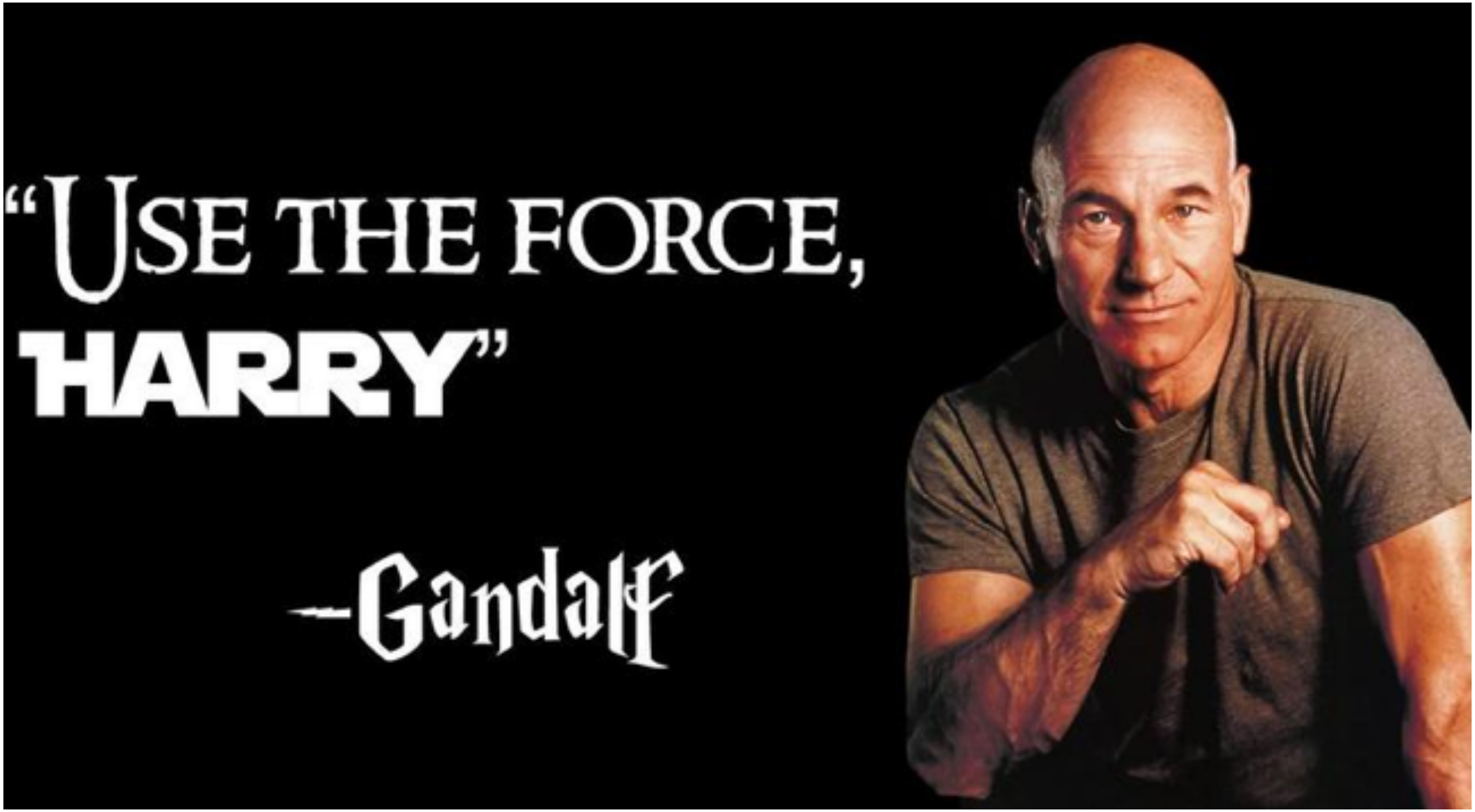
Happy May the 4th!!!!1!



Android Security

CS 642

Drew Davidson

Some Slides taken from John Mitchell

Lecture Roadmap

- What is Android?
  - History
  - Design
- Exploits
  - System Defenses
- Other Attacks
  - Threats
  - Defenses

What is Android?

