IAIK TU Graz

Mobile Security Research

Mobile Security 2022

Florian Draschbacher
florian.draschbacher@iaik.tugraz.at

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

Practicals

- **You should have started already!**
- Deadline 8th of June
- Questions?
  - Ask now
  - Send me an email

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

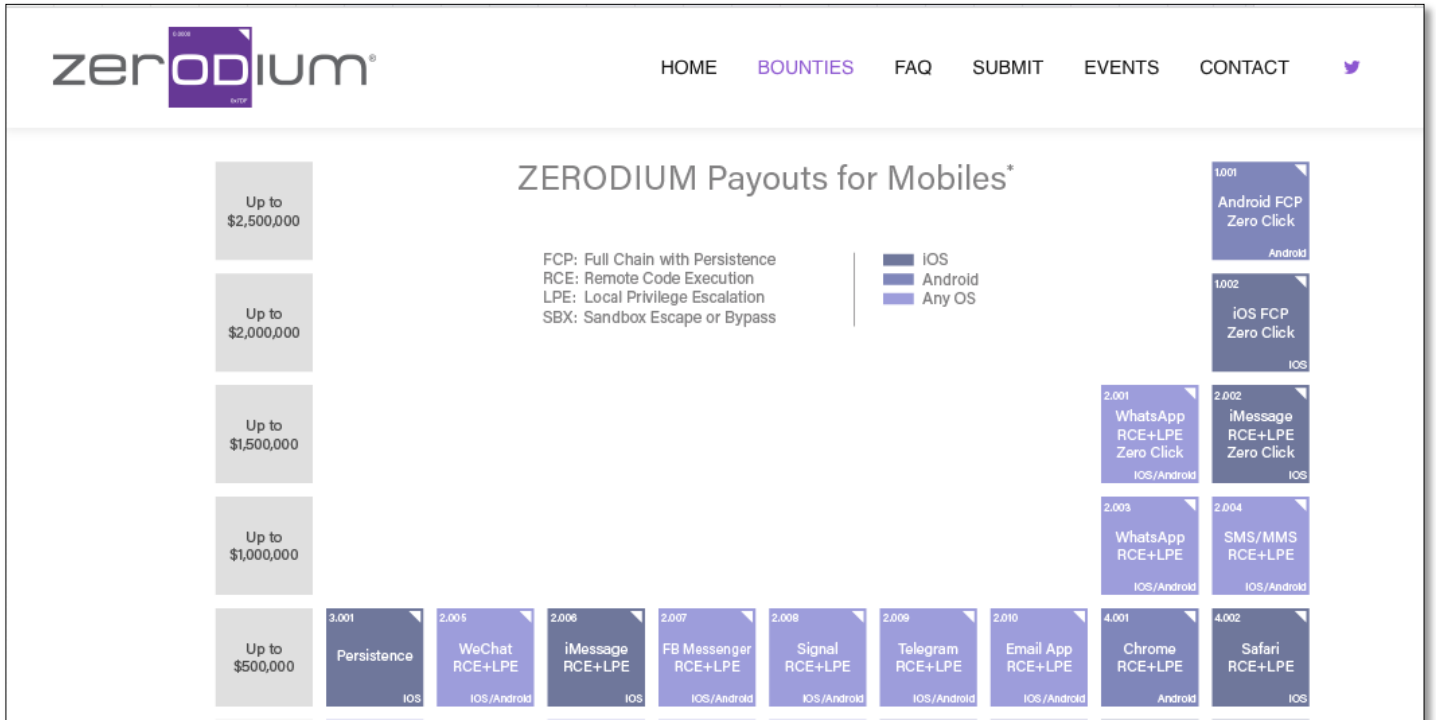
- **Why?**
  - Improve the security of mobile ecosystems
  - Develop new attack methods and corresponding defenses
  - Find and eliminate malware
  - Earn some bug bounties or academic title along the way!

