E-KRAAL Innovation Hub

TOPIC: Mobile Operating Systems and Architecture

PRESENTER: Christian Kisutsa & Team

Whoami - Christian

Speciality: Digital Forensics, Mobile Security and Network Security Monitoring

Projects: https://github.com/xtiankisutsa

Blog: www.shadowinfosec.io

Twitter: @xtian_kisutsa

Whoami - Francis

Speciality: Security Software Engineer | Researcher

Twitter: https://twitter.com/mutisy4

Blog: Coming soon!!

Whoami - Martin

Speciality: Reversing Engineering

Twitter: https://twitter.com/0xhexski

Blog: Coming soon!!

Why are we here???

- Understand mobile operating systems
- Reverse engineer mobile apps
- Statically analyze mobile apps
- Dynamically analyze mobile apps
- Be able to perform mobile app assessments

Where to start??

- Be INTERESTED or PASSIONATE
- Be able to GOOGLE
- READ books, blogs, articles, papers etc.
- WATCH videos, tutorials, walkthroughs etc.
- PRACTICE, PRACTICE, PRACTICE

Key Takeaways!

- Learn something new!!
- Gain some PRACTICAL skills and TECHNICAL knowledge
  - How to break android apps
  - How to identify potential vulns
- Understand the CONCEPTS on mobile security
- Inculcate a hacker MINDSET on how to accomplish tasks

Mobile Operating Systems

Apple’s Strategy

- Apple’s iOS ecosystem is quite closed
  - iOS is closed source
  - iOS and iOS apps can only run on Apple devices
  - You can install apps only from the Apple Store
  - Tricky to “jailbreak” iOS devices

- How come are they still around?
  - They were the first ~> significant chunk of market share
  - They make great products, and people know about it

Google’s Strategy

- Android Inc. started developing Android in 2003
  - Google purchased them in 2005

- Google needed to catch up
  - The “Open Handset Alliance” (84 companies)
  - Ecosystem is much more open
    - Android / AOSP is open source
    - Android can run on many different devices, even non-Google ones
    - Easy to inspect Android apps / reverse them / modify them
    - Easy to install apps you develop (“side loading”)
    - Developers can do many more things
    - Quite easy to “jailbreak” Android devices

Android vs iOS recap

	Android	iOS
Open-source OS?	✓	✗
Can OS run on non-Google/Apple device?	✓	✗
Can you run custom OS?	✓	✗
Can you sideload apps?	✓	✗
Can you run custom/modified apps?	✓	✗
Is it easy to tinker with apps?	✓	✗
Easy access to emulator?	✓	✗

Android vs iOS recap

	Android	iOS	Can this cause security problems?
Open-source OS?	✓	✗	
Can OS run on non-Google/Apple device?	✓	✗	
Can you run custom OS?	✓	✗	
Can you sideload apps?	✓	✗	
Can you run custom/modified apps?	✓	✗	
Is it easy to tinker with apps?	✓	✗	
Easy access to emulator?	✓	✗	

Android vs iOS recap

The reason may differ from what you expect!

	Android	iOS	Can this cause security problems?
Open-source OS?	✓	✗	~
Can OS run on non-Google/Apple device?	✓	✗	**Yes!**
Can you run custom OS?	✓	✗	**Yes!**
Can you sideload apps?	✓	✗	**Yes!**
Can you run custom/modified apps?	✓	✗	**Yes!**
Is it easy to tinker with apps?	✓	✗	**Yes!**
Easy access to emulator?	✓	✗	**Probably not**

Android Architecture

From the eyes of an app

- Android is based on Linux
- Each app has its own Linux user ID*
- Each app lives in its own security *sandbox*
  - Standard Linux process isolation
  - Restricted file system permissions

* There are ways to setup apps so that they share the user ID. See "sharedUserId".

App Installation

- The Android framework creates a new Linux user

- Each app is given a private directory
  - Also called "Internal Storage"
  - No other app can access it*

* There are ways to setup apps so that they share the user ID. See "sharedUserId".

App Isolation

- Apps are run in separate processes
- Apps being in sandbox means that they can't
  - talk to each other
  - do anything security-sensitive
- Q: how can apps do anything interesting?
- This is when architecture & security get mixed up

