ISEC TU Graz

Mobile Security Research

Mobile Security 2026

Florian Draschbacher
florian.draschbacher@tugraz.at

Some slides based on material by **Yanick Fratantonio**

Introduction

What's this presentation about?

- You developed an intuition about the security of mobile systems
  - How can you use that knowledge in actual security research?
  - Systematically analyze attack surfaces
  - Identify and classify vulnerabilities

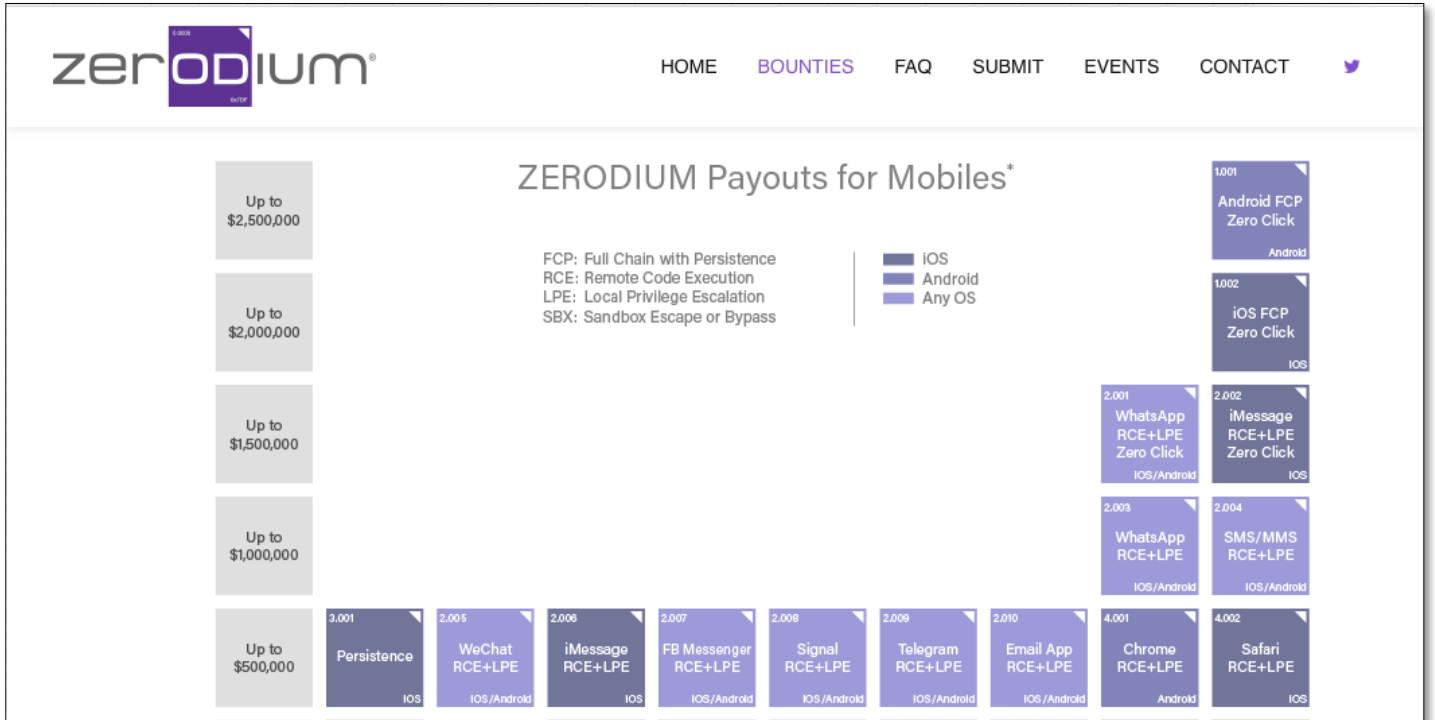
- **Why?**
  - Improve the security of mobile ecosystems
  - Develop new attack methods and corresponding defenses
  - Find and eliminate malware
  - Earn some bug bounties or academic title along the way!

Who is joining the hunt?

- Security Research Community
  - Universities
  - Companies
- Device Manufacturers
- Gray markets
  - State Actors
    - Military, intelligence services
  - Companies
    - Selling exploitation kits as products
  - Black Hat Hackers
    - Ransoms, selling stolen data, …

Bug Bounty Programs

- Manufacturers realised they are competing with exploit gray markets
- E.g. Zerodium pays up to 2.5m$
  - Sells exploits to governments

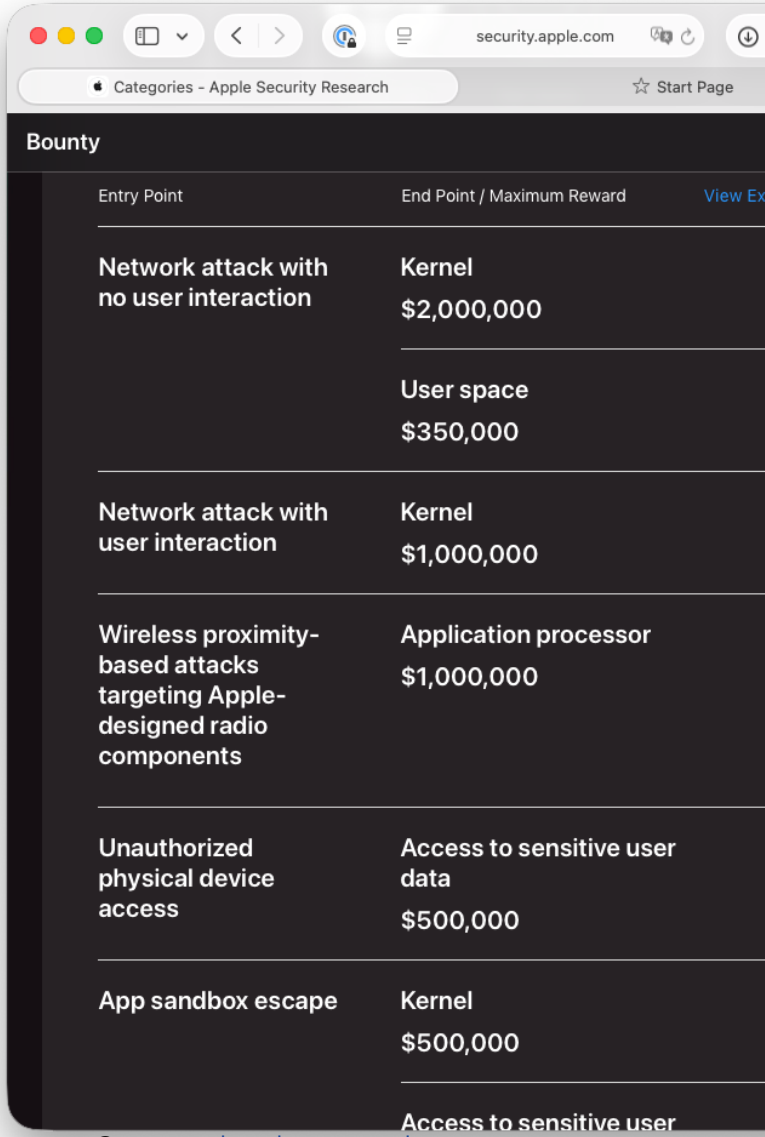


Source: zerodium.com

Bug Bounty Programs

- Incentivize white hat hackers to invest their time
  - Convince black hat hackers to join the good side
- In theory: Clearly specified scope and payout levels
  - E.g. Meta: Social Engineering Attacks are out of scope
- In practice: Companies are still reluctant to pay bounty

Source: arstechnica.com



Source: developer.apple.com

Play Store Bug Bounty

- Google even offers bug bounties for the most popular apps on Play Store

Google Play Security Reward Program Rules

The Google Play Security Reward Program (GPSRP) is a vulnerability reward program offered by Google Play in collaboration with the developers of certain popular Android apps. It recognizes the contributions of security researchers who invest their time and effort to help make apps on Google Play more secure.

Category	1) Remote/no user interaction	2) User must follow a link, vulnerable app must be already installed	3) User must install malicious app or victim app is configured in a non-default way	4) Attacker must be on the same network (e.g. MiTM)
Arbitrary code execution	$20,000	$10,000	$4,000	$1,000
Theft of sensitive data	$5,000	$3,000	$1,000	$500

Organization/Developer	Package Name(s)
8bit Solutions LLC	com.x8bit.bitwarden
Airbnb	com.airbnb.android
Alibaba	com.alibaba.aliexpresshd
Amazon	com.amazon.mShop.android.shopping, com.amazon.avod.thirdpartyclient, com.ar
Ayopop	com.ayopop
Coinbase	com.coinbase.android, org.toshi, com.coinbase.pro
delight.im	im.delight.letters
Dropbox	com.dropbox.android, com.dropbox.paper

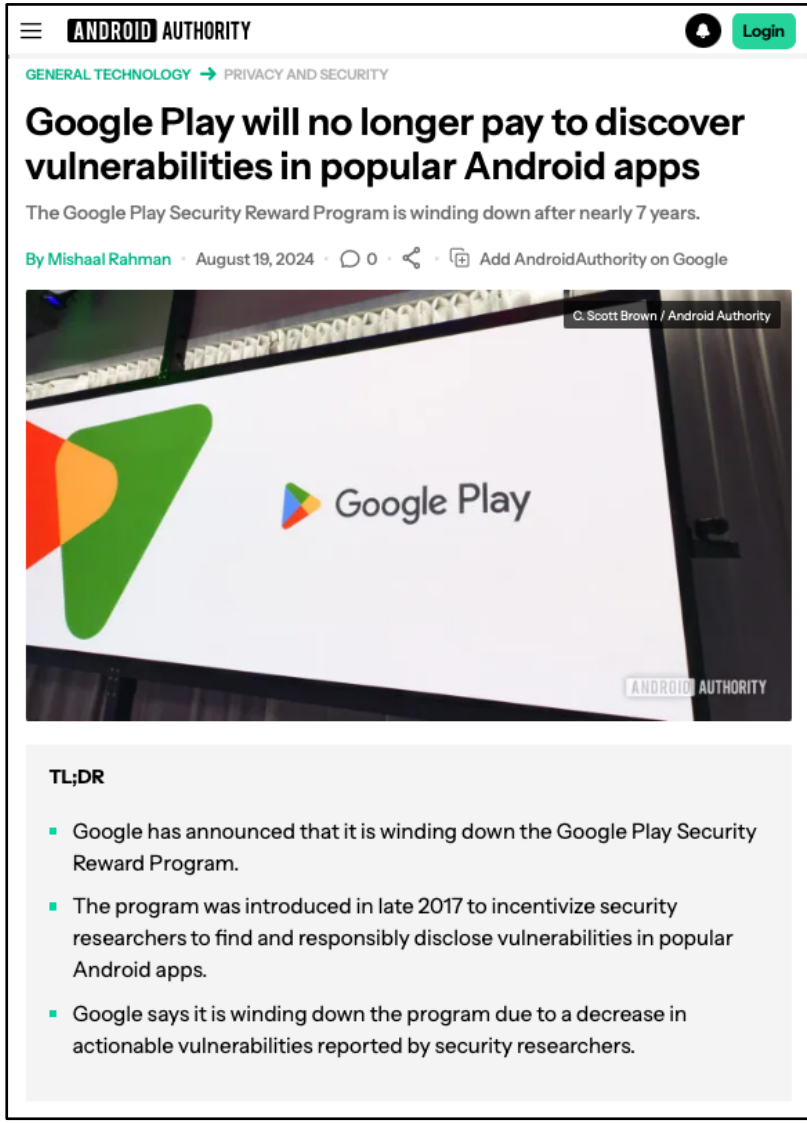
Source: bughunters.google.com

Scope

Only applications developed by Google, by participating developers (in the list below), or with 100 million or more installs are in scope. Only vulnerabilities that work on Android 6.0 devices (with the most up-to-date patches) and higher will qualify.

Bug Bounty Programs: Trends

- Big tech shifts a lot of capital to AI resources
- Raised payouts for high and critical severity findings
  - Lowered payouts for lower severity
- AI tools lead to a huge increase in submissions
  - Until recently, many were completely hallucinated
  - Some vendors stopped or paused bug bounty programs



Legal aspects (Austria)

I am no lawyer!

- Legal to analyse / modify services or products for identifying security bugs?
  - Very difficult question!

- General questions
  - Which jurisdiction applies?
    - **User country!**
  - What services / products / parts are covered?
    - Very imprecise formulation of ‘computer program’
    - In general: **Sequence of computer instructions and corresponding source code**
      - Plus: Material that supported the development process
    - What about the application package?

Legal aspects (Austria)

I am no lawyer!