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

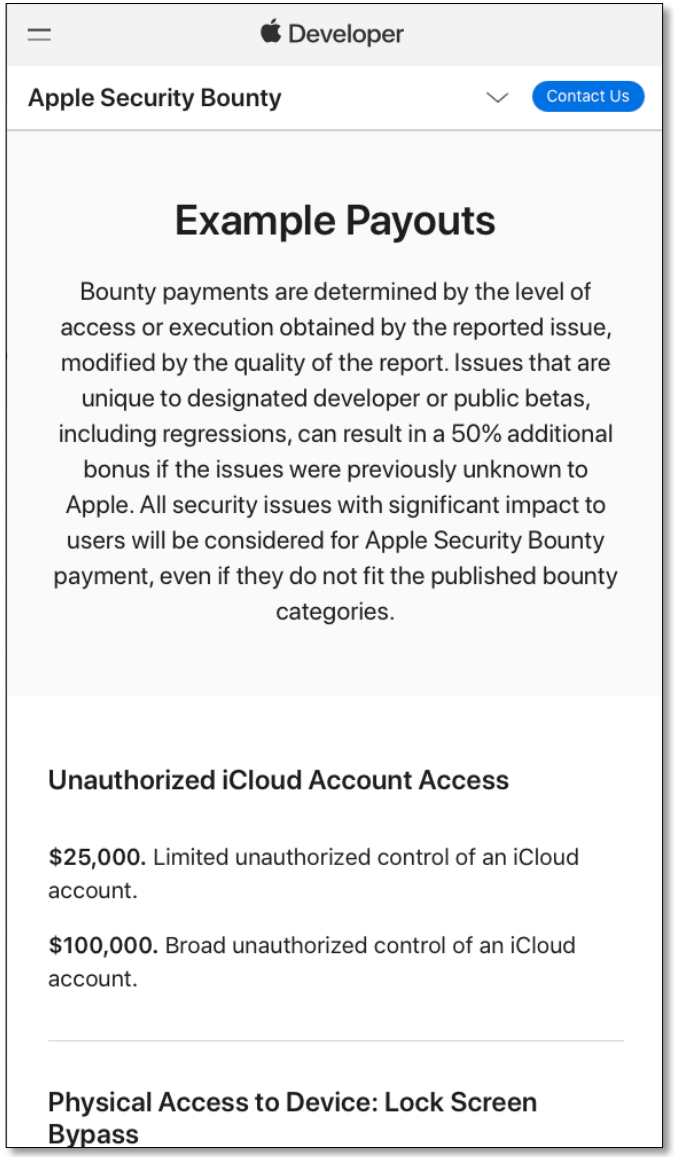


Source: zerodium.com

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

IAIK TU Graz

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.

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): 6 months of imprisonment
  - Using hacking tools (§126c): 6 months of imprisonment
  - Malicious intent is crucial!
    - Stealing sensitive data, preventing operation, …

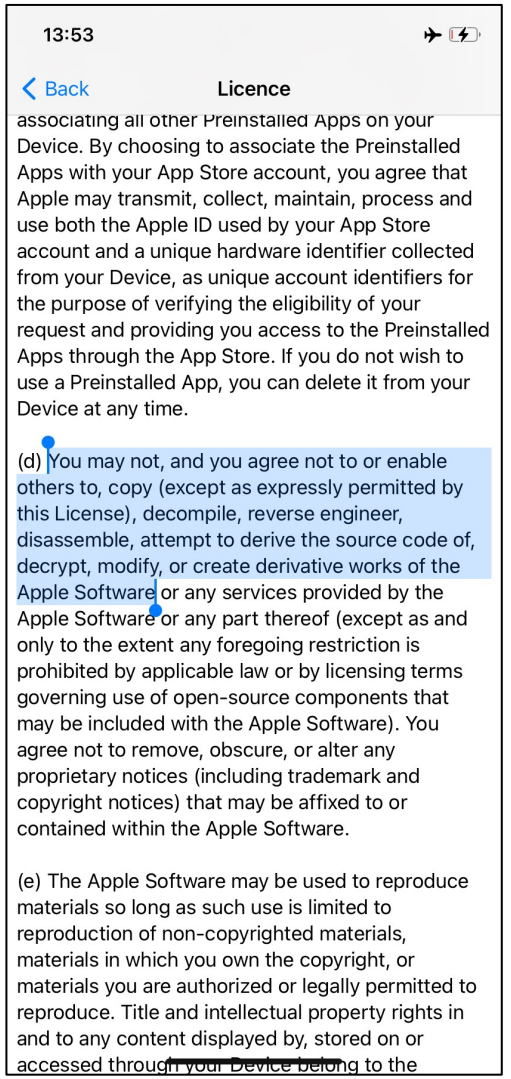
- Urheberrecht / Copyright law
  - Assuming you are entitled to the use of some software:
  - Decompilation only allowed for ensuring interoperability (§40e)
    - Exception for matters affecting public safety (§41)
    - Decision by EU Court of Justice: Decompilation also for fixing bugs (Source: lexology.com)
  - 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
    - In those cases: Considered part of the purchase contract

    Source: linux-magazin.de
- Gesetz gegen unlauteren Wettbewerb (~Trade Secret Law)
  - Reverse-engineering for extracting trade secrets is legal (§26d)
- Datenschutzgesetz (Data Protection Law)
  - Illegal to extract personal data
- Information on foreign jurisdictions:
  - E.g. from Electronic Frontier Foundation