Android Framework Architecture

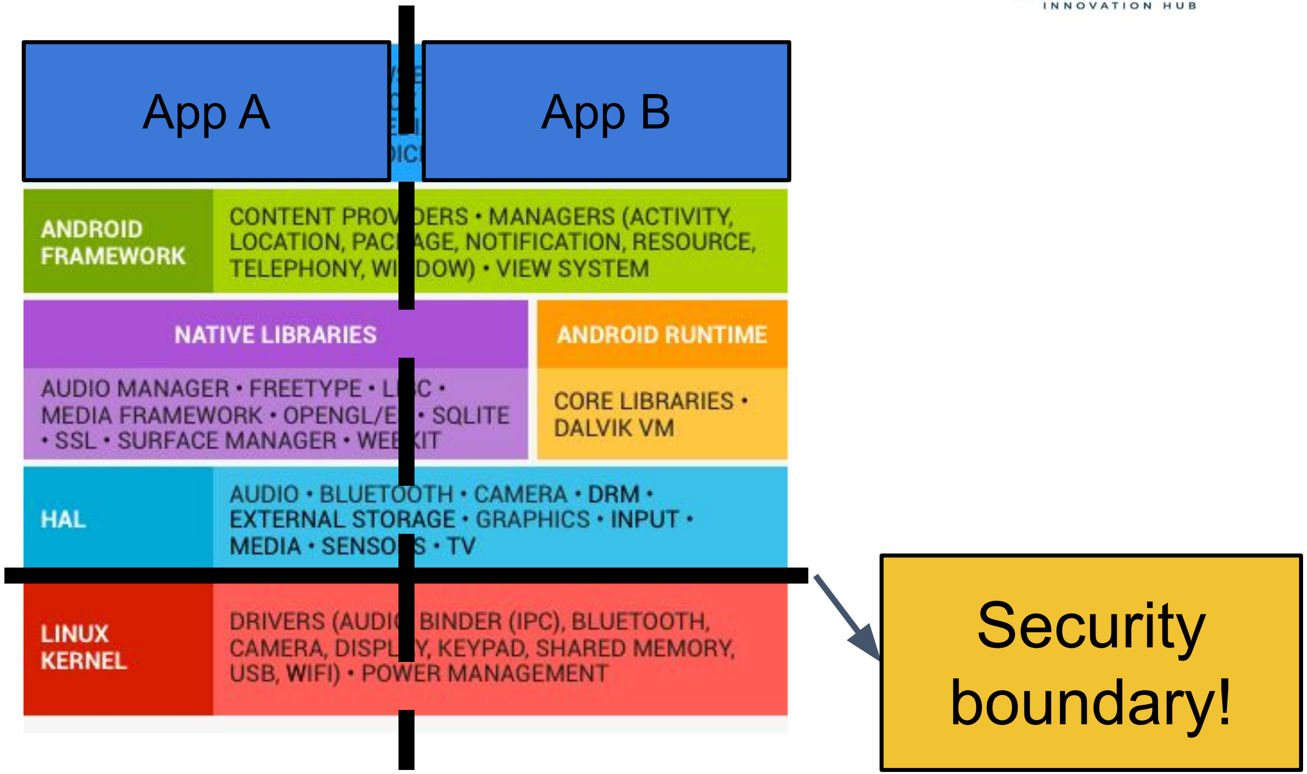


Image from https://source.android.com/security

Asking favors to the OS, aka "syscalls"

- Traditional OSes (like Windows, Linux, Android) have two worlds: user-space vs. kernel-space

- User-space is where user processes and apps live
  - They can't do much by themselves

- Kernel-space is where the actual OS lives
  - The OS is the God on your machine & information

