Mobile Security

Course Overview

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

Welcome

Course objectives:

- Understand security trade-offs of most popular mobile platforms
- Understand privacy and security threats to mobile applications and how to defend against them
- Understand how to design and implement secure mobile applications
- Understand how to manage mobile applications and devices in an enterprise environment

Stanford | Center for Professional Development

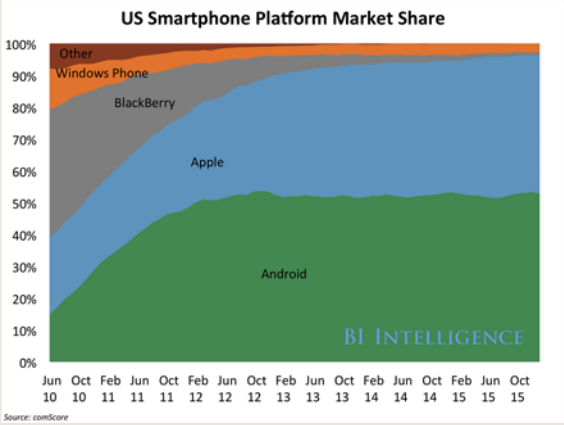
Mobile Security

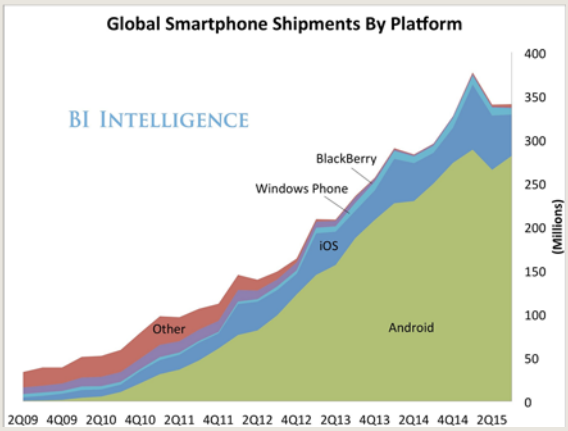
Markets

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

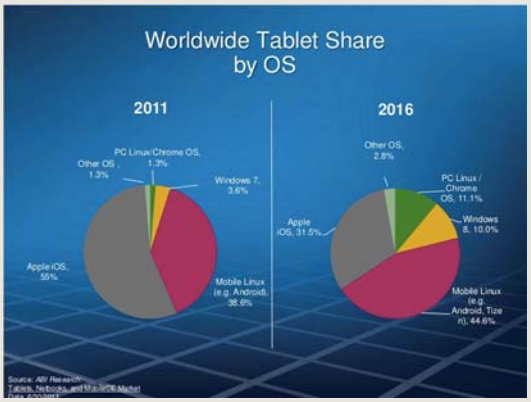
Smartphone Market Share: US & Global





Stanford | Center for Professional Development

Tablets and Watches



Top Smartwatch Operating Systems with Shipments, Market Share and 5-Year CAGR Growth
(Units in Millions)

SmartWatch OS	2016 Units (M)	2016 Share	2020 Units (M)	2020 Share	2016 - 2020 CAGR
watchOS	14.0	49.4%	31.0	37.6%	22%
Android Wear	6.1	21.4%	28.8	35.0%	48%
RTOS	1.4	5.0%	8.3	10.1%	56%
Tizen	3.2	11.3%	5.4	6.6%	14%
Android	1.0	3.6%	4.3	5.2%	44%
Linux	0.6	2.3%	2.3	2.8%	37%
Pebble OS	2.0	7.0%	2.2	2.7%	3%
Total	**28.3**	**100.0%**	**82.5**	**100.0%**	**31%**
Android Wear + Android	7.1	25.0%	33.2	40.2%	47%

Source: IDC Worldwide Quarterly Wearable Device Tracker, March 17, 2015

Mobile Security

Types of Apps & Overview of Mobile OS Security Architecture

Stanford Advanced Computer Security Certificate

Types of Mobile Applications

Web Apps

Various Web UI techniques: JSF, GWT, HTML5+REST, etc... Quick to develop with traditional web skills but less use of device capabilities.

Hybrid Apps

HTML5 wrapped in native shell. Provisioned and accessed as a native app. Leverages web skills and HTML5 with appearance of native app.