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

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

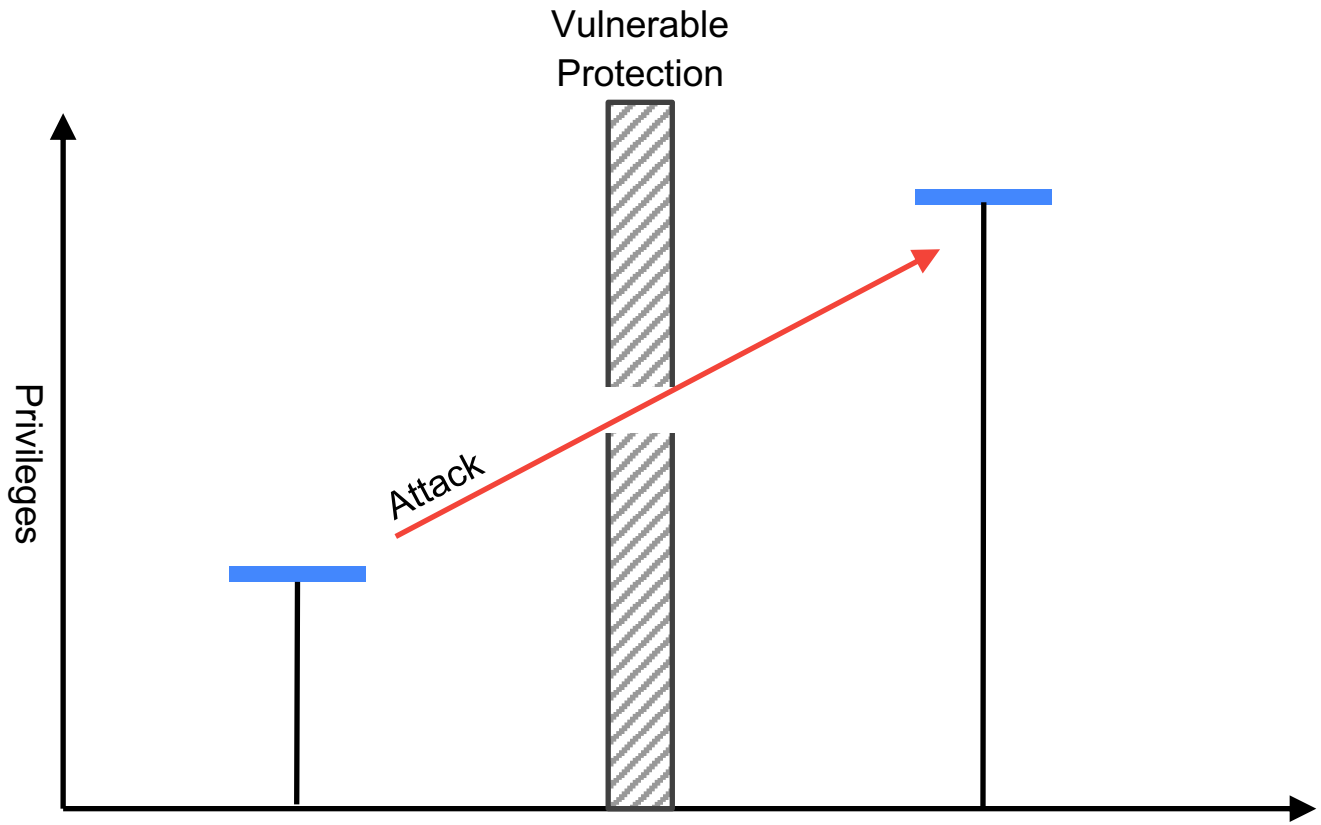
- Were not meant to be possible
- Have negative security repercussions

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

Exploit

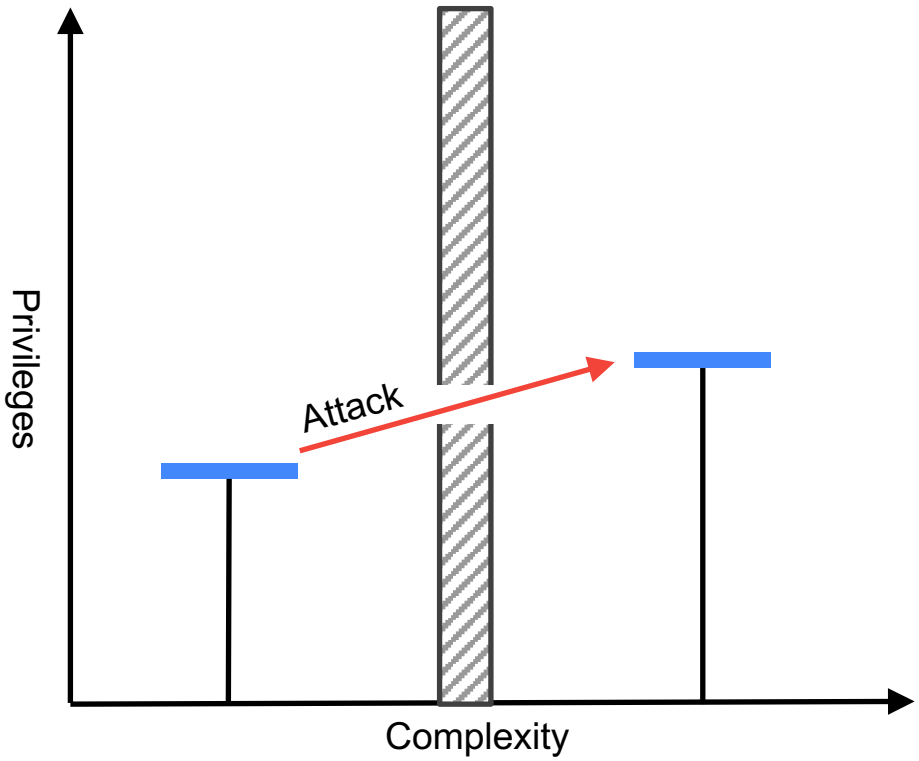
A vulnerability is only a theoretical problem until someone finds an exploit