Android Framework Architecture

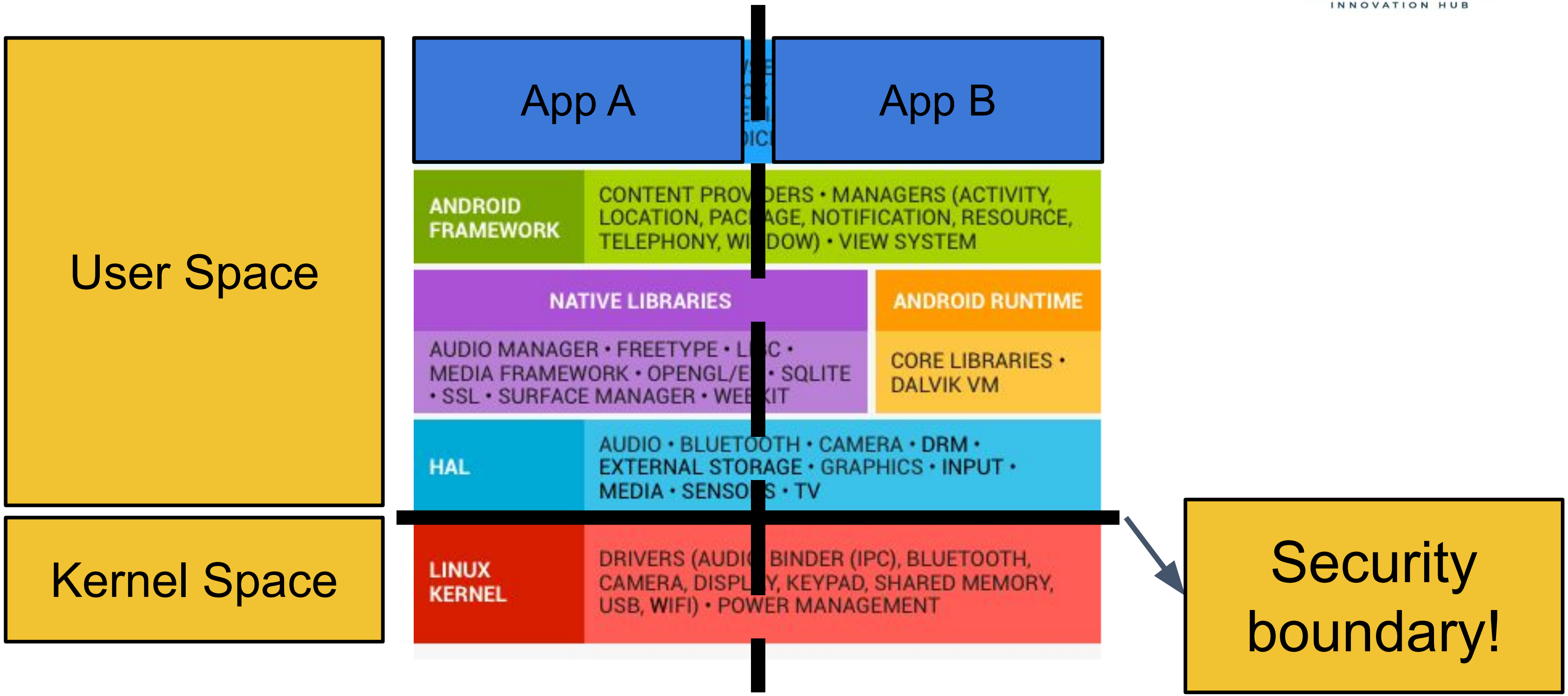


Image from
https://source.android.com/security

Example: Storing a File

- Let's say a process wants to save a file on the hard drive

- The process has no access to the physical hard drive
  - It would be too dangerous!

- The process needs to ask the OS
  - Would you mind saving file X with content ABC?

Example: Storing a File

- The developer uses high-level APIs

```
{
 ...
 OutputStreamWriter writer = new OutputStreamWriter(...)
 writer.write(data);
 writer.close();
 ...
}
```

- Under the hood, the process needs to ask the OS
  - Would you mind writing "data" in file XYZ?

Example: Storing a File

- Going down: Java ~> libc ~> syscalls
- fd = open(const char *filename, int flags, umode_t mode)
- n = write(unsigned int fd, char *buf, size_t count)
- close(unsigned int fd);

Not all requests are as easy as opening a file...

- Get current location?
- Send an SMS?
- Display something to the UI?
- Play a sound?
- Talk to other apps!?

Android Framework Architecture

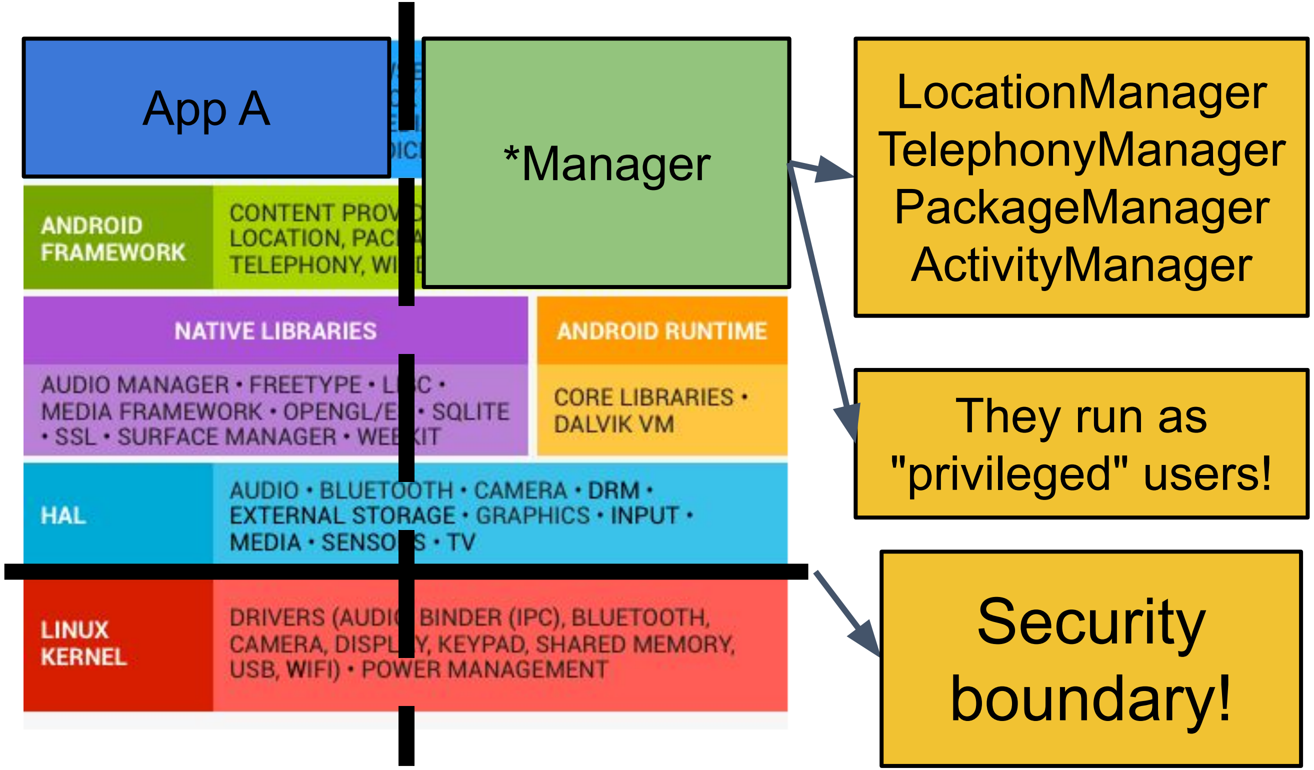


Image from https://source.android.com/security

Example: getLastLocation()

- App invokes Android API
  - LocationManager.getLastLocation() (ref)
  - We are still within the app's sandbox!

