# Mobile Device and Platform Security – Part II

John Mitchell

# Two lectures on mobile security

- Introduction: platforms and trends
- Threat categories
  - Physical, platform malware, malicious apps
- Defense against physical theft
- Malware threats
- System architecture and defenses
  - Apple iOS security features and app security model
  - Android security features and app security model
- Security app development
  - WebView – secure app and web interface dev
  - Device fragmentation

Thurs

Tues

# ANDROID

History and early decisions

# Android history

- Android, Inc founded by Andy Rubin around 2005
  - Worked with HTC-built device with a physical keyboard
  - Scrapped Blackberry-like phone when iPhone came out
  - First Android phone HTC Dream, Oct 2008 (T-Mobile G1): touchscreen and keyboard
- Open-source software project
- Backed and acquired by Google

# HTC Dream

- First phone had
  - Android 1.6 (Donut)
  - 3.15 megapixel rear camera with auto-focus
  - 3.2 inch touchscreen
  - Gmail, Google Maps, Search, Google Talk, You Tube, calendar, contacts, alarm

# Android ecosystem

- Open-source software distributed by Google
  - Business goal: increase number of users and devices linked to core Google products
- Multiple hardware vendors
  - Each customize software for their products
- Open marketplace for apps
  - Aim for scale

Aside: open-source OS successful pre-mobile

# App market

- Self-signed apps
- App permissions
  - granted on user installation
- Open market
  - Bad apps may show up on
  - Shifts focus from remote exploit to privilege escalation


# ANDROID PLATFORM

Theft and loss protection

Predictive security

- Look for malicious code in apps

Privacy advisor

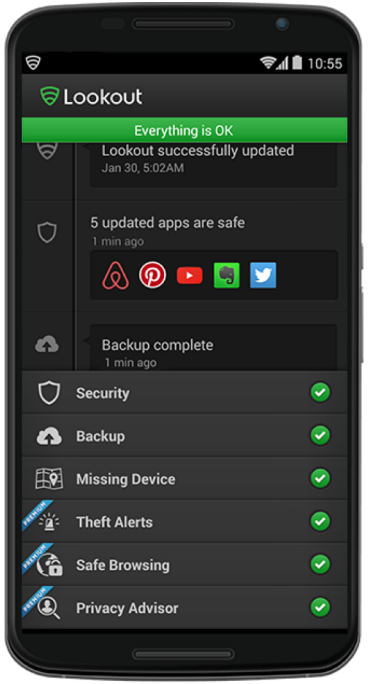
- See if app can access private information

Locate lost phone

- Map location and make a sound

Lock and wipe

- Web interface to remotely remove data
- Data backup
- Store and retrieve from cloud




https://www.lookout.com/android

# Device lock and unlock

- Similar PIN and fingerprint
- Fingerprint API lets users
  - Unlock device
  - Securely sign in to apps
  - Use Android Pay
  - Purchase on Play Store

# ANDROID PLATFORM

Managed code

# Managed code overview

- Java programming language
- Bytecode execution environment
  - Verifier
  - Run-time checks
  - Memory safety
- Permission checking
  - Stack inspection

# Java language overview

Classes and Inheritance

- Object features
- Encapsulation
- Inheritance

Types and Subtyping

- Primitive and ref types
- Interfaces; arrays
- Exception hierarchy

Generics

- Subtype polymorphism. generic programming

Virtual machine

- Loader and initialization
- Linker and verifier
- Bytecode interpreter

Security

- Java “sandbox”
- Type safety
- Stack inspection

# Managed code overview

Java programming language

Bytecode execution environment

- Verifier
- Run-time checks
- Memory safety

Permission checking

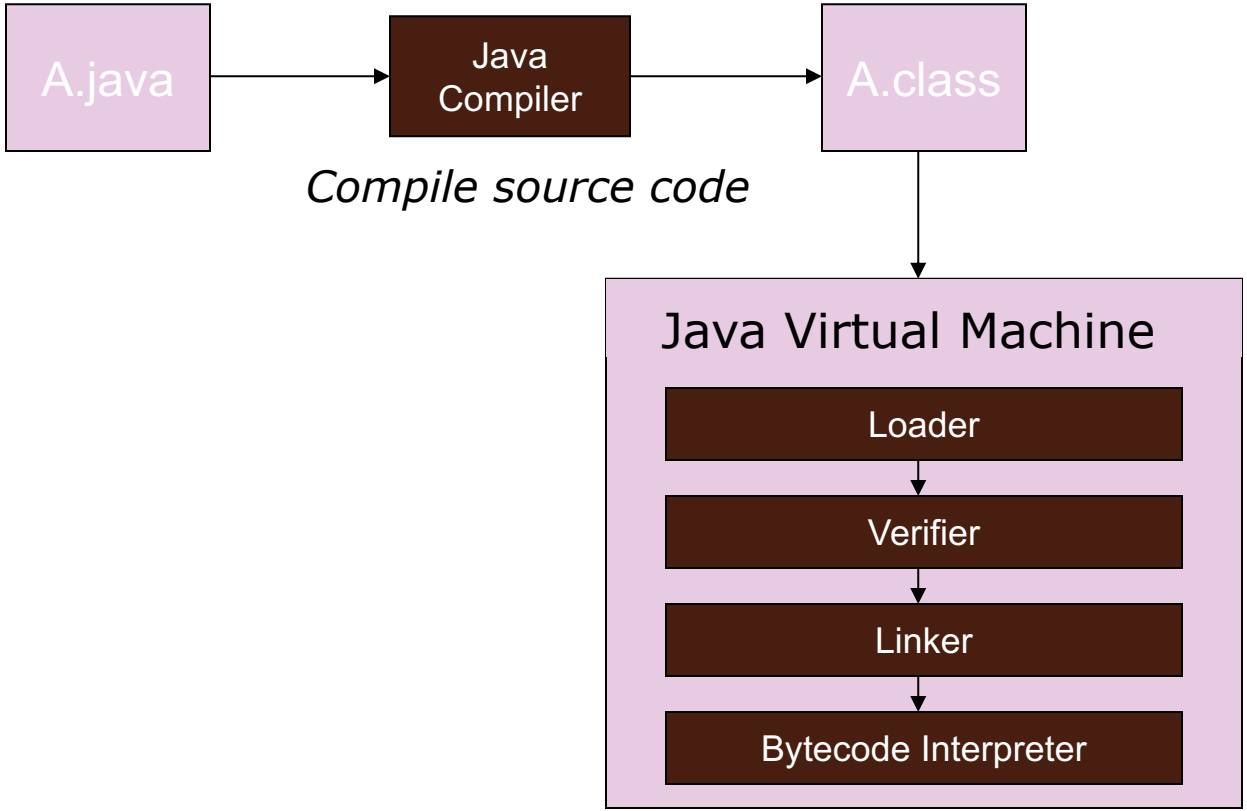
- Stack inspection

# Java Implementation

- Compiler and Virtual Machine
  - Compiler produces bytecode
  - Virtual machine loads classes on demand, verifies bytecode properties, interprets bytecode

- Why this design?
  - Bytecode interpreter “manages” code execution safely
  - Minimize machine-dependent part of implementation

# Java Virtual Machine Architecture



# JVM Linker and Verifier

- Linker
  - Adds compiled class or interface to runtime system
  - Creates static fields and initializes them
  - Resolves names
    - Checks symbolic names and replaces with direct references
- Verifier
  - Check bytecode of a class or interface before loaded
  - Throw exception if error occurs

# Verifier

- Bytecode may not come from standard compiler
  - Evil hacker may write dangerous bytecode
- Verifier checks correctness of bytecode
  - Every instruction must have a valid operation code
  - Every branch instruction must branch to the start of some other instruction, not middle of instruction
  - Every method must have a structurally correct signature
  - Every instruction obeys the Java type discipline
    - Last condition is fairly complicated .