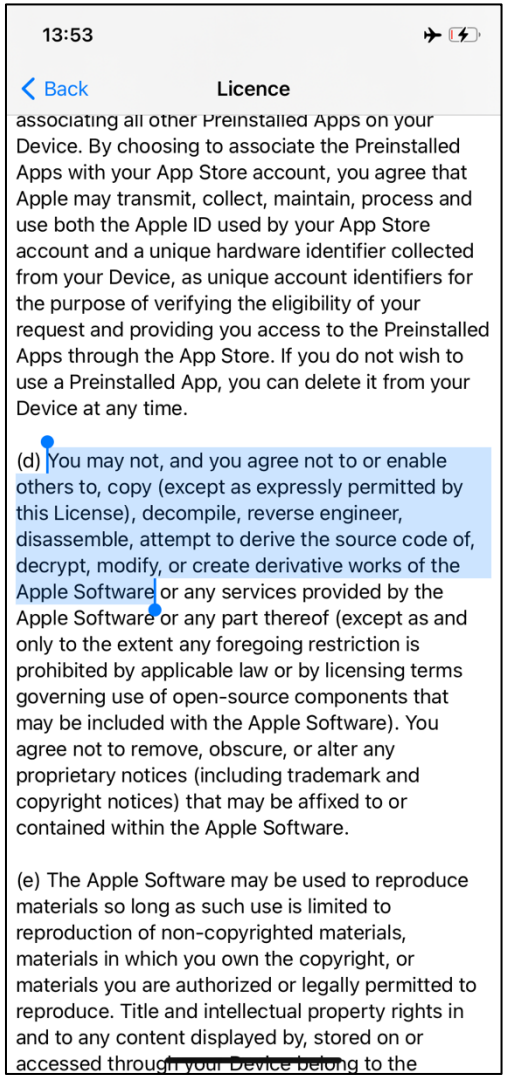
- Strafgesetzbuch / Legal Code
  - Compromising a computer system (§118a): 24-36 months of imprisonment
  - Using hacking tools (§126c): 6-36 months of imprisonment
  - Malicious intent is crucial!
    - Stealing sensitive data, preventing operation, …

- Urheberrecht / Copyright law
  - Assuming you are entitled to the use of some software:
  - Decompilation only allowed for ensuring interoperability (§40e)
    - Exception for matters affecting public safety (§41)
    - Decision by EU Court of Justice: Decompilation also for fixing bugs (Source: lexology.com)
  - Modifications without redistribution allowed! (§40d)
  - Violation: 6 months of imprisonment

Legal aspects (Austria)

I am still no lawyer!

- End User License Agreements (EULA)
  - Often stricter than copyright laws
  - Only legally binding if shown prior to purchase / download
    - Source: linux-magazin.de
- Gesetz gegen unlauteren Wettbewerb (~Trade Secret Law)
  - Reverse-engineering for extracting trade secrets is legal (§26d)
  - Unless agreed otherwise
- Datenschutzgesetz (Data Protection Law)
  - Illegal to extract personal data
- Information on foreign jurisdictions:
  - E.g. from Electronic Frontier Foundation


13:53

Back Licence

associating all other Preinstalled Apps on your Device. By choosing to associate the Preinstalled Apps with your App Store account, you agree that Apple may transmit, collect, maintain, process and use both the Apple ID used by your App Store account and a unique hardware identifier collected from your Device, as unique account identifiers for the purpose of verifying the eligibility of your request and providing you access to the Preinstalled Apps through the App Store. If you do not wish to use a Preinstalled App, you can delete it from your Device at any time.

(d) You may not, and you agree not to or enable others to, copy (except as expressly permitted by this License), decompile, reverse engineer, disassemble, attempt to derive the source code of, decrypt, modify, or create derivative works of the Apple Software or any services provided by the Apple Software or any part thereof (except as and only to the extent any foregoing restriction is prohibited by applicable law or by licensing terms governing use of open-source components that may be included with the Apple Software). You agree not to remove, obscure, or alter any proprietary notices (including trademark and copyright notices) that may be affixed to or contained within the Apple Software.

(e) The Apple Software may be used to reproduce materials so long as such use is limited to reproduction of non-copyrighted materials, materials in which you own the copyright, or materials you are authorized or legally permitted to reproduce. Title and intellectual property rights in and to any content displayed by, stored on or accessed through your Device belong to the

iOS License Agreement

Want to be on the safe side in your research?

- Obtain permission from the service / product provider
- Join bug bounty programs

Safe Harbor Provisions

- We consider these terms to provide you authorisation, including under the Computer Fraud and Abuse Act (CFAA), to test the security of the products and systems identified as in-scope below. These terms do not give you authorisation to intentionally access company data or data from another person's account without their express consent, including (but not limited to) personally identifiable information or data relating to an identified or identifiable natural person.
- If Meta determines in its sole discretion that you have complied in all respects with these Bug Bounty Programme Terms in reporting a security issue to Meta, we will not initiate a complaint to law enforcement or pursue a civil action against you, to include civil actions under the CFAA in connection with the research underlying your report and DMCA claims against you for circumventing the technological measures that we have used to protect the applications in scope. Meta will also not pursue legal action against you for clear accidental or good faith violations of its policy or these terms.

Source: www.facebook.com/whitehat

Important: Check if there are exemptions!

Systematizations

or

A Graphical Intuition on Security

IT Security

We generally concentrate on protecting or attacking assets

Specifically, their

- Confidentiality: Prevent leakage
- Integrity: Prevent modification
- Availability: Prevent destruction

Assets

- **User Data**
  Passwords, Credentials, Activity Logs, Location, Input data, …
- **Application Data**
  Firmware, Private Keys, Certificates, API Endpoints, Copyrighted material
- **Computing Resources**
  Keep the system / service operational even in face of an attacker

Security Vulnerabilities

Question: **What is a security vulnerability?**

Answer: **A weakness that allows an attacker to perform actions that**

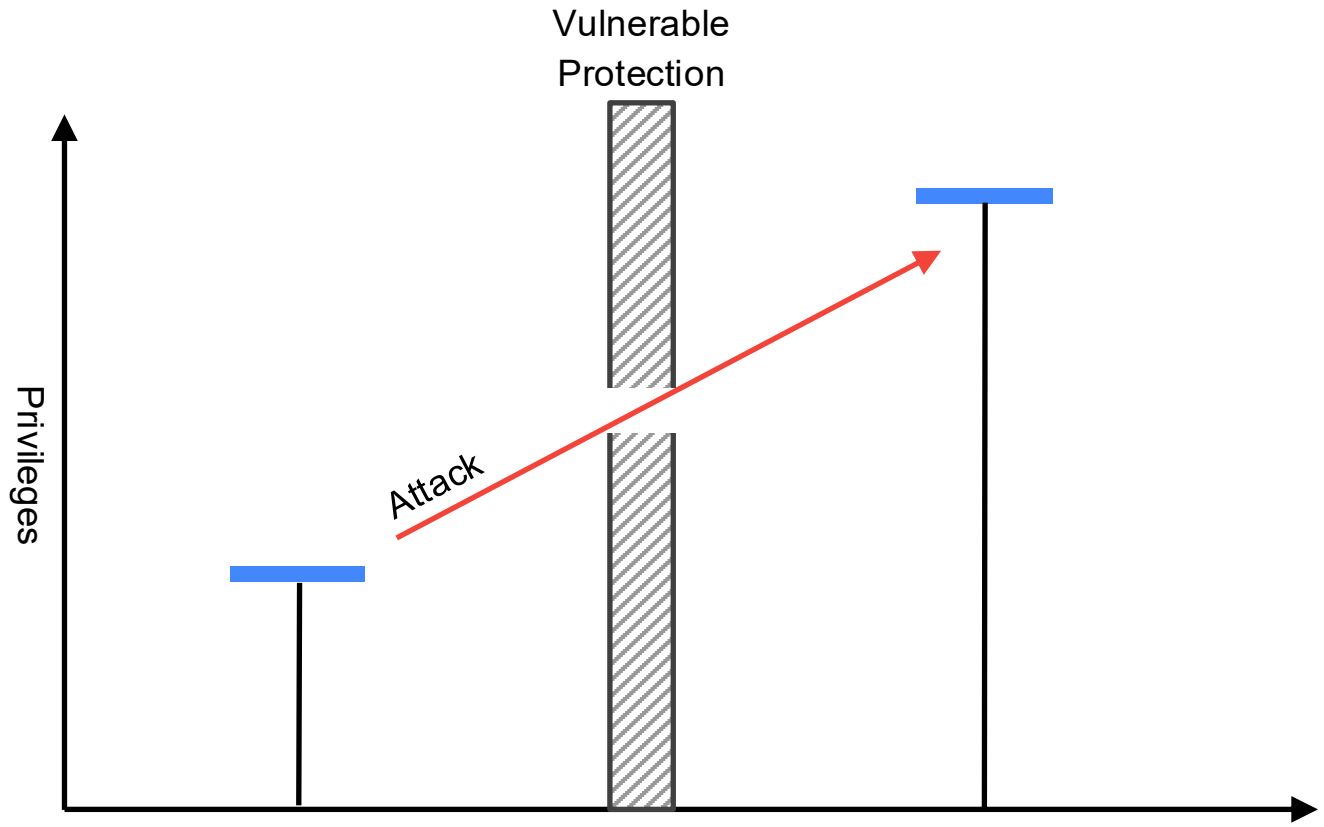
- Were not meant to be possible
- Have negative security repercussions

- Some vulnerabilities are much more important than others
- Various aspects to take into account

Exploit

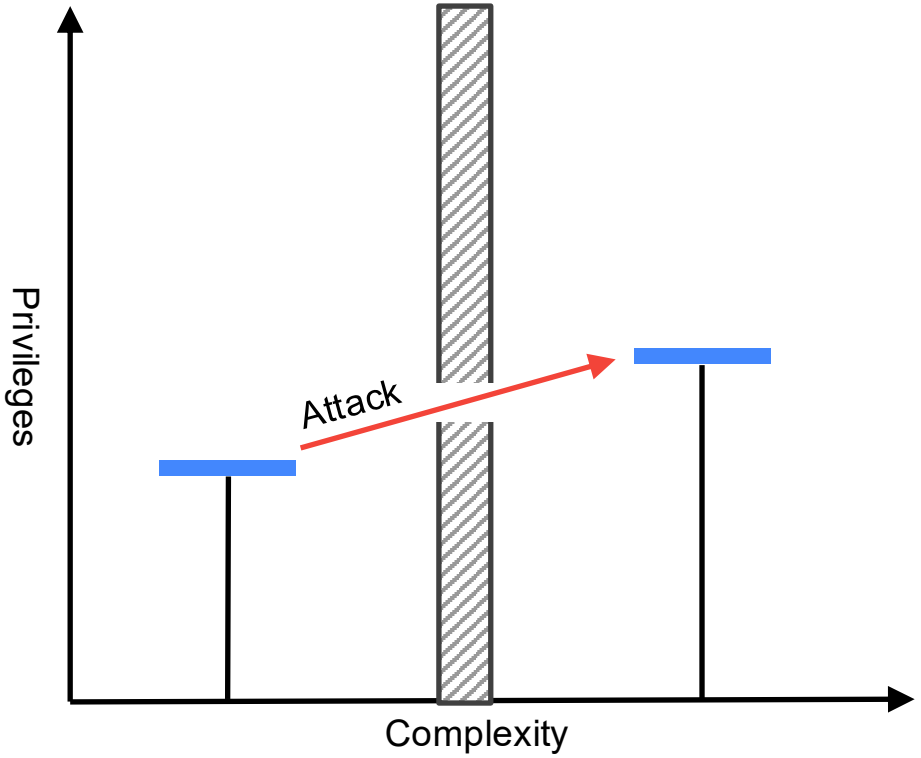
A vulnerability is only a theoretical problem until someone finds an exploit

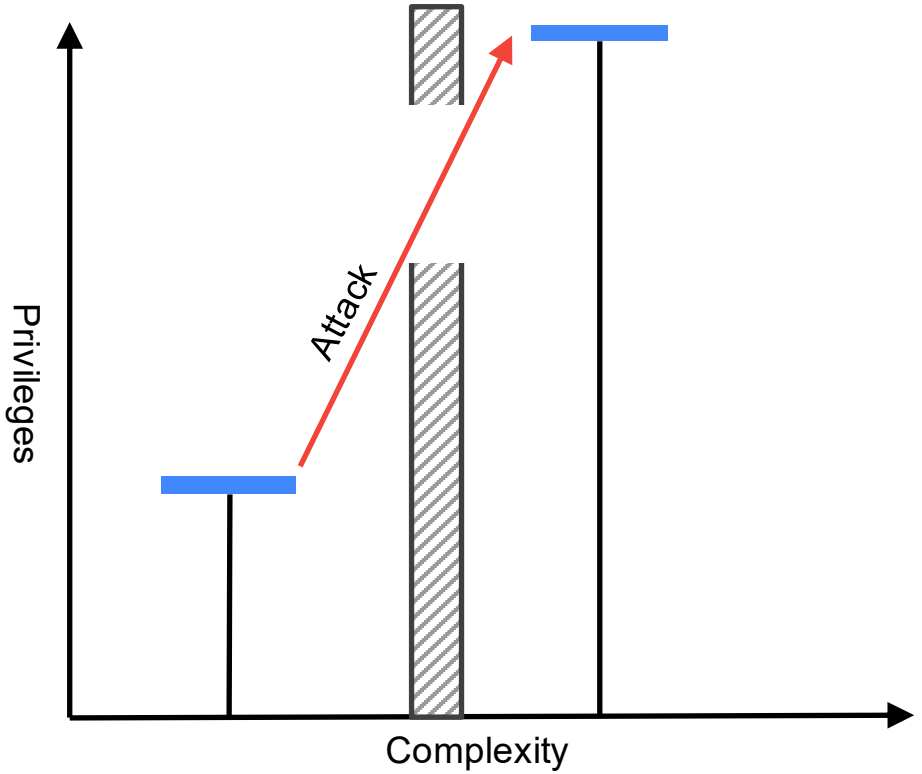
- Make use of the vulnerability for **elevating privileges**



Exploits

- Intuitively, we are looking for vulnerabilities that allow exploits that
  - Maximize the impact (the privilege gain)
  - Minimise our efforts (the complexity)





Types of Exploits

In principle: Exploiting any vulnerability allows Elevation of Privilege (EOP)