- Actual implementation of the privileged API
  - LocationManagerService.getLastLocation() (ref)
  - We are in a "privileged" service

- How do we go from one side to the other one?

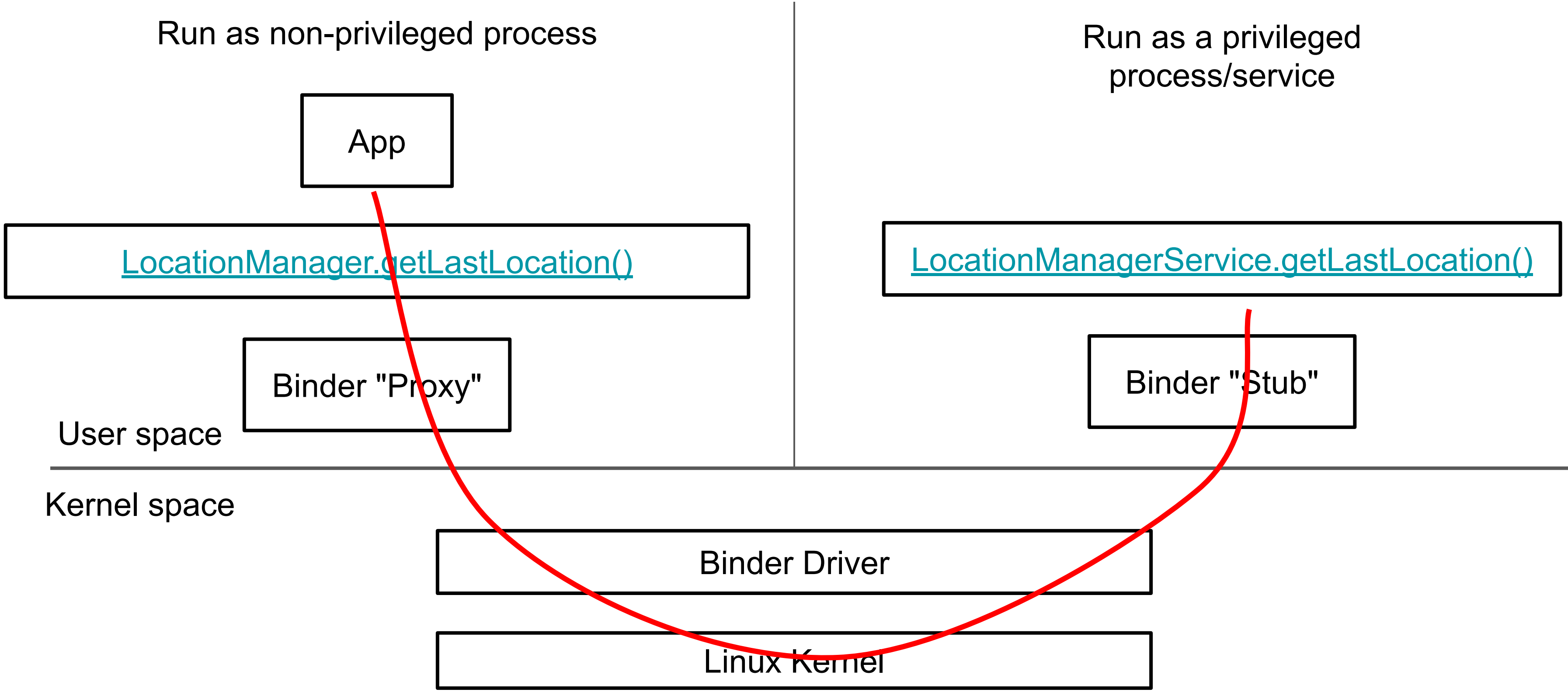
Crossing the bridge

- Binder!

- Binder: one of the main Android's "extensions" over Linux

- It allows for
  - Remote Procedure Call (RPC)
  - Inter-Process Communication (IPC)

Binder RPC


Run as non-privileged process
App
LocationManager.getLastLocation()
Binder "Proxy"
User space
Run as a privileged process/service
LocationManagerService.getLastLocation()
Binder "Stub"
Kernel space
Binder Driver
Linux Kernel