# Bytecode interpreter / JIT

- Standard Java virtual machine interprets instructions
  - Perform run-time checks such as array bounds
  - Possible to compile bytecode class file to native code
- Java programs can call native methods
  - Typically functions written in C

- Just-in-time compiler (JIT)
  - Translate set of bytecodes into native code, including checks
- Ahead-of-time (AOT)
  - Similar principles but prior to loading into runtime system

# Type Safety of Java

- Run-time type checking
  - All casts are checked to make sure type safe
  - All array references are checked to make sure the array index is within the array bounds
  - References are tested to make sure they are not null before they are dereferenced.
- Additional features
  - Automatic garbage collection
  - No pointer arithmetic

If program accesses memory, that memory is allocated to the program and declared with correct type

# Managed code overview

Java programming language

Bytecode execution environment

- Verifier
- Run-time checks
- Memory safety

Permission checking

- Stack inspection

# Managed code overview

- Java programming language
- Bytecode execution environment
  - Verifier
  - Run-time checks
  - Memory safety
- Permission checking
  - Stack inspection

# ANDROID PLATFORM

Platform security model

# Android platform model

Architecture components

- Operating system, runtime environment
- Application sandbox
- Exploit prevention

Permission system

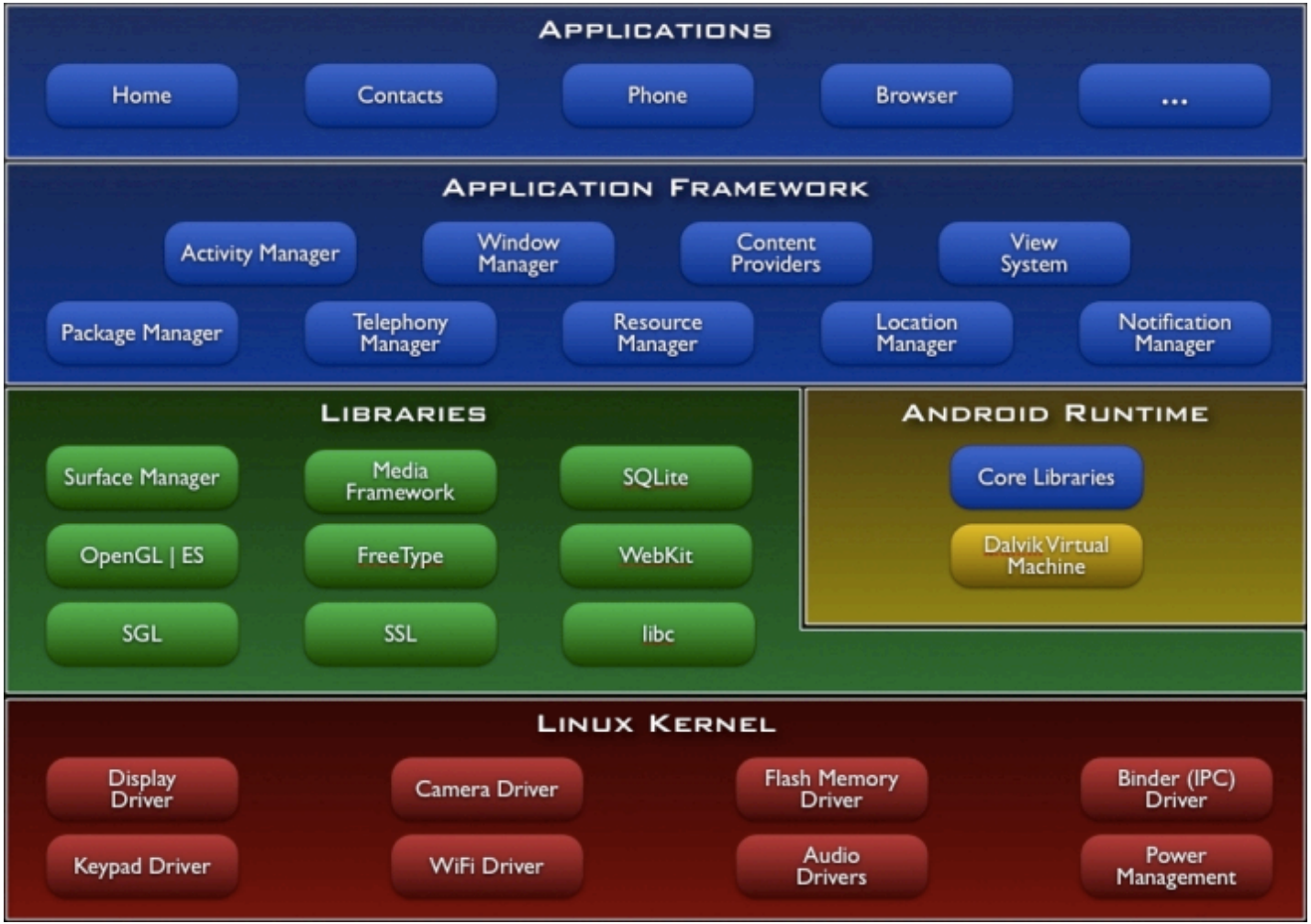
- Granted at install time
- Checked at run time

Inter-app communication

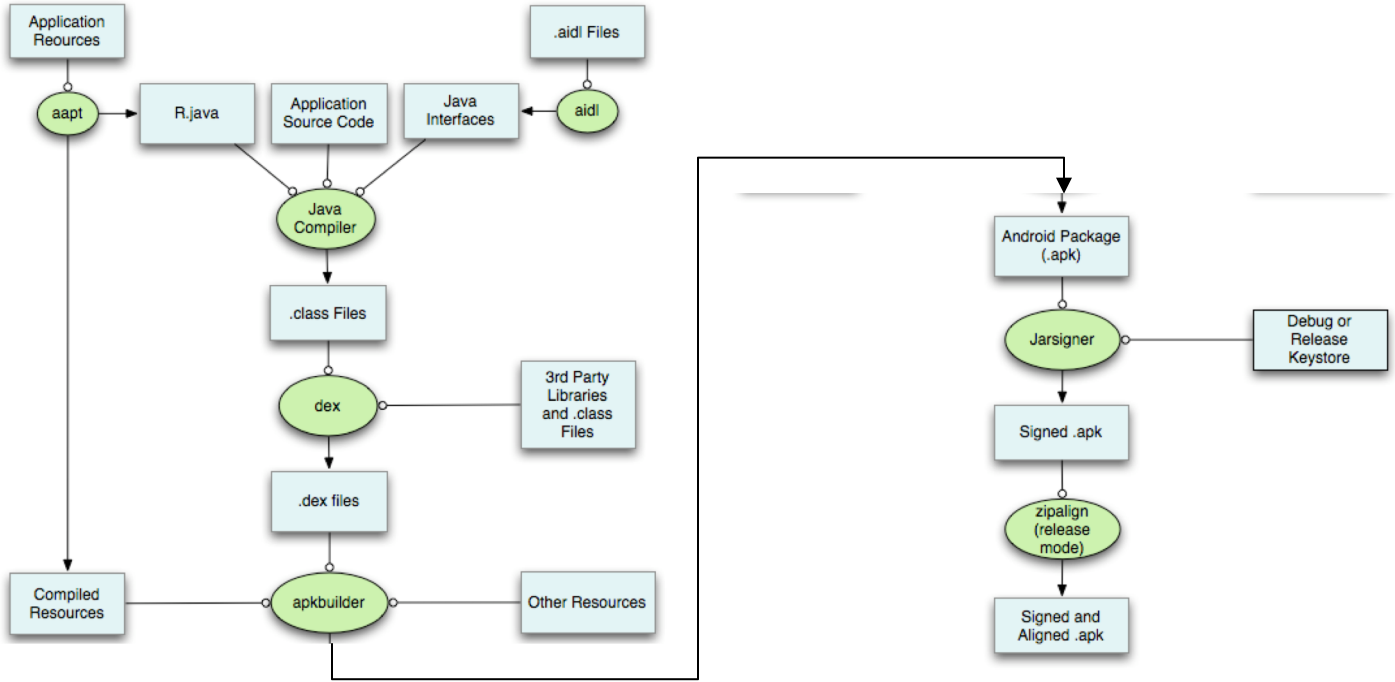
- Intent system
- Permission redelegation (intent input checking)

# Android platform summary

- Linux kernel, browser, SQL-lite database
- Software for secure network communication
  - Open SSL, Bouncy Castle crypto API and Java library
- C language infrastructure
- Java platform for running applications
  - Dalvik bytecode, virtual machine / Android runtime (ART)


