Uninstallable by Design

The Role of Pre-installed Apps in Android's Security Landscape

OWASP Switzerland Community Meetup
February 2025

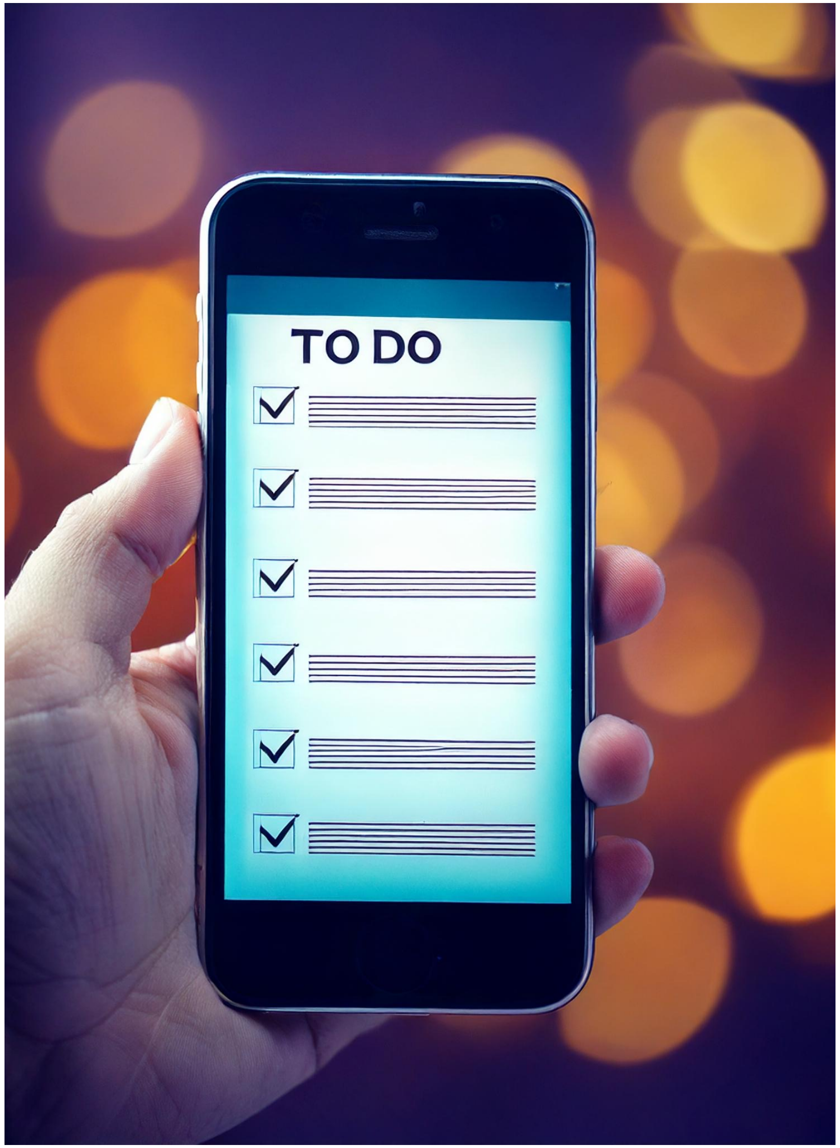
Content

Focus is on the Android OS for smartphones:

- What are pre-installed apps?
- How Android Verified Boot works
- Why we can’t uninstall pre-installed apps

Academic case study

- How we analysed pre-installed apps
- Challenges and solutions
- Future Work in this field



Who am I

Thomas Sutter
PhD Student @ Uni-Bern
Computer Science
https://7homassutter.github.io

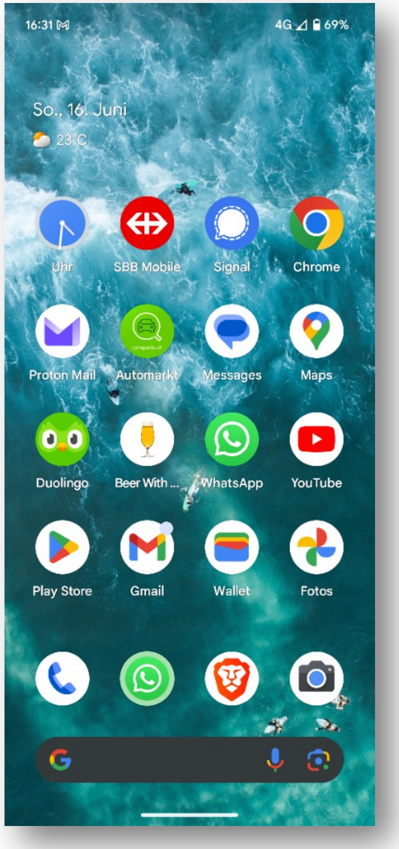
Software Engineering Group
Institute of Computer Science
University of Bern
https://seg.inf.unibe.ch/

Information Security Group
Institute of Computer Science
Zurich University of Applied Sciences
https://www.zhaw.ch/en/engineering/institutes-centres/init/information-security

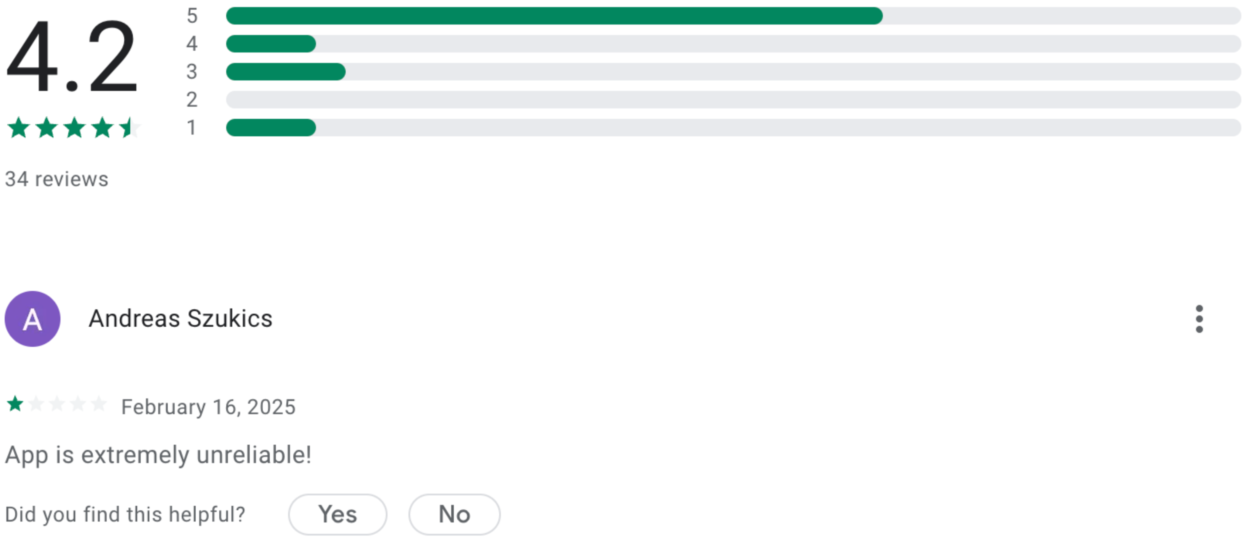
What are pre-installed apps?

Different Types of Apps

Userspace Apps



- Apps installed over an app store
- User reviews available
- Users can decide to uninstall apps
- Apps have limited access rights
- Apps are stored in the ".apk" file format, which is basically a zip file



4.2 ★★★★☆

34 reviews

A Andreas Szukics

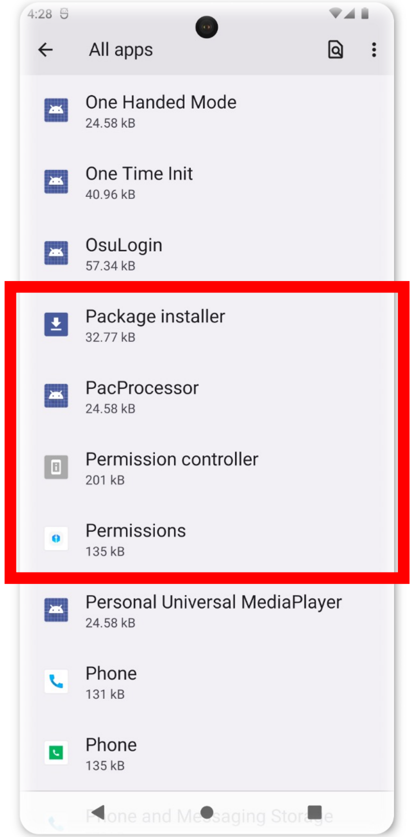
★☆☆☆☆ February 16, 2025

App is extremely unreliable!