Binder details

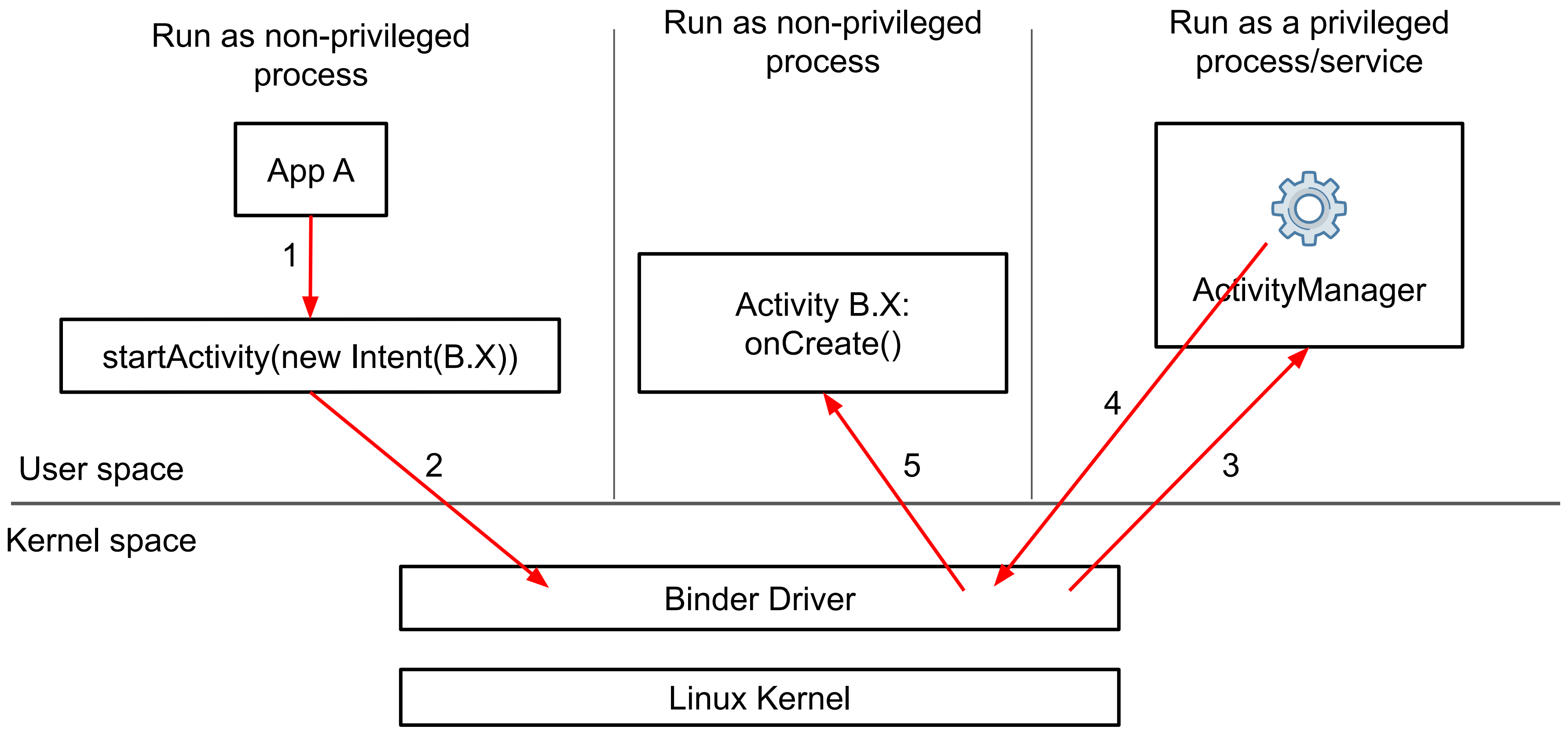
- Proxy and Stub are automatically generated starting from AIDL

- Binder internals
  - /dev/binder
  - ioctl syscall
    - Multi-purpose syscall, to talk to drivers
    - The Binder kernel driver takes care of it, dispatches messages and returns replies

Binder as IPC mechanism

- How do apps talk to each other?
- High-level API: Intents
- Under the hood: Binder calls!

Binder IPC: A → B.X



iOS Architecture

iOS apps

- Written in Objective C, now also in Swift
- Code is compiled directly to assembly (ARM)
  - There is no bytecode!
  - Much more difficult to reverse engineer
- But it gets worse...

iOS apps

- Every "method invocation" is transformed to a call to objc_msgSend(self, "method selector")

- "method selector" is a pointer to a string!
  - The string specifies which method should be actually invoked
  - backward dataflow + string analysis just to determine which method is called??

- First big difference: disassembling it is not as trivial as with Dalvik bytecode
  -

iOS apps

- In the general case: apps need to be published on the Apple Store before they can be installed on devices
  - Details later

- The app is shipped/installed encrypted
  - It is decrypted at run-time when the app starts

- iOS apps are developed with MAC OS' xcode IDE
  - Developing iOS apps on Windows/Linux is a mess...

iOS modern security features

- Code signing
  - Only "signed code" will be executed by the kernel
    - It needs to be signed by Apple itself
  - Only pages in memory that come from signed sources will be executed
  - Apps can't change their behavior & update themselves
  - Apps can't download+execute arbitrary code

- The code is signed when Apple "approves" the app to the Apple store

- *Very* big difference wrt Android!