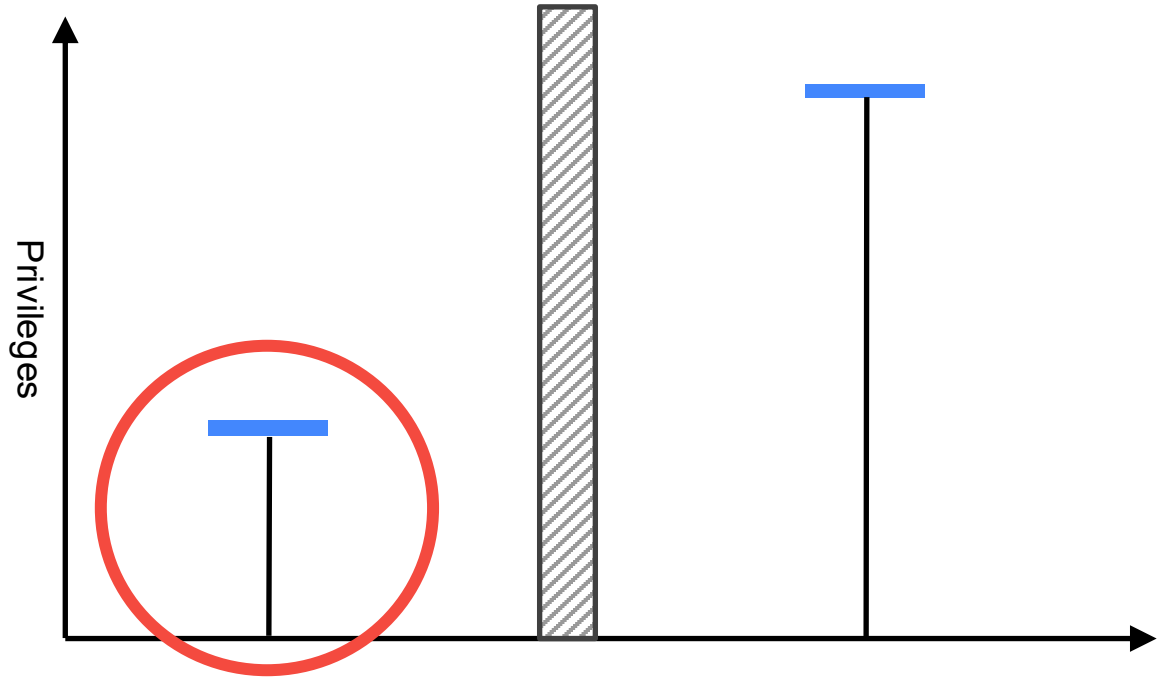
- E.g. Attacker with root execution ➔ TEE OS code execution

More specific terms for common types

- Remote Code Execution (RCE)
- Denial of Service (DOS)
- Information Disclosure (ID)

Threat Model

- When thinking about a specific exploit, we need to consider the threat model
- “How much can the attacker legitimately do already?”



Threat Model

The threat model is a list of assumptions about the attacker

- For attacks: Assume an attacker that is **not more powerful** than X
- For defense: Assume an attacker that is **at least as powerful** as X

- E.g. “Attacker can install & run a malicious app as root”
  - The attacker can get all permissions
  - The attacker cannot run code in the TEE

- Only a way to elevate permission beyond threat model qualifies as exploit
  - Usually requires some background knowledge on target platform!
  - The assumed threat model is essential for describing an exploit!

Attacker Model

A simplified notion for common threat models

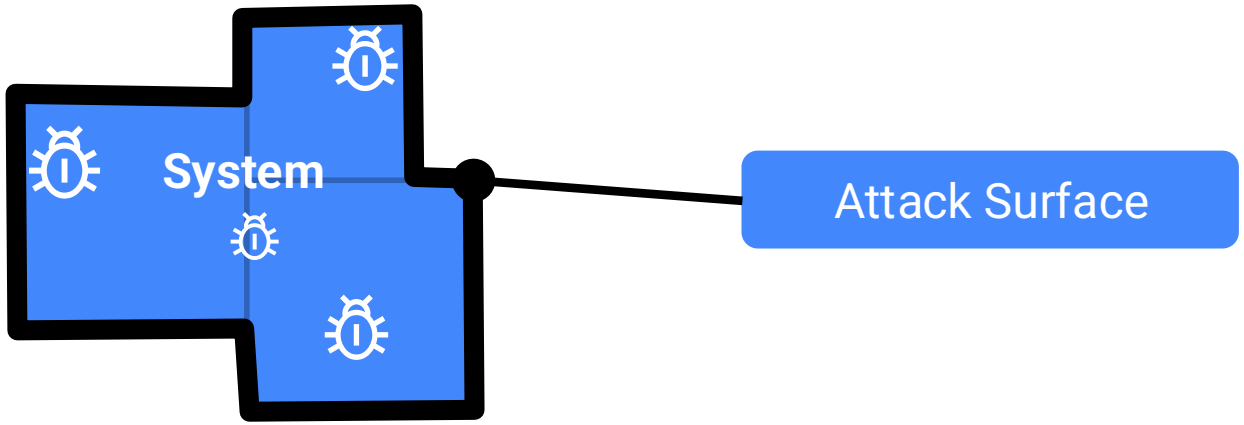
- Remote attacker
  - Attacker can lure victim to visit web page, reading email, receive SMS
  - Attacker can send a message to WhatsApp, Messenger, etc
- Proximity attacker
  - Attacker is physically in the environment of the victim
- Physical attacker
  - Attacker has physical access to the device
- Local attacker
  - Malicious app on the device

Source: source.android.com

Attack Surface

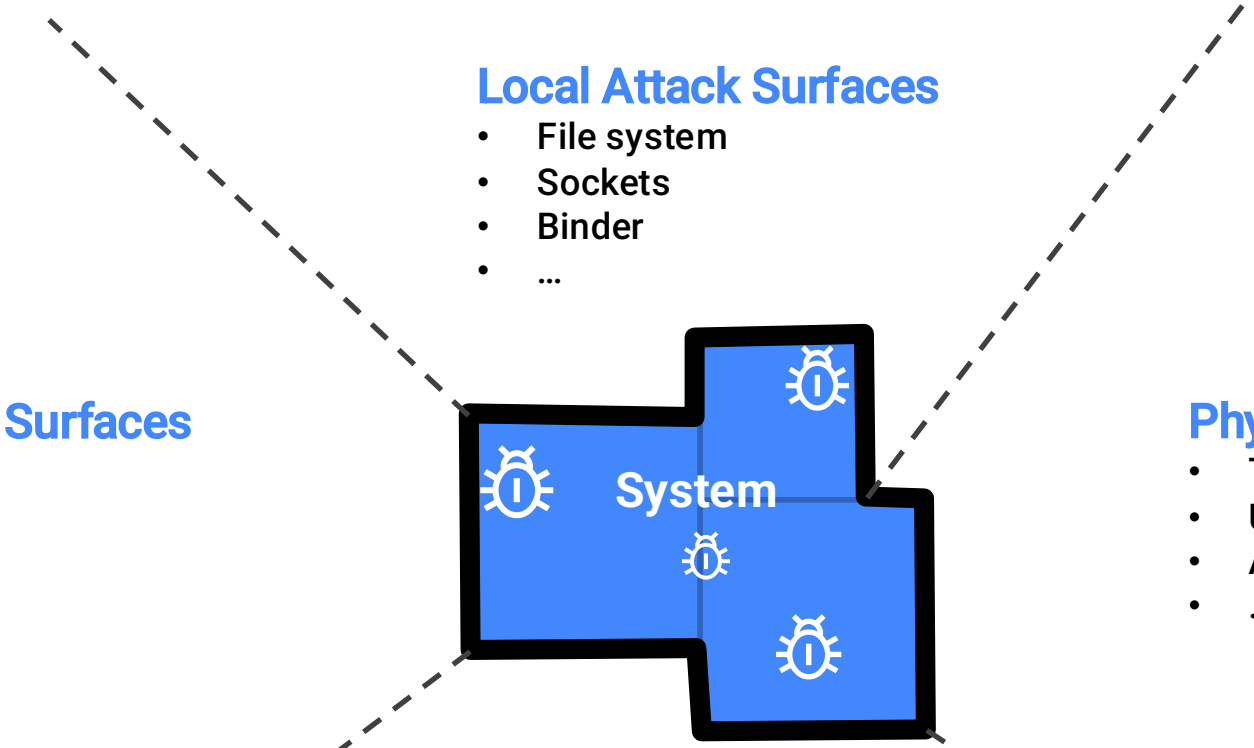
The parts of a system that can be reached by some attacker

- Parts that process data from user, file, network, IPC, …



Attack Surface and Attacker Model

Total attack surface is composed from **different types** of attack surfaces



Local Attack Surfaces
- File system
- Sockets
- Binder
- …

Proximal Attack Surfaces
- Wifi
- Bluetooth
- NFC
- …

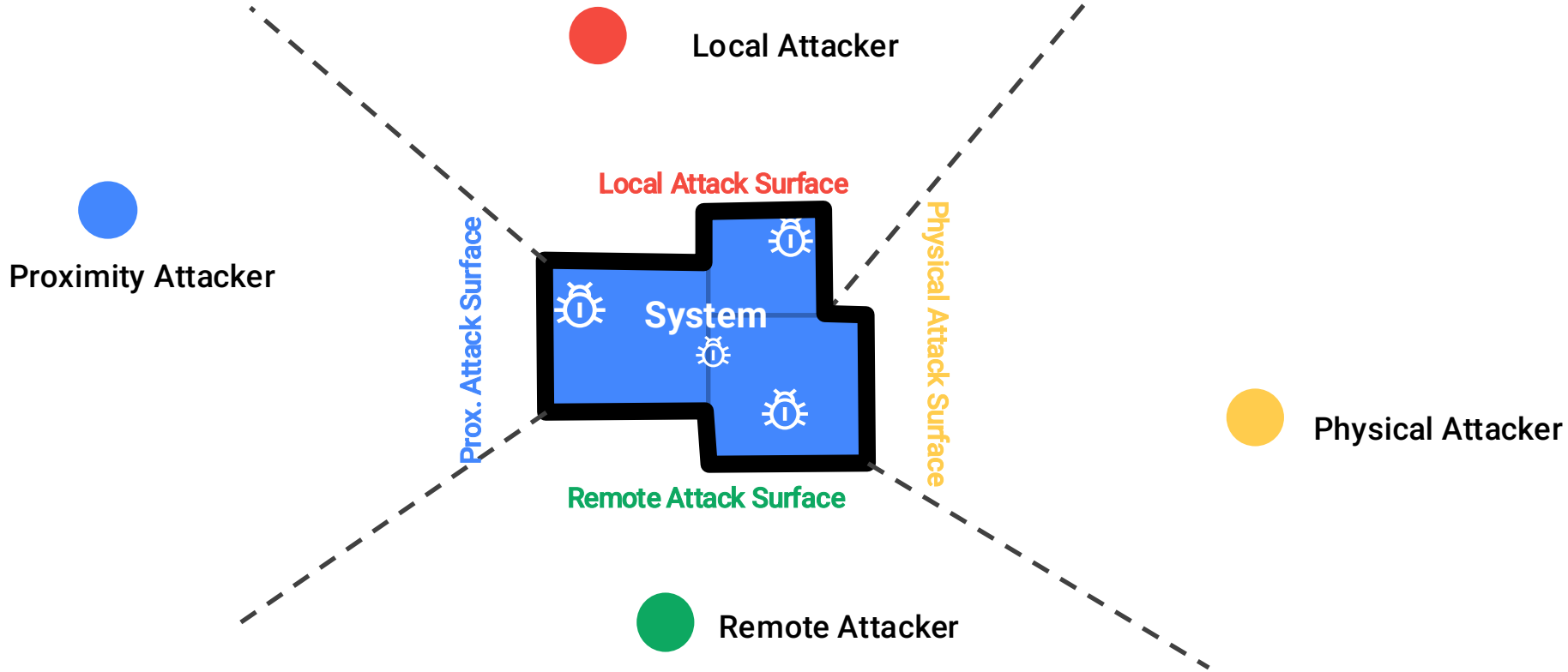
Physical Attack Surfaces
- Take the device apart
- USB
- ADB
- …