Did you find this helpful? Yes No

Different Types of Apps

Pre-installed Apps



4:28

All apps

One Handed Mode
24.58 kB

One Time Init
40.96 kB

OsuLogin
57.34 kB

Package installer
32.77 kB

PacProcessor
24.58 kB

Permission controller
201 kB

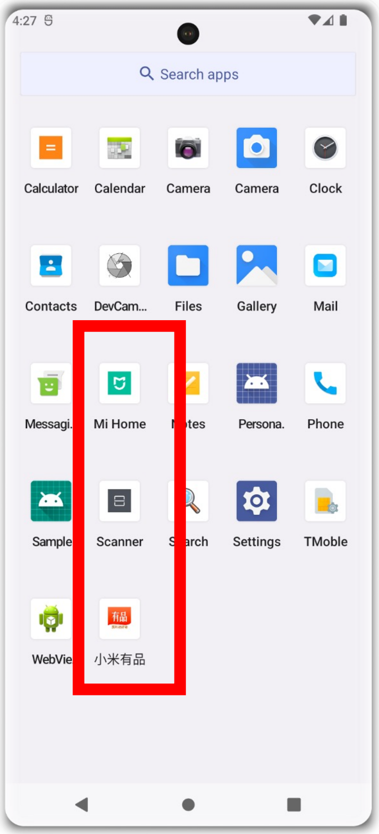
Permissions
135 kB

Personal Universal MediaPlayer
24.58 kB

Phone
131 kB

Phone
135 kB

Phone and Messaging Storage

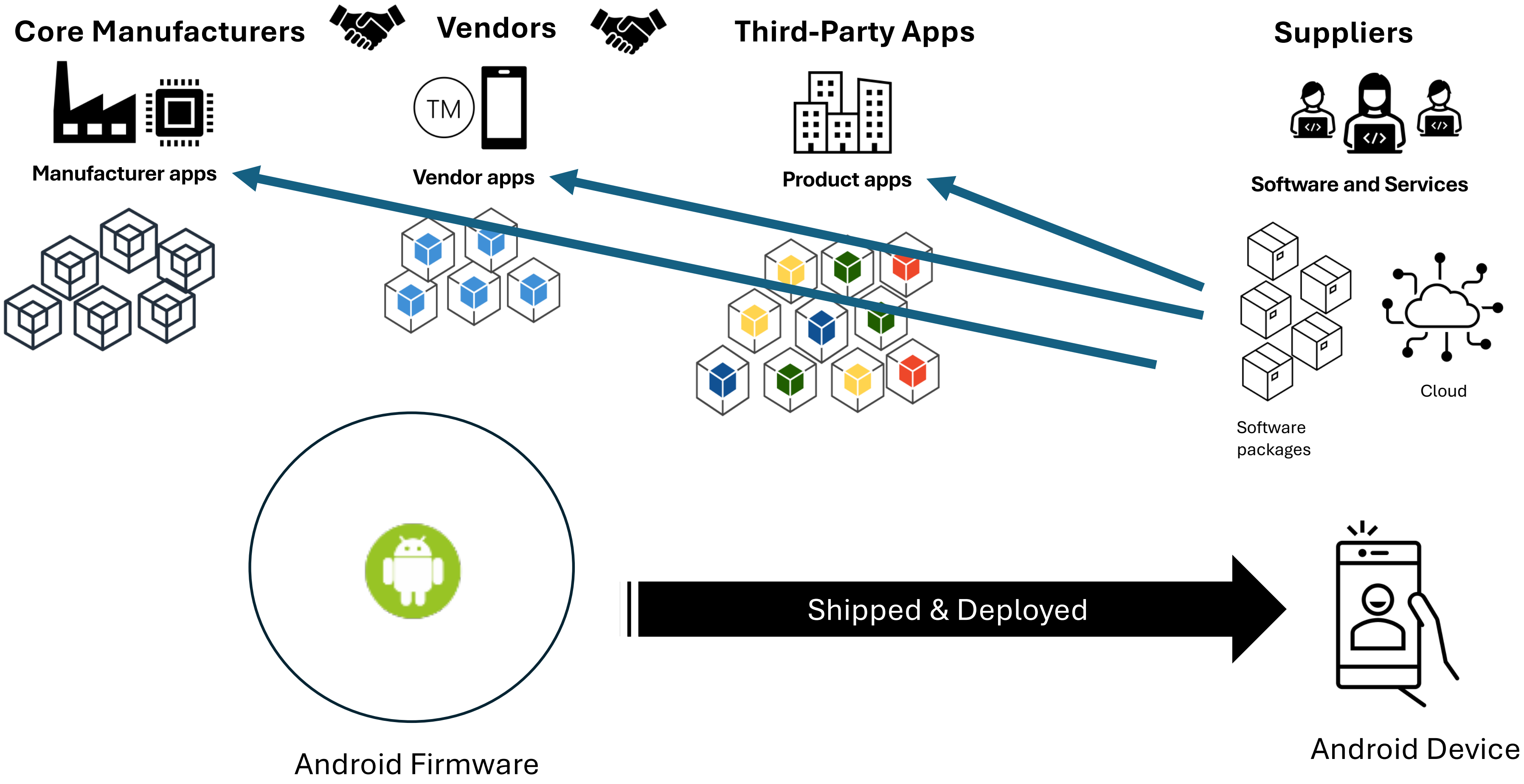


Pre-installed apps are all the Android applications that are shipped with a stock device. For instance:

- Telephone app
- Contacts app
- System service
  - Headless apps
- …

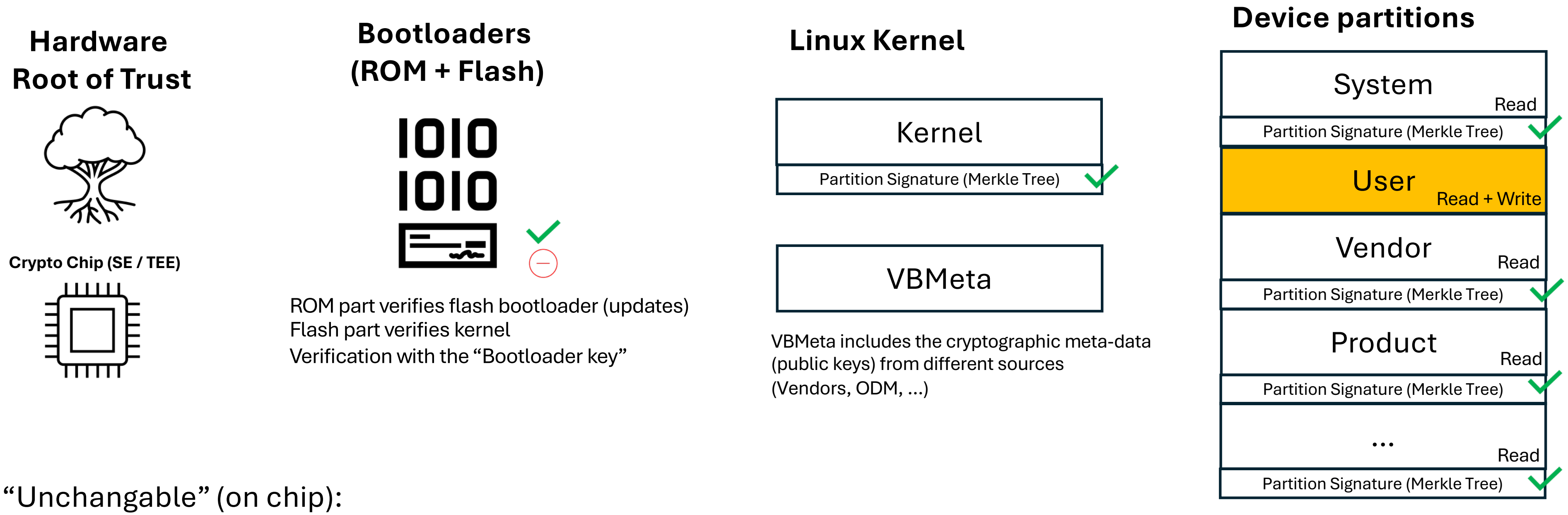
How do these apps end up on our phones?

Android's Supply Chain



Why are these apps interesting?

Android Verified Boot (simplified)



Hardware Root of Trust

Crypto Chip (SE / TEE)

Bootloaders (ROM + Flash)

✓

⊖

ROM part verifies flash bootloader (updates)
Flash part verifies kernel
Verification with the “Bootloader key”

Linux Kernel

Kernel

Partition Signature (Merkle Tree) ✓

VBMeta

VBMeta includes the cryptographic meta-data (public keys) from different sources (Vendors, ODM, …)

Device partitions

System	Read
Partition Signature (Merkle Tree) ✓	
User	Read + Write
Vendor	Read
Partition Signature (Merkle Tree) ✓	
Product	Read
Partition Signature (Merkle Tree) ✓	
…	Read
Partition Signature (Merkle Tree) ✓	

“Unchangable” (on chip):

- Bootloader key (asymmetric)
- Device unique key (symmetric)

Programmable:

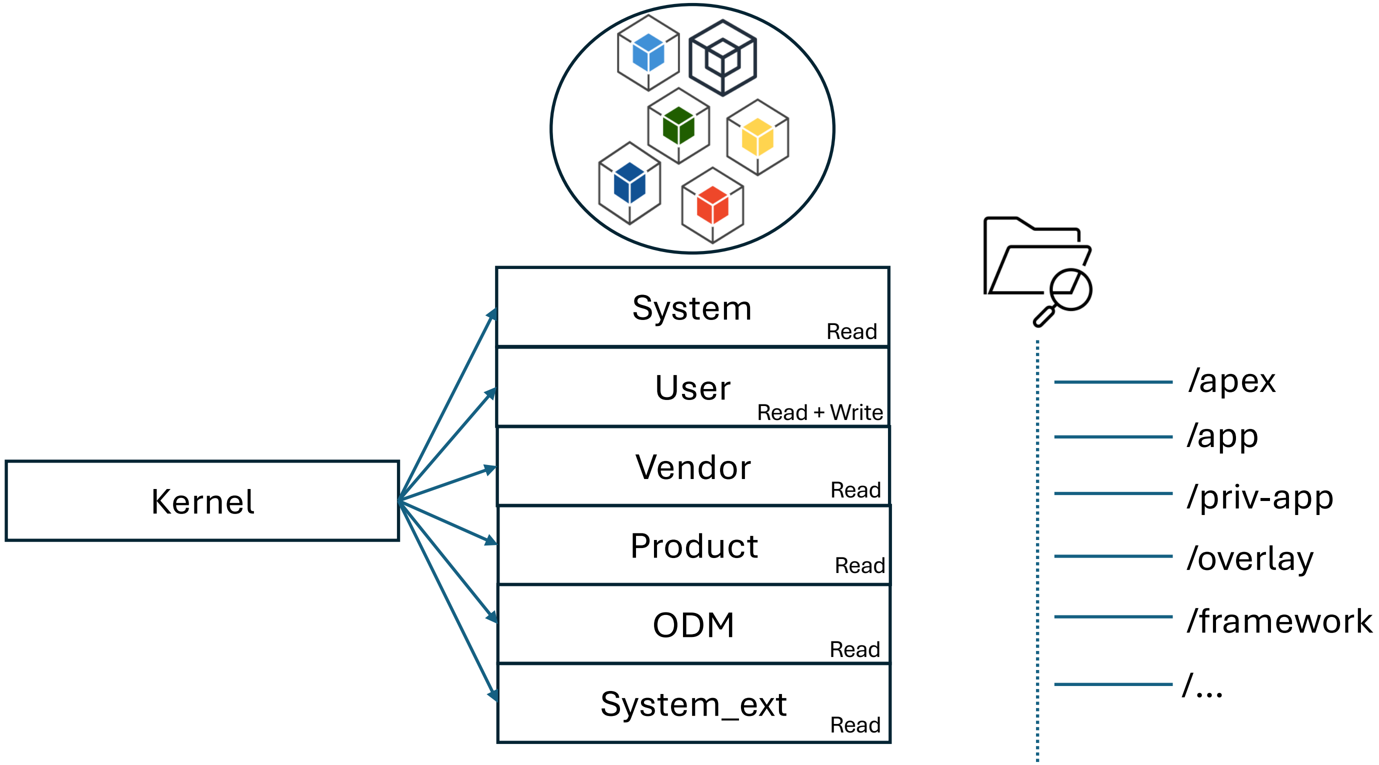
- Android Root of Trust (asymmetric)

DM-Verity only works for read-only partitions
Verification is done before mount and during
FS-verity for specific files runtime

Locating The Apps

- **boot partition.**
  - *kernel*
  - *ramdisk*
- **init_boot partition**
- **system partition**
- **odm partition**
- **odm_dlkm partition**
- **recovery partition**
- **cache partition**
- **misc partition**
- **userdata partition**
- **metadata partition**
- **vendor partition**
- **vendor_dlkm partition**
- **radio partition**
- **tos partition**
- **pvmfw partition**

…



More infos: https://source.android.com/docs/core/architecture/partitions

Why are partitions read-only anyways?

Minimizing attack surface during runtime

Sandboxing

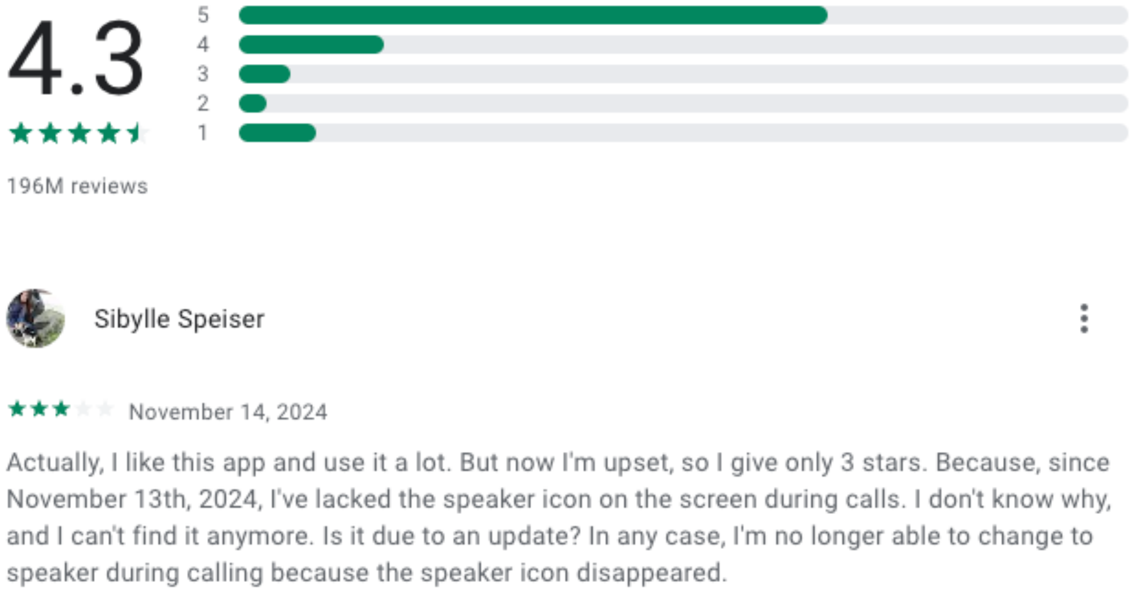
Performance

User Experience

Files on read-only partitions cannot simply be deleted

Consequenlty, pre-installed apps cannot be uninstalled

- Users have no way to remove these apps
  - Only by jailbreaking or rooting, which is not a real option for normal users



4.3

★★★★☆

196M reviews

Sibylle Speiser

★★★☆☆ November 14, 2024

Actually, I like this app and use it a lot. But now I'm upset, so I give only 3 stars. Because, since November 13th, 2024, I've lacked the speaker icon on the screen during calls. I don't know why, and I can't find it anymore. Is it due to an update? In any case, I'm no longer able to change to speaker during calling because the speaker icon disappeared.