- A lot of things to different people
  - The fabled gPhone
    - Invites comparison to the iPhone
  - An internet of things (IoT) platform
  - An operating system for your car?
- Too big to explain in this lecture
  - We'll introduce some security features as needed
  - More to learn

(Ancient) History of Android

- 2003: Andy Rubin cofounds Android Inc to build a web-connected smartphone
- 2005: Google acquires Android Inc
- 2007: iPhone Gen I released
- 2008: HTC Dream (G1) released

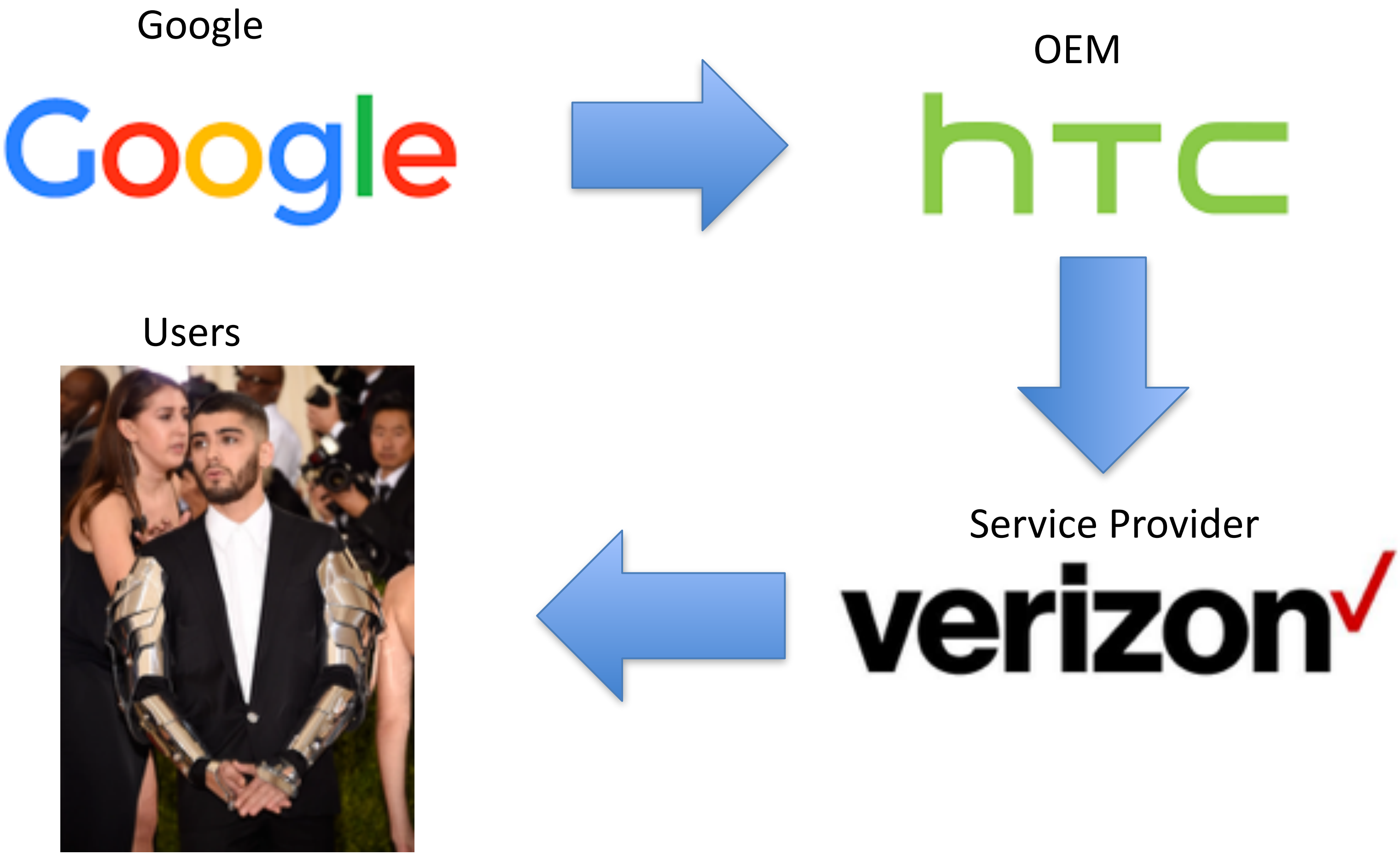
Android Design

- More than an Operating System
  - A specialized Linux distro, at the lowest level
  - A framework for running Android "apps"
  - An entire ecosystem for smartphone users