Remote Attack Surfaces
- Network stack
- SMS/MMS
- Browser
- …

Bug Picture: Google / Apache 2.0

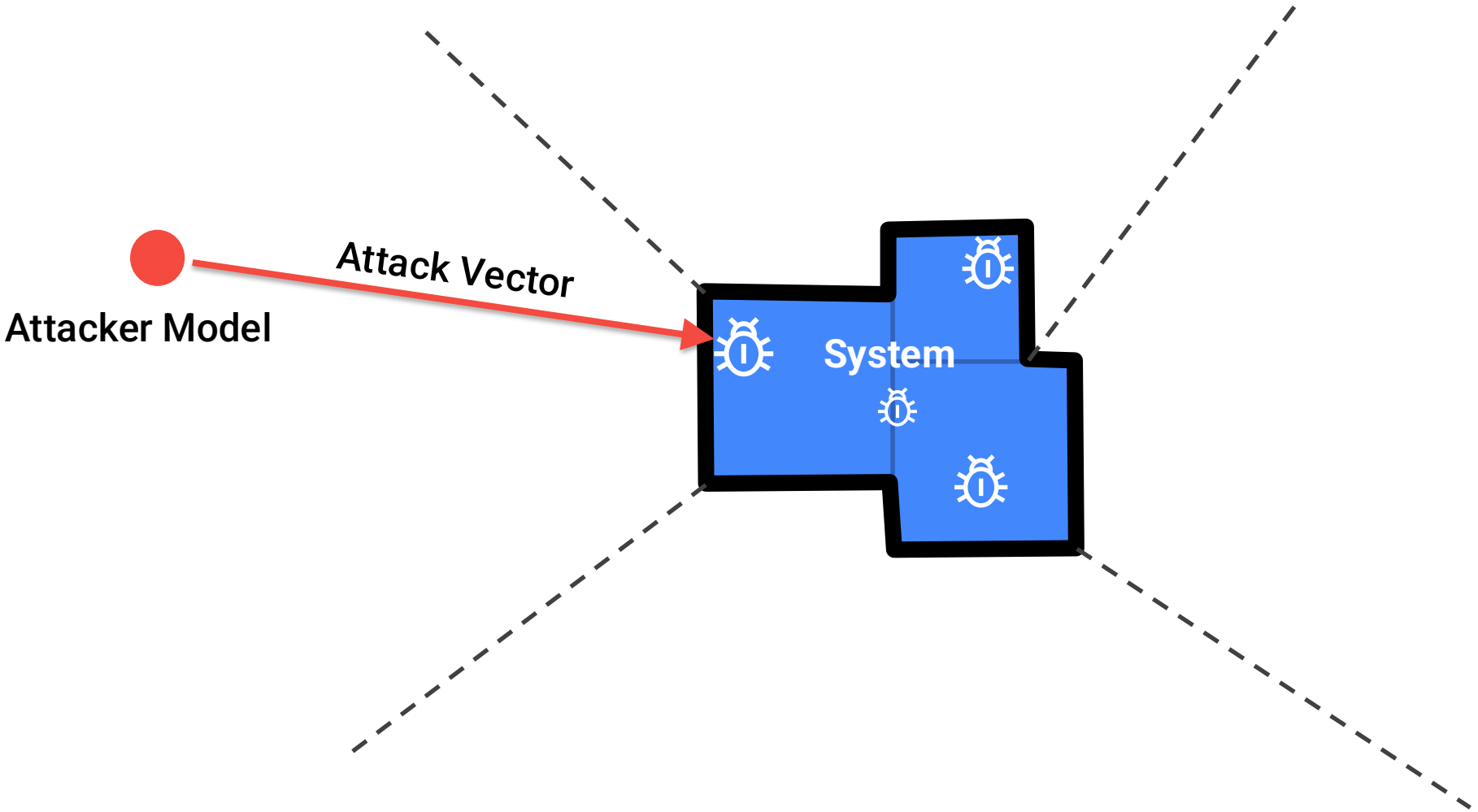
Attack Surfaces and Attacker Model

Different attack surface types reachable with different attacker / threat model



Attack Vector

Specific path an attack takes through the attack surface to exploit a vulnerability



Example Scenario

Cf. Stagefright!

- **Threat Model:**
  Remote Attacker who knows victim's phone number
- **Attack Surface:**
  Media parser library - Part of remote attack surface due to automated MMS parsing
- **Attack Vector:**
  Sending a maliciously crafted image file
- **Vulnerability:**
  Buffer overflow due to lack of input sanity checks
- **Exploit Type:**
  Remote Code Execution (RCE)

Threat Modelling

Offers a structured answer for **“what could possibly go wrong”**

Analyse system to identify

- All assets
- Possible attack surfaces
- Relevant threat models and attack vectors

Picture: Google / Apache 2.0

Multiple uses

- Part of design and implementation phase
- Good starting point for research as well!
- Helps evaluate impact or effectiveness of attacks or defenses

Vulnerability Disclosure

Vulnerability Disclosure

Now that you've discovered a vulnerability, how do your ethically report it?

- Write a report and/or minimal exploit
  - Ensure the vulnerability is clearly described and can be reproduced
- Responsible / Coordinated disclosure:
  - Report the find to the vendor and give them time to triage the issue
  - Only disclose publicly if vulnerability is fixed and/or ~90 days passed
- Many big vendors have at least an information page on reporting bugs
  - Find out whom to contact

More information: OWASP: Vulnerability Disclosure Cheat Sheet

The story of an Android bug (after submission)

1. Triaging
2. Assignment of severity score
3. Working on a fix
4. Fix is committed and tested
5. Patch is released as part of Android Security Bulletin
   - Or quarterly update
6. Bug can be tracked via its CVE number
   - Common vulnerabilities and exposures
   - Assigned by vendor for “serious” vulnerabilities
7. Bug bounty is paid

Android Security Bulletin

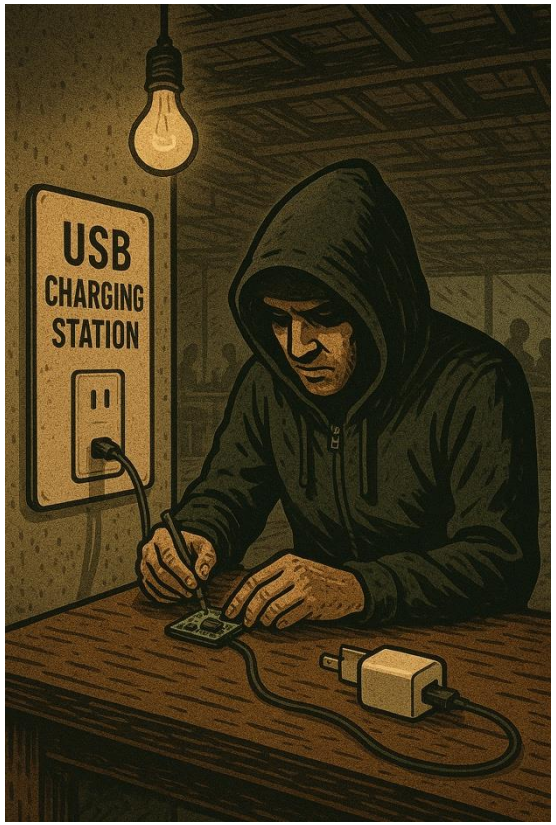
Monthly AOSP updates fix reported security vulnerabilities

- Every update comes with a bulletin describing the fixes
  - Detailed vulnerability list including CVE and references to fix commits
- Device manufacturers integrate fixes to release security patches
  - Android Security Patch Level on device tells about most recent security update
- Device-specific bulletins also available for some manufacturers

Source: source.android.com

My Mobile Security Research Journey

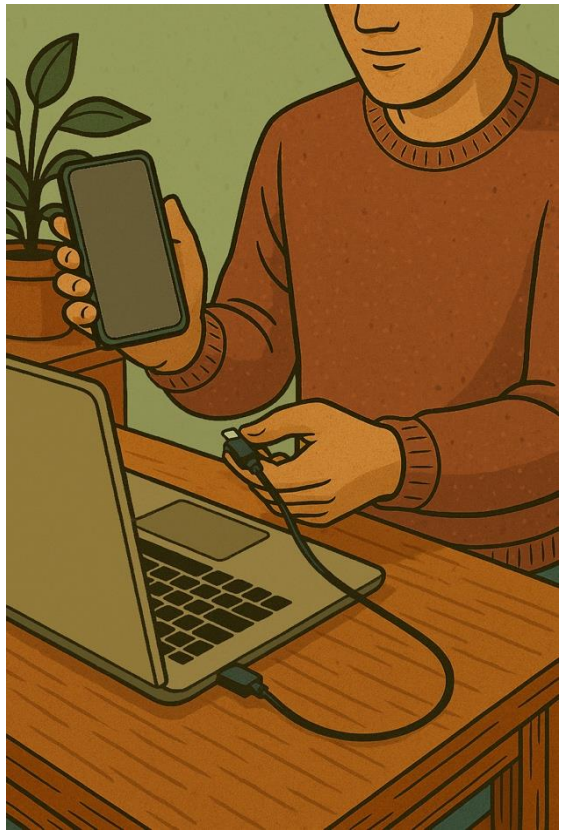
JuiceJacking (~2011)

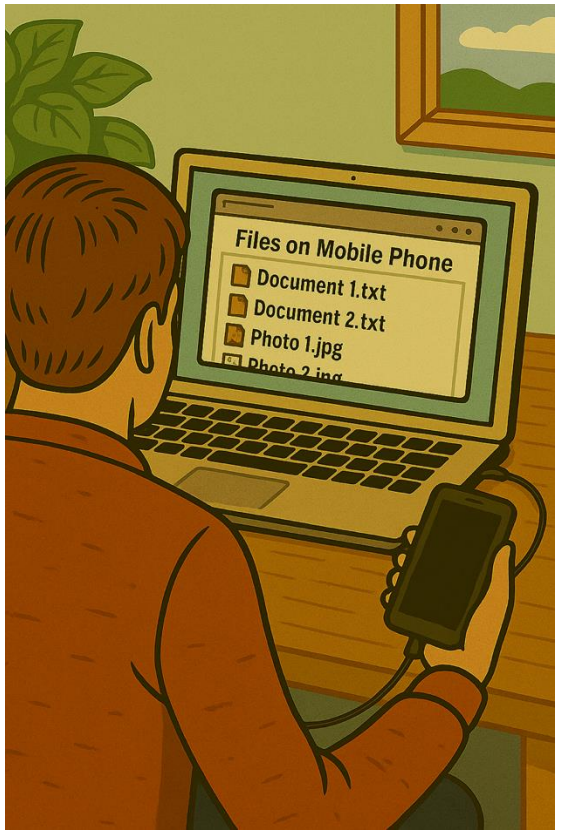


Stealthy data extraction through manipulated phone chargers

Airports, Train Stations, Museums, Hotel Rooms, Rental Cars, Power Banks, …

JuiceJacking Mitigations (2013-)





iOS and Android require user consent for USB file access

USB on Mobile

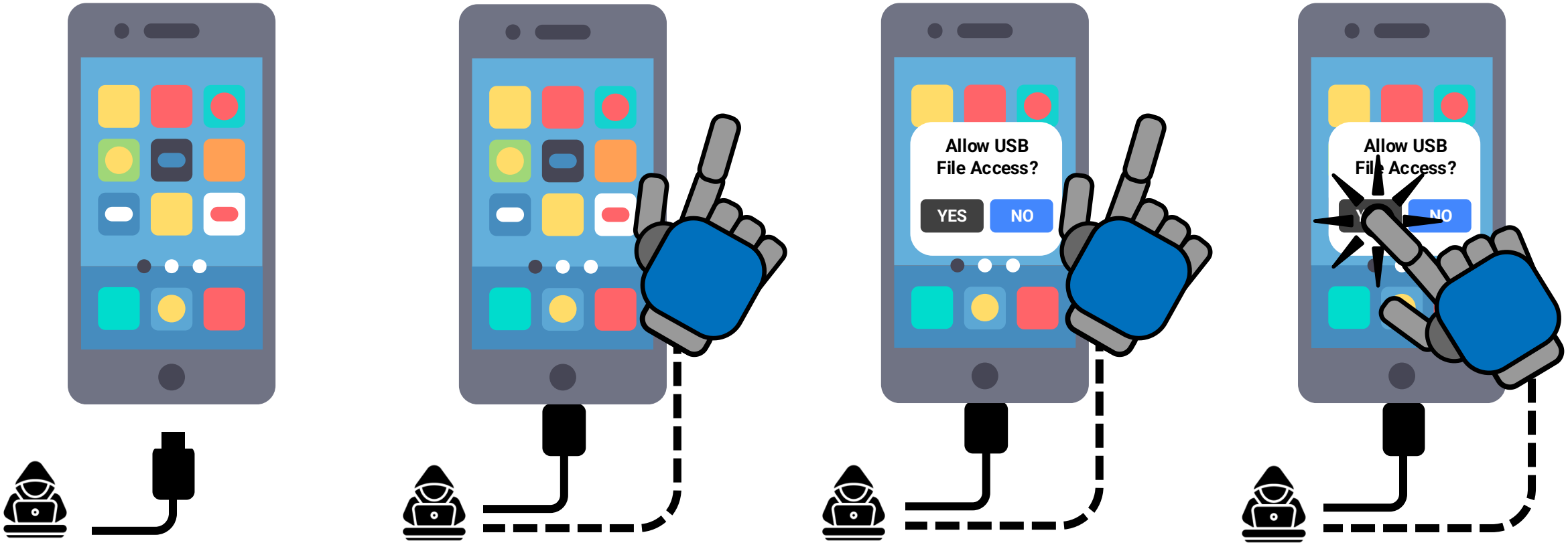
Mobile devices use multi-function USB-C ports

- **Power**
  Charge phone or supply peripherals
- **USB Host**
  Data exchange with peripherals
- **USB Device**
  Data exchange with computer

Important: A USB port either acts as USB host or USB device at a given time

→ Foundation of JuiceJacking mitigation

ChoiceJacking



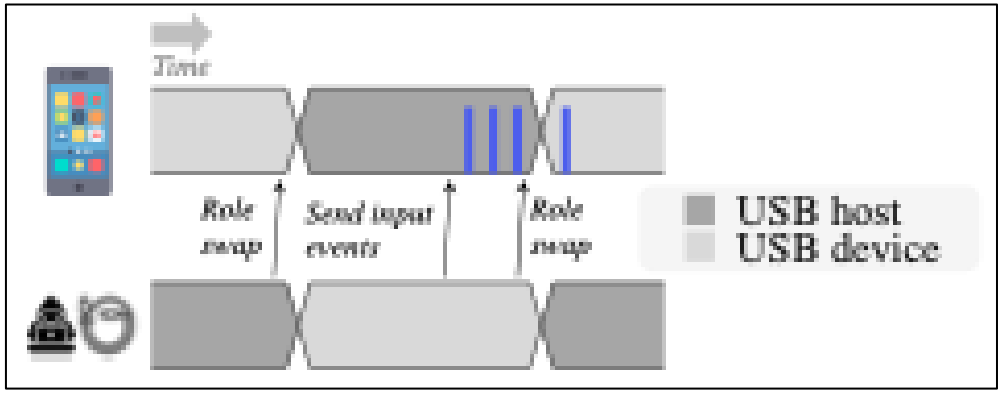
Core Principle: Establish input event channel in addition to file channel

Florian Draschbacher, Lukas Maar, Mathias Oberhuber, and Stefan Mangard. *ChoiceJacking: Compromising Mobile Devices through Malicious Chargers like a Decade Ago.* In: USENIX Security 2025.