- Users have no way to remove these apps
  - Only by jailbreaking or rooting, which is not a real option for normal users

- No app store for pre-installed apps
  - No review possibility
  - No publicly known security controls
  - No app classification or description
  - Apps cannot easily be installed

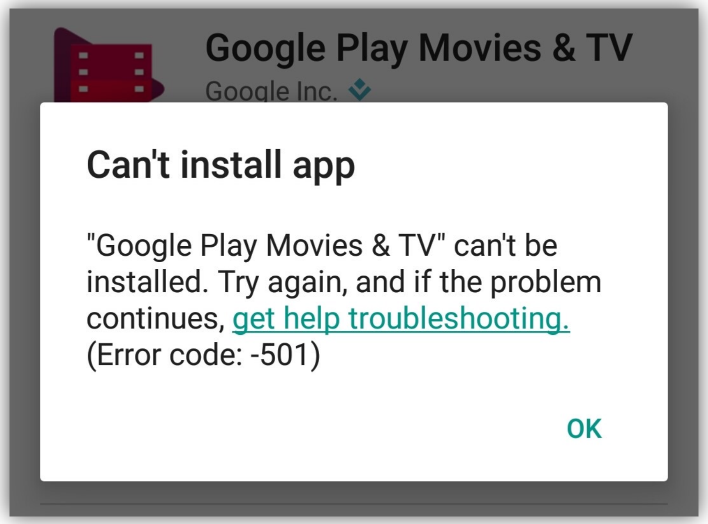
- Users have no way to remove these apps
    - Only by jailbreaking or rooting, which is not a real option for normal users
- No app store for pre-installed apps
    - No review possibility
    - No publicly known security controls
    - No app classification or description
    - Apps cannot easily be installed
- Pre-installed app have lots of permissions

- Users have no way to remove these apps
  - Only by jailbreaking or rooting, which is not a real option for normal users

- No app store for pre-installed apps
  - No review possibility
  - No publicly known security controls
  - No app classification or description
  - Apps cannot easily be installed

- Pre-installed app have lots of permissions

- Patching cycles are long and vendors sell space on their products



Google Play Movies & TV

Google Inc.

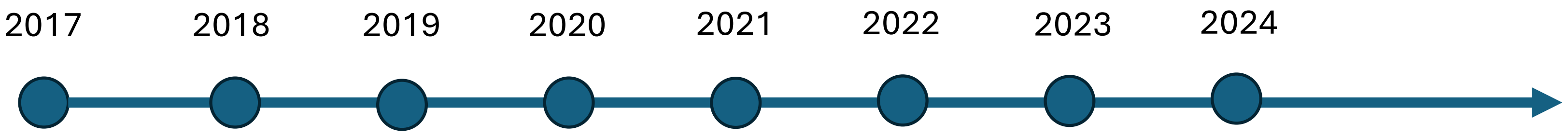
Can't install app

"Google Play Movies & TV" can't be installed. Try again, and if the problem continues, get help troubleshooting. (Error code: -501)

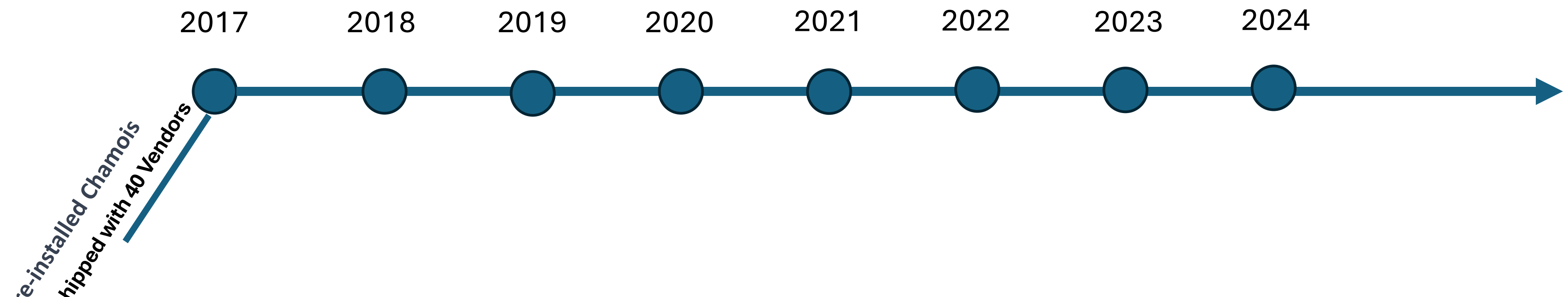
- Users have no way to remove these apps
  - Only by jailbreaking or rooting, which is not a real option for normal users
- No app store for pre-installed apps
  - No review possibility
  - No publicly known security controls
  - No app classification or description
  - Apps cannot easily be installed
- Pre-installed app have lots of permissions
- Patching cycles are long and vendors sell space on their products
- Security practioners have problems analysing them on scale and in-depth at runtime
  - Installation is not simply possible
  - Dependencies to files and services

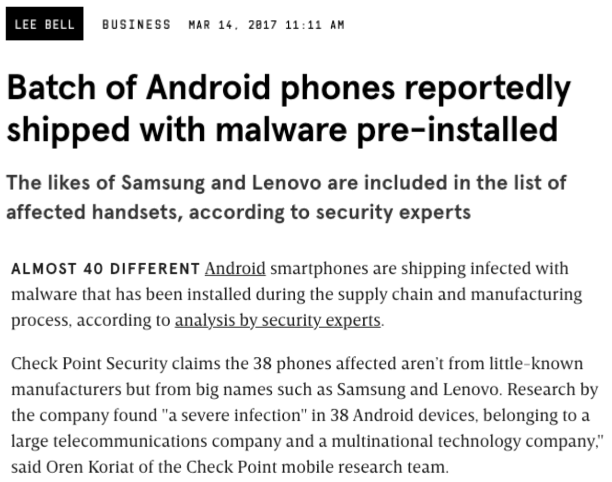
Pre-Installed Nightmares?

History of Incidents



History of Incidents





LEE BELL BUSINESS MAR 14, 2017 11:11 AM

Batch of Android phones reportedly shipped with malware pre-installed

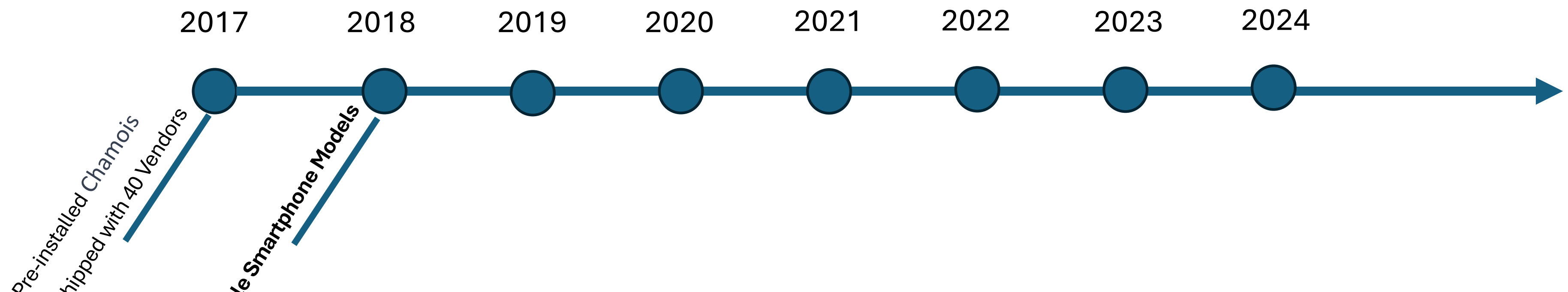
The likes of Samsung and Lenovo are included in the list of affected handsets, according to security experts

ALMOST 40 DIFFERENT Android smartphones are shipping infected with malware that has been installed during the supply chain and manufacturing process, according to analysis by security experts.

Check Point Security claims the 38 phones affected aren't from little-known manufacturers but from big names such as Samsung and Lenovo. Research by the company found "a severe infection" in 38 Android devices, belonging to a large telecommunications company and a multinational technology company," said Oren Koriat of the Check Point mobile research team.

https://www.wired.com/story/android-phones-hiding-pre-installed-malware/

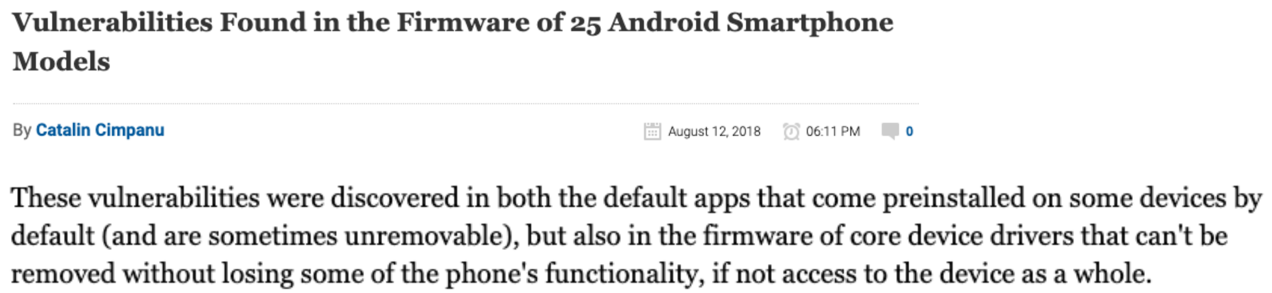
History of Incidents



2017 2018 2019 2020 2021 2022 2023 2024

Pre-installed Chamois Malware shipped with 40 Vendors

25 Vulnerable Smartphone Models



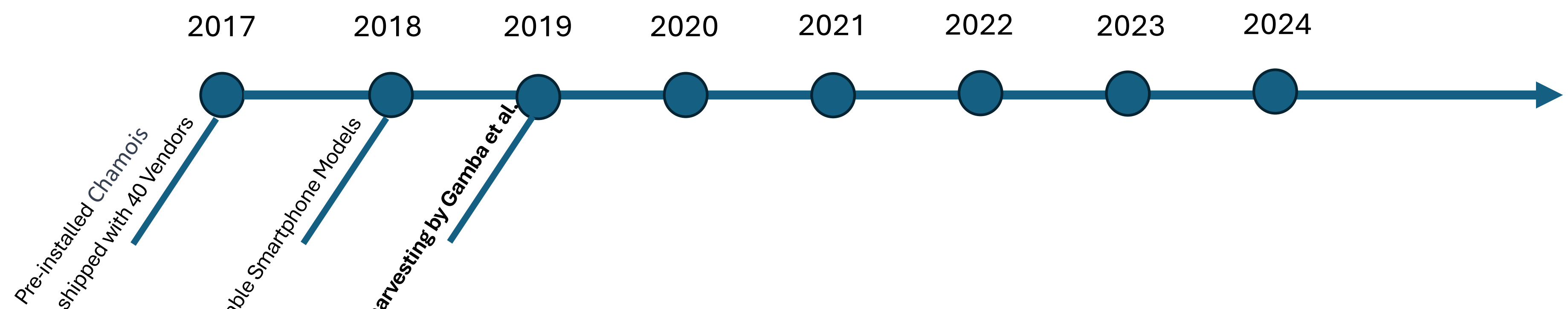
Vulnerabilities Found in the Firmware of 25 Android Smartphone Models

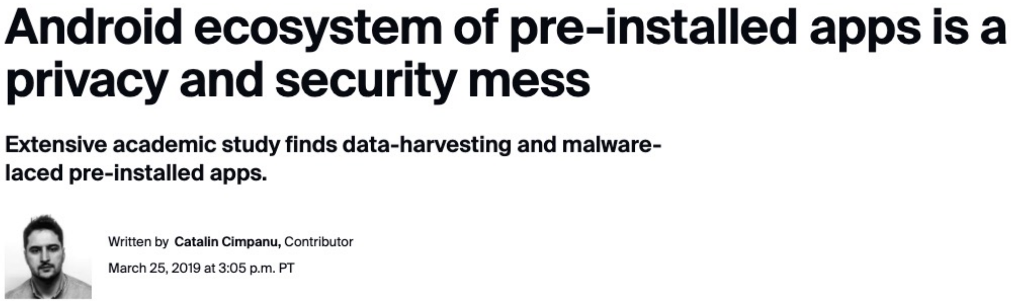
By **Catalin Cimpanu** August 12, 2018 06:11 PM 0

These vulnerabilities were discovered in both the default apps that come preinstalled on some devices by default (and are sometimes unremovable), but also in the firmware of core device drivers that can't be removed without losing some of the phone's functionality, if not access to the device as a whole.

https://www.bleepingcomputer.com/news/security/vulnerabilities-found-in-the-firmware-of-25-android-smartphone-models/

History of Incidents


2017
2018
2019
2020
2021
2022
2023
2024
Pre-installed Chamois
Malware shipped with 40 Vendors
25 Vulnerable Smartphone Models
Data harvesting by Gamba et al.