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

# Managed code runs in app sandbox




Application development process: source code to bytecode

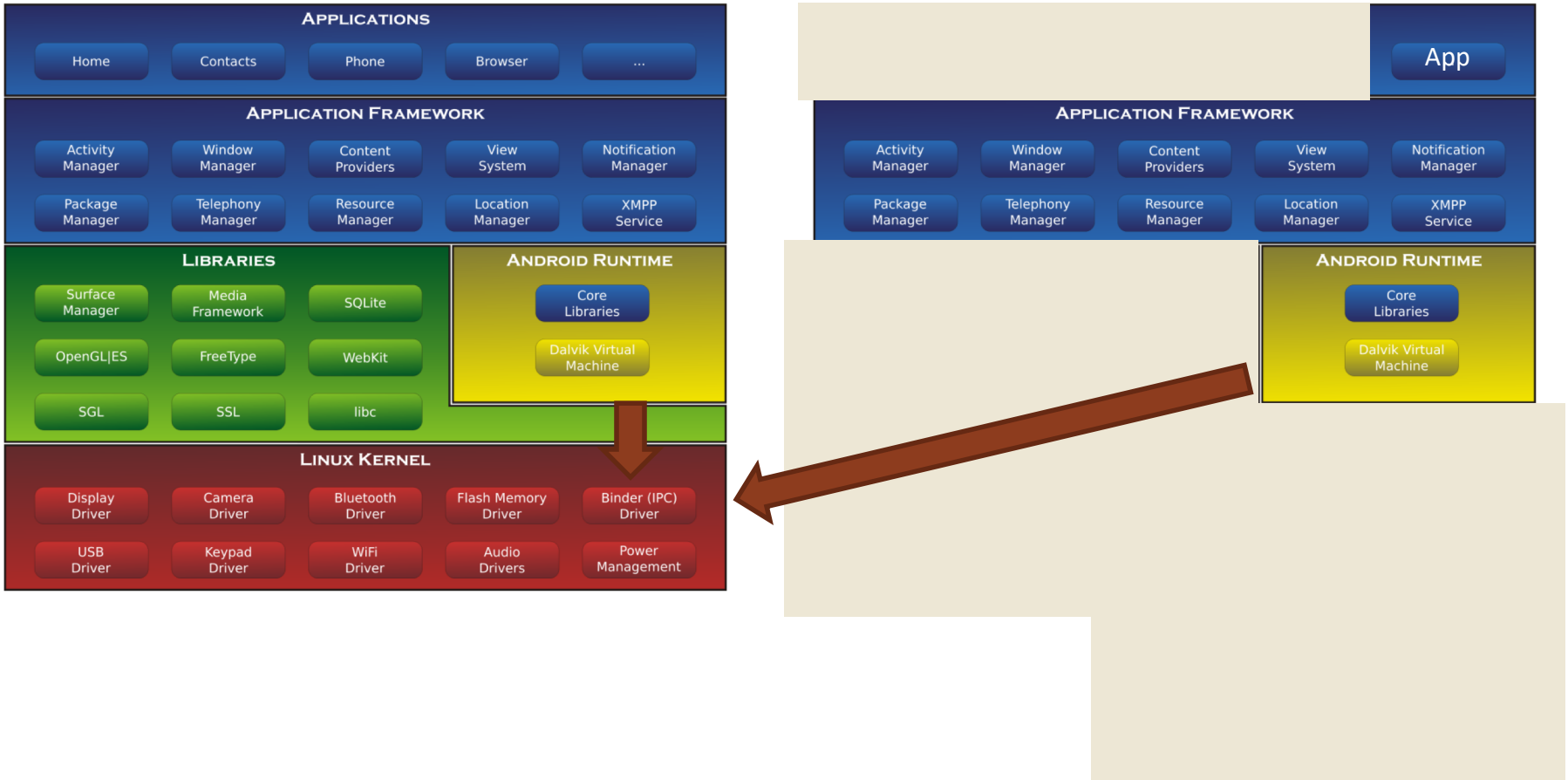
# Security Features

- Isolation
  - Multi-user Linux operating system
  - Each application normally runs as a different user

- Communication between applications
  - May share same Linux user ID
    - Access files from each other
    - May share same Linux process and Dalvik VM
  - Communicate through application framework
    - “Intents,” based on Binder, discussed in a few slides

# Application sandbox

- Application sandbox
  - Each application runs with its UID in its own runtime environment
    - Provides CPU protection, memory protection
    - Only ping, zygote (spawn another process) run as root
- Applications announce permission requirement
  - Create a whitelist model – user grants access at install time
- Communication between applications
  - May share same Linux user ID
    - Access files from each other
    - May share same Linux process and runtime environment
  - Or communicate through application framework
    - “Intents,” reference monitor checks permissions


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service
Libraries
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management
App
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service
Android Runtime
Core Libraries
Dalvik Virtual Machine

# Android platform model

Architecture components

- Operating system, runtime environment
- Application sandbox
- Exploit prevention

Permission system

- Granted at install time
- Checked at run time

Inter-app communication

- Intent system
- Permission redelegation (intent input checking)

# Exploit prevention

- Open source: public review, no obscurity
- Goals
  - Prevent remote attacks, privilege escalation
  - Secure drivers, media codecs, new and custom features
- Overflow prevention
  - ProPolice stack protection
    - First on the ARM architecture
  - Some heap overflow protections
    - Chunk consolidation in DL malloc (from OpenBSD)
- ASLR
  - Avoided in initial release
    - Many pre-linked images for performance
  - Later developed and contributed by Bojinov, Boneh

# dlmalloc (Doug Lea)

- Stores meta data in band
- Heap consolidation attack
  - Heap overflow can overwrite pointers to previous and next unconsolidated chunks
  - Overwriting these pointers allows remote code execution
- Change to improve security
  - Check integrity of forward and backward pointers
    - Simply check that back-forward-back = back,  f-b-f=f
  - Increases the difficulty of heap overflow

# Android platform model

Architecture components

- Operating system, runtime environment
- Application sandbox
- Exploit prevention

Permission system

- Granted at install time
- Checked at run time

Inter-app communication

- Intent system
- Permission redelegation (intent input checking)

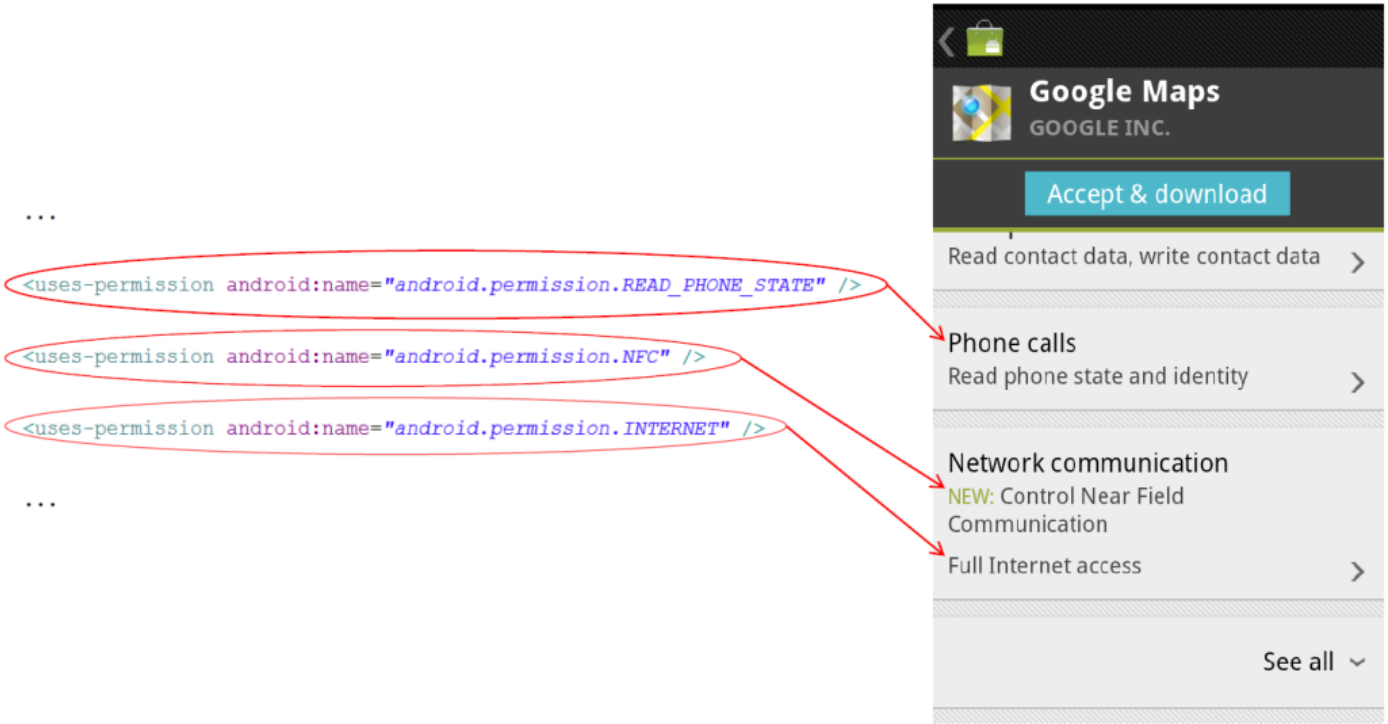
# Android market

- Self-signed apps
- App permissions granted on user installation
- Open market
  - Bad applications may show up on market
  - Shifts focus from remote exploit to privilege escalation

# Android permissions

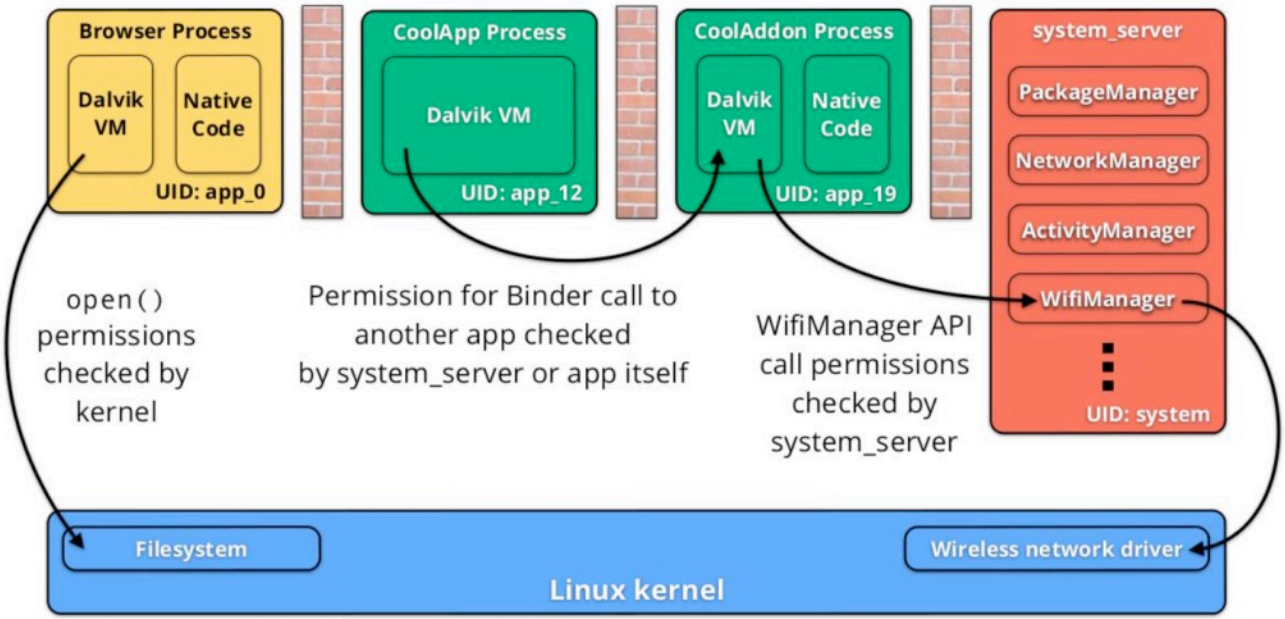
- Example of permissions provided by Android
  - “android.permission.INTERNET”
  - “android.permission.READ_EXTERNAL_STORAGE
  - “android.permission.SEND_SMS”
  - “android.permission.BLUETOOTH”
- Also possible to define custom permissions