Code signing exceptions

- The fact that Apple needs to sign ALL code that needs to be executed is very constraining; there are exceptions

- Enterprise / universities: they want to deploy their own apps, without having Apple in the loop
    - Ad-hoc, modified devices that can provision the code they want

Code signing exceptions

- Code signing, in theory, affects all code + libraries

- But browsers need to JIT javascript!
  - JIT: Javascript is Just-In-Time-compiled to assembly and executed
  - Javascript is served from within web pages, Apple can't preemptively sign them all

- One exception: "mobilesafari" can create a WX area + JIT
  - But, to prevent abuse, only ONE such area can be created

Data Execution Prevention

- Code and data sections
  - Code sections are not writable
  - Data sections are not executable
- We have the same protection in Linux...
  - Common bypass: code reuse / ROP to create WX memory ⇒ stage two
- ... but in iOS is trickier: you can't create WX memory
  - The entire payload needs to be via ROP

Apple store app reviews

- Developers develop an iOS app, then they submit it to the store for approval

- The review process seems more thorough wrt Google's
  - I have friends whose app was rejected because the UI was not nice enough

- If Apple approves, the app/code is signed and can be executed on Apple devices

Code Signing Bypass

- Is there any way to bypass Apple's code signing?
  - Bypass ⇔ developer can execute additional code after the approval

- People have found bugs in the code signature verification process... but is there something deeper?

- "Jekyll on iOS: When Benign Apps Become Evil"
  - Write app so that it contains an hidden memory corruption vuln
  - Hope that Apple doesn't find it (that's a fair assumption!)
  - After approval, exploit the vuln to gain arbitrary code execution

Provisioning System

- Individual developers need to test their apps!

- Steps for developer
  - He/She creates a dev account with Apple (and pay an annual fee)
  - He/She generates crypto keys
  - He/She generates a "provision file" + specify WHICH device to use
  - He/She can now run her app on that device

- Modern versions of xcode/iOS allow developers/users to install apps outside the Apple store, but only if the source code is available

- This is one of the many reasons why doing research on iOS is a pain...

Security Mechanisms

- Apple toolchain supports the classic protection mechanisms
- ASLR, stack canaries, …

App sandbox

- Privilege separation: apps have users/groups, they run in different processes with different users

- Sandbox is stricter than Android's
    - Apps can only read/write their own files
    - Apps can read "external files" only via user interaction / input
    - Example
        - To transfer a file from an app to another one, the app needs to show a UI picker to the user -- the user needs to be involved, she can't be left outside the loop

Permissions

- In the past
  - All apps run in the same type of sandbox
  - If it's ok for third-party app to have access to capability X, then all third-party apps have access to it
  - There was no permission system, and apps were able to access address book, GPS info, wifi AP names by using public APIs
  - BUT: in early versions of iOS, only system apps could be installed!
    - There was no Apple store for third-party apps!
- Modern versions
  - Permission system like Android's runtime permission model
  - Third-party apps can't ask for "draw on top",

Public vs. Private APIs

- Public APIs: APIs described in Apple's documentation
  - Apps can use these without problems

- Private APIs
  - There are a number of APIs that are "private"
  - They are not discussed in the documentation
  - "Interesting" design: private APIs ARE available within the memory space of every app -- they are just "hidden"
  - But there are of course many tricks to "discover them" at run-time…
  - … and to invoke them

Public vs. Private APIs

- Apple's Term of Service:
  - "Apps should NOT use private APIs"
  - This is ground for rejection

- But apps can technically do so... and it's not simple to analyze them...

- "iRiS: Vetting Private API Abuse in iOS Applications"
  - 7% of apps on the Apple store made use of private APIs...

Additional Hardening

- There is no "shell" in iOS
  - You either run stuff in the context of the exploited device, or you need to bring all the code you need with you

- iOS is based on BSD, but some calls are disabled
  - system(), fork(), exec() are disabled
  - An app CANNOT start a new app or launch a binary tool to process data
  - An app CAN start a new thread, but it will be inside the main app

Additional Hardening

- Secure boot (similar to Android's)
- Data encryption (similar to Android's)
  - UID key (embedded in hw, not accessible via sw) ⇒ device key
  - User-provided passcode (unlock) + UID key ⇒ passcode key
  - Four protection classes
    - Complete: file is protected and can be accessed only when device is unlocked
    - Complete unless open: file can be open only when unlocked, but then it's accessible
    - Protection until first user auth: file is protected until device boot + unlock first time
    - No protection

Additional Hardening

- Kernel Integrity Protection (KIP)
  - After the iOS kernel completes initialization, the memory controller deniers writes to the protected physical memory region

- Touch ID / Face ID
  - Touch ID: nothing special
  - Face ID: high-tech stuff, "invisible grid of dots projected towards your face to build a 3D model"
    - Claim from a company: "it has been hacked" (news article)
    - But the news is quite old and I didn't see follow ups. They may have over claimed.