Android ecosystem of pre-installed apps is a privacy and security mess

Extensive academic study finds data-harvesting and malware-laced pre-installed apps.

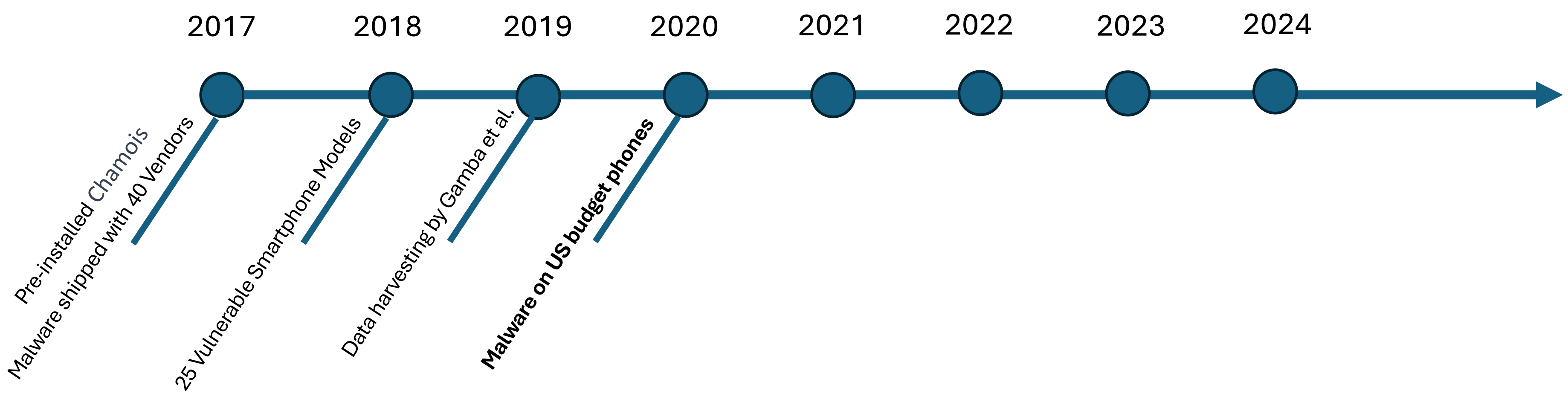
Written by **Catalin Cimpanu,** Contributor

March 25, 2019 at 3:05 p.m. PT

https://www.zdnet.com/article/android-ecosystem-of-pre-installed-apps-is-a-privacy-and-security-mess/

Paper: “An Analysis of Pre-installed Android Software” by Gamba et al.

History of Incidents



2017

2018

2019

2020

2021

2022

2023

2024

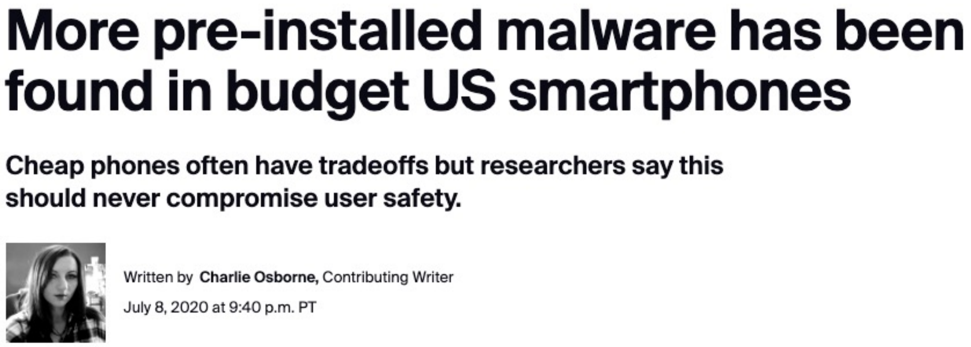
Pre-installed Chamois

Malware shipped with 40 Vendors

25 Vulnerable Smartphone Models

Data harvesting by Gamba et al.

Malware on US budget phones



More pre-installed malware has been found in budget US smartphones

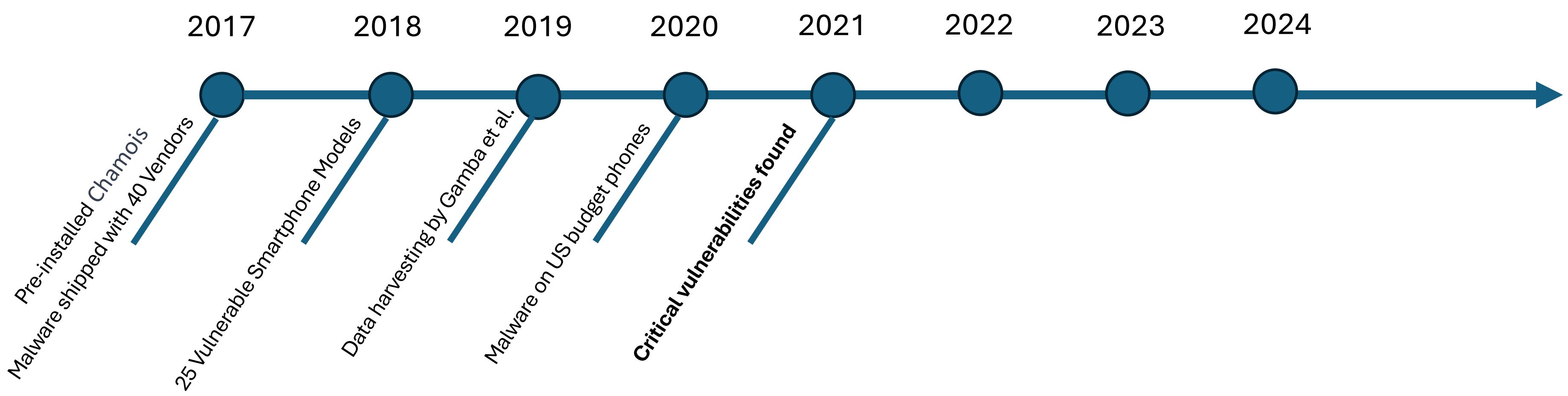
Cheap phones often have tradeoffs but researchers say this should never compromise user safety.

Written by **Charlie Osborne,** Contributing Writer

July 8, 2020 at 9:40 p.m. PT

https://www.zdnet.com/article/more-pre-installed-malware-has-been-found-in-budget-us-smartphones/

History of Incidents



2017 2018 2019 2020 2021 2022 2023 2024

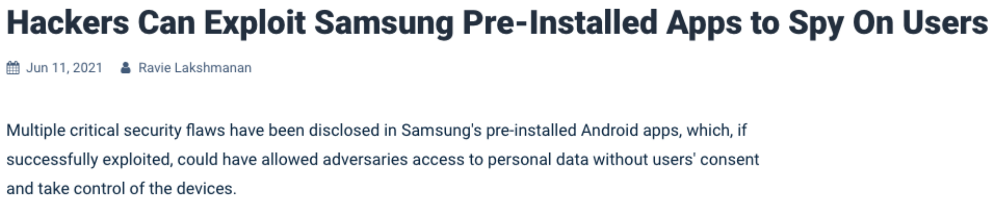
Pre-installed Chamois Malware shipped with 40 Vendors

25 Vulnerable Smartphone Models

Data harvesting by Gamba et al.

Malware on US budget phones

Critical vulnerabilities found



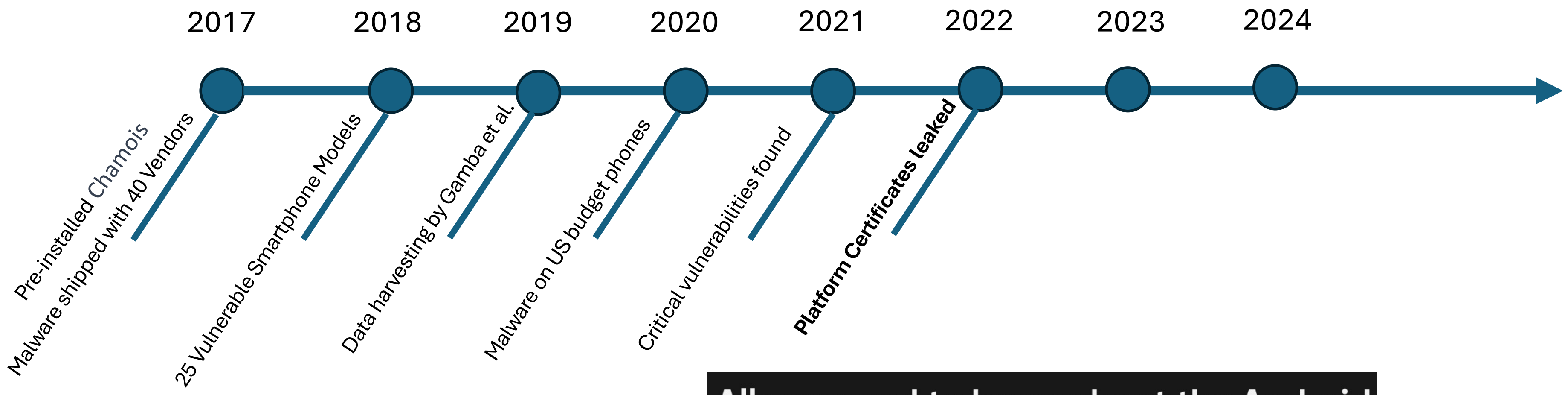
Hackers Can Exploit Samsung Pre-Installed Apps to Spy On Users

Jun 11, 2021 Ravie Lakshmanan

Multiple critical security flaws have been disclosed in Samsung's pre-installed Android apps, which, if successfully exploited, could have allowed adversaries access to personal data without users' consent and take control of the devices.

https://thehackernews.com/2021/06/hackers-can-exploit-samsung-pre.html

History of Incidents



2017 – Pre-installed Chamois Malware shipped with 40 Vendors

2018 – 25 Vulnerable Smartphone Models

2019 – Data harvesting by Gamba et al.