ChoiceJacking: 3 Attack Techniques

- **T1:** Flaw in Android Open Accessory Protocol (AOAP)
  - Against specification: Concurrent HID events and file exchange (MTP) possible

- **T2:** Race Condition in Android input dispatcher
  - Process events after role swap



- **T3:** Pivoting via Bluetooth HID
  - iOS and Android!

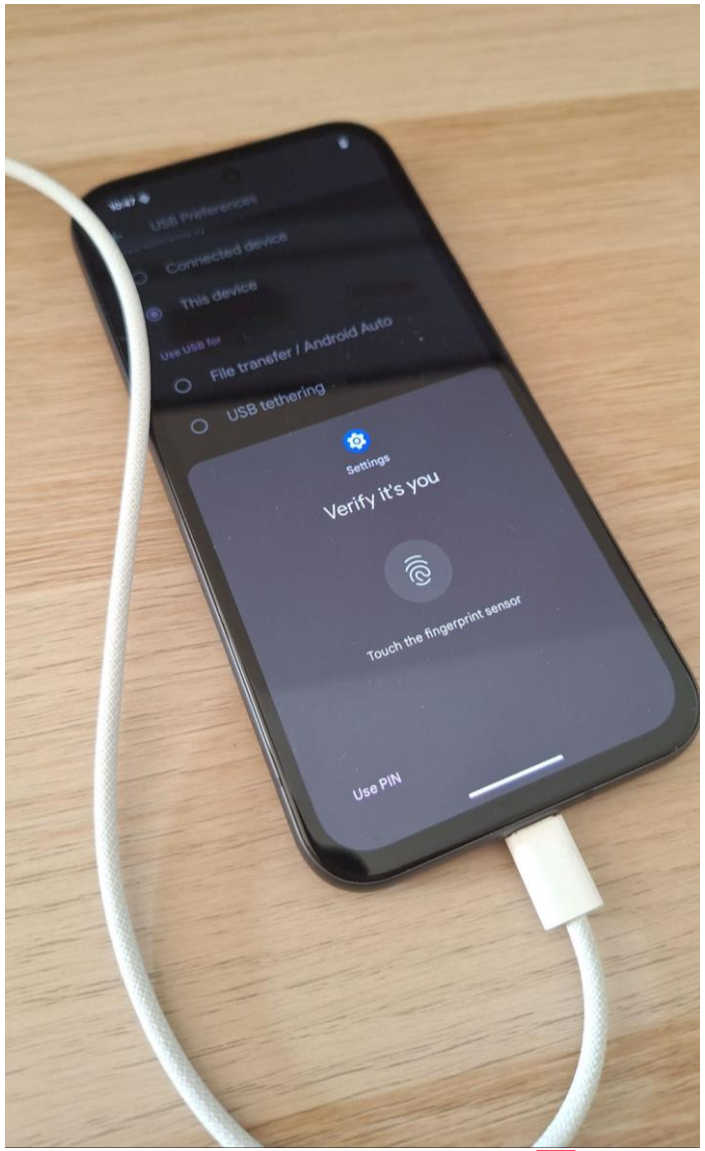
ISEC TU Graz

ChoiceJacking: Evaluation

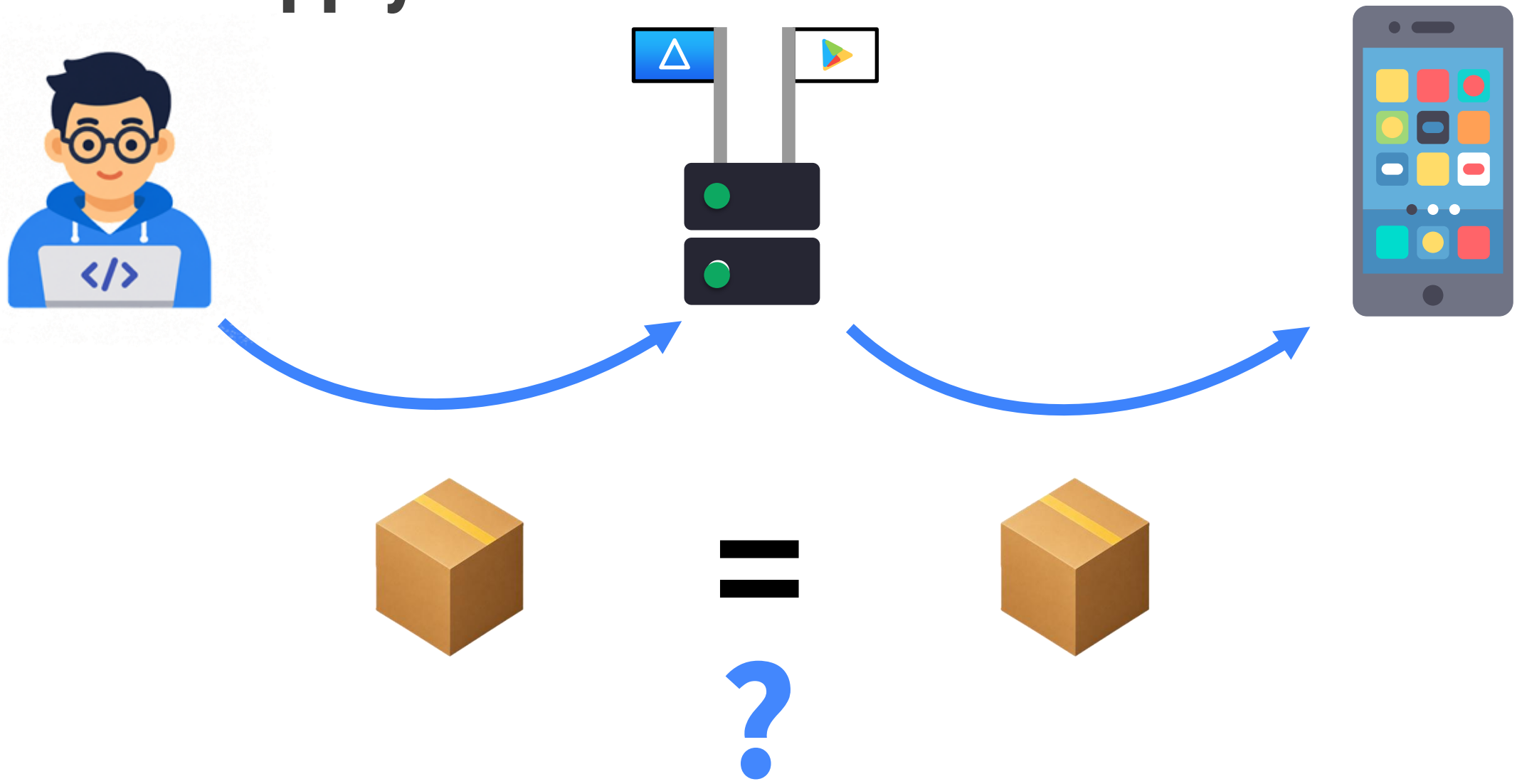
- 11 devices across Android and iOS
  - Samsung, Xiaomi, Vivo, Oppo, Huawei, Honor, Google, Apple
- All susceptible to at least 1 technique
- Some attacks faster than a human blink
  - Samsung: ~130 milliseconds
- Even worked on locked screens for some vendors

ChoiceJacking: Disclosure

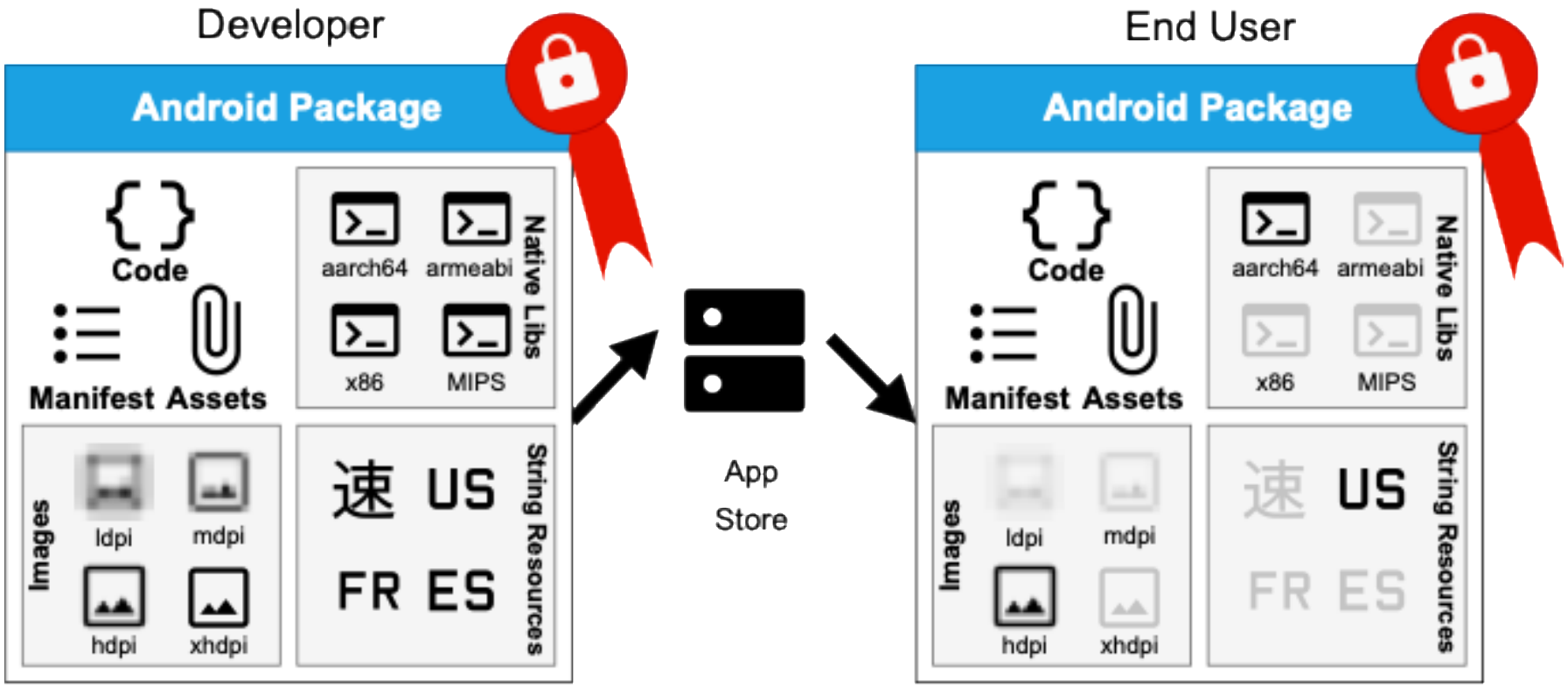
- Responsibly disclosed 21 security issues
  - 16 vendor-specific, 4 upstream Android, 1 iOS issue
- 5 CVEs, 1 high severity
  - Samsung, Apple, Google, …
- Fixes in Android 15 and iOS 18.4
  - Require authentication for USB file access



Mobile Supply Chains

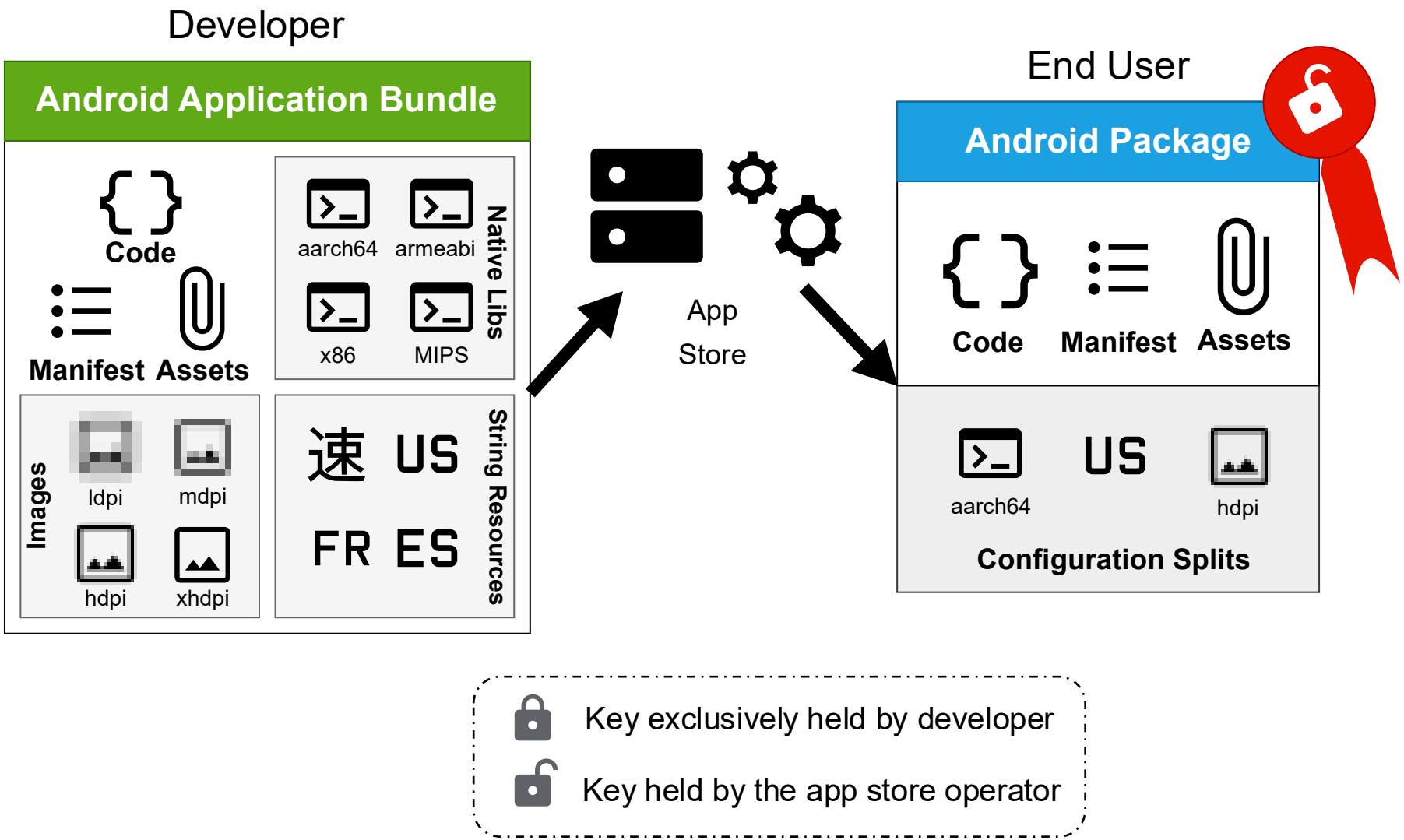


Direct APK Distribution (-2018)