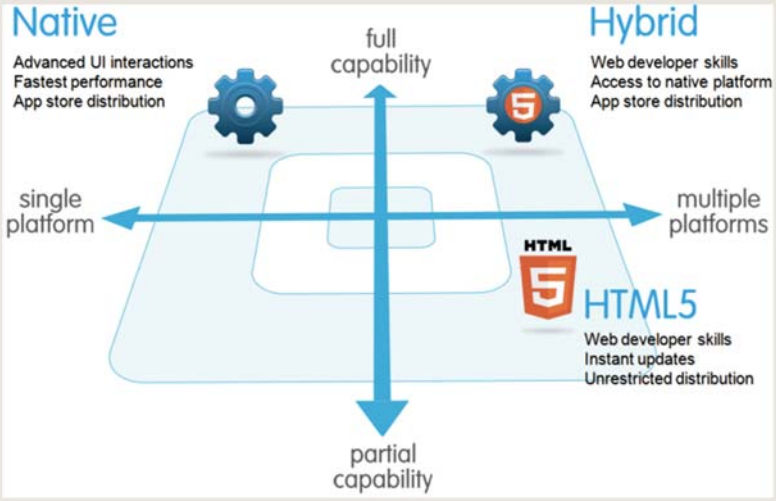
Hybrid Apps +

Hybrid app tweaked with native code to access device capabilities. Requires more time and expertise than Hybrid Apps

Native App

Platform-specific written with OS specific SDK to deliver fullest capabilities. Requires additional expertise and time

Trade-offs of application types



https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

iOS vs Android: Basic Comparison

	Android	iOS
Native Development Language	Java	Objective C
Base Operating System	Linux	Mach
Sandboxing	Each App is a User / DAC	Seatbelt / MAC
Memory Protection	DEP (>Froyo), ASLR	DEP, ASLR
Interprocess Communication	Intents, Content Providers, Binder	URLs & App Extensions
Encryption & Key Management Support	Third-party libraries available	Data Protection API & Keychain part of OS
App Store Market	Open; Apps ask for permissions; Bouncer	Closed; Apple App Store; vetting proprietary; apps must be signed

Stanford | Center for Professional Development

Mobile Security

Threats to Mobile Applications

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

Threats

Privacy: data leakage, identifier leakage, …

Security: phishing, malware & drive-bys, malicious intents (e.g, permission escalation), …

Where to test/protect against threats

	Android	iOS
Data Leakage / Privacy		Applications/Data directory
Misconfiguration		App Bundles
Implementation Vulnerabilities	Web Vulns (XSS, SQL Injection, etc)	Web Vulns & Traditional Vulns (Buffer/Integer Overflows, Format Strings)
Phishing	Activities vulnerable to malicious intents	
Malware	Custom app stores (e.g., China)	

Mobile Security

Mobile Malware

Stanford Advanced Computer Security Certificate

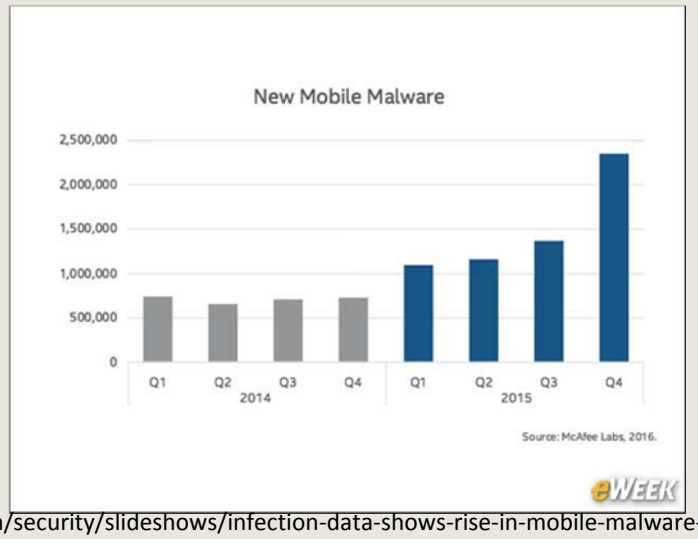
Stanford | Center for Professional Development

Mobile Malware Examples

- DroidDream (Android)
  - Over 58 apps uploaded to Google app market
  - Conducts data theft; send credentials to attackers
- Ikee (iOS)
  - Worm capabilities (targeted default ssh pwd)
  - Worked only on jailbroken phones with ssh installed
- Zitmo (Symbian,BlackBerry,Windows,Android)
  - Propagates via SMS; claims to install a “security certificate”
  - Captures info from SMS; aimed at defeating 2-factor auth
  - Works with Zeus botnet; timed with user PC infection