# Android permission model




https://www.owasp.org/images/3/3e/Danelon_OWASP_EU_Tour_2013.pdf

# Android permission model




https://www.owasp.org/images/3/3e/Danelon_OWASP_EU_Tour_2013.pdf

# Android platform model

Architecture components

- Operating system, runtime environment
- Application sandbox
- Exploit prevention

Permission system

- Granted at install time
- Checked at run time

Inter-app communication

- Intent system
- Permission redelegation (intent input checking)

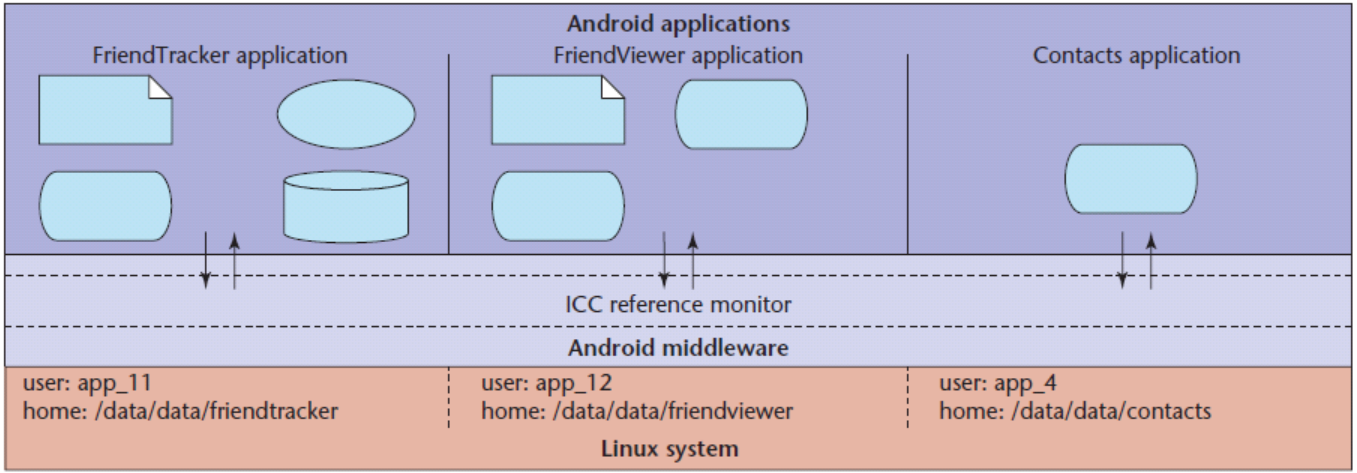
# Application development concepts

- Activity – one-user task
  - Example: scroll through your inbox
  - Email client comprises many activities

- Intents – asynchronous messaging system
  - Fire an intent to switch from one activity to another
  - Example: email app has inbox, compose activity, viewer activity
    - User click on inbox entry fires an intent to the viewer activity, which then allows user to view that email

# Android Intents

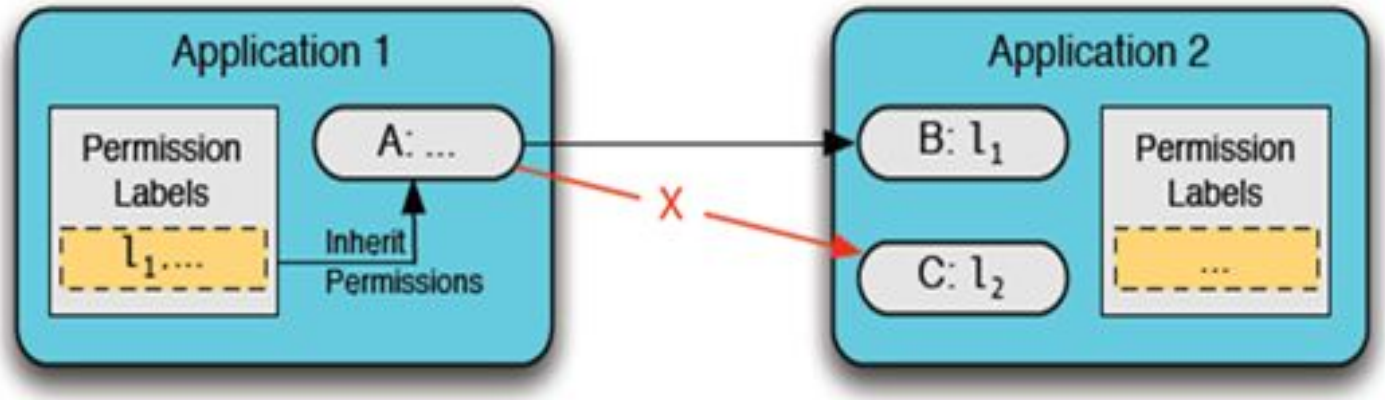
- Intent is a bundle of information, e.g.,
  - action to be taken
  - data to act on
  - category of component to handle the intent
  - instructions on how to launch a target activity
- Routing can be
  - Explicit: delivered only to a specific receiver
  - Implicit: all components that have registered to receive that action will get the message




Layers of security

- Each application executes as its own user identity
- Android middleware has reference monitor that mediates the establishment of inter-component communication (ICC)

Source: Penn State group Android security paper




MAC Policy Enforcement in Android. This is how applications access components of other applications via the reference monitor. Component A can access components B and C if permission labels of application 1 are equal or dominate labels of application 2.

Source: Penn State group, Android security tutorial

# Security issues with intents

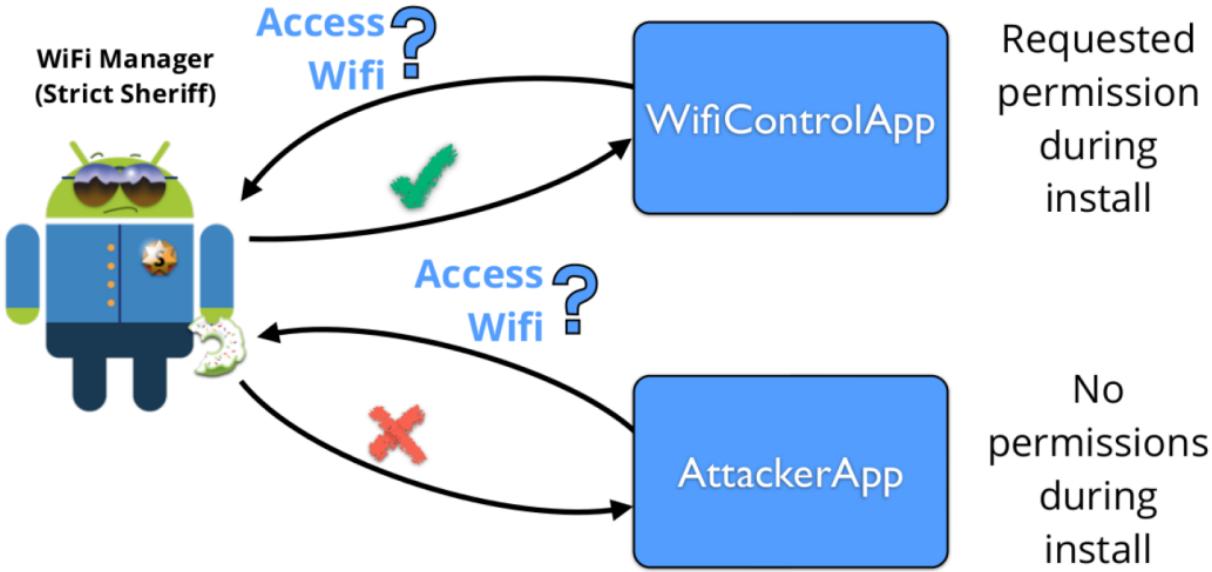
- Sender of an intent may
  - Verify that the recipient has a permission by specifying a permission with the method call
  - Use explicit intents to send the message to a single component

- Receivers must implement appropriate input checking to handle malicious intents