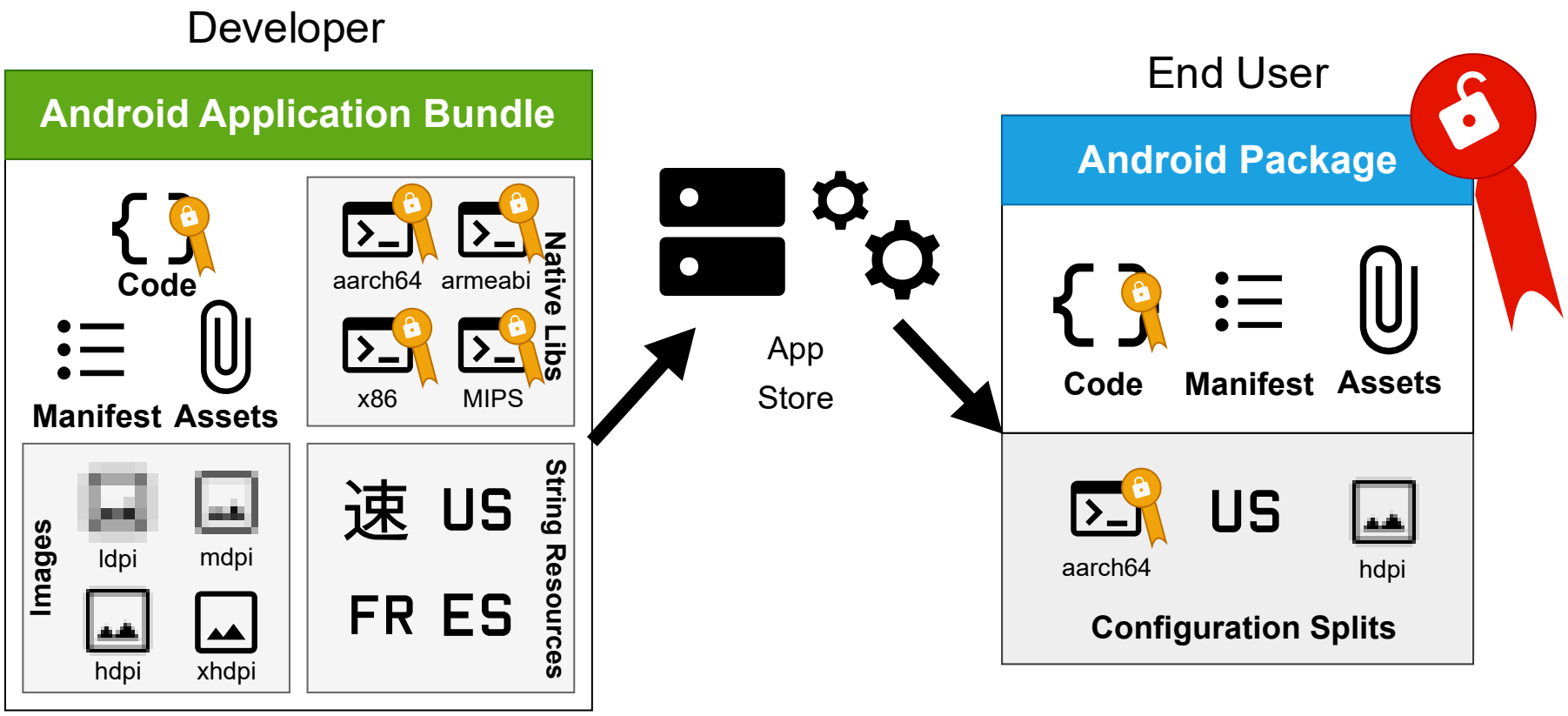


Key exclusively held by developer

Android App Bundle (AAB; 2018-)



Code Transparency for Android App Bundle (2020-)



Key exclusively held by developer

Key held by the app store operator

Manifest Problems

We identified design and implementation flaws exploitable in 5 attacks

Attack	Attacker	Condition
Stripping CT	Any	None
Modifying Assets	Any	Relevant Assets
Inject Shared Library	Privileged / Unprivileged	None
Debuggable Flag	Privileged	None
Backup Opt-In	Privileged	< Android 14

Goal: Code execution in context of target app OR Data extraction

Large-Scale Analyses

- Apps that use AAB and Code Transparency
  - Google Play: 46% (AAB), 0.0014% (CT) of 3.3m apps
  - Huawei AppGallery: 0.04% (AAB), 0% (CT) of 240k apps

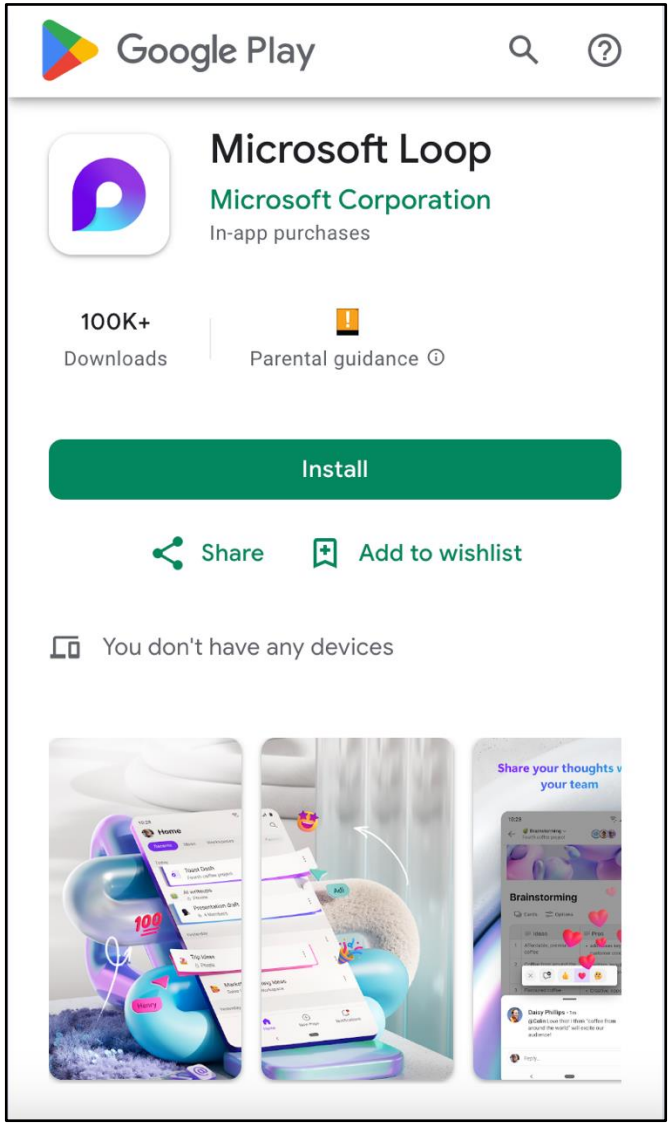
- Executable Assets in Popular Google Play Apps
  - 22% of 6648 apps cannot use CT (DEX/SO in assets)
  - 52% susceptible to code execution through asset manipulation
    - If they used CT

Case Study

- Microsoft Loop (uses Code Transparency)
- Inject shared static library into app manifest:

```
<uses-static-library
    android:name="com.attack.library"
    android:version="1"
    android:certDigest="f7…9d">
</uses-static-library>
```

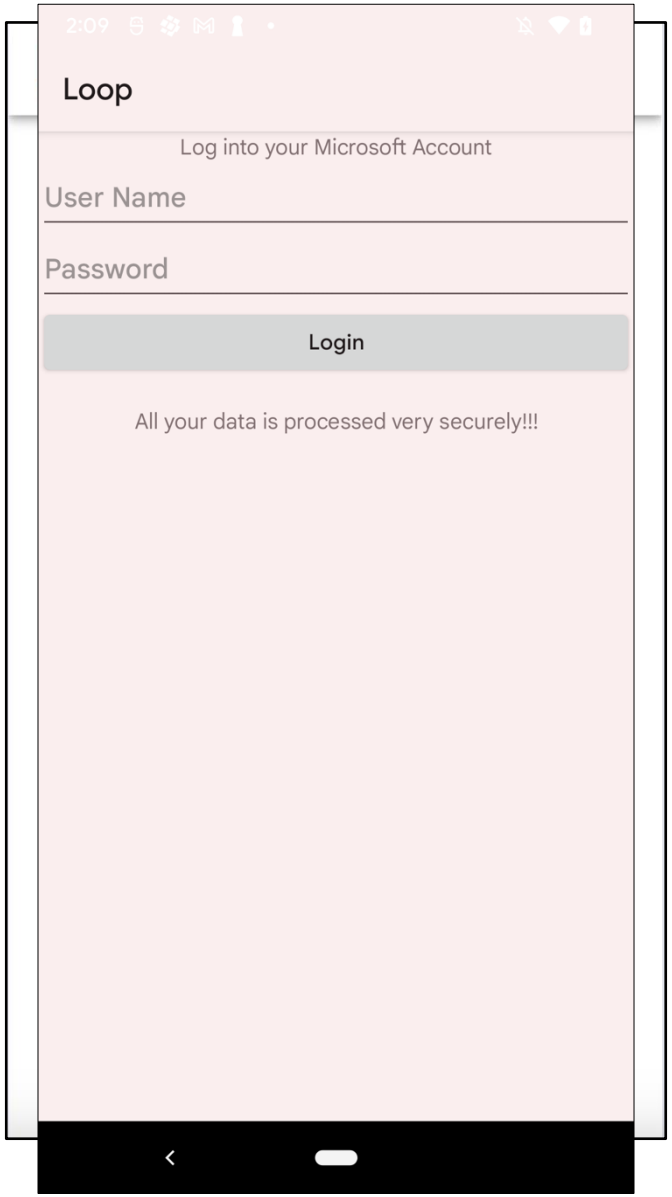
- Bundletool still reports intact CT
- Static library executes in context of MS Loop



Case Study

- Microsoft Loop (uses Code Transparency)
- Inject shared static library into app manifest:

```
<uses-static-library
    android:name="com.attack.library"
    android:version="1"
    android:certDigest="f7…9d">
</uses-static-library>
```



- Bundletool still reports intact CT
- Static library executes in context of MS Loop

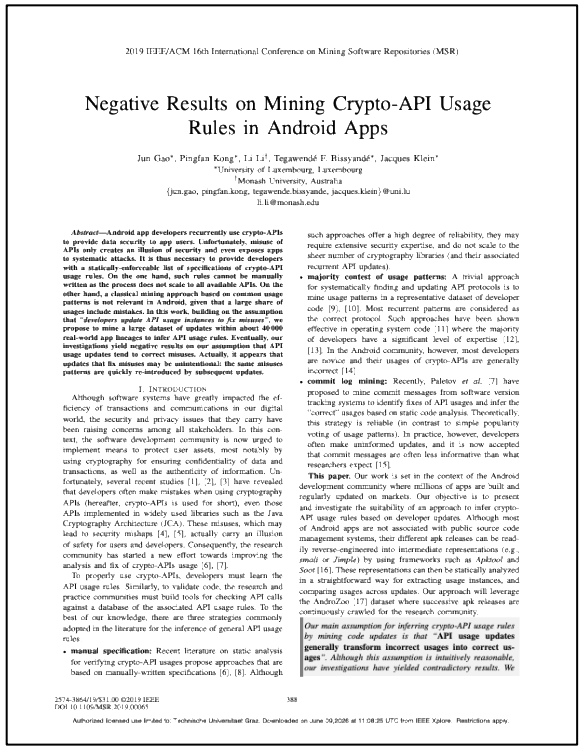
API Misuse

“Developers keep re-introducing (crypto) API misuse into their codebase”

- [Gao+2019]

- Developers unable or unwilling to patch issues
- High prevalence of API misuse in apps
- Still no systemic solution for many cases
  - 3rd party mitigations are limited & purpose-specific

→ Automated On-Device Mitigation for Crypto API Misuse



2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

Negative Results on Mining Crypto-API Usage Rules in Android Apps

Jun Gao*, Pingfan Kong*, Li Li†, Tegawendé F. Bissyandé*, Jacques Klein*
*University of Luxembourg, Luxembourg
†Monash University, Australia
{jun.gao, pingfan.kong, tegawende.bissyande, jacques.klein}@uni.lu
li.li@monash.edu

***Abstract*—Android app developers recurrently use crypto-APIs to provide data security to app users. Unfortunately, misuse of APIs only creates an illusion of security and even exposes apps to systematic attacks. It is thus necessary to provide developers with a statically-enforceable list of specifications of crypto-API usage rules. On the one hand, such rules cannot be manually written as the process does not scale to all available APIs. On the other hand, a classical mining approach based on common usage patterns is not relevant in Android, given that a large share of usages include mistakes. In this work, building on the assumption that *"developers update API usage instances to fix misuses"*, we propose to mine a large dataset of updates within about 40 000 real-world app lineages to infer API usage rules. Eventually, our investigations yield negative results on our assumption that API usage updates tend to correct misuses. Actually, it appears that updates that fix misuses may be unintentional: the same misuses patterns are quickly re-introduced by subsequent updates.**

I. Introduction

Although software systems have greatly impacted the efficiency of transactions and communications in our digital world, the security and privacy issues that they carry have been raising concerns among all stakeholders. In this context, the software development community is now urged to implement means to protect user assets, most notably by using cryptography for ensuring confidentiality of data and transactions, as well as the authenticity of information. Unfortunately, several recent studies [1], [2], [3] have revealed that developers often make mistakes when using cryptography APIs (hereafter, crypto-APIs is used for short), even those APIs implemented in widely used libraries such as the Java Cryptography Architecture (JCA). These misuses, which may lead to security mishaps [4], [5], actually carry an illusion of safety for users and developers. Consequently, the research community has started a new effort towards improving the analysis and fix of crypto-APIs usage [6], [7].