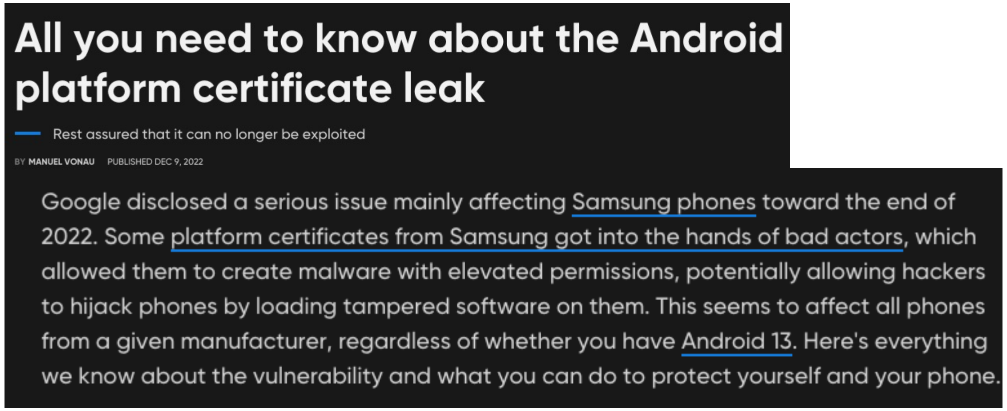
2020 – Malware on US budget phones

2021 – Critical vulnerabilities found

2022 – **Platform Certificates leaked**

2023

2024



All you need to know about the Android platform certificate leak

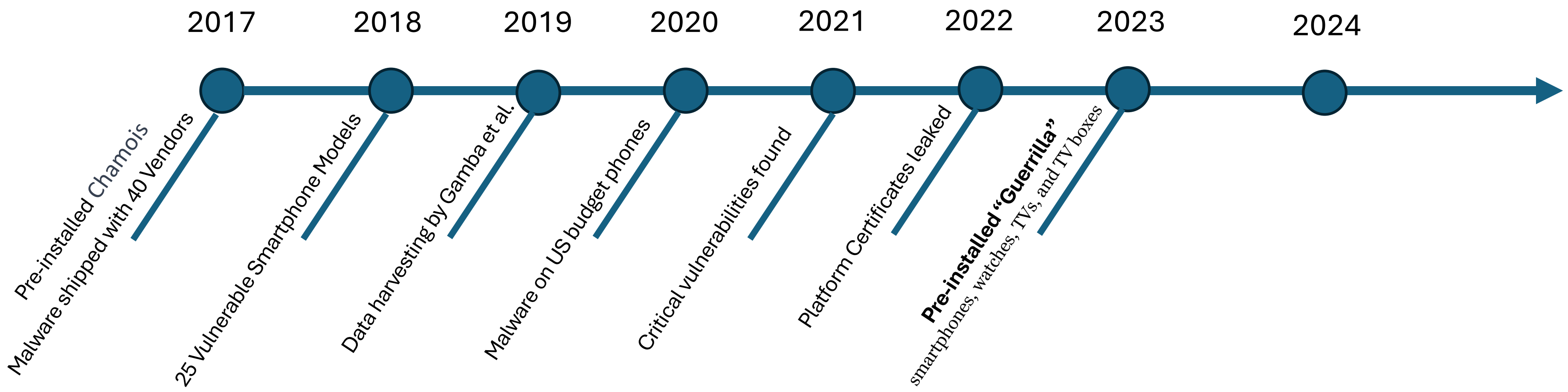
Rest assured that it can no longer be exploited

BY MANUEL VONAU PUBLISHED DEC 9, 2022

Google disclosed a serious issue mainly affecting Samsung phones toward the end of 2022. Some platform certificates from Samsung got into the hands of bad actors, which allowed them to create malware with elevated permissions, potentially allowing hackers to hijack phones by loading tampered software on them. This seems to affect all phones from a given manufacturer, regardless of whether you have Android 13. Here's everything we know about the vulnerability and what you can do to protect yourself and your phone.

https://www.androidpolice.com/android-platform-certificate-leak-explainer/

History of Incidents



2017

Pre-installed Chamois
Malware shipped with 40 Vendors

2018

25 Vulnerable Smartphone Models

2019

Data harvesting by Gamba et al.

2020

Malware on US budget phones

2021

Critical vulnerabilities found

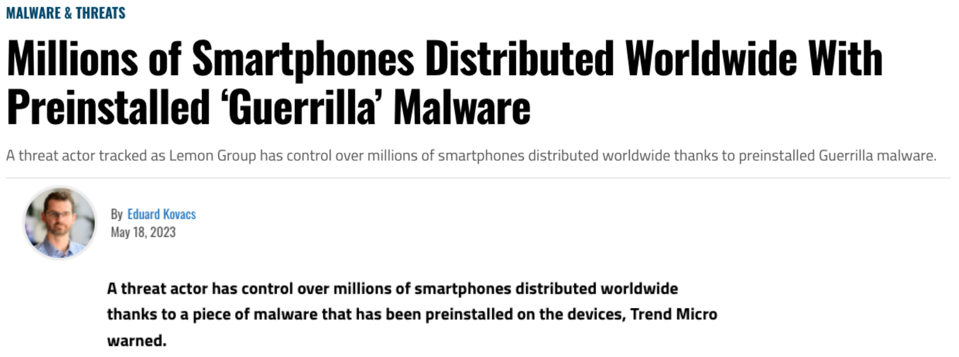
2022

Platform Certificates leaked

2023

Pre-installed “Guerrilla”
smartphones, watches, TVs, and TV boxes

2024



MALWARE & THREATS

Millions of Smartphones Distributed Worldwide With Preinstalled ‘Guerrilla’ Malware

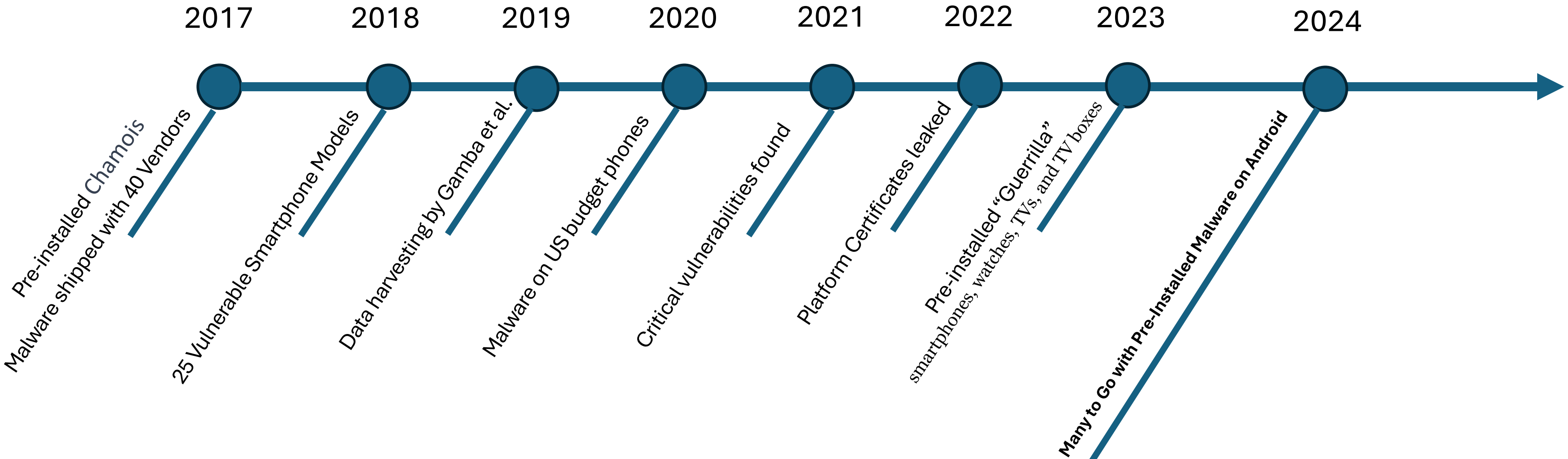
A threat actor tracked as Lemon Group has control over millions of smartphones distributed worldwide thanks to preinstalled Guerrilla malware.

By Eduard Kovacs
May 18, 2023

A threat actor has control over millions of smartphones distributed worldwide thanks to a piece of malware that has been preinstalled on the devices, Trend Micro warned.

https://www.securityweek.com/millions-of-smartphones-distributed-worldwide-with-preinstalled-guerrilla-malware/

History of Incidents



2017

Pre-installed Chamois

Malware shipped with 40 Vendors

2018

25 Vulnerable Smartphone Models

2019

Data harvesting by Gamba et al.

2020

Malware on US budget phones

2021

Critical vulnerabilities found

2022

Platform Certificates leaked

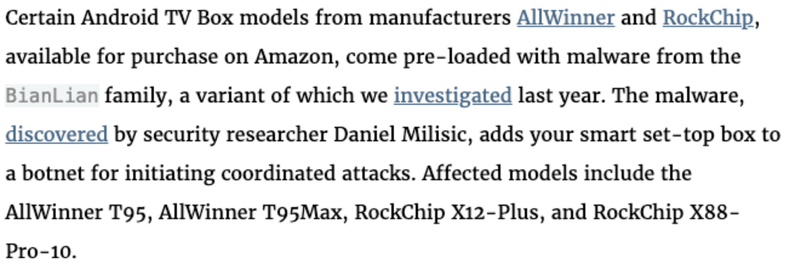
2023

Pre-installed “Guerrilla”

smartphones, watches, TVs, and TV boxes

2024

One Down, Many to Go with Pre-Installed Malware on Android



Certain Android TV Box models from manufacturers AllWinner and RockChip, available for purchase on Amazon, come pre-loaded with malware from the BianLian family, a variant of which we investigated last year. The malware, discovered by security researcher Daniel Milisic, adds your smart set-top box to a botnet for initiating coordinated attacks. Affected models include the AllWinner T95, AllWinner T95Max, RockChip X12-Plus, and RockChip X88-Pro-10.

https://www.eff.org/deeplinks/2024/11/one-down-many-go-pre-installed-malware-android

Research Motivation

According to our research, and by research I mean “googling”, there seems to be an issue with the security and privacy of pre-installed apps…

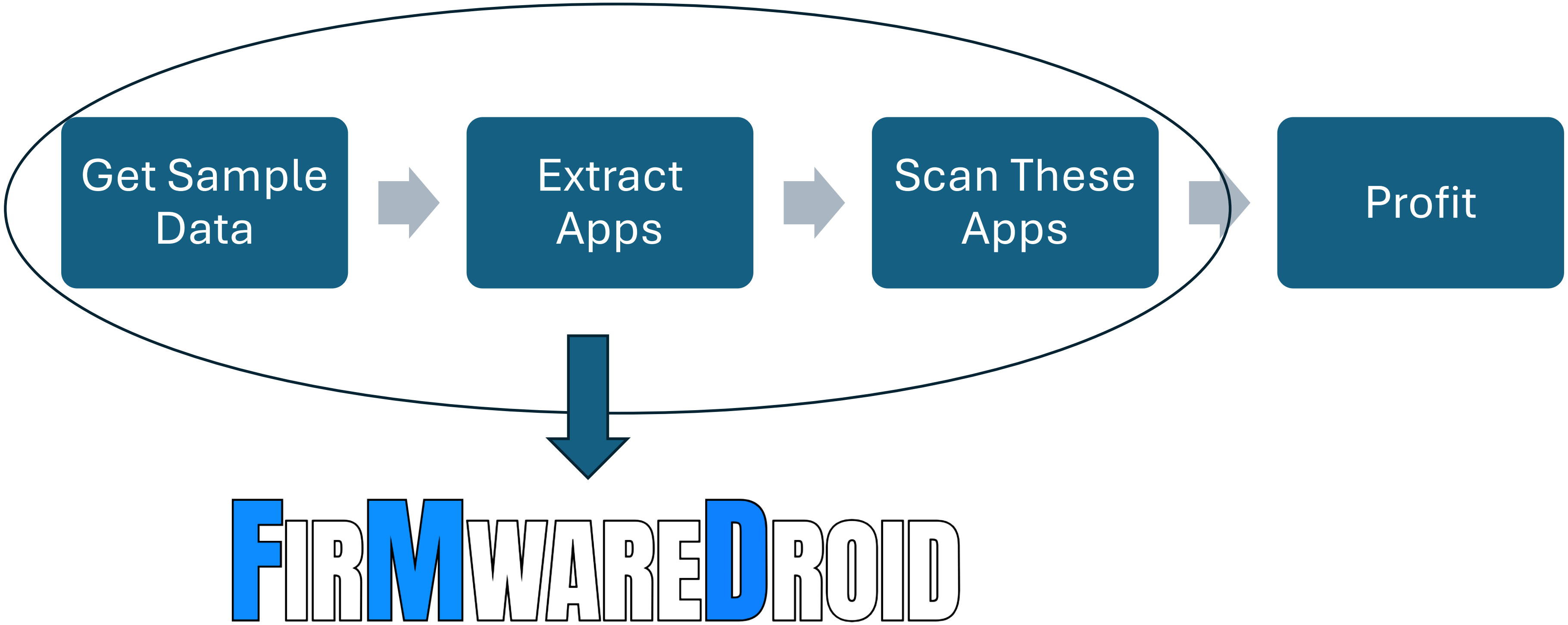
Our main goal is to make the analysis less laboursome.