# Attack: Permission redelegation

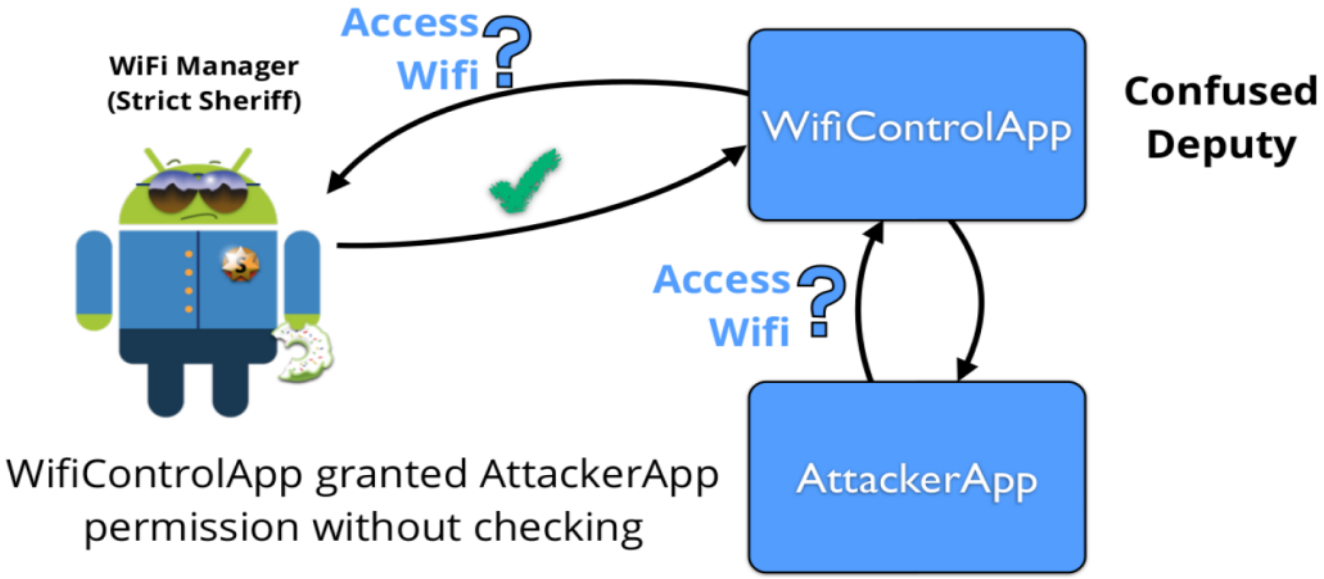
- Idea: an application without a permission gains additional privileges through another application
- Example of the “confused deputy” problem

# Permission redelegation




https://www.owasp.org/images/3/3e/Danelon_OWASP_EU_Tour_2013.pdf

# Permission redelegation



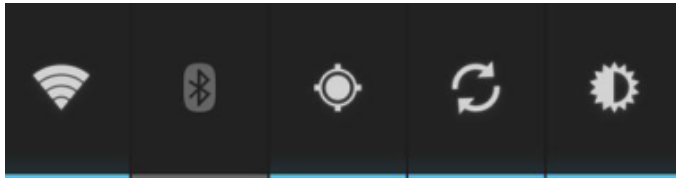

https://www.owasp.org/images/3/3e/Danelon_OWASP_EU_Tour_2013.pdf

# How could this happen?

- App w/ permissions exposes a public interface
- Study in 2011
  - Examine 872 apps
  - 320 of these (37%) have permissions and at least one type of public component
  - Construct attacks using 15 vulnerabilities in 5 apps
- Reference
  - Permission Re-Delegation: Attacks and Defenses, Adrienne Felt, Helen Wang, Alexander Moshchuk, Steven Hanna, Erika Chin, Usenix 2011

# Example: power control widget

- Default widgets provided by Android, present on all devices



- Can change Wi-fi, BT, GPS, Data Sync, Screen Brightness with only one click
- Uses Intent to communicate the event of switching settings
- A malicious app without permissions can send a fake Intent to the Power Control Widget, simulating click to switch settings

https://www.owasp.org/images/3/3e/Danelon_OWASP_EU_Tour_2013.pdf

# Vulnerable versions (in red)

| Version | Codename | API | Distribution |
|---|---|---|---|
| **1.6** | **Donut** | **4** | **0.10%** |
| **2.1** | **Eclair** | **7** | **1.50%** |
| **2.2** | **Froyo** | **8** | **3.20%** |
| 2.3 - 2.3.2 | Gingerbread | 9 | 0.10% |
| 2.3.3 - 2.3.7 | | 10 | 36.40% |
| 3.2 | Honeycomb | 13 | 0.10% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 25.60% |
| 4.1.x | **Jelly Bean** | 16 | 29.00% |
| **4.2.x** | | **17** | **4.00%** |

Principle of least privilege helps but is not solution by itself

Apps with permissions need to manage security

https://www.owasp.org/images/3/3e/Danelon_OWASP_EU_Tour_2013.pdf

# Android platform model

Architecture components

- Operating system, runtime environment
- Application sandbox
- Exploit prevention

Permission system

- Granted at install time
- Checked at run time

Inter-app communication

- Intent system
- Permission redelegation (intent input checking)

# ANDROID PLATFORM

Mobile web apps

# Outline

- Mobile web apps
  - Use WebView Java objects, implemented based on WebKit browser
  - “JavaScript bridge” lets web content use Java objects exported by app

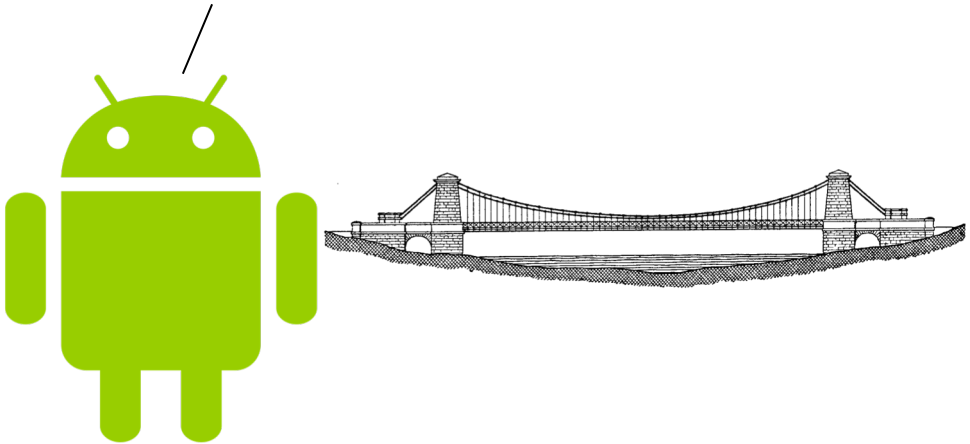
- Security problems
  - WebView does not isolate bridge access by frame or origin
  - App environment may leak sensitive web information in URLs
  - WebView does not provide security indicators
  - …

# Mobile Web Apps

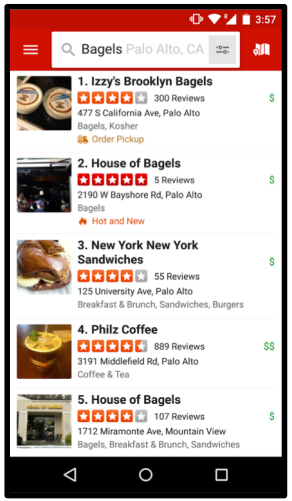
Mobile web app: embeds a fully functional web browser as a UI element