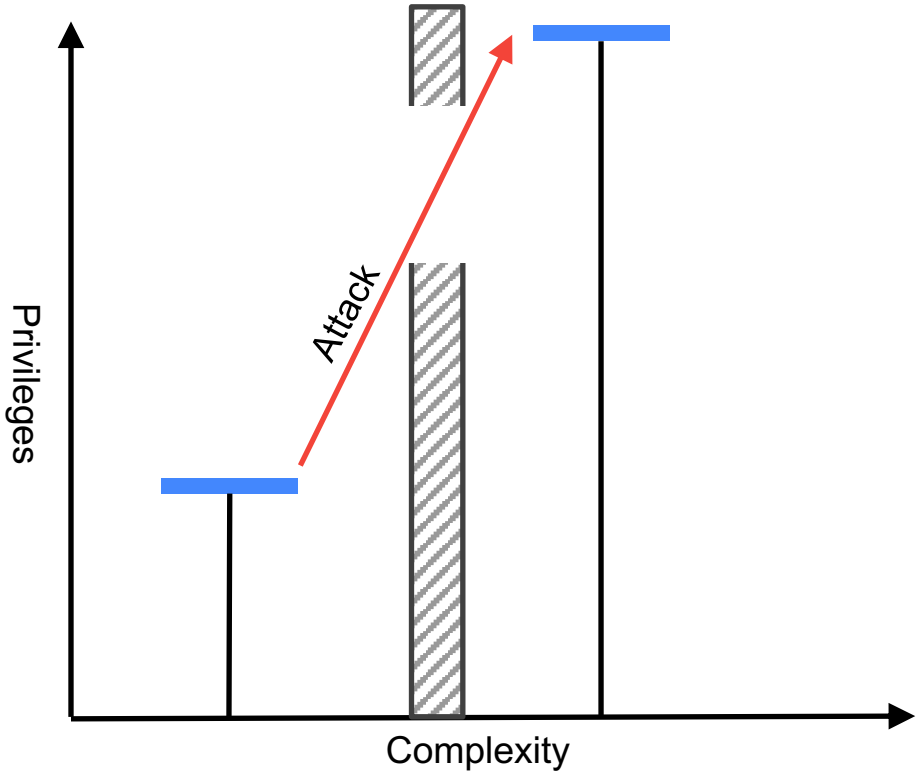
- Make use of the vulnerability for **elevating privileges**



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)

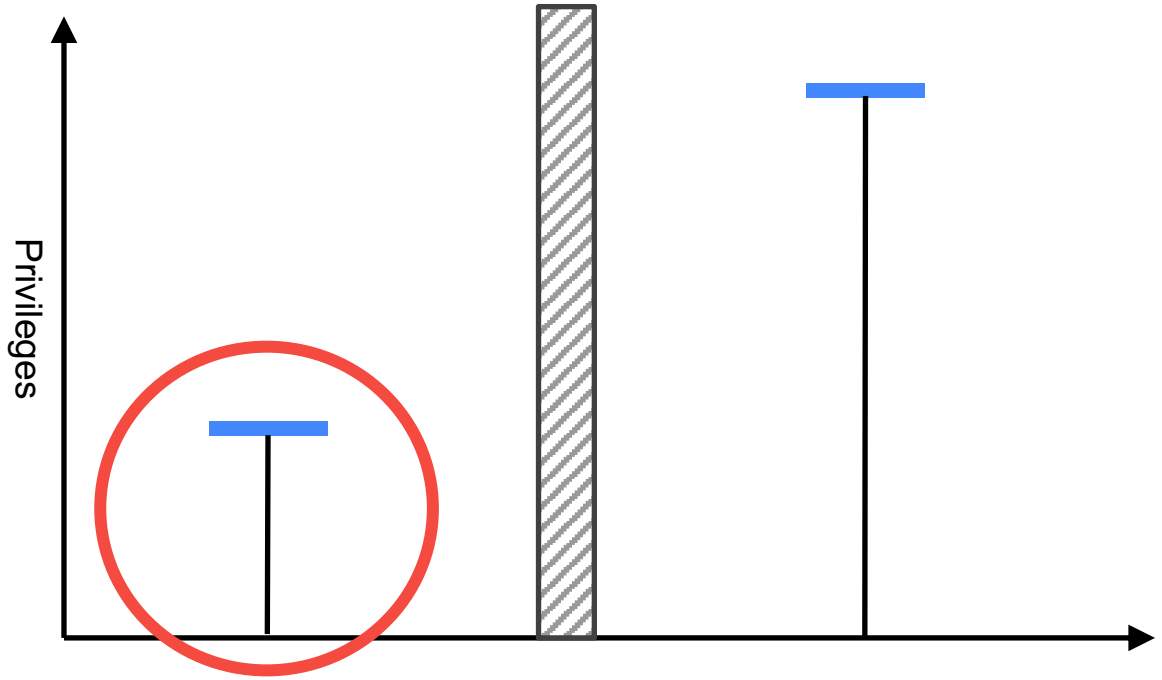
- E.g. Attacker with root execution ➔ TEE OS code execution