Android Open Source Project

Apps
App Store (Google Play)
Development tools
Closed-Source Components

From Google to You



Google

OEM

Service Provider

Users

Android Exploits

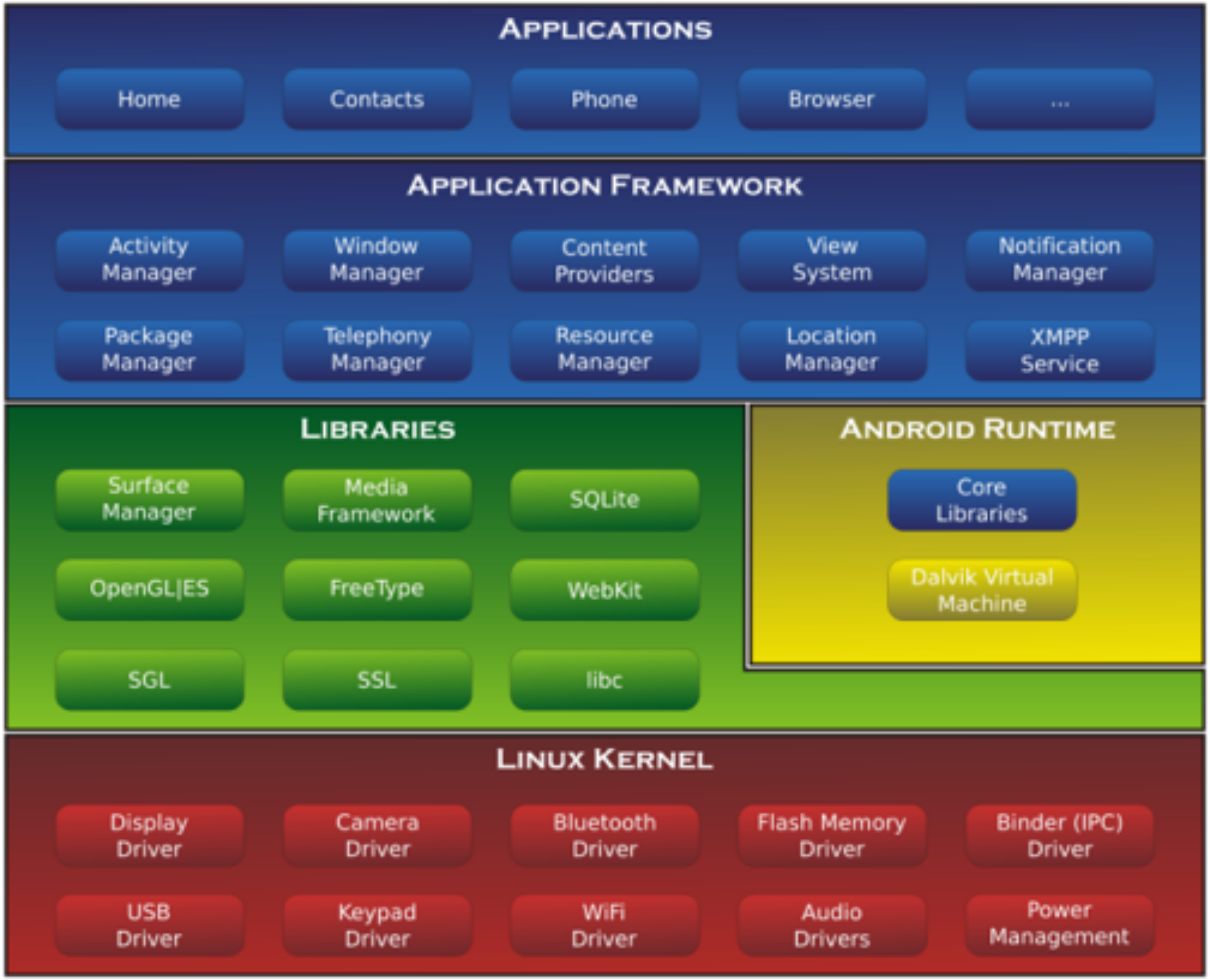
What is an Android Exploit?

- Working definition:

  An action that occurs in contravention of the security model of an Architecture

- Examples:
  - Privilege Escalation: User code runs as root
  - Data Exfiltration: App steals another’s data
  - DOS: App renders device unusable