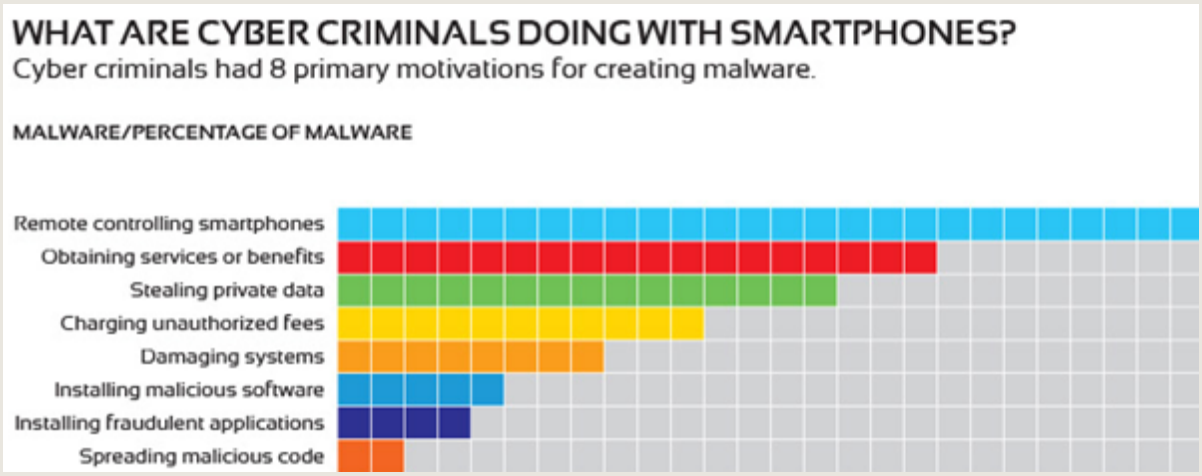
Stanford | Center for Professional Development

Mobile Malware Growth



http://www.eweek.com/security/slideshows/infection-data-shows-rise-in-mobile-malware-development.html

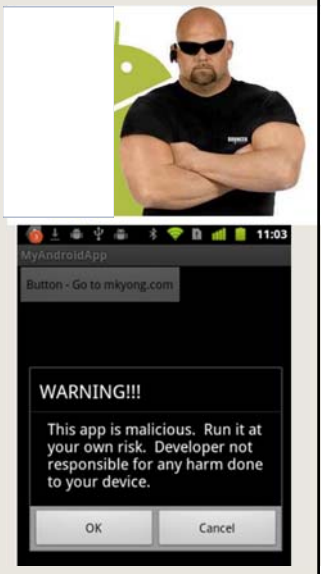
What are cybercriminals doing with mobile malware?



App Store Security: Google Bouncer

- Dynamic analysis / runs apps submitted to Google Play before posting on the app store
- Emulated Android environment; runs for 5 minutes
- Oberheide/Miller fingerprinted Bouncer environment: Linux/QEMU, from Google infrastructure, allows network access, with contacts / emulated SD card
- Bouncer emulates inputs, clicks, etc

https://jon.oberheide.org/files/summercon12-bouncer.pdf



Mobile Security

Case Studies

Stanford Advanced Computer Security Certificate

Things that have gone wrong (FTC cases)

1) FTC vs. Fandango / Credit Karma. SSL Certificate Verification Disabled. Not using TLS correctly (e.g., don't enable allowsAnyHTTPSCertificateForHost on iOS); secure by default in iOS 9. (2014)

2) FTC vs. BabyBus. COPPA violation. Collection of geo data about children without parental authorization. (2014)

3) FTC vs. Vulcun. Bypassing Android permissions. (2016)

More at https://www.ftc.gov/news-events/media-resources/mobile-technology

OWASP Mobile Top 10

M1: Weak Server Side Controls
M2: Insecure Data Storage
M3: Insufficient Transport Layer Protection
M4: Unintended Data Leakage
M5: Poor Authorization and Authentication
M6: Broken Cryptography
M7: Client Side Injection
M8: Security Decisions Via Untrusted Inputs
M9: Improper Session Handling
M10: Lack of Binary Protections