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

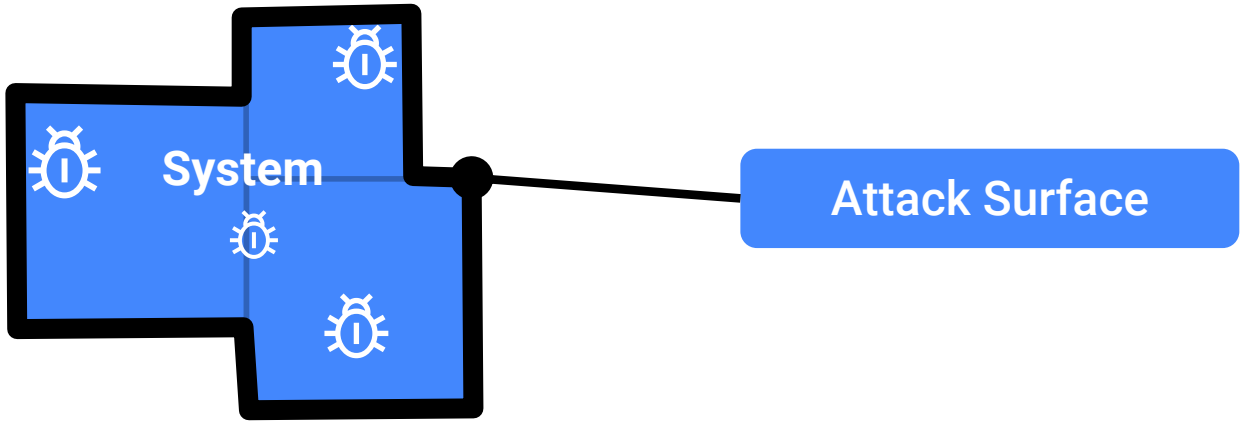
- Remote attacker
  - Attacker can lure victim to visit web page, reading email, receive SMS
  - Attacker can send a message to WhatsApp, Messenger, etc
- Proximal attacker
  - Attacker is physically in the environment of the victim
  - Same WiFi network, can send malformed Wifi / Bluetooth packets
- Local attacker
  - Attacker can run code on the victim's device (via installed app)
  - Attacker has physical access to the device

Source: source.android.com

Attack Surface

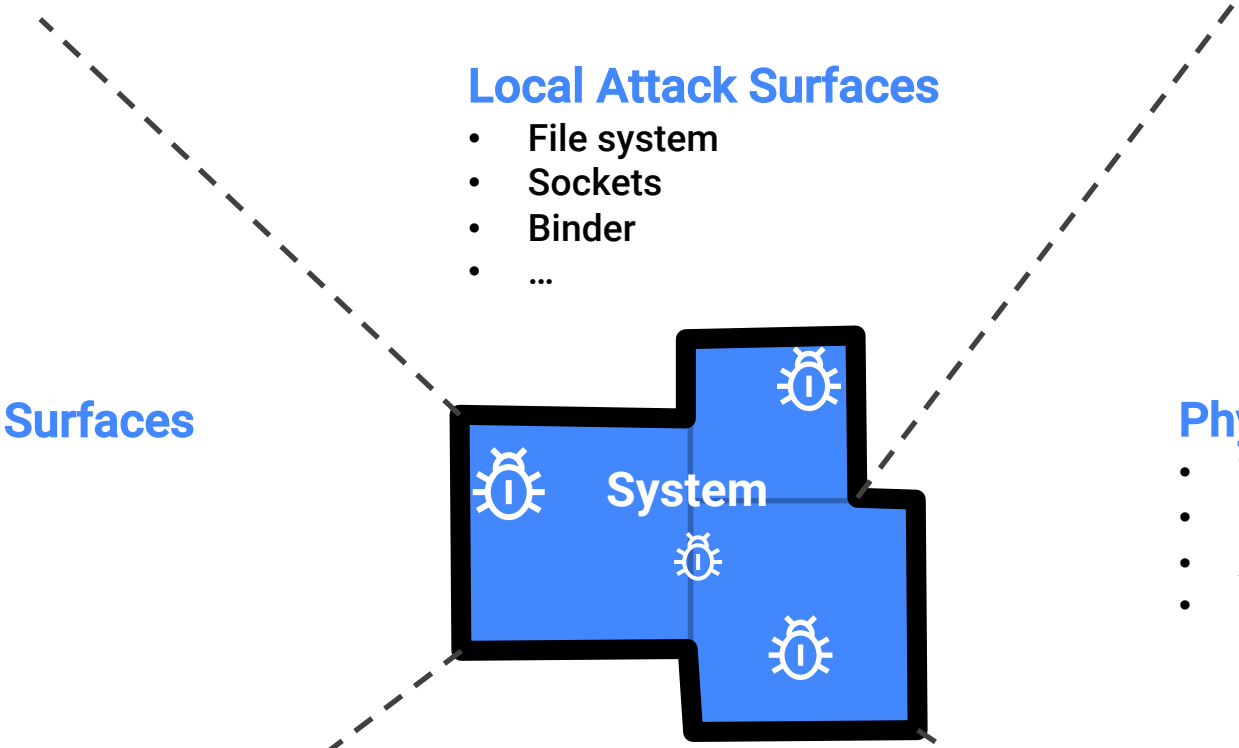
The parts of a system that can be reached by some attacker