Additional Hardening

- The Secure Enclave is a separate, high-security processor
  - The analogous of ARM TrustZone
- The Secure Enclave checks that its own software is signed by Apple before booting
- Many security-related APIs rely on it
  - Encryption-related material, Touch ID, Face ID, Apple Pay

Pointer Authentication Codes

- Pointer Authentication Codes (PAC)
  - PAC-protected pointers contain the address + crypto/auth tag
  - An attacker can't arbitrarily alter a pointer to make it point somewhere else -- even if she has full write capabilities!

- Goal: mitigate memory corruption vulns, making them more difficult to exploit
  - But not impossible; some bypass techniques are known
  - Example: pointer leak and local reuse, exchange puts ⇔ system

- More info by Qualcomm: link

Jailbreak

- iOS devices are very "closed down"
- Users want to do more stuff with their devices
- Jailbreak: the process of getting "root" on these devices

Jailbreak

- Disable iOS security checks
- Install arbitrary apps (even third-party closed-source)
- Gain root access
- Load arbitrary customization
    - Custom UI, custom kernel, ...

Jailbreak

- Cydia is a package manager for iOS mobile apps
- Used on jailbroken iPhones to install third-party apps
- Cydia Substrate: framework to modify iOS with extensions

Jailbreak: history and current status

- First jailbreak for iPhone 1.0 in 2007
- Several jailbreaks for other versions follow
- "jailbreakme.com" remote jailbreak (2011)
  - Jailbreak by just visiting a URL
  - People were trolling Apple by going to Apple stores and visiting this website with the "free to use" devices there
- Current status:
  - There is a big community of people that PRETEND jailbreaks
  - People that find these bugs/exploit them got pissed, stopped releasing
  - Apple would fix them immediately anyways...

Reversing iOS apps

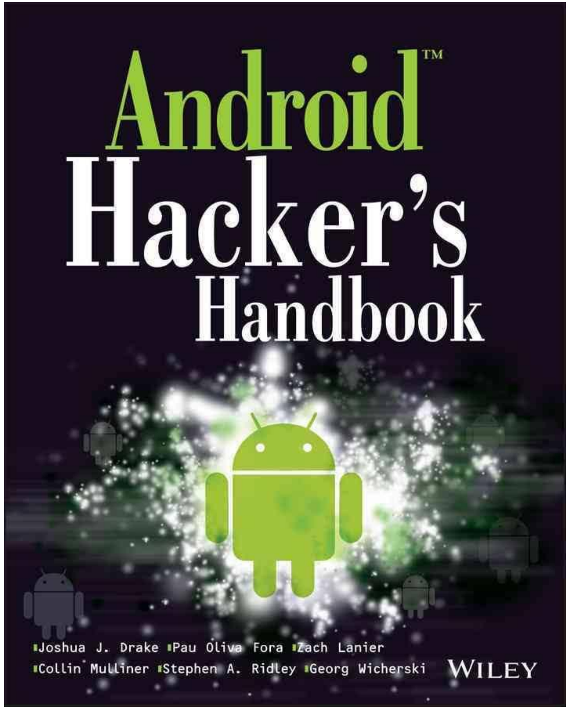
- Run the app in a jailbroken device and dump memory to get the decrypted app's content
    - Tool: Clutch (load, set breakpoint, dump, …)

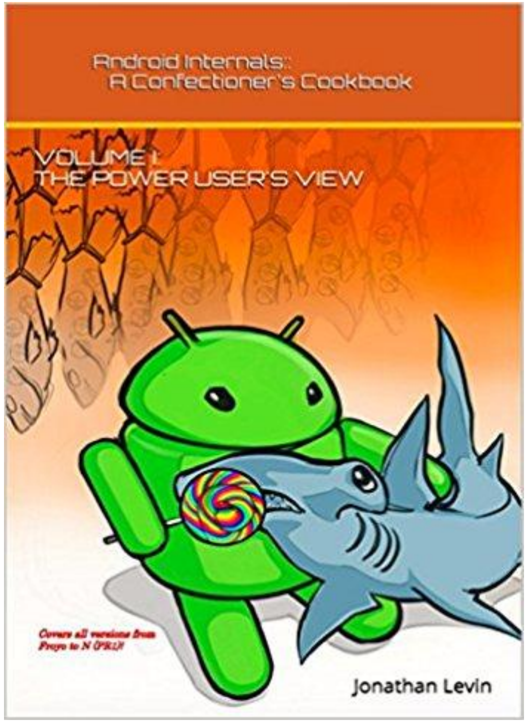
- Analyze it in IDA, Ghidra, Radare, Cutter, Hopper etc

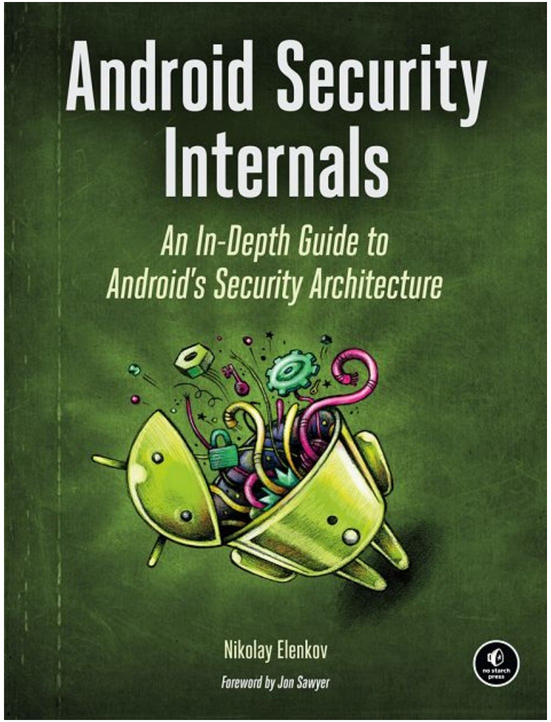
- Tools like frida work on iOS as well…

- Use cydia to install openssh / other tools to have a decent debugging/devel environment