To properly use crypto-APIs, developers must learn the API usage rules. Similarly, to validate code, the research and practice communities must build tools for checking API calls against a database of the associated API usage rules. To the best of our knowledge, there are three strategies commonly adopted in the literature for the inference of general API usage rules

- **manual specification:** Recent literature on static analysis for verifying crypto-API usages propose approaches that are based on manually-written specifications [6], [8]. Although such approaches offer a high degree of reliability, they may require extensive security expertise, and do not scale to the sheer number of cryptography libraries (and their associated recurrent API updates).
- **majority contest of usage patterns:** A trivial approach for systematically finding and updating API protocols is to mine usage patterns in a representative dataset of developer code [9], [10]. Most recurrent patterns are considered as the correct protocol. Such approaches have been shown effective in operating system code [11] where the majority of developers have a significant level of expertise [12], [13]. In the Android community, however, most developers are novice and their usages of crypto-APIs are generally incorrect [14].
- **commit log mining:** Recently, Paletov *et al.* [7] have proposed to mine commit messages from software version tracking systems to identify fixes of API usages and infer the "correct" usages based on static code analysis. Theoretically, this strategy is reliable (in contrast to simple popularity voting of usage patterns). In practice, however, developers often make uninformed updates, and it is now accepted that commit messages are often less informative than what researchers expect [15].

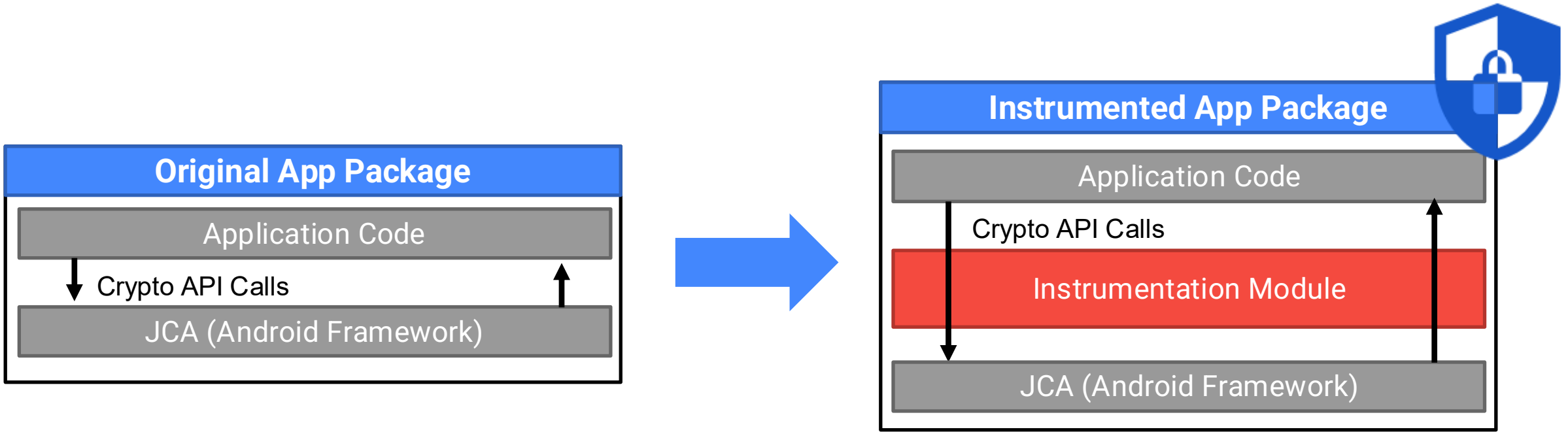
This paper. Our work is set in the context of the Android development community where millions of apps are built and regularly updated on markets. Our objective is to present and investigate the suitability of an approach to infer crypto-API usage rules based on developer updates. Although most of Android apps are not associated with public source code management systems, their different apk releases can be readily reverse-engineered into intermediate representations (e.g., *smali* or *Jimple*) by using frameworks such as *Apktool* and *Soot* [16]. These representations can then be statically analyzed in a straightforward way for extracting usage instances, and comparing usages across updates. Our approach will leverage the AndroZoo [17] dataset where successive apk releases are continuously crawled for the research community.

> *Our main assumption for inferring crypto-API usage rules by mining code updates is that* **"API usage updates generally transform incorrect usages into correct usages"**. *Although this assumption is intuitively reasonable, our investigations have yielded contradictory results. We*

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00065

CryptoShield

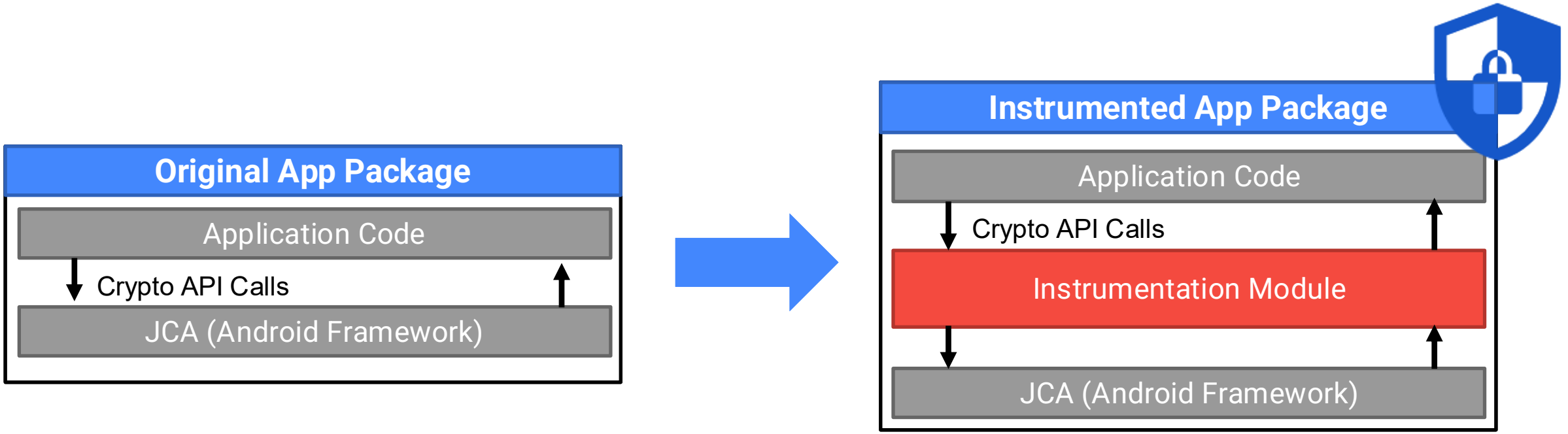
1. Define mitigation for 10 most critical crypto misuse cases
2. On-device agent that injects instrumentation module



Florian Draschbacher and Johannes Feichtner. *CryptoShield: Automatic On-Device Mitigation for Crypto API Misuse in Android Applications.* In: AsiaCCS 2023.

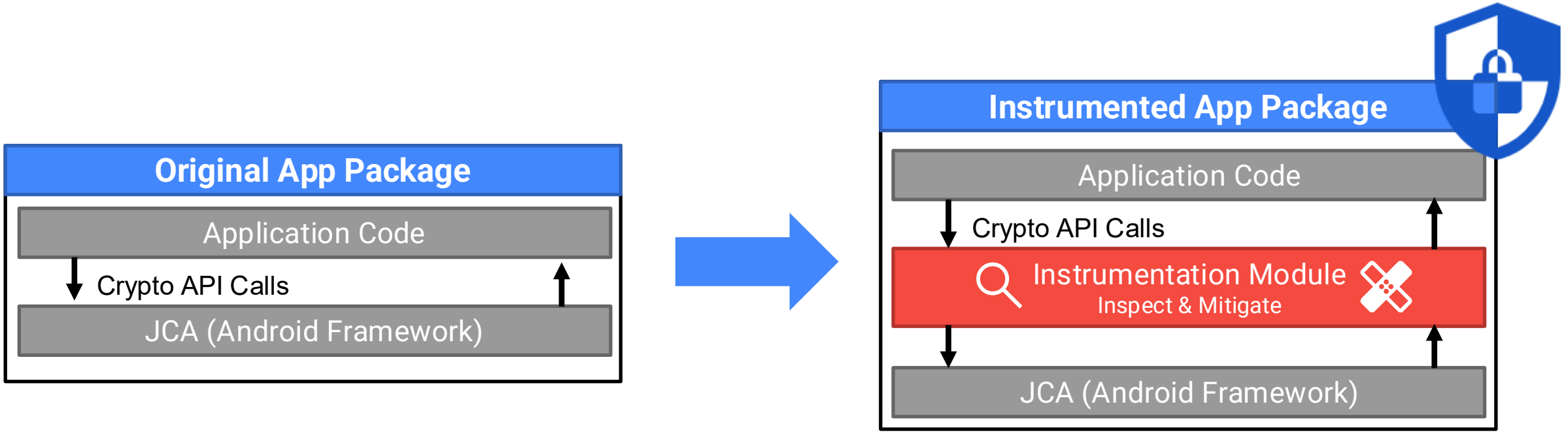
CryptoShield

1. Define mitigation for 10 most critical crypto misuse cases
2. On-device agent that injects instrumentation module



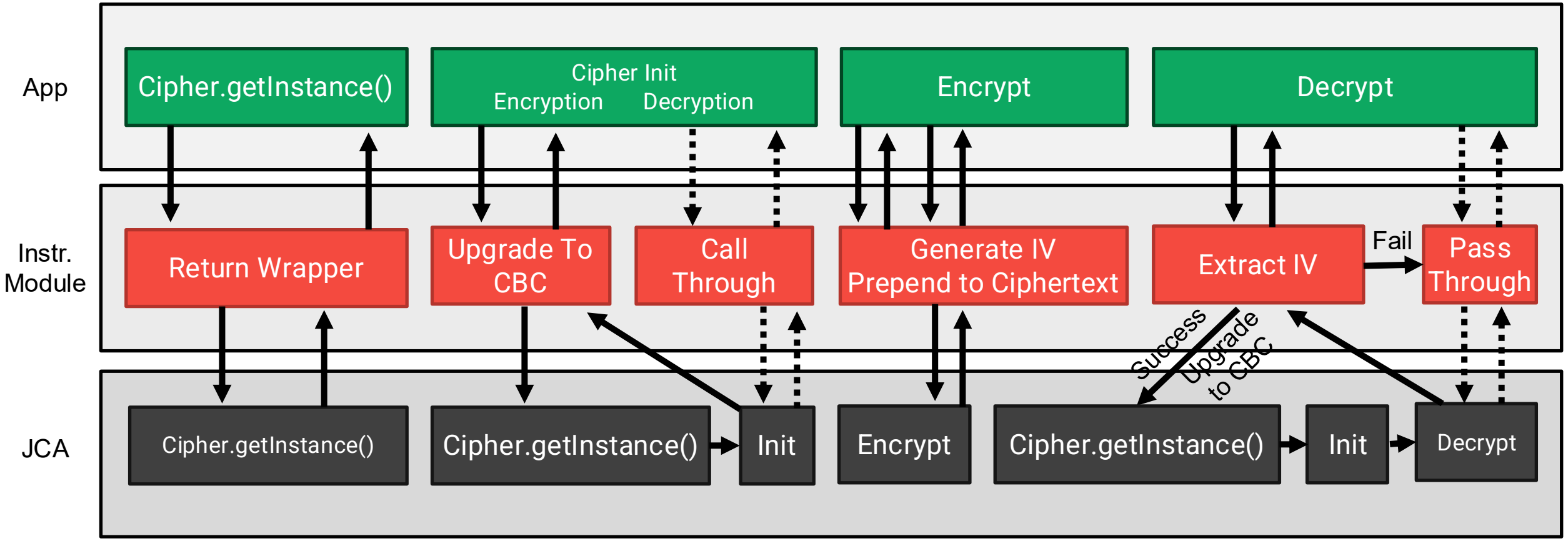
CryptoShield

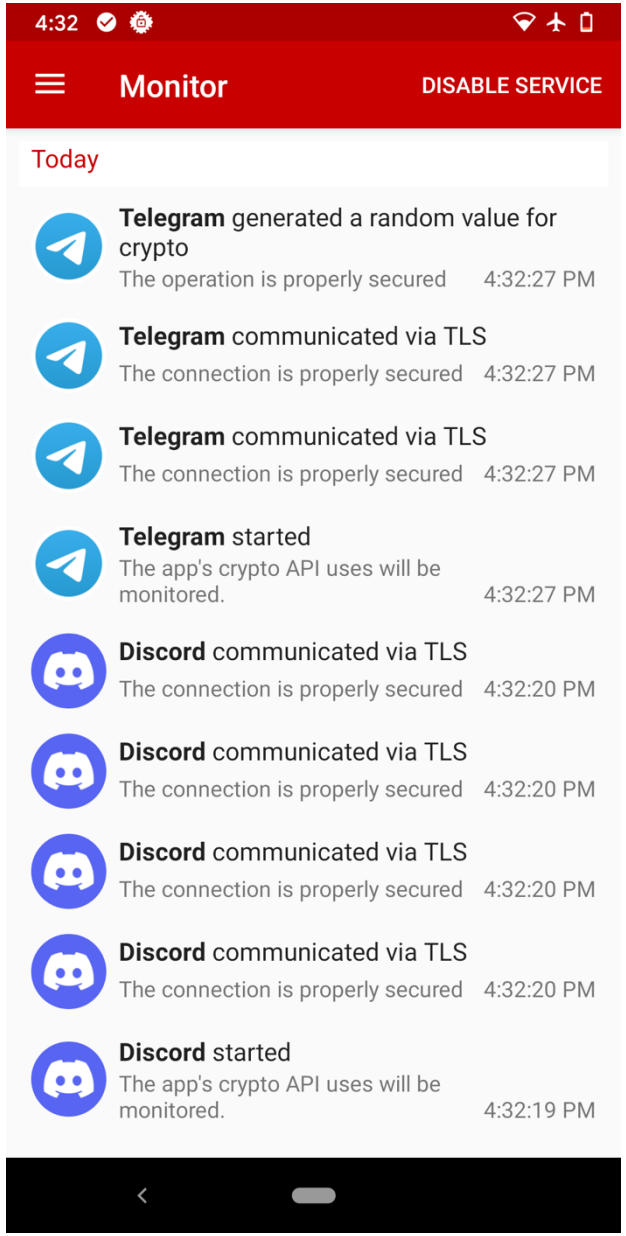
1. Define mitigation for 10 most critical crypto misuse cases
2. On-device agent that injects instrumentation module