Summary

- Markets: Smartphones, Tablets, Watches
- Application architectures (native, hybrid, web-based)
- Threats to privacy and security
- FTC case studies: things that have gone wrong
- OWASP Mobile Top 10
- Ecosystem-level defenses

Stanford | Center for Professional Development

Mobile Security

Unix Security Review

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

Unix security review

Principle of Least Privilege

Unix security fundamentals

- File system permissions
- Process isolation, UID, setuid

Least privilege examples

- Qmail
- Android app separation

Principle of Least Privilege

Assume compartmentalization and isolation

- Separate the system into isolated compartments
- Limit interaction between compartments

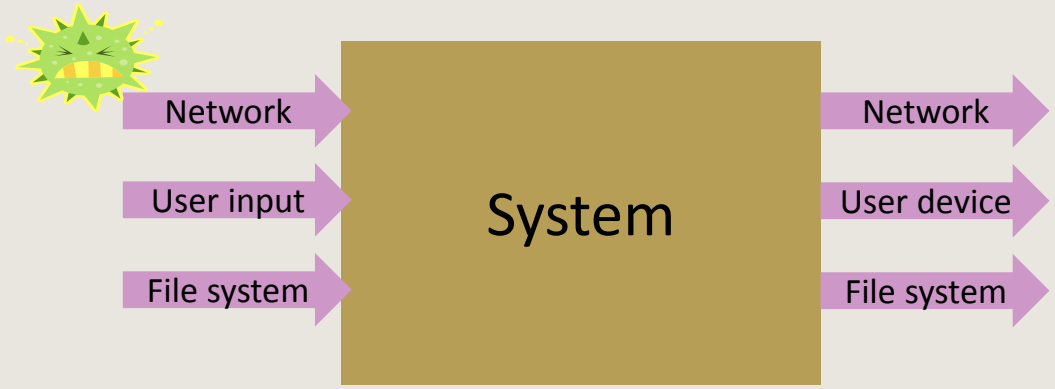
What's a privilege?

- Ability to access or modify a resource

Principle of Least Privilege

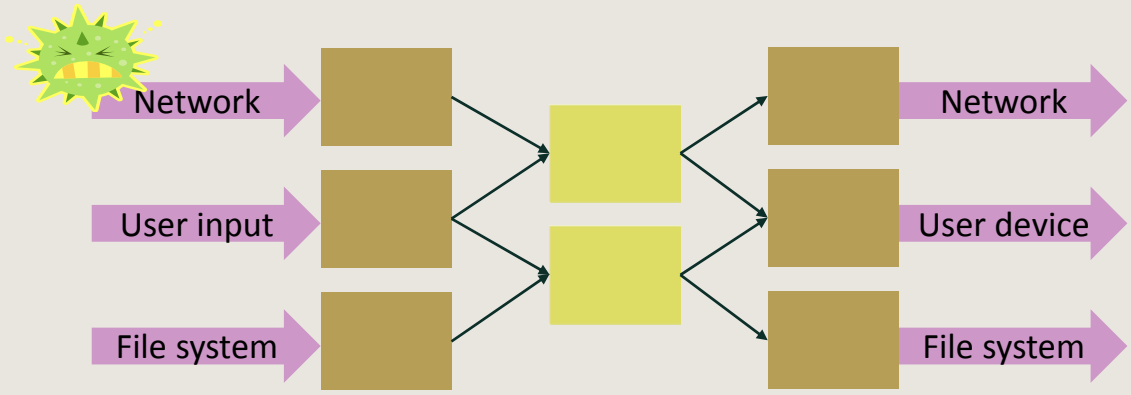
- A system module should only have the minimal privileges needed for its intended purposes

Monolithic design



Stanford | Center for Professional Development

Component design



Stanford | Center for Professional Development

Principle of Least Privilege

- A system module should only have the minimal privileges needed for its intended purposes

Stanford | Center for Professional Development

Mobile Security

Unix Security Fundamentals

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

Unix security review

Principle of Least Privilege

Unix security fundamentals

- File system permissions
- Process isolation, UID, setuid

Least privilege examples

- Qmail
- Android app separation

Unix access control

Process has user id

- Inherit from creating process
- Process can change id
  - Restricted set of options
- Special “root” id
  - All access allowed

File has access control list (ACL)