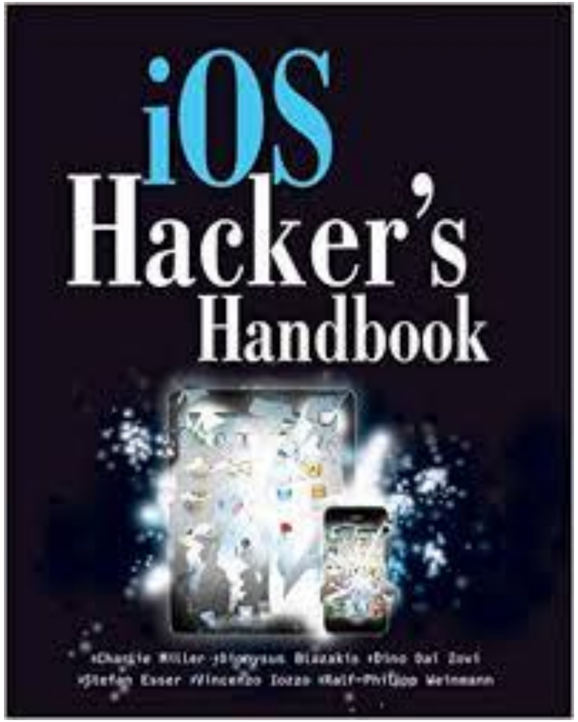
Recommended Reading (Android)

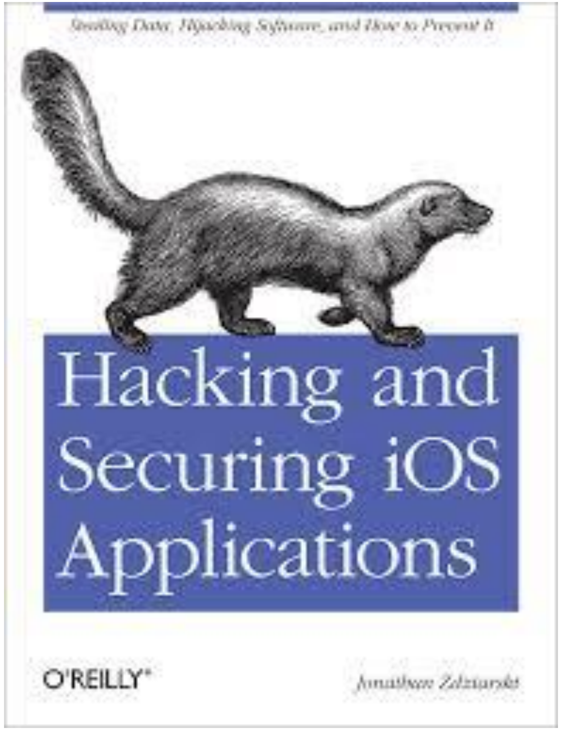


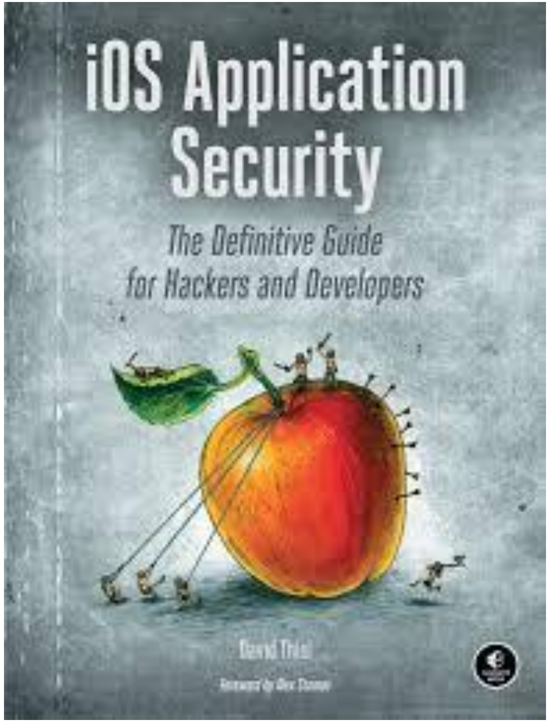




Recommended Reading (iOS)







References

- https://mobisec.reyammer.io/
- https://mobilesecuritywiki.com
- https://github.com/sindresorhus/awesome
- https://github.com/rednaga
- https://github.com/OWASP/owasp-mstg