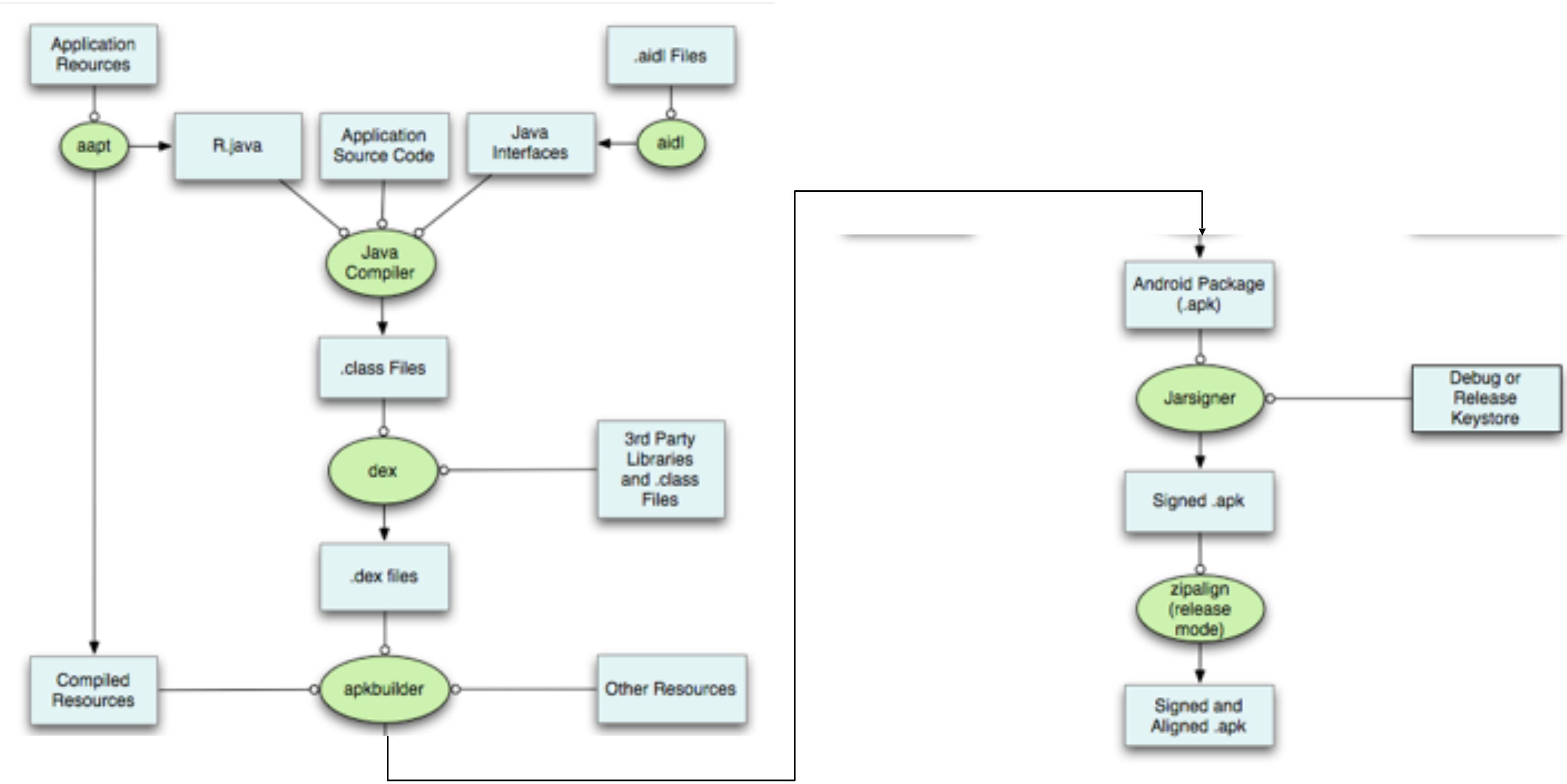
Multi-Layered Architecture



Application Design

- Each app runs within an independent instance of the Dalvik Virtual Machine (DVM)
  - Apps largely run bytecode
  - Each app runs as its own user, i.e. there is a separate UID for each app

App Deployment



Intra-Application Security

- Signed code
  - Prevents out-of-band rewrites
- Java-style Sandbox protections
  - Bytecode verifier prevents ill-formed programs
  - Runtime checks against buffer overflows, etc.
  - Could use the security manager for policies
- Android Lifecycle, App Killer
  - System may pause an app
  - System may kill an app with too many resources

Inter-Application Security

- OS level protections
  - Separate UIDs give apps distinct privileges
  - Minimizes privilege escalation
- Binder IPC
  - Kernel mediates communication between apps
  - Receiving app must register for incoming messages

OS Protection

- ASLR
  - – Makes it statistically impossible/improbable to know if you’re smashing the stack effectively
- Dlmalloc
  - – Makes it much harder to spray the heap

Google Play (Store)

- Largest distribution channel for apps
  - Kill switch
  - Google Bouncer
  - “Wisdom” of the crowds