- Grants permission to user ids
- Owner, group, other

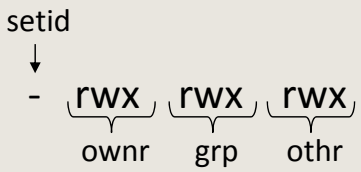
	File 1	File 2	...
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
...			
User m	Read	write	write

Unix file access control list

Each file has owner and group

Permissions set by owner

- Read, write, execute
- Owner, group, other
- Represented by vector of four octal values



Only owner, root can change permissions

- This privilege cannot be delegated or shared

Setid bits – Discuss in a few slides

Process effective user id (EUID)

Each process has three Ids (+ more under Linux)

- Real user ID (RUID)
  - same as the user ID of parent (unless changed)
  - used to determine which user started the process
- Effective user ID (EUID)
  - from set user ID bit on the file being executed, or sys call
  - determines the permissions for process
    - file access and port binding
- Saved user ID (SUID)
  - So previous EUID can be restored

Real group ID, effective group ID, used similarly

Process Operations and IDs

Fork and Exec

- Inherit three IDs, except exec of file with setuid bit

Setuid system call

- seteuid(newid) can set EUID to
  - Real ID or saved ID, regardless of current EUID
  - Any ID, if EUID=0

Details are actually more complicated

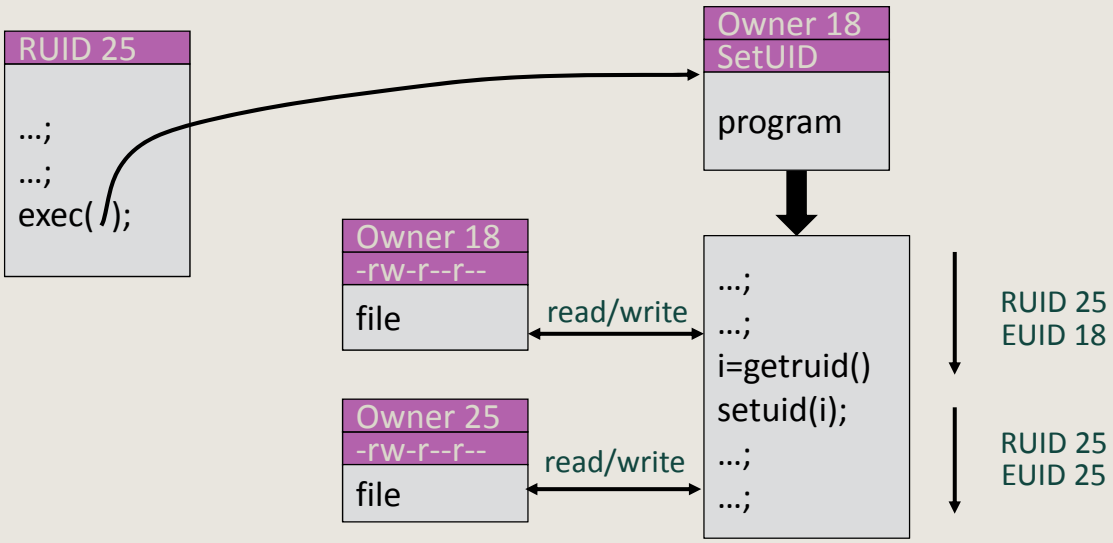
- Several different calls: setuid, seteuid, setreuid

Setid bits on executable Unix file

Three setid bits

- Setuid – set EUID of process to ID of file owner
- Setgid – set EGID of process to GID of file
- Sticky
  - Off: if user has write permission on directory, can rename or remove files, even if not owner
  - On: only file owner, directory owner, and root can rename or remove file in the directory

Example



Unix summary

Good things

- Some protection from most users
- Flexible enough to make things possible

Main limitations

- Too tempting to use root privileges
- No way to assume some root privileges without all root privileges