- 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
- …

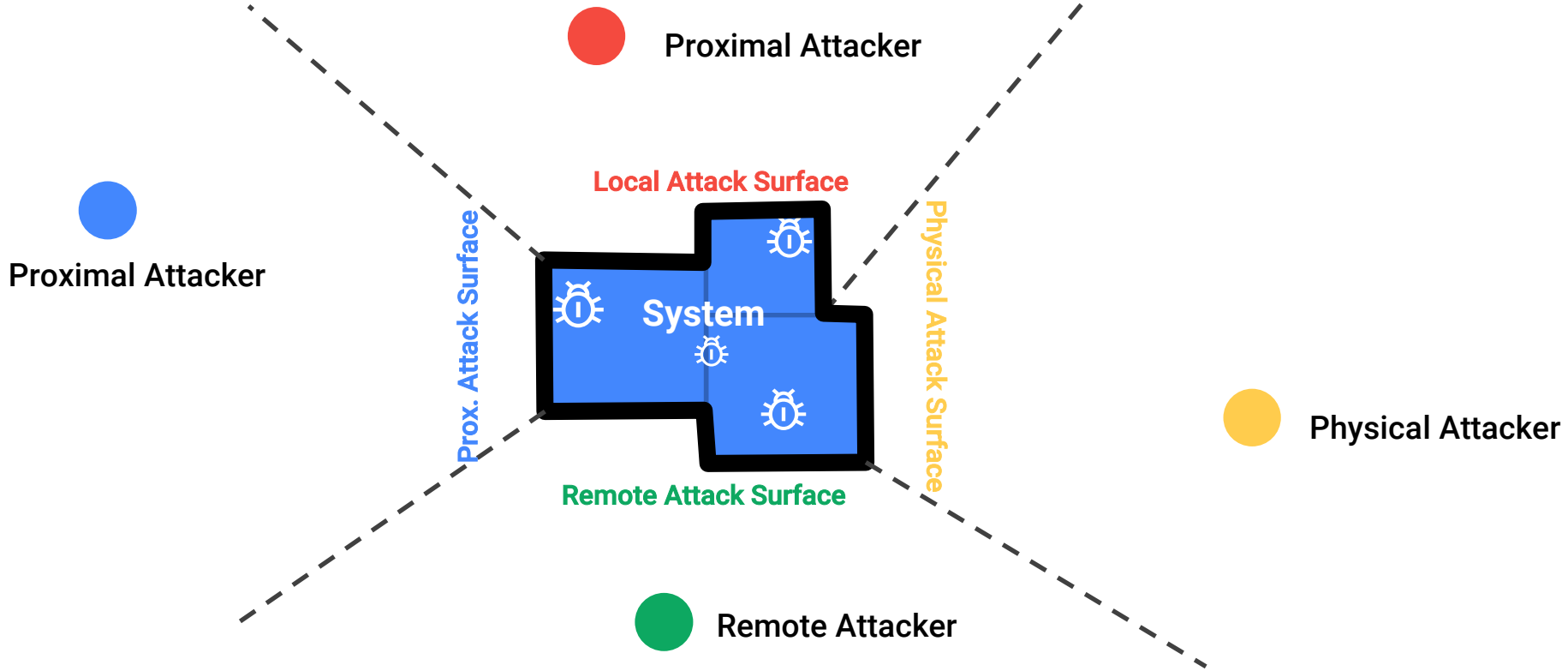
Physical Attack Surfaces
- Take the device apart
- USB
- ADB
- …