Exploits Still Happen

- Confused deputy
  - Stagefright
- Data exfiltration
  - Sensor side-channels
    - Microphone, Gyroscope
  - App misconfiguration
    - Facebook Debug log
- Denial of Service
  - Exception loops
  - Battery drain

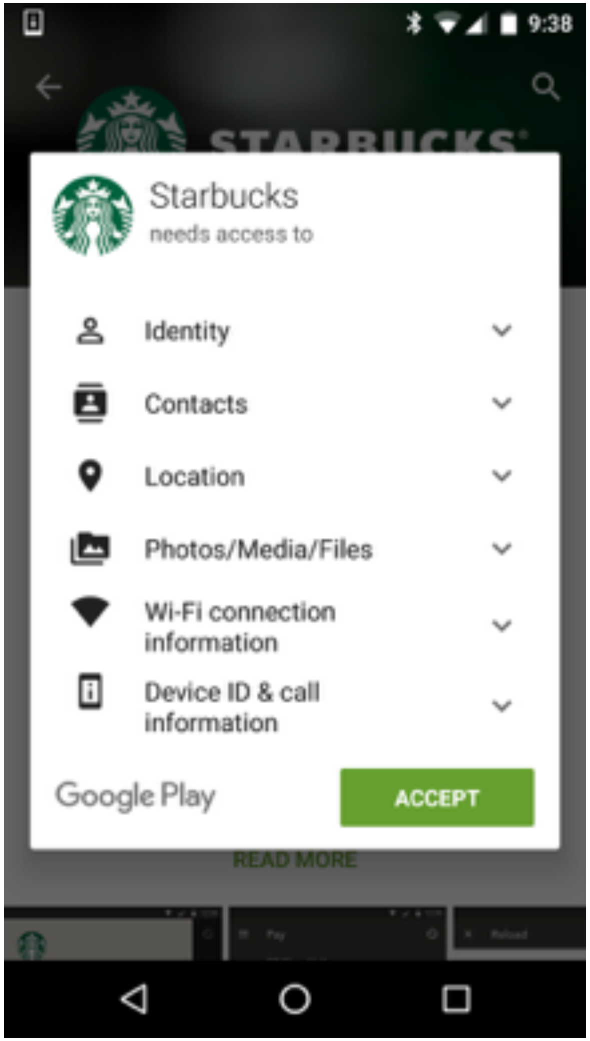
Other Threats

Shady Code

- The previous definition of exploit was somewhat weak
  - What happens when the security model is insufficient?
- Enable “PII attacks”
  - Broadly, attacks that leverage your personally identifiable information