Mobile Security

Least Privilege Examples

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

Unix security review

Principle of Least Privilege

Unix security fundamentals

- File system permissions
- Process isolation, UID, setuid

Least privilege examples

- Qmail
- Android app separation

Stanford | Center for Professional Development

Qmail design

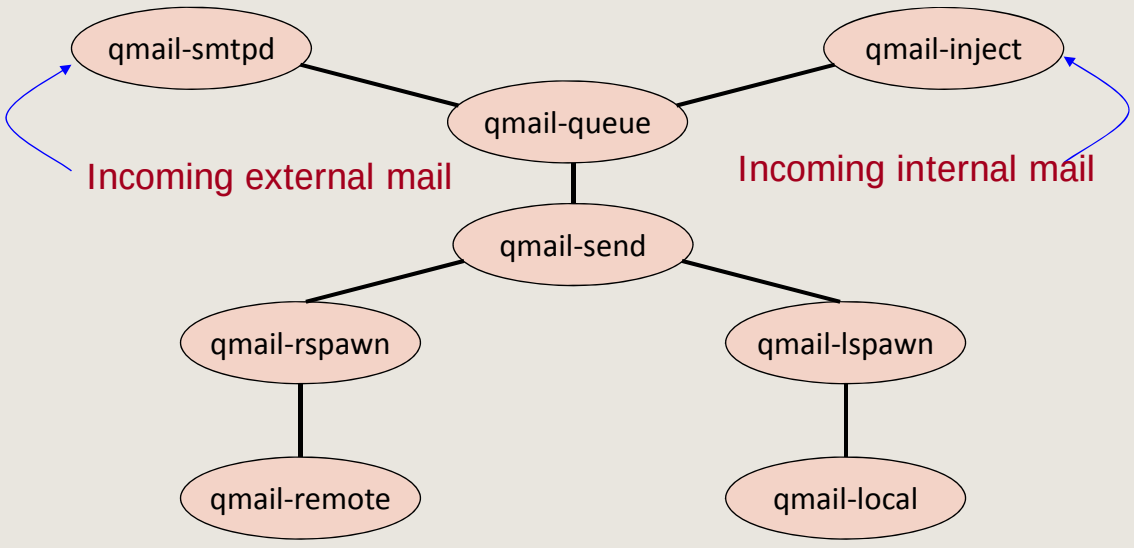
Isolation based on OS isolation

- Separate modules run as separate “users”
- Each user only has access to specific resources

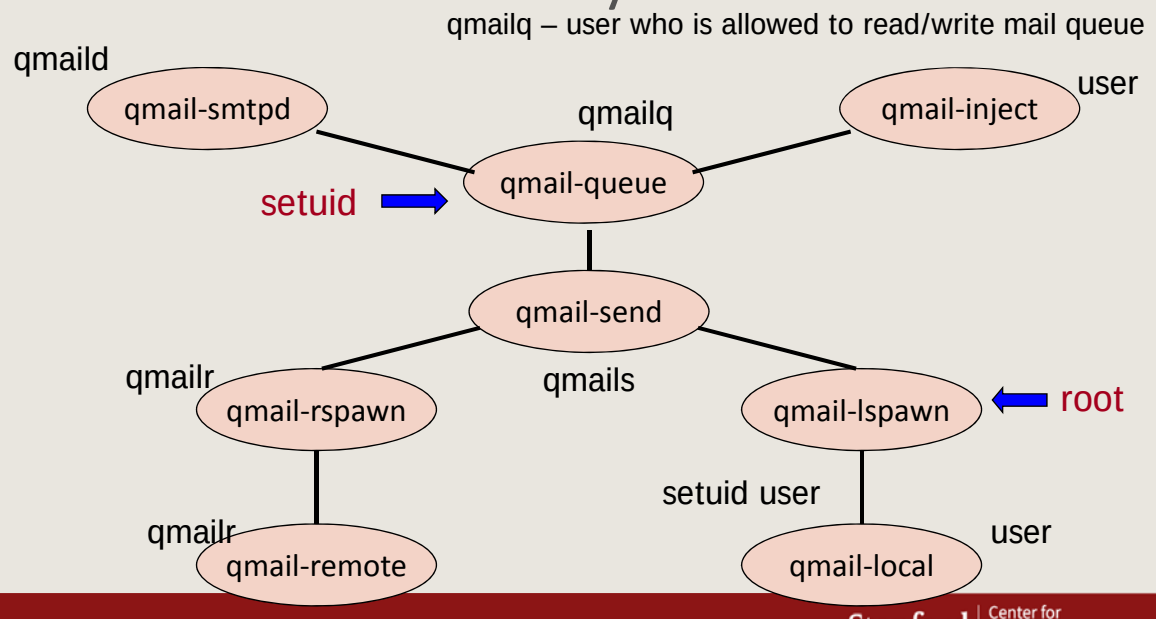
Least privilege

- Minimal privileges for each UID
- Only one “setuid” program
  - › setuid allows a program to run as different users
- Only one “root” program
  - › root program has all privileges