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

Bug Picture: Google / Apache 2.0

Attack Surfaces and Attacker Model

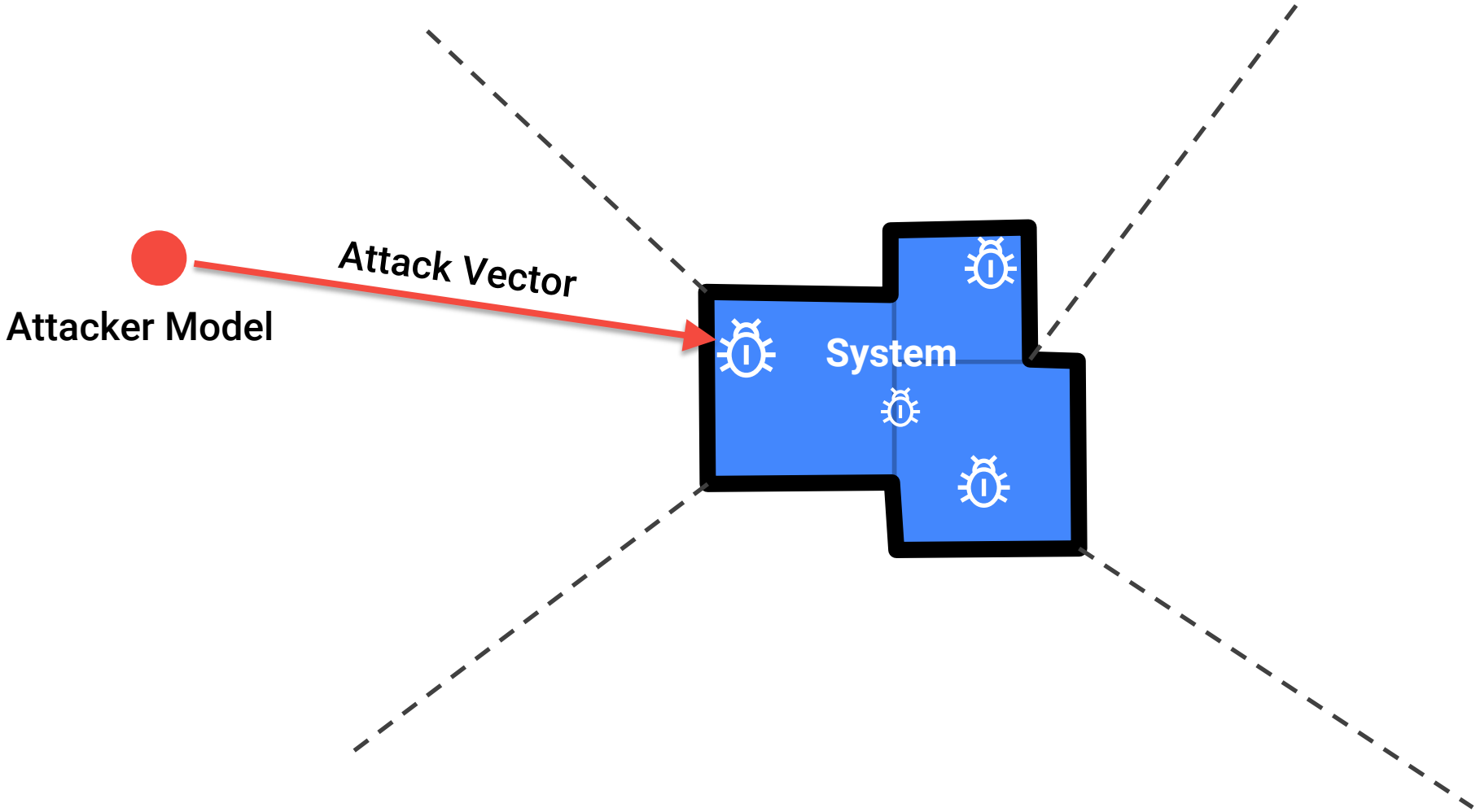
Different attack surface types reachable with different attacker / threat model



Note: Keep attacker model and attack surface separate despite correspondence!

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

Finding and exploiting vulnerabilities

Locating Vulnerabilities

- Now that we know what vulnerabilities are, how can we find them?
  - Requires creativity and perseverence!

1. Get inspiration
2. Decide on attacker & threat model
3. Enumerate reachable attack surfaces
4. Reconnaisance
5. Exploit vulnerability
6. Chain multiple exploits?

Getting inspiration

- Study recent research publications
  - Conferences, Journals, Blogs, Websites, Twitter
- Study Android Security Bulletin / iOS Release Notes
  - Contain references to fixed CVEs
- Keep track of new technological advancements
  - Newest iOS / Android versions
  - Covid Contact Tracing, …

Enumerating attack surfaces

- Consider all layers of the system
  - User, Apps, System, TEE, Hardware

- Sort all identified attack surfaces to obtain a prioritized list
  - Start with lowest requirements in terms of practicality
  - High privileges (e.g. running in kernel space)
  - Non-memory-safe programming languages (C/C++)
  - Complex data formats / state machines
  - Hasn’t been in the spotlight before

Reconnaisance

Now that you know where to look, how to actually find a bug?

- Manual Analysis
  - Find code that is used, accessible and vulnerable!
- Automated Analysis
  - Taint Tracking
    - Find code paths between a specified source and sink
  - Symbolic Execution
    - What input is needed for getting to specific output / execution point?
  - Requires some sort of manual a priori decision
    - Specific vulnerability class, target APIs, …

Fuzzing

Idea: Automatically feed random input to target process and provoke crashes

- How to generate data?
  - Mutation from some valid input data
  - Random mutations ➔ dumb fuzzing
  - From some format specification ➔ smart fuzzing
- How to feed data to target?
  - Easy for e.g. file viewers or cmd tools, not so much for GUI
- How to identify **exploitable** bugs?