Shady Code Defenses

- Android Permissions
  - – Install-time permissions



Starbucks
needs access to

Identity

Contacts

Location

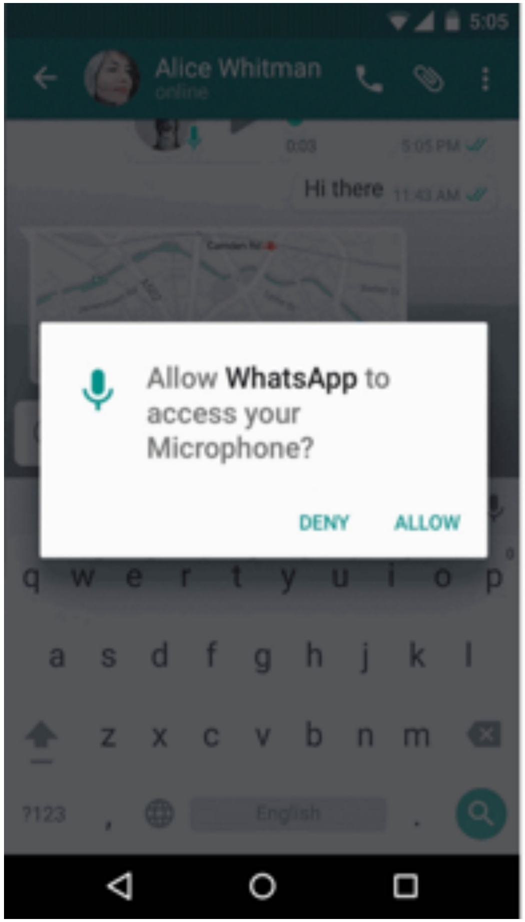
Photos/Media/Files

Wi-Fi connection information

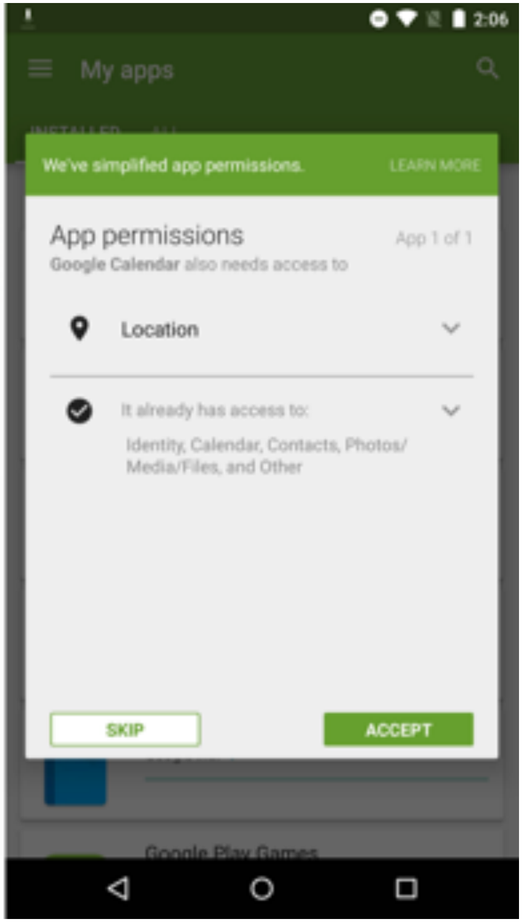
Device ID & call information

Shady Code Defenses

- Android Permissions
  - Runtime
  - Update-Time



Allow **WhatsApp** to access your Microphone?



We've simplified app permissions.

App permissions App 1 of 1

Google Calendar also needs access to

Location

It already has access to:

Identity, Calendar, Contacts, Photos/Media/Files, and Other

Category	Permission	Description
Your Accounts	AUTHENTICATE_ACCOUNTS	Act as an account authenticator
	MANAGE_ACCOUNTS	Manage accounts list
	USE_CREDENTIALS	Use authentication credentials
Network Communication	**INTERNET**	**Full Internet access**
	ACCESS_NETWORK_STATE	View network state
Your Personal Information	**READ_CONTACTS**	**Read contact data**
	WRITE_CONTACTS	Write contact data
System Tools	WRITE_SETTINGS	Modify global system settings
	WRITE_SYNC_SETTINGS	Write sync settings (e.g. Contact sync)
	READ_SYNC_SETTINGS	Read whether sync is enabled
	READ_SYNC_STATS	Read history of syncs
Your Accounts	GET_ACCOUNTS	Discover known accounts
Extra/Custom	WRITE_SECURE_SETTINGS	Modify secure system settings

What’s the Problem with Permissions?

- Admittedly, a step up over the Desktop
  - Arguably, table stakes for such a personal device

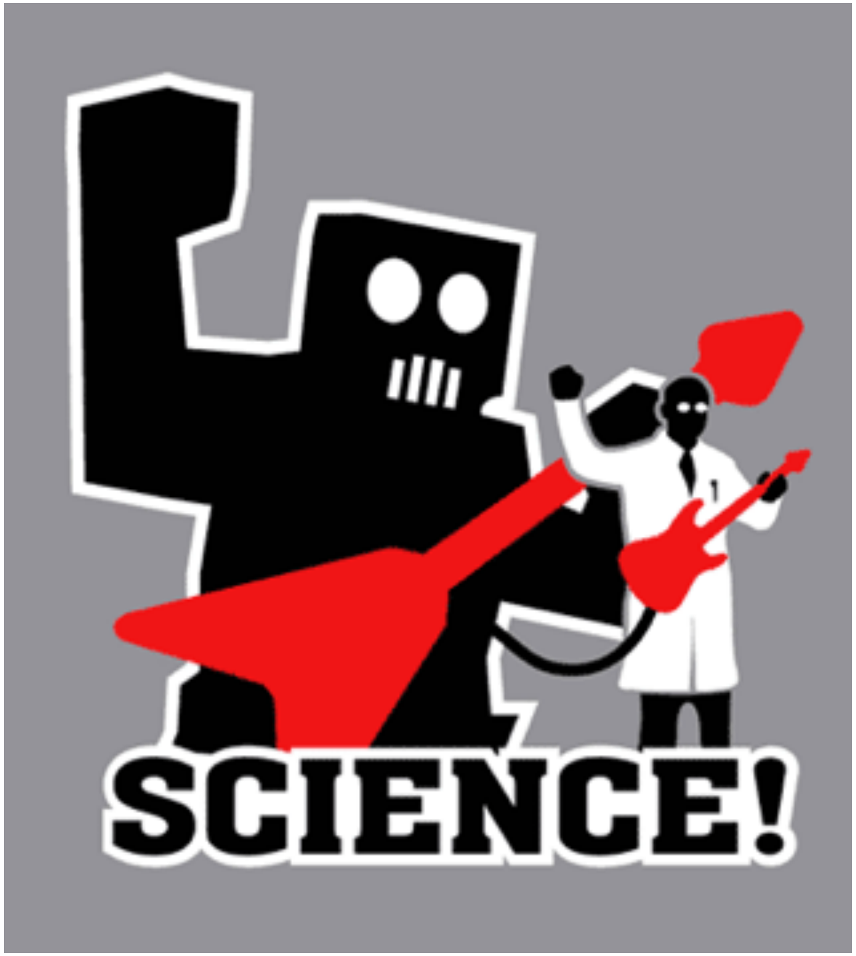
- “Permission entanglement”
  - You may control when a permission is used, but not *how*
    - Permissions are per-app thus shared with libraries
    - A single permission may be used in various ways
    - Composite effect of permissions exceed sum