Structure of qmail



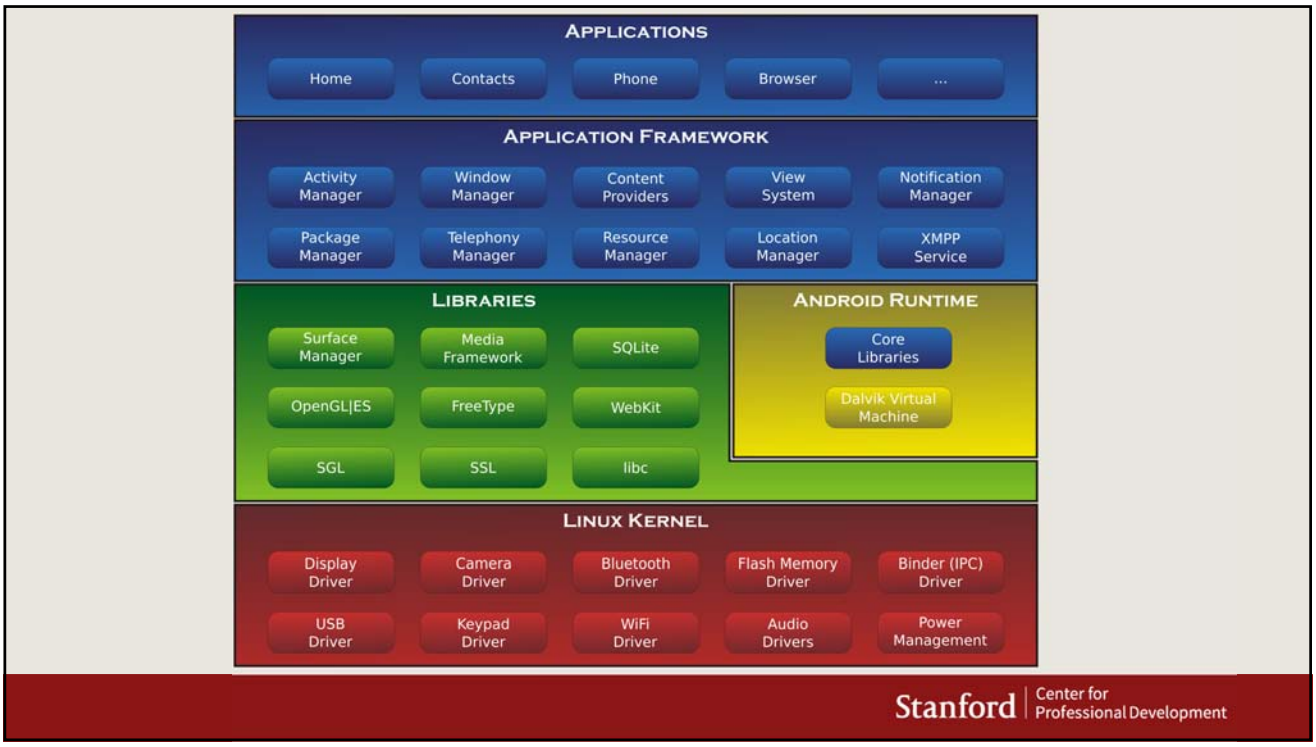
Isolation by Unix UIDs

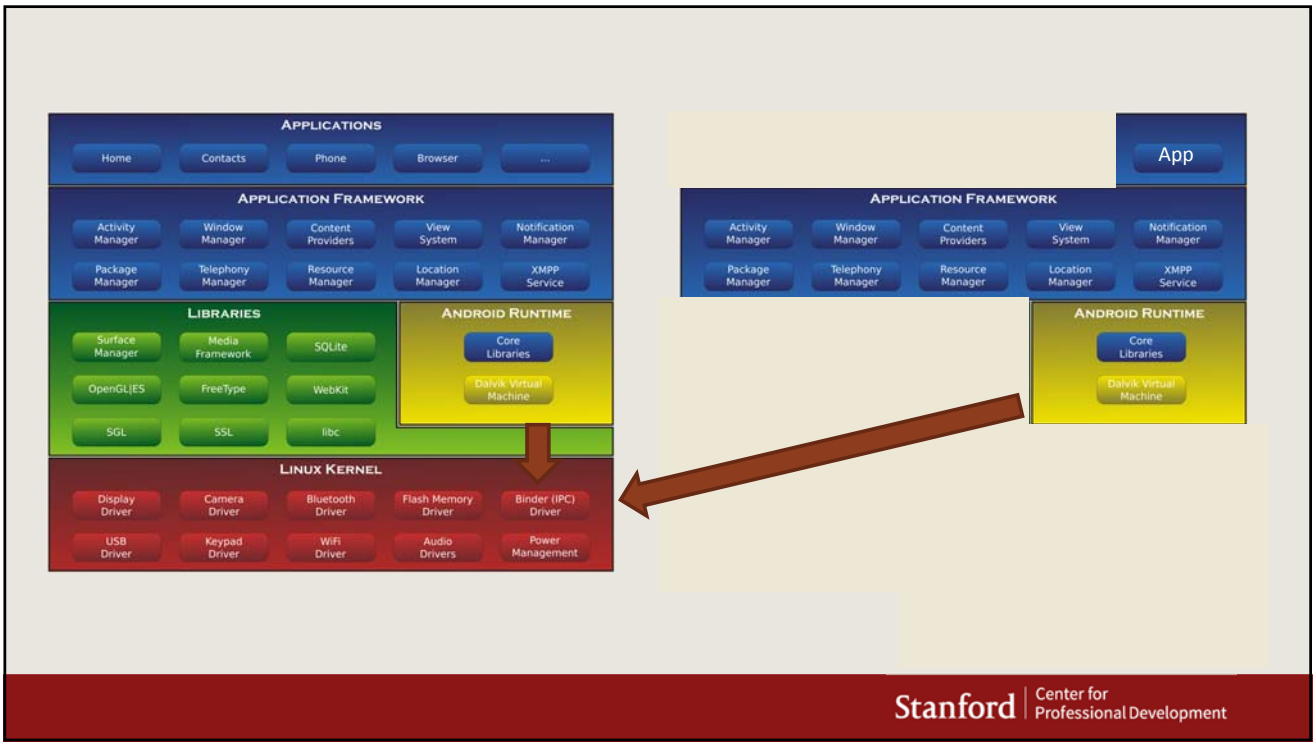


Android process isolation

Android application sandbox

- Isolation: Each application runs with own UID in own VM
  - Provides memory protection
  - Communication limited to using Unix domain sockets
  - Only ping, zygote (spawn another process) run as root
- Interaction: reference monitor checks permissions on inter-component communication
- Least Privilege: Applications announces permission
  - User grants access at install time