- **Basically, lowering the bar for researchers to start analysing these apps**
  - Have some tooling to support the analysis
  - Figuring out if there is any good existing tools for the job

- Compare different Android firmware
- Learn what is shipped on our Android devices

- Hopefully, find some cool vulnerabilities
- Maybe writing some papers about it

Plan for the Case Study



Get Sample Data

Extract Apps

Scan These Apps

Profit

Getting The Sample Data

Where to find the data?

- Extracting firmware/apps from devices?
    - Crowed-sourcing?

Where to find the data?

- ~~Extracting firmware/apps from devices?~~
    - ~~Crowed-sourcing?~~

- Official vendor website?
    - Mostly not existing

Where to Firmware Samples?

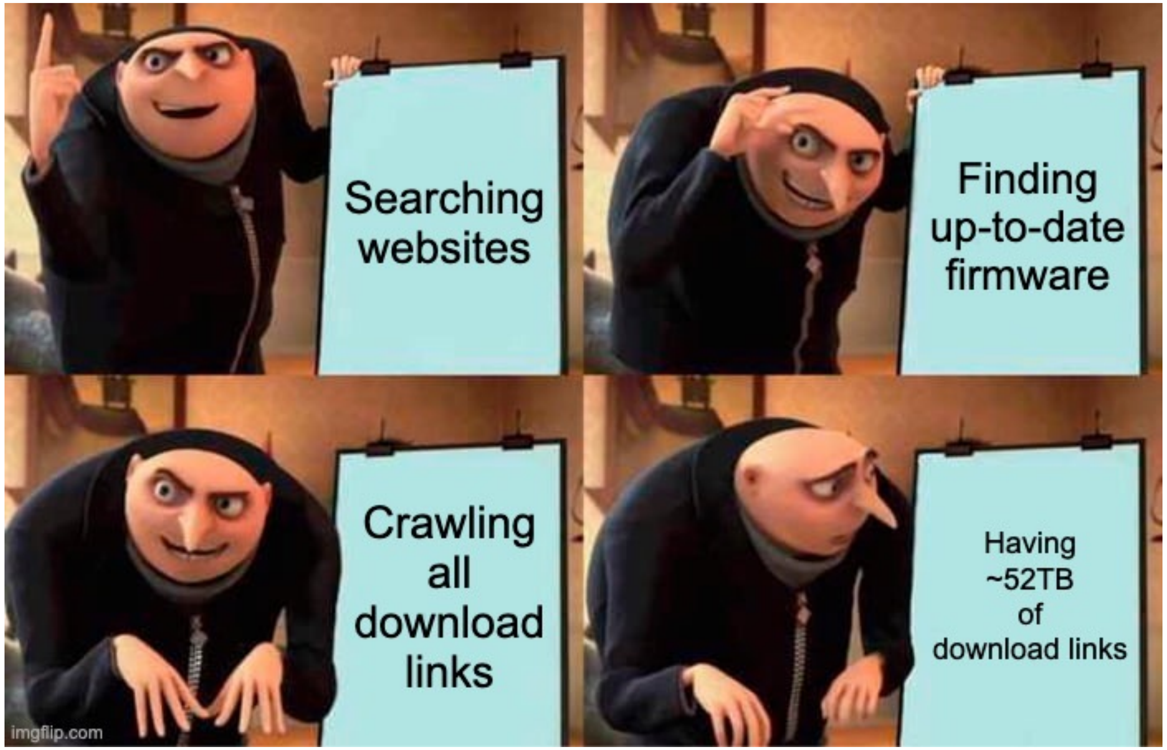
- ~~Extracting firmware/apps from devices?~~
  - ~~Crowed-sourcing?~~

- ~~Official vendor website?~~
  - ~~Mostly not existing~~

- Shady webforums
  - Many websites claiming to have ”official” stock firmware
    - Firmware taken from update servers
    - Custom firmware
    - Fake “stock” firmware

Crawling For Firmware

Developed a small 321 lines of code web-crawler. Just to collect some download links from various websites…


Searching websites
Finding up-to-date firmware
Crawling all download links
Having ~52TB of download links
imgflip.com

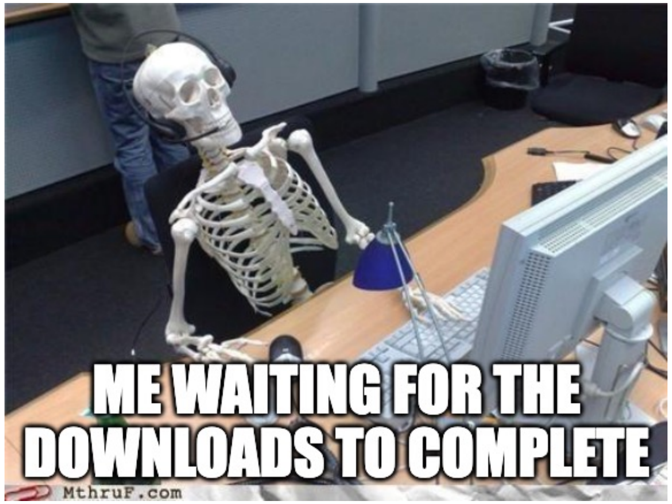
…worked better than expected

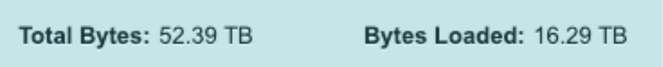
Let’s Get Research Data

2021: We collected around ~10TB of firmware samples
2023: We collected links for ~52TB and downloaded another ~16TB of samples for our current study

Basically, we found download links to all Android versions (v4 to v15)

- For free
- Including official stock ROMs, fake stock ROMS, and customs ROMs




Total Bytes: 52.39 TB **Bytes Loaded:** 16.29 TB

Data is available for academic researchers…
For the others: I might have a text file with some download links, that I’m eventually willing to share

A Word on File Formats

Compression Formats

.zip
.tar
.gzip
.7z
.lz4
.bin
.dat
.br
.nb0
.pac
.ubi
.ozip
...

Partition Formats

.ext2
.ext4
.sparse
.f2fs
.erofs
.payload_bin
.img
.bin
.raw
.super
...

Android container formats

.apex
.apk
.aab
...

Not including formats for kernel, radio, tos, pvmfw, cache, ...

Developing an Extraction Routine

- Maybe some of you know the tool “unblob”
  - Upsides:
    - Really handy to extract compressed files and some partitions
    - Supports Android sparse images
  - Downsides:
    - Performance is often slow in case of large firmware
    - Android custom firmware formats not supported
    - Extraction depth is hard to set
    - Came out years after me implementing an extraction routine

- Developed an own extraction routine with a mix of state-of-the-art tools and mounting tools to handle Android’s mix of file formats
  - 2021: We were able to extract ~8’000 samples
  - 2024: We were able to extract ~20k samples
    - Still open challenges in doing this stable, generic, and fast

Digging Into The Data

Developing FirmwareDroid (FMD): The Pipeline

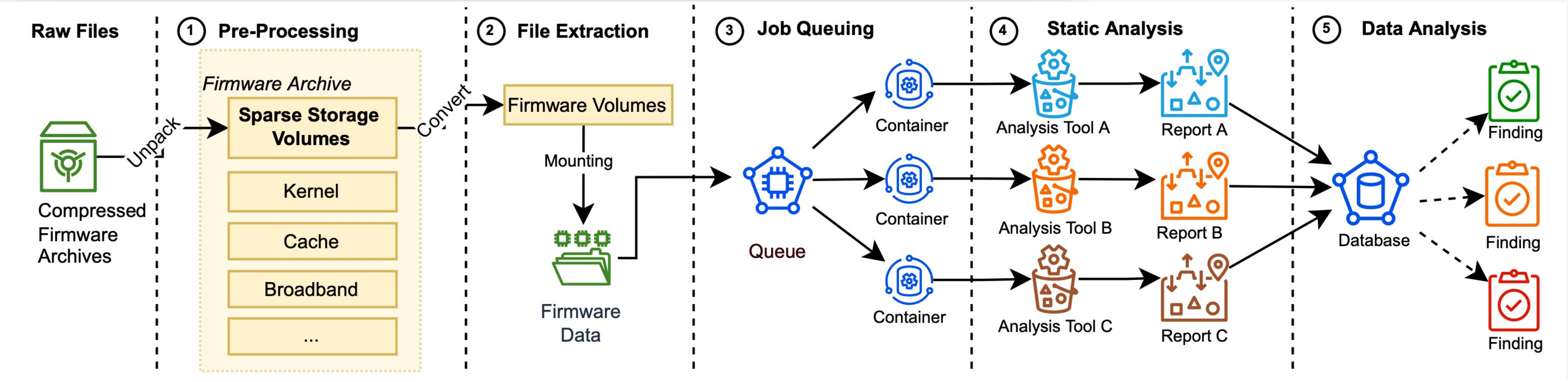
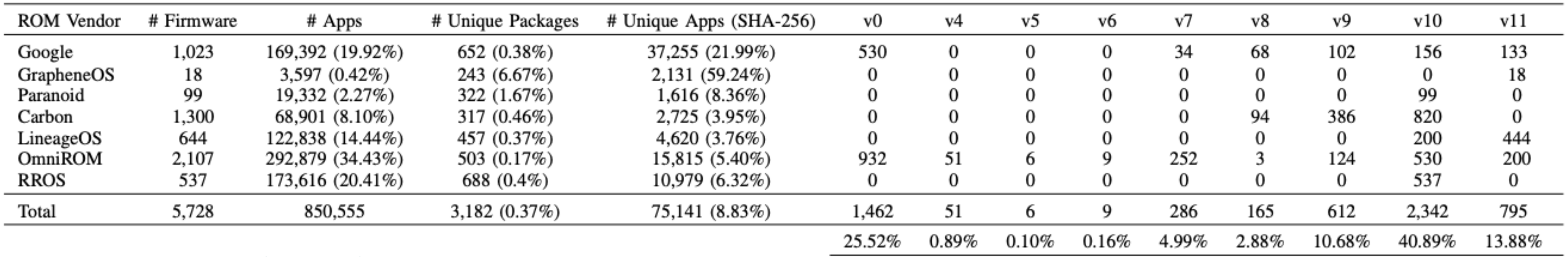


Image from our paper: https://ieeexplore.ieee.org/document/10172951

Implemented this tool back in 2020 as part of my master thesis and we did some experiments:

- Detecting Malware
- Searching Vulnerabilities
- **Permission Analysis**
- **Tracker Analysis**