Chaining Exploits

- Attackers can take different bugs and „chain“ them

- Example of a chain
  - RCE bug to go from „remote attacker“ ➔ „code execution in unprivileged app“
  - EOP bug in kernel ➔ Root code execution
  - EOP bug in a TEE‘s interface only visible to root ➔ TEE code execution

- All iOS jailbreaks are complex exploit chains

- Interesting read: **Chainspotting: Building Exploit Chains with Logic Bugs**
  - Chain of 11 bugs across 6 unique applications
  - Net effect: Remote attacker can install and run arbitrary APKs

Source: labs.f-secure.com

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
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

Preventing Vulnerabilities & Exploitation

Preventing vulnerabilities & exploitation

Computer programs are designed and constructed by humans

- They will always contain some flaws

Still, we can

- Use tools to prevent certain kinds of implementation bugs
- Make it hard to exploit vulnerabilities
- Build defenses against known kinds of attacks

Vulnerability Prevention

- Write code in memory-safe languages
  - Java over C/C++, Kotlin over Java!
- Use compiler-assisted protections
  - Integer overflow sanitizations, Java lint checkers, …
- Integrate security testing into CI pipelines
  - Automatic fuzzing as part of build process
- Use common sense!
  - Read documentation, don’t rely on stackoverflow or self-proclaimed experts™

Preventing Exploitation

Techniques that do not remove / fix vulnerabilities, but make exploitation harder

- Stack canaries
  - Ensure return pointer hasn't been overwritten
- Address Space Layout Randomization (ASLR)
  - Interesting target functions are at different addresses for every run
- Pointer Authentication (PAC)
  - Guard against manipulation of pointers

Defense in Depth

Traditionally, we only use defenses that restrict what attacker could do without it

- In theory, anything else would be useless, right?

For example:

- Why use EncryptedSharedPreferences Source: developer.android.com
  - Defense against root attackers?
  - We could simply hook the ESP reads/writes using root!
- When app's private folder is already encrypted with File Based Encryption
  - Defense against cold-boot attackers
- And app's private folder is not accessible to other apps
  - Defense against local non-root attackers

Defense in Depth

Implementing redundant defense mechanisms

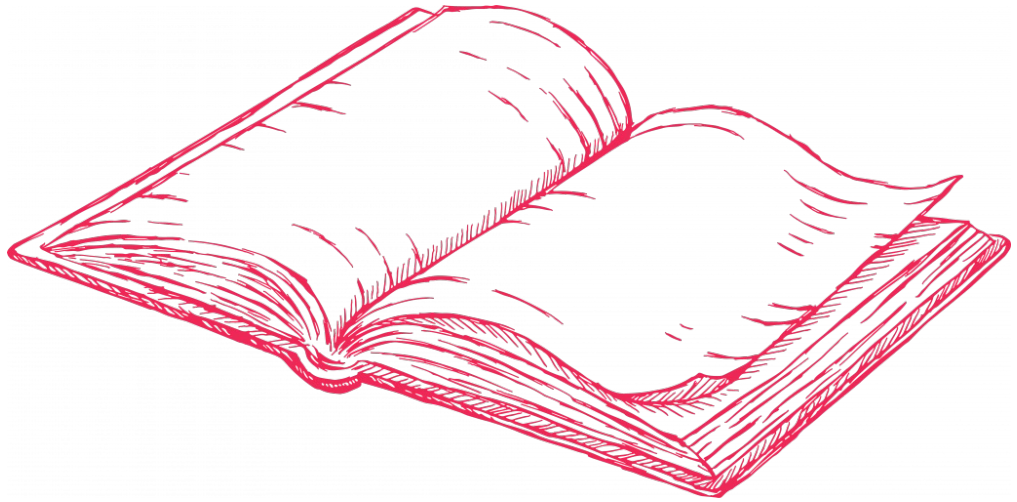
- Redundancy is still useful as long as different mechanisms are used
- Even if a bug in one mechanism is found, the other keeps protection in place
  - It's less likely to find a bug that affects both mechanisms
- **Example:** App sandbox is implemented using Linux UID and SELinux policy

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



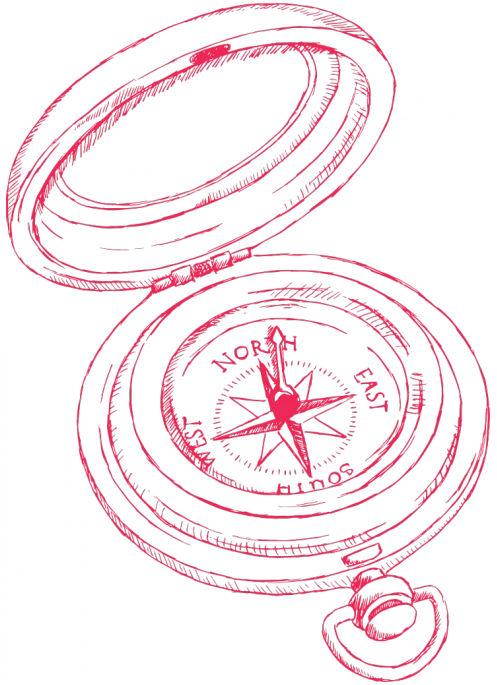
All slides are available on the course website

Lecture recordings are available from TeachCenter

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 IAIK!
  - Master theses
  - Research and teaching

We're hiring!

Send me an email!

florian.draschbacher@iaik.tugraz.at

Outlook

- 03.06.2022
  - Q&A for Assignment 2
- 10.06.2022
  - Assignment 2 Presentations
- 24.06.2022, 15:00-17:00, HS i2
  - Exam option 1
- 01.07.2022, 10:00-12:00, HS i12
  - Exam option 2

Exam registration open now