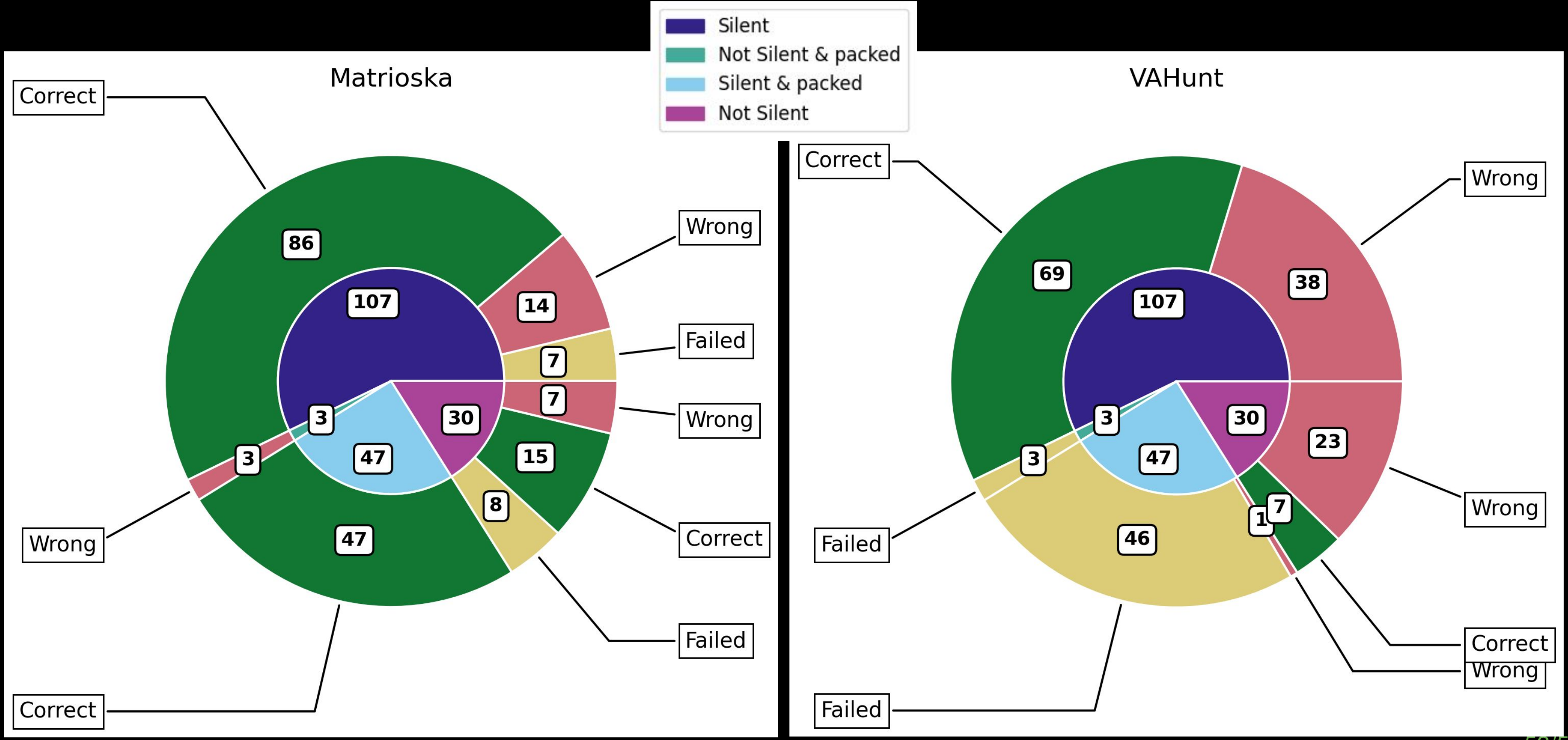
RQ3: Matrioska's Evaluation

- Matrioska is still subject to some **false positives** and **false negatives**
- Some legitimate apps might have APKs in their assets directory, or use stub components
- Malicious apps could delay detonation