# JavaScript Bridge

```
Obj foo = new Object();
addJavascriptInterface(foo, ‘f’);
```

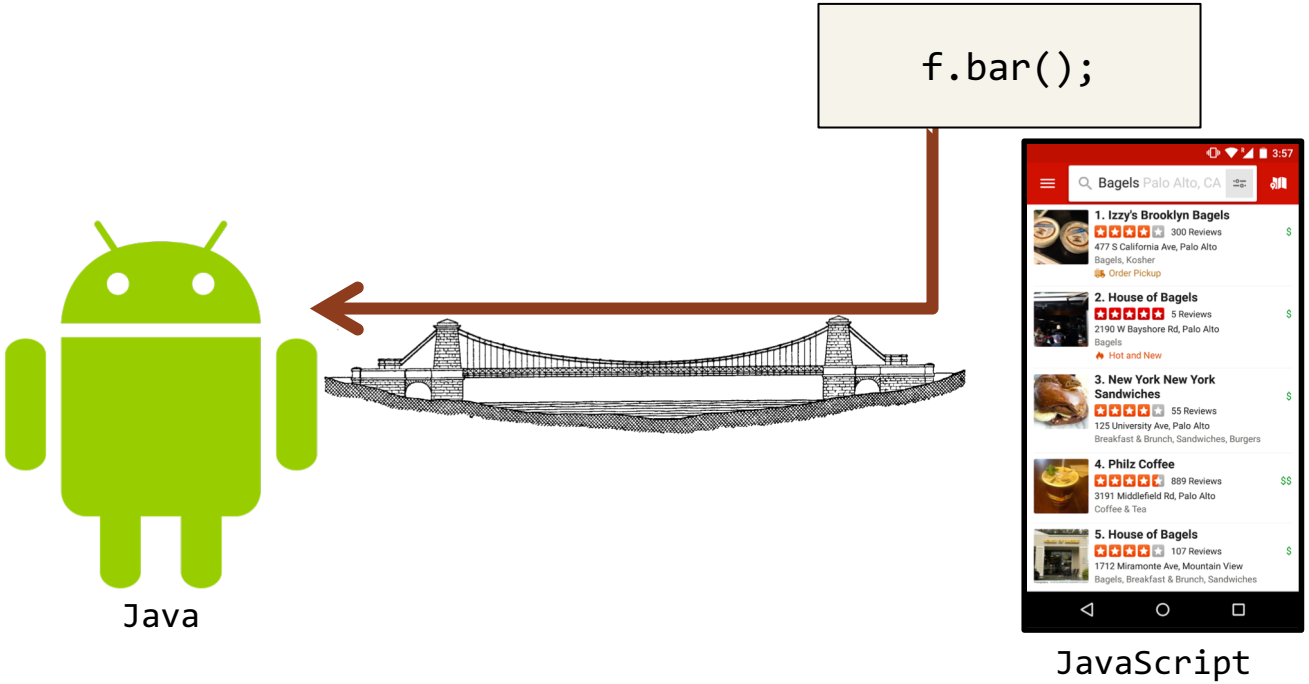


Java




JavaScript

# JavaScript Bridge

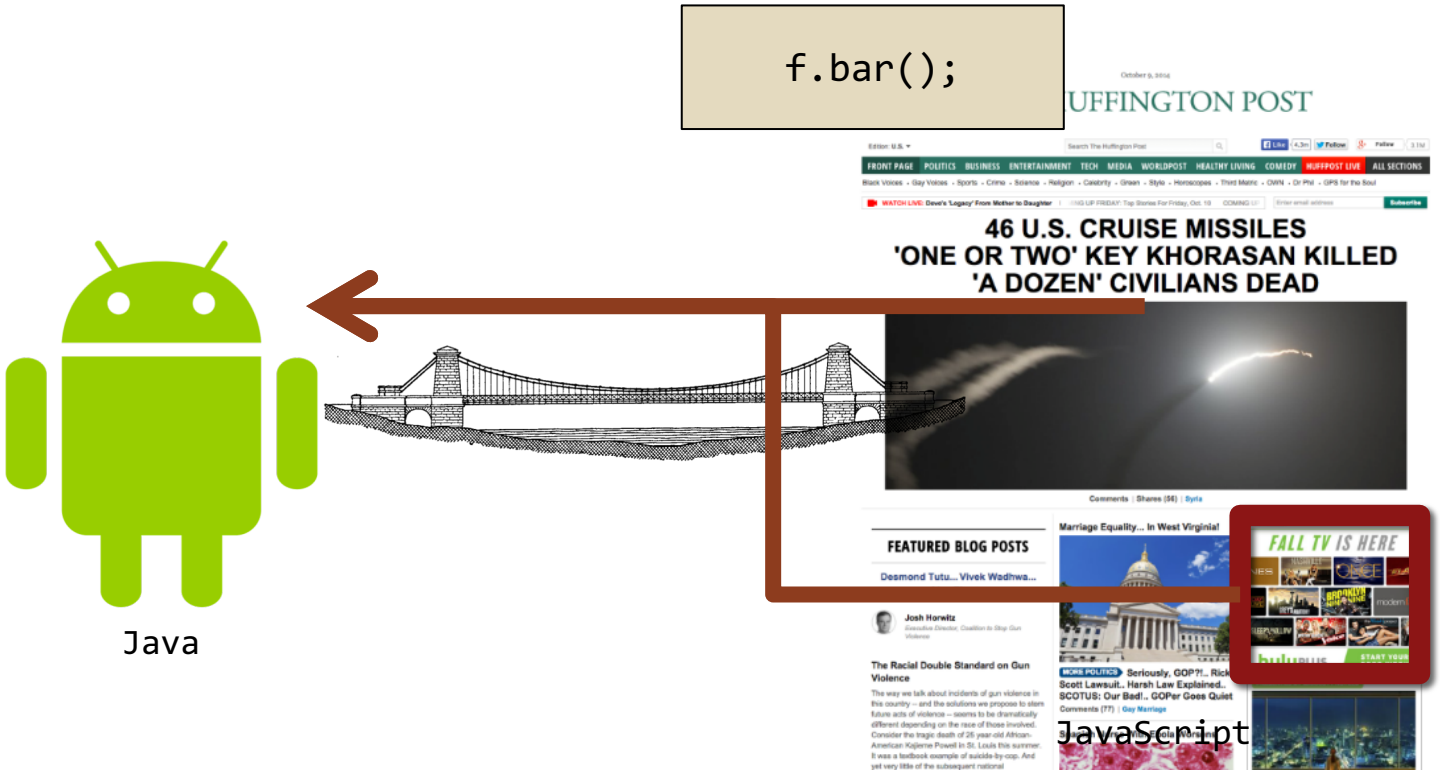


# Security Concerns

Who can access the bridge?

- Everyone

# No origin distinction in WebView calls



# Analysis of Public Apps

- How many mobile web apps?
- How many use JavaScript Bridge?
- How many vulnerable?

# Experimental Results

- 737,828 free apps from Google Play (Oct ’13)
- 563,109 apps embed a browser
- 219,404 use the JavaScript Bridge
- 107,974 have at least one security violation

# Most significant vulnerabilities

1. Loading untrusted web content

2. Leaking URLs to foreign apps

3. Exposing state changing navigation to foreign apps

# Loading untrusted web content

*“You should restrict the web-pages that can load inside your WebView with a whitelist.”*

*- Facebook*

*“…only loading content from trusted sources into WebView will help protect users.”*

*- Adrian Ludwig, Google*

# Forms of navigation

```
// In app code
myWebView.loadUrl(“foo.com”);

<!-- In HTML -->
<a href=“foo.com”>click!</a>

<!-- More HTML -->
<iframe src=“foo.com”/>

// In JavaScript
window.location = “foo.com”;
```

# Implementing navigation whitelist

```
public boolean shouldOverrideUrlLoading(
      WebView view, String url){

   // False -> Load URL in WebView
   // True  -> Prevent the URL load


}
```

```
public boolean shouldOverrideUrlLoading(
      WebView view, String url){

   String host = new URL(url).getHost();
   if(host.equals(“stanford.edu”))
      return false;
   log(“Overrode URL: ” + url);
   return true;
}
```

# Reach Untrusted Content?

- 40,084 apps with full URLs and use JavaScript Bridge
- 13,683 apps (34%) can reach untrusted content

# Exposing sensitive information in URLs

- Android apps communicate using intents
  - An implicit intent is delivered to any app whose filter matches
  - An intent filter can declare zero or more <data> elements, such as
    - mimeType - e.g., android:mimeType="video/mpeg“
    - scheme - e.g., android:scheme="http"

- When a WebView loads a page, an intent is sent to the app
  - Another app can register a filter that might match this intent
  - If the URL contains sensitive information, this information can be stolen

# Example

OAuth protocol for browser-based web authentication

- Used by Google, Facebook, LinkedIn and other identity providers
- In some configurations, may return a session token as part of a URL

Mobile app developers may try to use OAuth through WebView

- A form of session token is returned as part of a URL
- Delivered through an implicit intent
- May reach any app with filter that specifies protocol scheme my_oauth

Malicious app may steal a session token from a vulnerable app

- Malicious app registers an implicit intent with scheme my_oauth
- Waits for a URL containing the form of session token returned by OAuth.