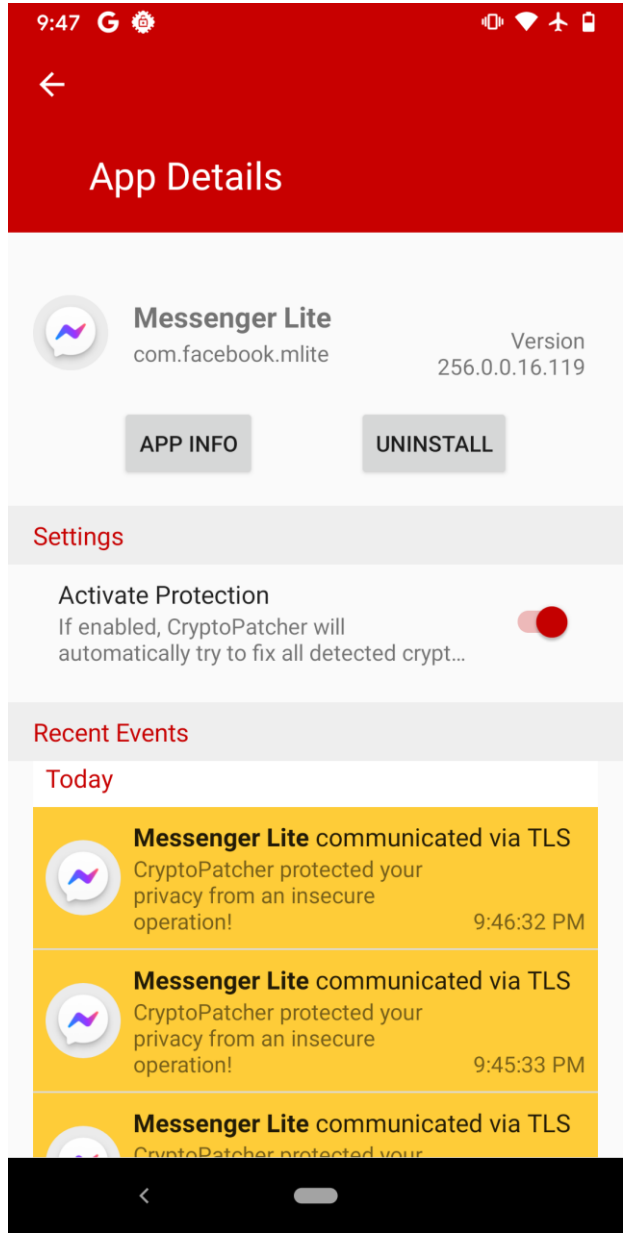


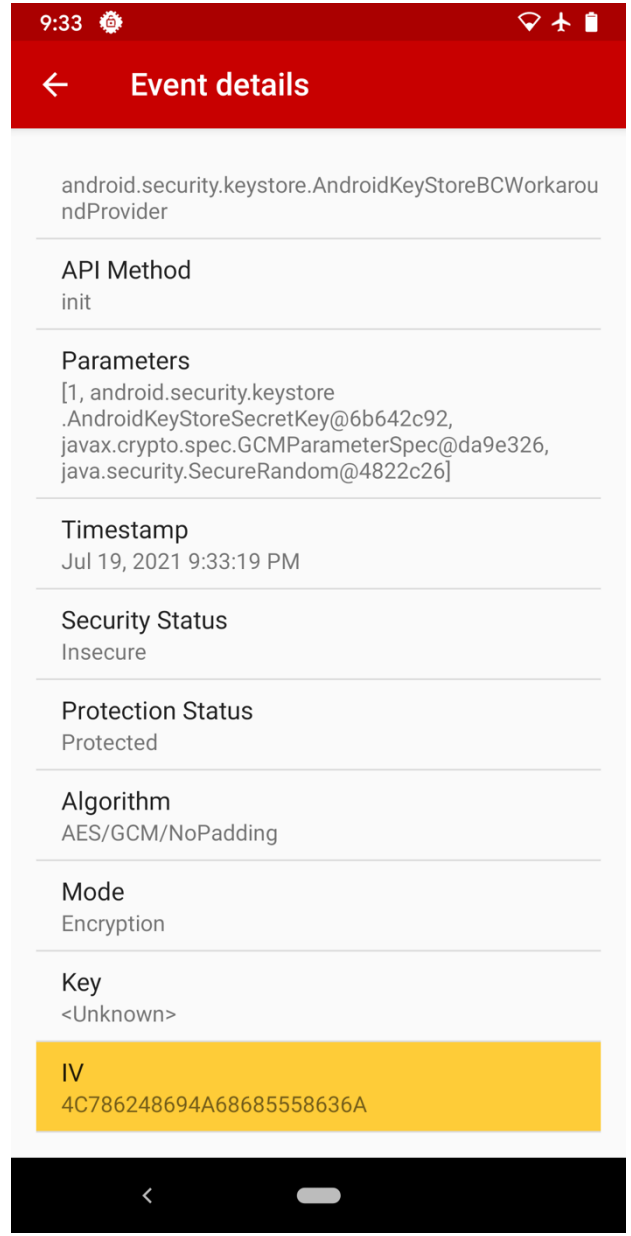
CryptoShield: Mitigation Procedures

Misuse Case: Implicit ECB Mode for AES




4:32
Monitor
DISABLE SERVICE
Today
Telegram generated a random value for crypto
The operation is properly secured 4:32:27 PM
Telegram communicated via TLS
The connection is properly secured 4:32:27 PM
Telegram communicated via TLS
The connection is properly secured 4:32:27 PM
Telegram started
The app's crypto API uses will be monitored. 4:32:27 PM
Discord communicated via TLS
The connection is properly secured 4:32:20 PM
Discord communicated via TLS
The connection is properly secured 4:32:20 PM
Discord communicated via TLS
The connection is properly secured 4:32:20 PM
Discord communicated via TLS
The connection is properly secured 4:32:20 PM
Discord started
The app's crypto API uses will be monitored. 4:32:19 PM


9:47
App Details
Messenger Lite
com.facebook.mlite
Version 256.0.0.16.119
APP INFO
UNINSTALL
Settings
Activate Protection
If enabled, CryptoPatcher will automatically try to fix all detected crypt...
Recent Events
Today
Messenger Lite communicated via TLS
CryptoPatcher protected your privacy from an insecure operation! 9:46:32 PM
Messenger Lite communicated via TLS
CryptoPatcher protected your privacy from an insecure operation! 9:45:33 PM
Messenger Lite communicated via TLS


9:33
Event details
android.security.keystore.AndroidKeyStoreBCWorkaroundProvider
API Method
init
Parameters
[1, android.security.keystore.AndroidKeyStoreSecretKey@6b642c92, javax.crypto.spec.GCMParameterSpec@da9e326, java.security.SecureRandom@4822c26]
Timestamp
Jul 19, 2021 9:33:19 PM
Security Status
Insecure
Protection Status
Protected
Algorithm
AES/GCM/NoPadding
Mode
Encryption
Key
<Unknown>
IV
4C786248694A68685558636A

ISEC TU Graz

CryptoShield: Evaluation

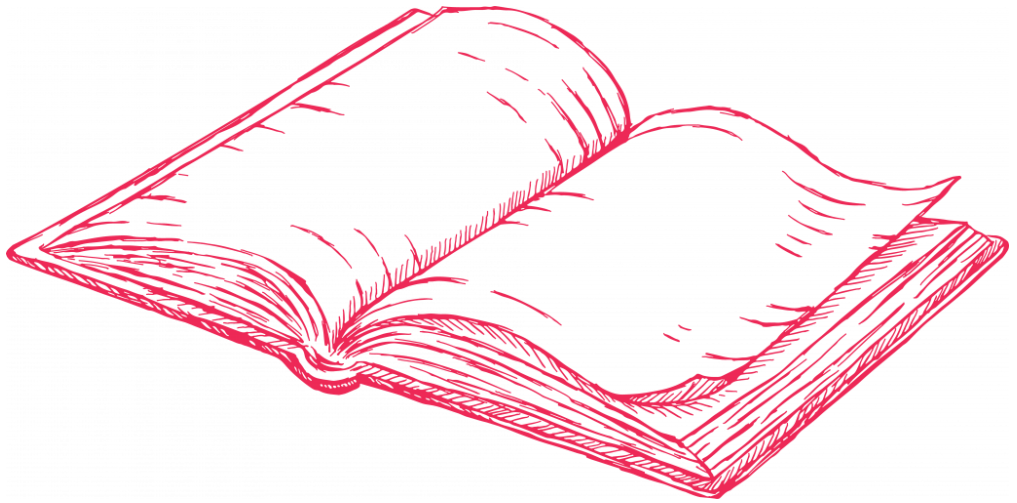
- Automated analysis
  - Crypto API misuse mitigated in 96 % of vulnerable apps
- Performance
  - Minimal size (5 %) and runtime (100 ms worst-case) overheads
  - Functionality retained for 92 % of 99 popular apps
- Synthetic detection benchmark
  - No FN, better FP than CryptoGuard
- Case studies of 2 critically vulnerable apps

Conclusion

Mobile Security

In this course, you learned about

- Key and Data Storage on **Mobile** devices
- iOS Platform **Security**
- iOS Application **Security**
- Android Platform **Security**
- Android Application **Security**
- **Mobile** Hardware Security
- **Mobile** Security Research

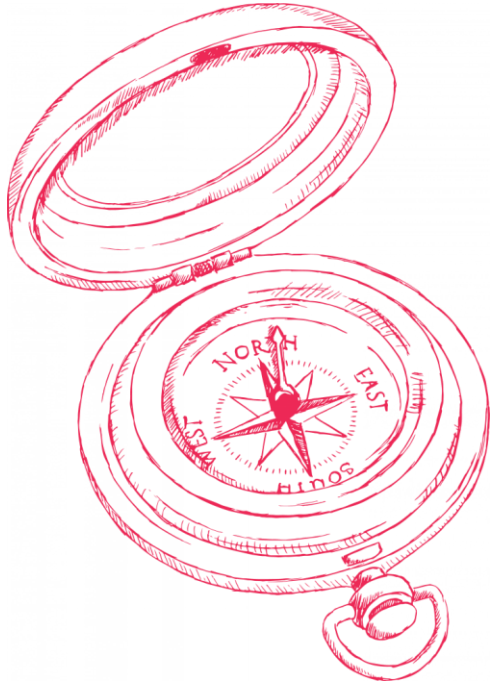


All slides are available on the course website

Further Resources 1

- Documentation
  - Android developer documentation
  - Android platform documentation and source code
  - Apple Platform Security

- Books
  - Jonathan Levin: MacOS and iOS Internals
  - Jonathan Levin: Android Internals
  - Aditya Gupta: The IoT Hacker's Handbook



Further Resources 2

- Online Courses
  - Mobile Systems and Smartphone Security
    - Includes 21 CTF-style challenges!
- Scientific Publications
  - dblp.org computer science bibliography
  - Google Scholar
  - Conference Proceedings
    - Usenix Security, NDSS, ACM CCS, IEEE S&P
- Vulnerability Writeups
  - Google Project Zero
  - NowSecure

What's next?

- Congrats, you're a **mobile security researcher** now ☺
- How can you find interesting new challenges in the domain?
  - Contribute in real-world security research
  - Deepen your knowledge
- Join ISEC!
  - Master theses
  - Research and teaching

Send me an email: florian.draschbacher@tugraz.at

GRAZ SECURITY WEEK 2026

Summer School for Cybersecurity at TU Graz

07.–11. SEPTEMBER

APPLY FOR A
STUDENT STIPEND!

Fill out the application survey and upload a CV + a short motivation letter

DEADLINE: June 28th

WWW.SECURITYWEEK.AT

Outlook

- 26.06.2026, 10:00-12:00, HS i12
  - Exam option 1
- 10.07.2026, 10:00-12:00, CCGEG002
  - Exam option 2

Course evaluation open now!

Exam registration open now