Case Study: 2021



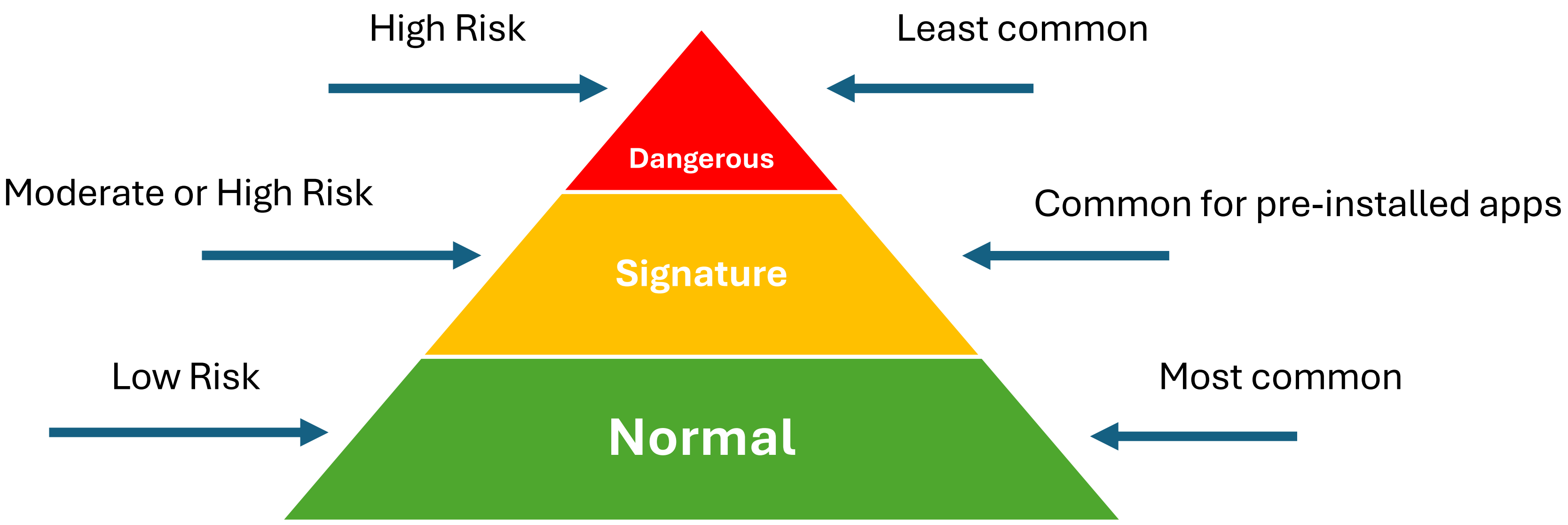
ROM Vendor	# Firmware	# Apps	# Unique Packages	# Unique Apps (SHA-256)	v0	v4	v5	v6	v7	v8	v9	v10	v11
Google	1,023	169,392 (19.92%)	652 (0.38%)	37,255 (21.99%)	530	0	0	0	34	68	102	156	133
GrapheneOS	18	3,597 (0.42%)	243 (6.67%)	2,131 (59.24%)	0	0	0	0	0	0	0	0	18
Paranoid	99	19,332 (2.27%)	322 (1.67%)	1,616 (8.36%)	0	0	0	0	0	0	0	99	0
Carbon	1,300	68,901 (8.10%)	317 (0.46%)	2,725 (3.95%)	0	0	0	0	0	94	386	820	0
LineageOS	644	122,838 (14.44%)	457 (0.37%)	4,620 (3.76%)	0	0	0	0	0	0	0	200	444
OmniROM	2,107	292,879 (34.43%)	503 (0.17%)	15,815 (5.40%)	932	51	6	9	252	3	124	530	200
RROS	537	173,616 (20.41%)	688 (0.4%)	10,979 (6.32%)	0	0	0	0	0	0	0	537	0
Total	5,728	850,555	3,182 (0.37%)	75,141 (8.83%)	1,462	51	6	9	286	165	612	2,342	795
					25.52%	0.89%	0.10%	0.16%	4.99%	2.88%	10.68%	40.89%	13.88%

Table from our paper: https://ieeexplore.ieee.org/document/10172951

- 5,728 firmware samples for experimenting with 75,141 unique apps
- We used firmware from seven vendors that share their firmware on an official channel and at no cost
- Focus:
  - Android 10 and 11
  - Static Analysis with existing tooling
    - Permissions used
    - Trackers identified

Android Permissions Expectations

There are three main permission levels on Android:



High Risk

Least common

Dangerous

Moderate or High Risk

Common for pre-installed apps

Signature

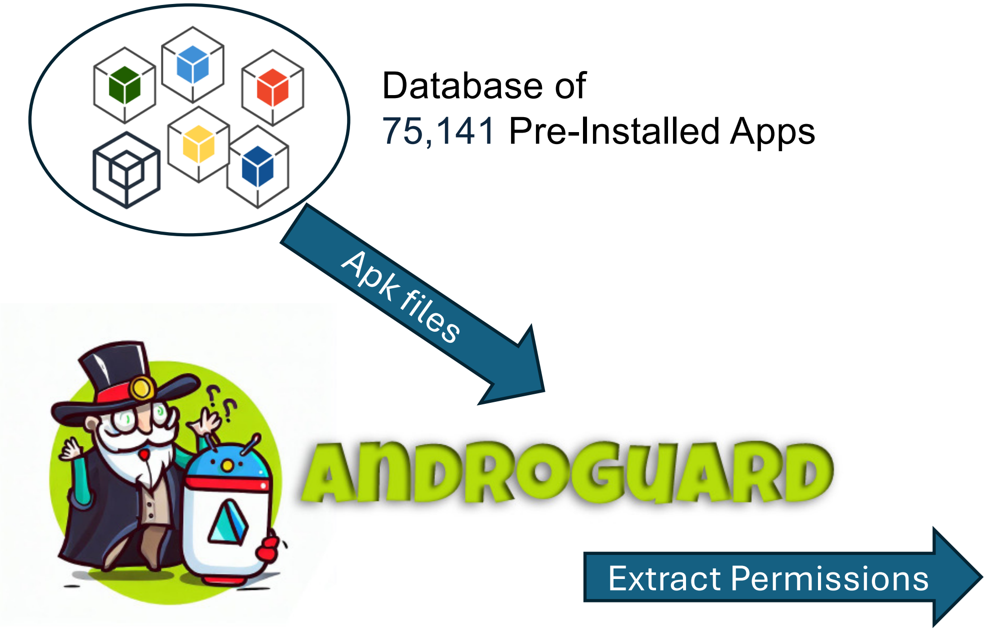
Low Risk

Most common

Normal

Hypothesis: If we scan the apps in our database, we expect to see a kind of pyramid distribution

Getting Permission Data



Database of
75,141 Pre-Installed Apps

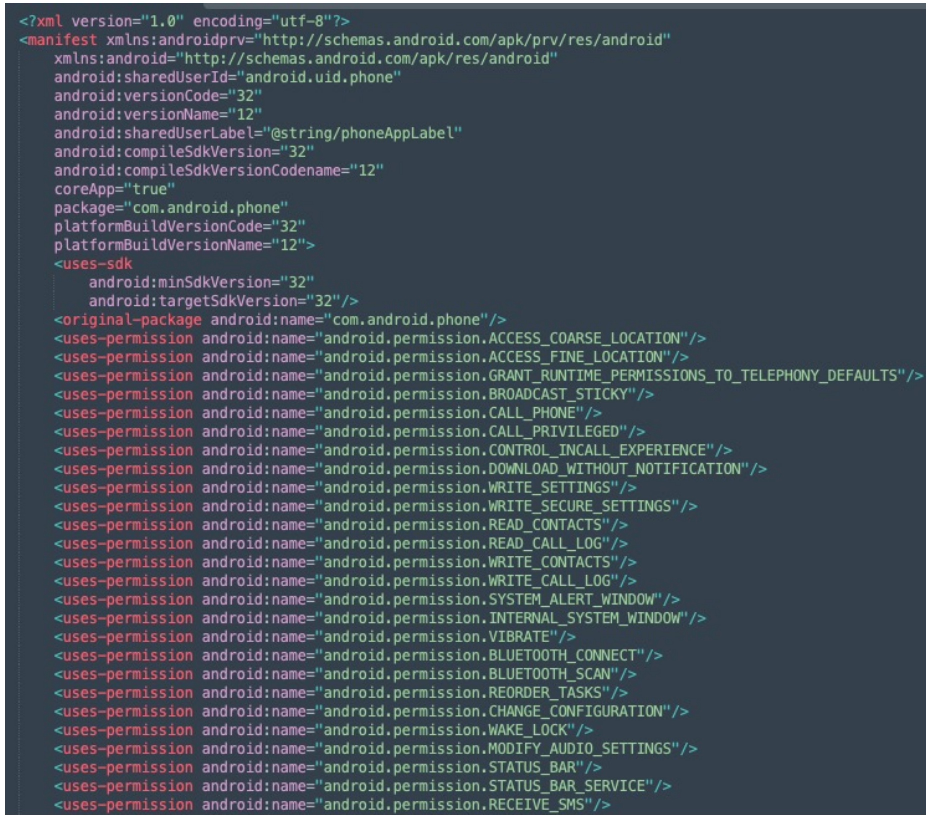
Apk files

Extract Permissions

Image source: https://github.com/androguard/androguard

"Androguard is a full python tool to play with Android files." – AndroGuard Devs

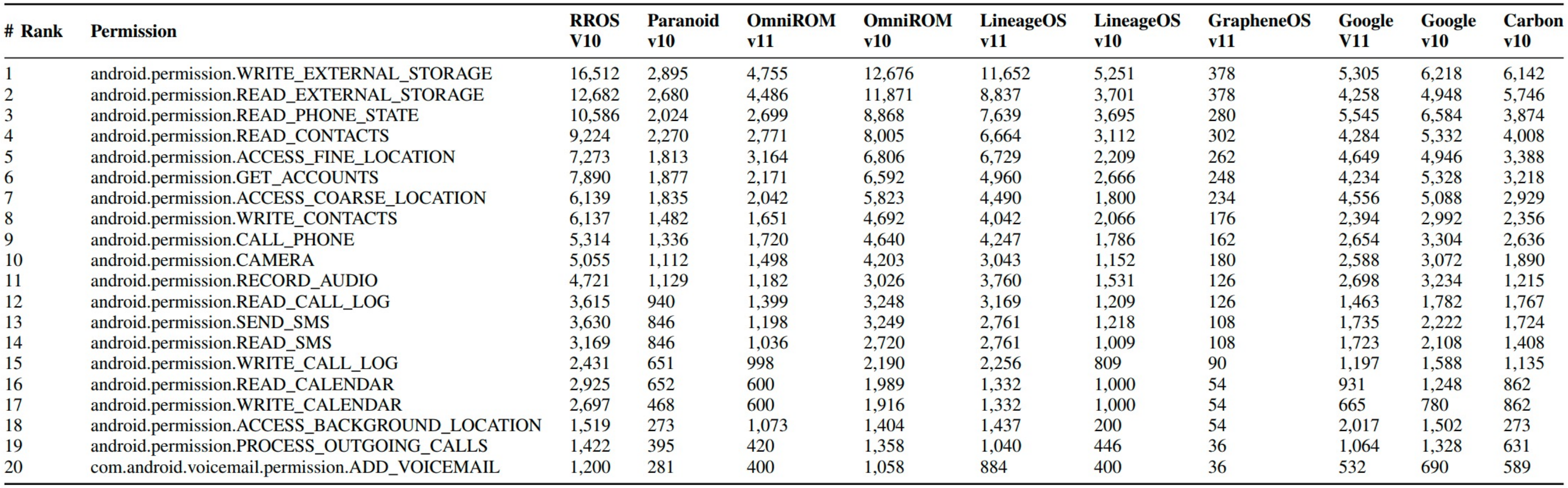
AndroidManifest.xml from "com.android.phone"



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:androidprv="http://schemas.android.com/apk/prv/res/android"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:sharedUserId="android.uid.phone"
    android:versionCode="32"
    android:versionName="12"
    android:sharedUserLabel="@string/phoneAppLabel"
    android:compileSdkVersion="32"
    android:compileSdkVersionCodename="12"
    coreApp="true"
    package="com.android.phone"
    platformBuildVersionCode="32"
    platformBuildVersionName="12">
    <uses-sdk
        android:minSdkVersion="32"
        android:targetSdkVersion="32"/>
    <original-package android:name="com.android.phone"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.GRANT_RUNTIME_PERMISSIONS_TO_TELEPHONY_DEFAULTS"/>
    <uses-permission android:name="android.permission.BROADCAST_STICKY"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.CALL_PRIVILEGED"/>
    <uses-permission android:name="android.permission.CONTROL_INCALL_EXPERIENCE"/>
    <uses-permission android:name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION"/>
    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
    <uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.READ_CALL_LOG"/>
    <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
    <uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.INTERNAL_SYSTEM_WINDOW"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
    <uses-permission android:name="android.permission.BLUETOOTH_SCAN"/>
    <uses-permission android:name="android.permission.REORDER_TASKS"/>
    <uses-permission android:name="android.permission.CHANGE_CONFIGURATION"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
    <uses-permission android:name="android.permission.STATUS_BAR"/>
    <uses-permission android:name="android.permission.STATUS_BAR_SERVICE"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