# Handling SSL Errors

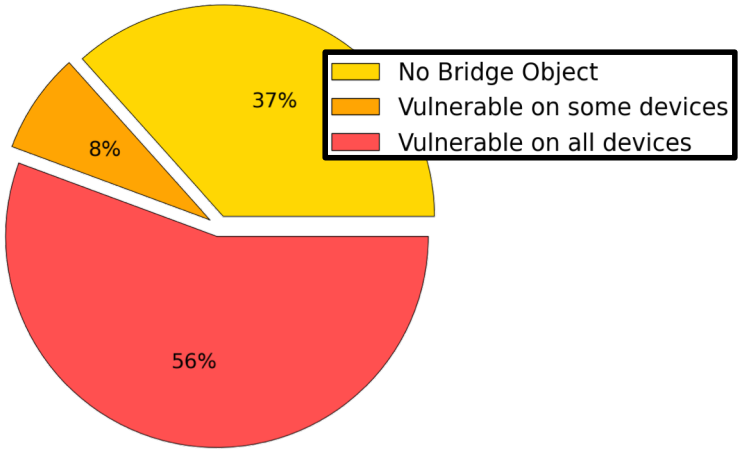
`onReceivedSslError`

1. `handler.proceed()`
2. `handler.cancel()`
3. `view.loadUrl(...)`

# Mishandling SSL Errors

117,974 apps implement `onReceivedSslError`

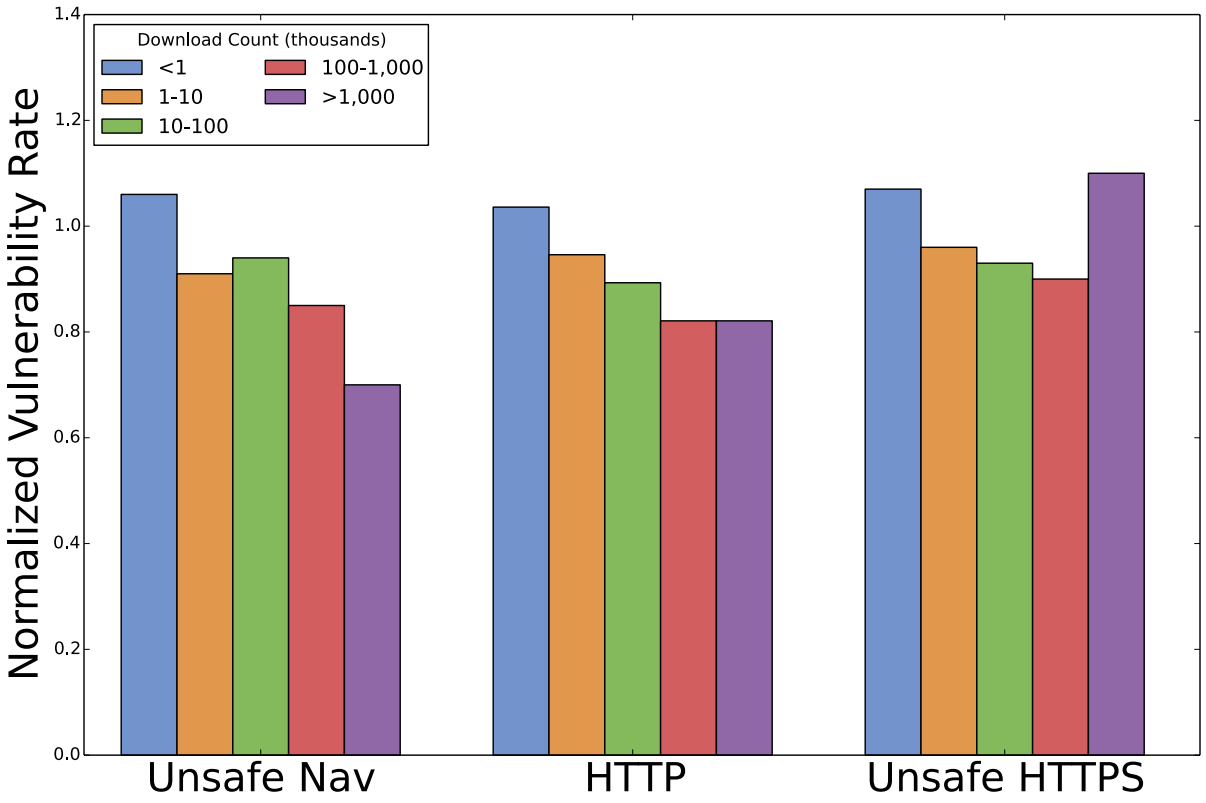
29,652 apps (25%) **must** ignore errors



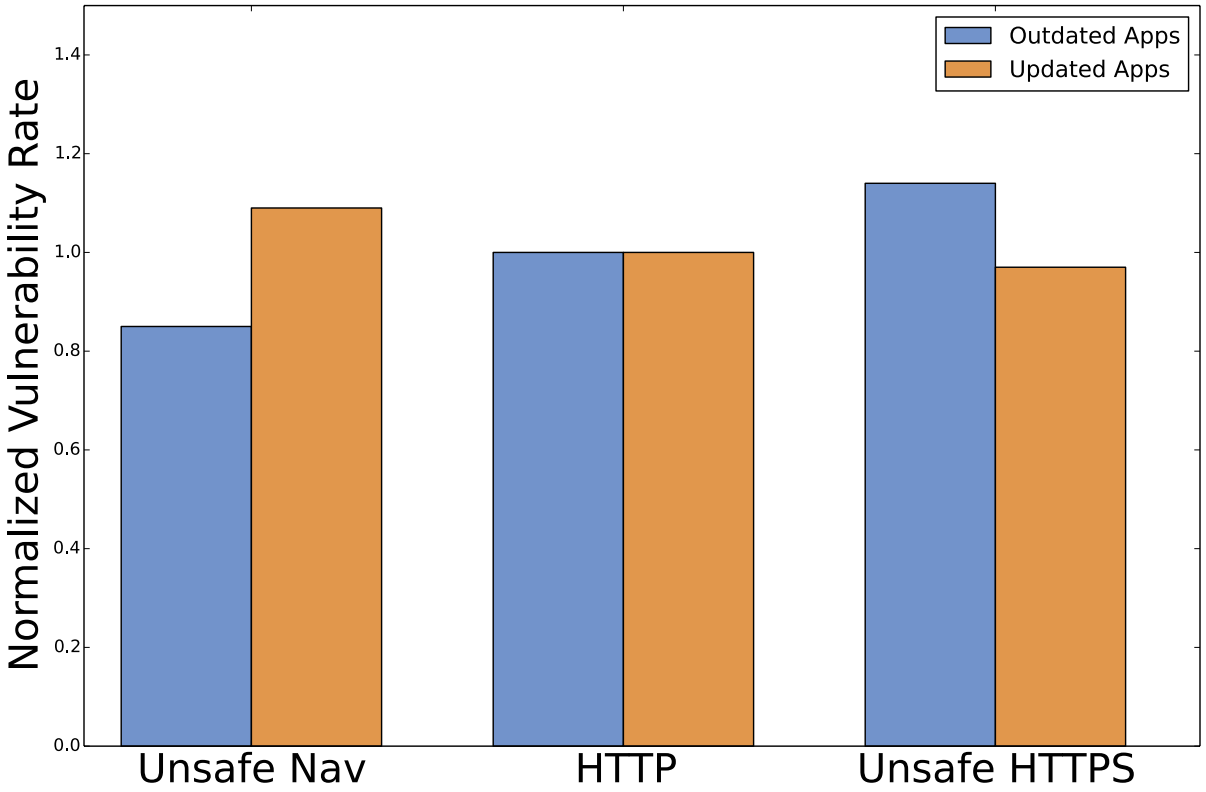
# Primary results

| Vulnerability | % Relevant | % Vulnerable |
|---|---|---|
| Unsafe Nav | 15 | 34 |
| HTTP | 40 | 56 |
| Unsafe HTTPS | 27 | 29 |

# Popularity


Download Count (thousands)
<1
1-10
10-100
100-1,000
>1,000
Normalized Vulnerability Rate
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Unsafe Nav
HTTP
Unsafe HTTPS

# Outdated Apps



# Libraries

29%
unsafe nav

51%
HTTP

53%
unsafe HTTPS

# Additional security issues

Based on 998,286 free web apps from June 2014

| Mobile Web App Feature | % Apps |
|---|---|
| JavaScript Enabled | 97 |
| JavaScript Bridge | 36 |
| shouldOverrideUrlLoading | 94 |
| shouldInterceptRequest | 47 |
| onReceivedSslError | 27 |
| postUrl | 2 |
| Custom URL Patterns | 10 |

| Vuln | % Relevant | % Vulnerable |
|---|---|---|
| Unsafe Navigation | 15 | 34 |
| Unsafe Retrieval | 40 | 56 |
| Unsafe SSL | 27 | 29 |
| Exposed POST | 2 | 7 |
| Leaky URL | 10 | 16 |

# Summary

- Mobile web apps
  - Use WebView Java objects, implemented based on WebKit browser
  - “JavaScript bridge” lets web content use Java objects exported by app

- Security problems
  - WebView does not isolate bridge access by frame or origin
  - App environment may leak sensitive web information in URLs
  - WebView does not provide security indicators
  - …
  - Many browser security mechanism are not automatically provided by WebView

# ANDROID PLATFORM

Target fragmentation

# Summary

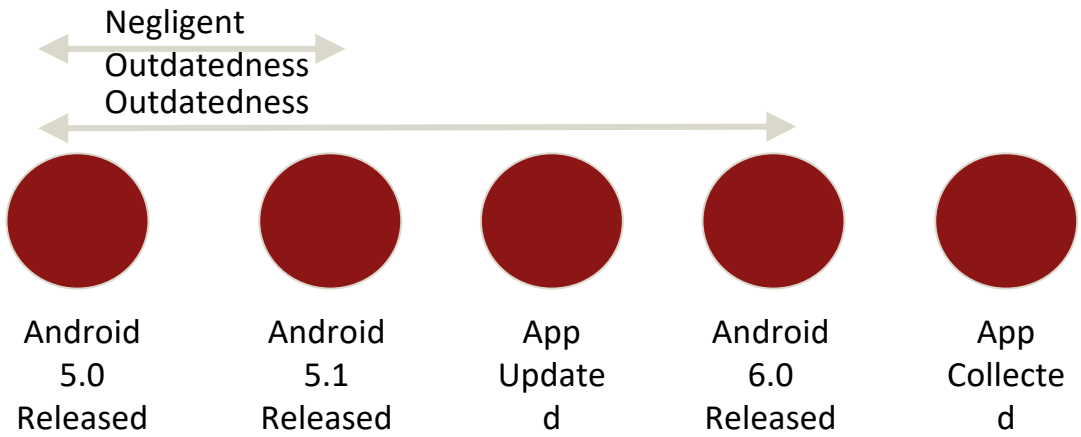
- Android apps can run using outdated OS behavior
  - The large majority of Android apps do this
  - Including popular and well maintained apps
- Outdated security code invisibly permeates the app ecosystem
  - “Patched” security vulnerabilities still exist in the wild
  - “Risky by default” behavior is widespread
-