Fixing Shady Code

- Fewer easy answers
  - One person’s privacy violation is another’s feature
    - Location-aware advertising?

Now Entering the Realm of Research

- What follows is a discussion of research prototypes
  - Unlike above, there are occasionally obvious reasons NOT to do these things



Data flow analysis

- Label the uses of permissions in the program
  - Sources: produce sensitive information
  - Sinks: interact with untrusted entities
- We’d like to know how these endpoints interact

- Tools
  - FlowDroid
  - Stamp

Example Endpoint permissions

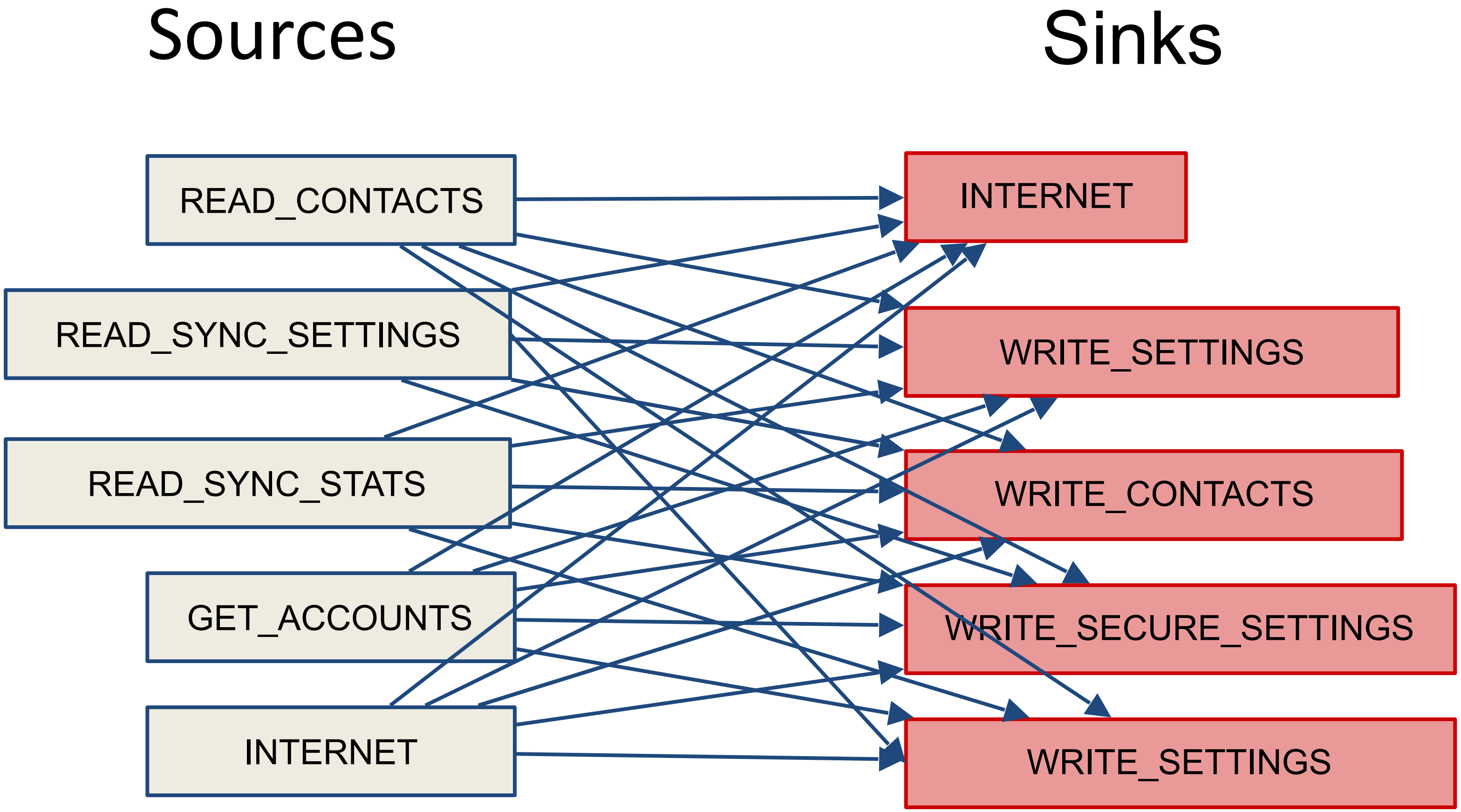
Sources

- Account data
- Audio
- Calendar
- Call log
- Camera
- Contacts
- Device Id
- Location
- Photos (Geotags)
- SD card data
- SMS

Sinks

- Internet (socket)
- SMS
- Email
- System Logs
- Webview/Browser
- File System
- Broadcast Message

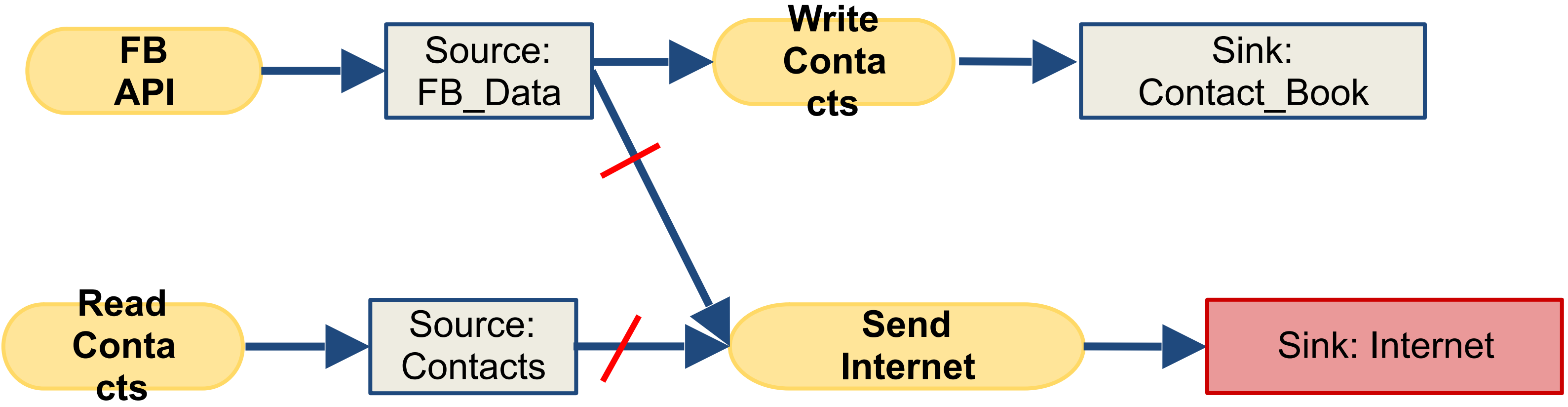
Possible Flows



Implementing Dataflow Analysis

- Identify what methods use which permissions
  - No canonical map!
- Identify what permissions actually do
  - Is it a source? Sink? BOTH?
- View the program as a Program Dependence Graph
  - Edges represent flows of control or data
  - Nodes represent abstract regions of code
  - Requires a program semantics / abstraction

Dataflow Analysis Example



Limitations of Dataflow Analysis

- Technical
  - Over-approximate
  - Requires deep knowledge of the system
    - Impractical without some manual modelling, at least on Android
- Practical

  …ideas?

(Dynamic) Taint Tracking

- Not the most media-savvy name
- Extend the system to record the provenance of data
    - Is it *tainted* by an input source?
- Tools
    - TaintDroid

Limitations of Dynamic Taint Tracking

- Technical limitations
  - Misses control dependencies
- Practical limitations
  - Slows execution
    - Could use it solely as an offline analysis

App Rewriting

- Change the behavior of the app
  - Reverse engineer it
  - Make some changes
  - Recompile it

DroidWeave

- To the board!

Conclusion

- Good luck on Finals!

- If you’re graduating, good luck in life!