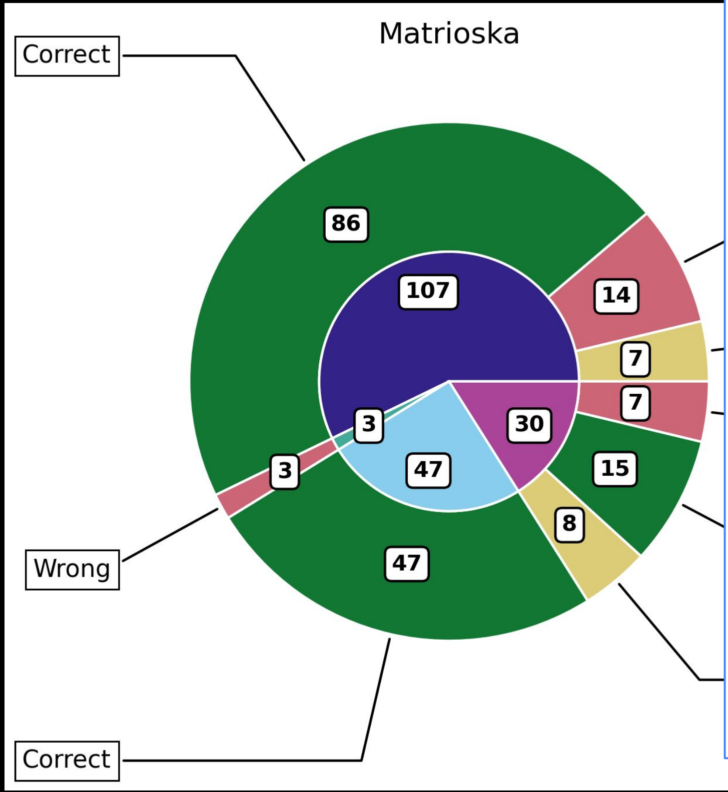
Video Demo

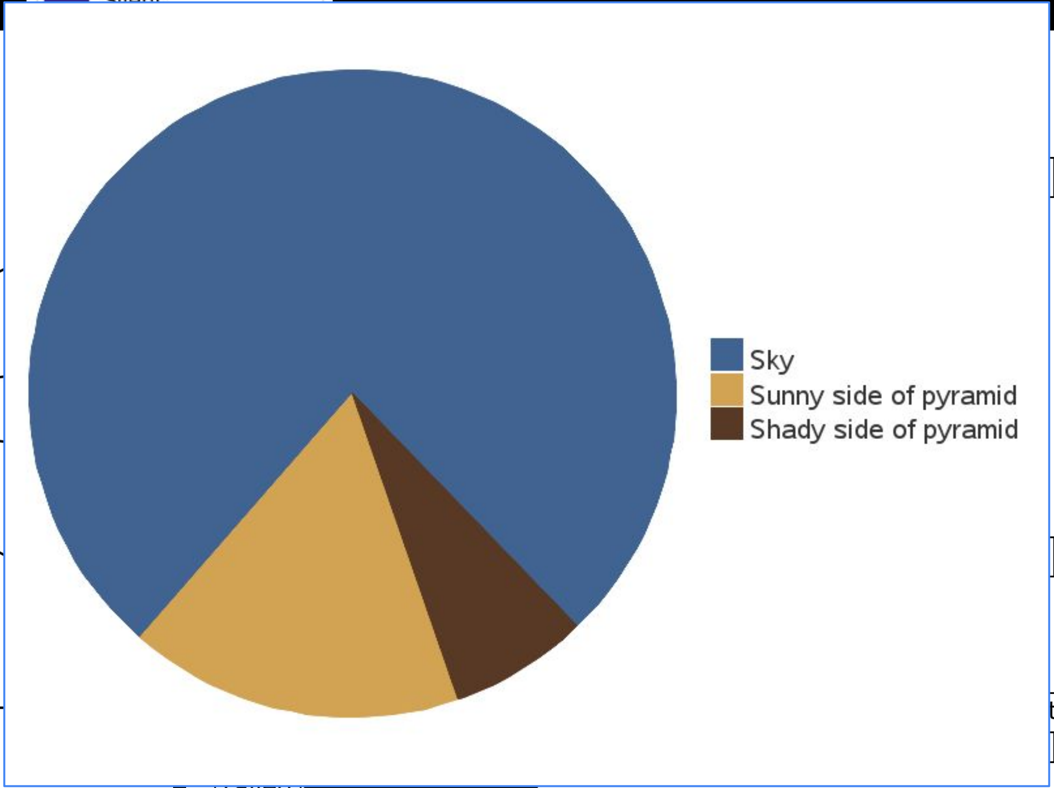


RQ3: Matrioska's Evaluation



RQ3: Matrioska's Evaluation





RQ3: Matrioska’s Evaluation

- VAHunt is limited by traditional static analysis limitations
    - It cannot analyze **packed** apps (i.e., apps decrypted at runtime)

Matrioska achieves **better performances** thanks to its dynamic analysis and signatures

Analyzing packed apps

- Packed APKs contain **encrypted payloads** or **obfuscated code**
- Medusa [10] is an open-source framework that can be used for dynamic DEX extraction
- Hooks into the ClassLoader at runtime to dump:
  - Decrypted DEX files
  - Loaded classes
  - de-obfuscated code before execution

[10] https://github.com/Ch0pin/medusa

RQ3: Matrioska‘s Evaluation

- Finally, we validated a total of **154 VBR malware**, in contrast to the 71,303 ones detected by the VAHunt’s paper

VBR malware are an existing **threat**–maybe even a growing one–but are **not currently prevalent** at the scale prior work suggested

Limitations

- **Code coverage**
  - Matrioska assumes immediate malware execution post-startup
- Assumes plugins are **silently** loaded
  - Interactions with the UI may be required

Future works

- Increase code coverage
  - Localize payload and try to reach it
  - Employ LLM-guided UI interaction tools
- Develop more stable frameworks for virtualization and hooking
  - Some of these have very unstable development cycles

Conclusions

- Existing defences against VBR malware exhibit several **limitations**
- Matrioska achieves better performances against the state-of-the-art and can effectively **safeguard** Android users

Questions?

Source: Chat-GPT

Full paper