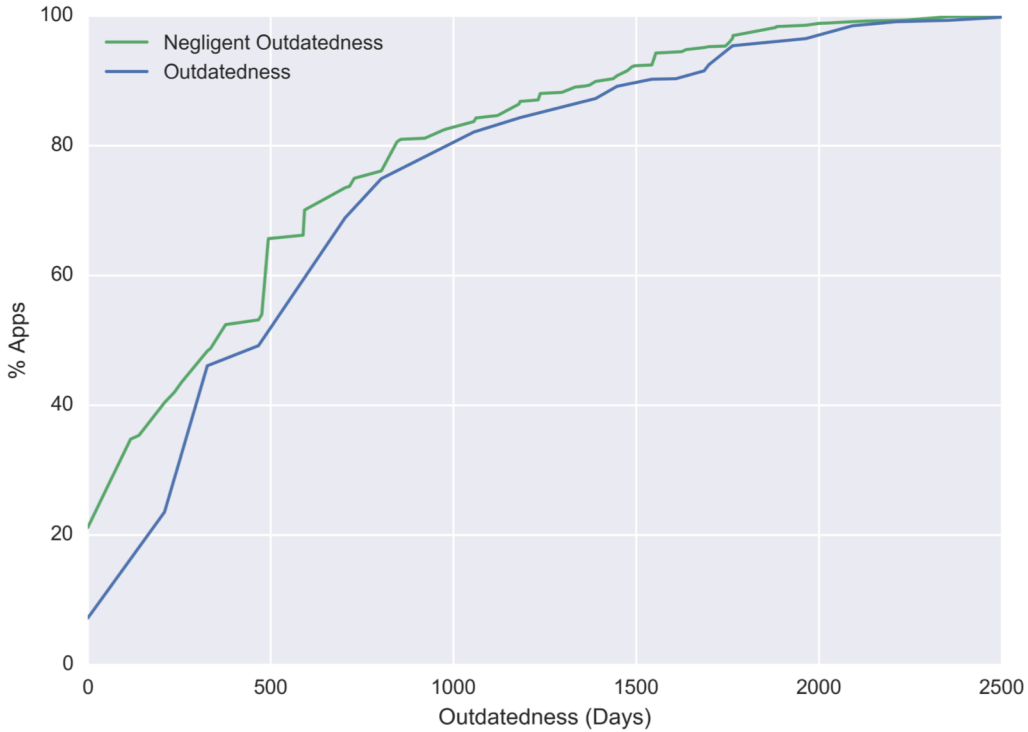
*“If the device is running Android 6.0 or higher… [the app] must request each dangerous permission that it needs while the app is running.*

- Android Developer Reference

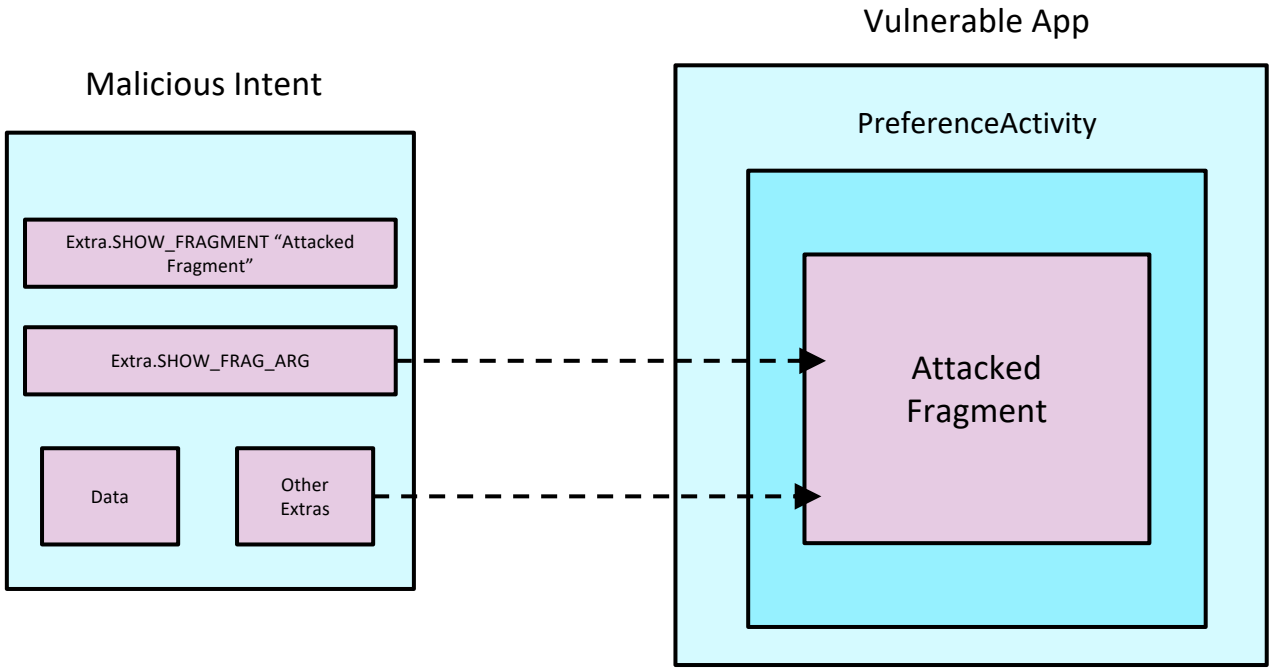
*“If the device is running Android 6.0 or higher **and your app's target SDK is 6.0 or higher** [the app] must request each dangerous permission that it needs while the app is running.*

- Android Developer Reference


Negligent
Outdatedness
Outdatedness
Android
5.0
Released
Android
5.1
Released
App
Update
d
Android
6.0
Released
App
Collecte
d


Negligent Outdatedness
Outdatedness
% Apps
Outdatedness (Days)
0
20
40
60
80
100
0
500
1000
1500
2000
2500

# Fragment Injection




securityintelligence.com/new-vulnerability-android-framework-fragment-injection/

# Fragment Injection

- Fixed in Android 4.4
- Developers implement isValidFragment to authorize fragments

```
// Put this in your app
protected boolean isValidFragment(String fName){
      return MyFrag.class.getName().equals(fName);
}
```

# Fragment Injection

›

Vulnerable if:

- Targets 4.3 or lower (31%)
- Some class inherits from PreferenceActivity (4.8%)
- That class is exported (1.1%)
- That class does not override isValidFragment (0.55%)

4.2% of apps vulnerable if no fix was ever implemented

# Mixed Content in WebView

- Major web browsers block Mixed Content
- In Android 5.0, WebViews block Mixed Content by default
- Can override default with setMixedContentMode()

# SOP for file:// URLs in WebView

- Android 4.1 separate file:// URLs are treated as unique origins
- Can override with setAllowFileAccessFromFileURLs()

# Summary

- Android apps can run using outdated OS behavior
  - The large majority of Android apps do this
  - Including popular and well maintained apps
- Outdated security code invisibly permeates the app ecosystem
  - “Patched” security vulnerabilities still exist in the wild
  - “Risky by default” behavior is widespread
-

# Two lectures on mobile security

- Introduction: platforms and trends
- Threat categories
  - Physical, platform malware, malicious apps
- Defense against physical theft
- Malware threats
- System architecture and defenses
  - Apple iOS security features and app security model
  - Android security features and app security model
- Security app development
  - WebView – secure app and web interface dev
  - Device fragmentation

Thurs

Tues

# Comparison: iOS vs Android

- App approval process
  - Android apps from open app store
  - iOS vendor-controlled store of vetted apps
- Application permissions
  - Android permission based on install-time manifest
  - All iOS apps have same set of “sandbox” privileges
- App programming language
  - Android apps written in Java; no buffer overflow…
  - iOS apps written in Objective-C

# Comparison

| | iOS | Android | Windows |
|---|---|---|---|
| Unix | x | x | |
| Windows | | | |
| Open market | | x | |
| Closed market | x | | |
| Vendor signed | x | | |
| Self-signed | | x | |
| User approval of permissions | | x | |
| Managed code | | x | |
| Native code | x | | |

# Comparison

| | iOS | Android | Windows |
|---|---|---|---|
| Unix | x | x | |
| Windows | | | x |
| Open market | | x | |
| Closed market | x | | x |
| Vendor signed | x | | |
| Self-signed | | x | x |
| User approval of permissions | | x | 7-> 8 |
| Managed code | | x | x |
| Native code | x | | |

# Two lectures on mobile security

- Introduction: platforms and trends
- Threat categories
  - Physical, platform malware, malicious apps
- Defense against physical theft
- Malware threats
- System architecture and defenses
  - Apple iOS security features and app security model
  - Android security features and app security model
- Security app development
  - WebView – secure app and web interface dev
  - Device fragmentation

Thurs

Tues