Permission Analysis

Top 20 requested dangerous permissions for pre-installed apps on Android 10 and 11



# Rank	Permission	RROS V10	Paranoid v10	OmniROM v11	OmniROM v10	LineageOS v11	LineageOS v10	GrapheneOS v11	Google V11	Google v10	Carbon v10
1	android.permission.WRITE_EXTERNAL_STORAGE	16,512	2,895	4,755	12,676	11,652	5,251	378	5,305	6,218	6,142
2	android.permission.READ_EXTERNAL_STORAGE	12,682	2,680	4,486	11,871	8,837	3,701	378	4,258	4,948	5,746
3	android.permission.READ_PHONE_STATE	10,586	2,024	2,699	8,868	7,639	3,695	280	5,545	6,584	3,874
4	android.permission.READ_CONTACTS	9,224	2,270	2,771	8,005	6,664	3,112	302	4,284	5,332	4,008
5	android.permission.ACCESS_FINE_LOCATION	7,273	1,813	3,164	6,806	6,729	2,209	262	4,649	4,946	3,388
6	android.permission.GET_ACCOUNTS	7,890	1,877	2,171	6,592	4,960	2,666	248	4,234	5,328	3,218
7	android.permission.ACCESS_COARSE_LOCATION	6,139	1,835	2,042	5,823	4,490	1,800	234	4,556	5,088	2,929
8	android.permission.WRITE_CONTACTS	6,137	1,482	1,651	4,692	4,042	2,066	176	2,394	2,992	2,356
9	android.permission.CALL_PHONE	5,314	1,336	1,720	4,640	4,247	1,786	162	2,654	3,304	2,636
10	android.permission.CAMERA	5,055	1,112	1,498	4,203	3,043	1,152	180	2,588	3,072	1,890
11	android.permission.RECORD_AUDIO	4,721	1,129	1,182	3,026	3,760	1,531	126	2,698	3,234	1,215
12	android.permission.READ_CALL_LOG	3,615	940	1,399	3,248	3,169	1,209	126	1,463	1,782	1,767
13	android.permission.SEND_SMS	3,630	846	1,198	3,249	2,761	1,218	108	1,735	2,222	1,724
14	android.permission.READ_SMS	3,169	846	1,036	2,720	2,761	1,009	108	1,723	2,108	1,408
15	android.permission.WRITE_CALL_LOG	2,431	651	998	2,190	2,256	809	90	1,197	1,588	1,135
16	android.permission.READ_CALENDAR	2,925	652	600	1,989	1,332	1,000	54	931	1,248	862
17	android.permission.WRITE_CALENDAR	2,697	468	600	1,916	1,332	1,000	54	665	780	862
18	android.permission.ACCESS_BACKGROUND_LOCATION	1,519	273	1,073	1,404	1,437	200	54	2,017	1,502	273
19	android.permission.PROCESS_OUTGOING_CALLS	1,422	395	420	1,358	1,040	446	36	1,064	1,328	631
20	com.android.voicemail.permission.ADD_VOICEMAIL	1,200	281	400	1,058	884	400	36	532	690	589

Table from our paper: https://ieeexplore.ieee.org/document/10172951

- 88.14% of the permissions are signature.
    - 28.57% of the permissions in our dataset are custom third-party permissions.
- 3.56% are dangerous.
- 8.21% are normal declared permissions.

Tracking The Trackers

A tracker is a piece of software that logs the behaviour of a program or user.

Exodus: https://github.com/Exodus-Privacy/exodus-core

«εxodus analyses Android applications. It looks for embedded trackers and lists them. A tracker is a piece of software meant to collect data about you or what you do. In a way, εxodus reports are a way of knowing what really are the ingredients of the cake you are eating. εxodus does not decompile applications, its analysis technique is entirely legal.” – via https://exodus-privacy.eu.org/en/page/what/

Image source: https://reports.exodus-privacy.eu.org/en/

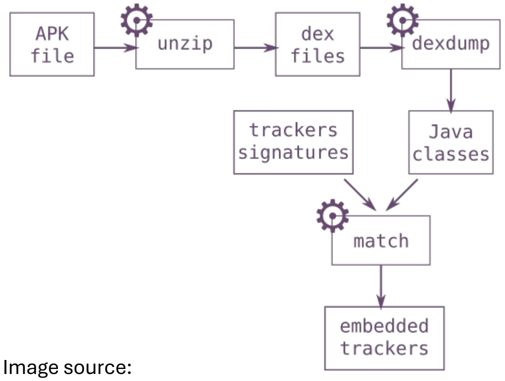


Image source:
https://exodus-privacy.eu.org/en/post/exodus_static_analysis/

Java Class Matching:

- Benefits:
  - Small False Positive Rate
- Downsides:
  - If the tracker is unkown we will miss it

Trackers Found

We found 41,175 known trackers with Exodus.

- Note: One app can have more than one tracker

In total 24,597 (**20.53%**) of the pre-installed apps in our dataset use trackers (have at least one).

Are these trackers all bad?
Do they consent?
What kind of data is collected?
When is the data collected and why?

There are still open research questions

… we need more insights and better tooling to answer this in a scientific way

Are we done yet?

Improve Tooling

- Most tools are far away from giving precise results:
  - Usually, high amounts of false positives
    - Vulnerabilities are often not exploitable
      - Lacking context awareness
      - Lack of configuration possibilities
- Re-Inventing the wheel
- Many tools do not scale well

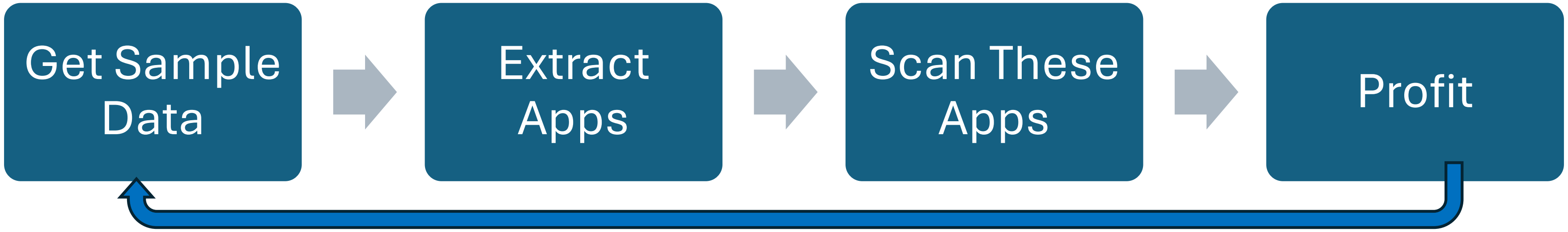
- **However, existing tools are used in research studies**
  - For instance, “AndroGuard” is use it many studies
  - Tools are used for comparison to develop new and better methods
  - Research tools are often unmaintained.
    - Thus, hard to setup after some time

State February 2025

- AndroGuard
- Quark-Engine
- APKiD
- Exodus-Core
- APKLeaks
- MobSFScan
- APKscan
- FlowDroid
- Trueseeing
- APKScan
- Deprecated:
  - Qark (deprecated, no updates by the author)
  - Androwarn (deprecated, no updates by the author)
  - SUPER Android Analyzer (deprecated, discontinued by the author)

- **YOUR TOOL HERE?**

Moving Forward



Get Sample Data

Extract Apps

Scan These Apps

Profit

Improve and repeat the experiments

Improvements from the original case study:

- Refactored the whole code base
    - Reimplemented the extraction process:
        - Better routines
        - More vendors supported
        - Added support for Android custom file formats
    - Added more tools
- Maybe program a GUI

Next Goal: Making the dynamic analysis of pre-installed apps possible

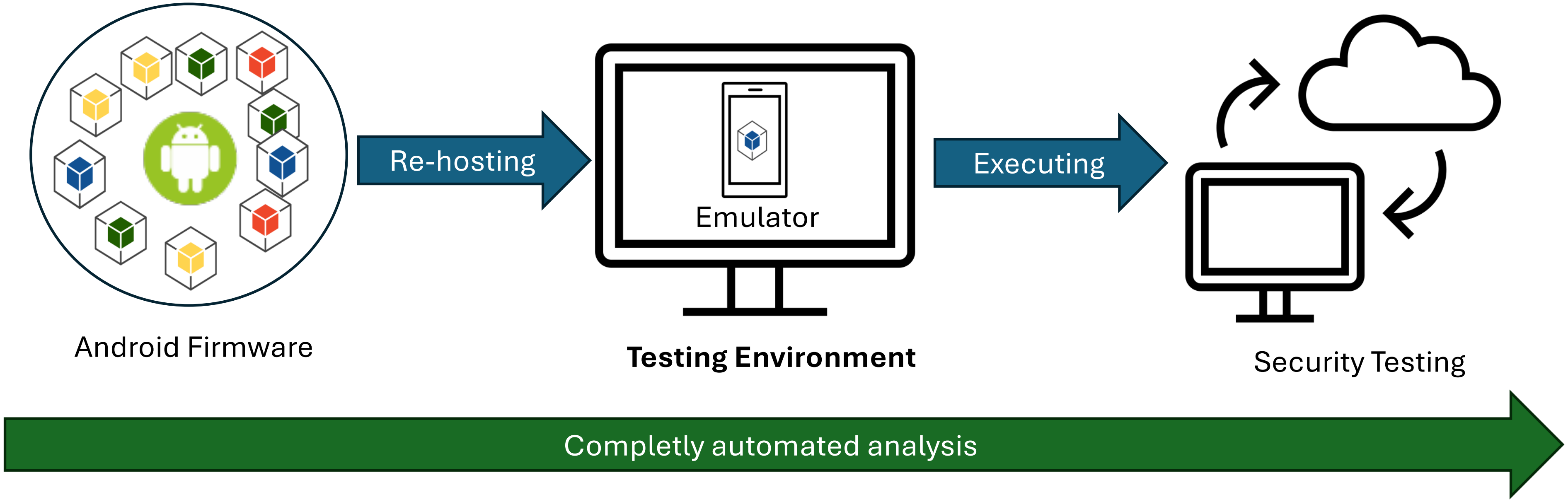
Challenges Testing Pre-Installed Apps

Static analysis is limited and can’t give us full insights

1. Problematic Installation
   - App privileges
     - Signature: App must be signed with a platform key
     - Privilege: App must live in /system/priv-apps/ (System partition are read-only)
   - Singleton Apps: App can’t be installed because it already exists in the testing environment
   - Ressources must be loaded before the app is started
   - Apps must be installed during first boot process
2. App runs under a shared UID
3. App is headless
4. App is dependent on custom hardware
5. App has software dependencies to other components (Collusion)
   - Companion apps
   - Deamons
   - Binaries (ARM, x86)
   - Custom Android framework
   - Remote servers
6. App prevents execution
   - CPU Architecture (x86 / ARMv8a)
   - Environment checks (emulator detection)
   - Root detection

Re-Hosting Pre-Installed Apps

Objective: Re-hosting pre-installed apps from device to the Android emulator



Re-hosting

Emulator

Executing

Android Firmware

Testing Environment

Security Testing

Completly automated analysis

THANK
YOU

Q & A

Material:

- Paper: *FirmwareDroid: Towards Automated Static Analysis of Pre-Installed Android Apps*
- Code: https://github.com/FirmwareDroid/FirmwareDroid
- Slides: https://7homassutter.github.io/publications/

Contact Info's:

- Linkedin: www.linkedin.com/in/thomas-sutter-0a2977169
- Mail: thomas (dot) sutter (at) unibe (dot) ch