Unix security review

Principle of Least Privilege

Unix security fundamentals

- File system permissions
- Process isolation, UID, setuid

Least privilege examples

- Qmail
- Android app separation

Mobile Security

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

Mobile Security

Bytecode Execution Environment

Stanford | Center for Professional Development

Managed code overview

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

Stanford | Center for Professional Development

Java Implementation

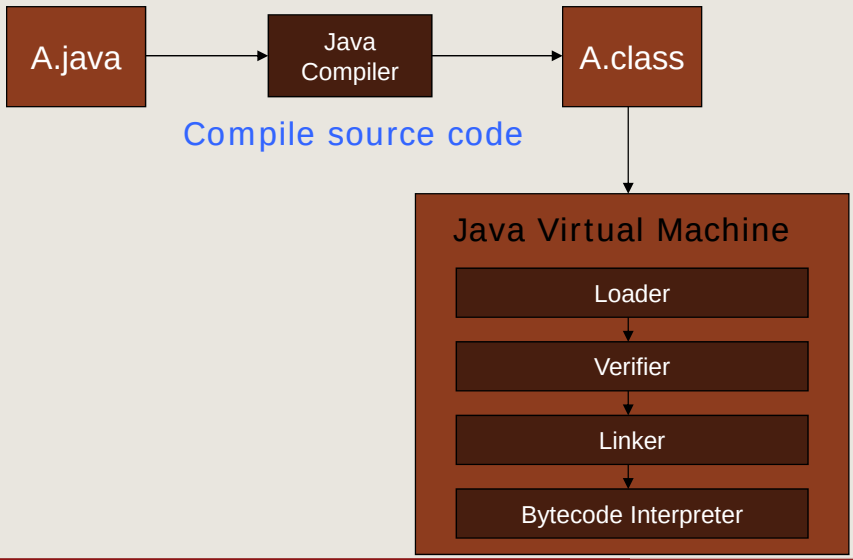
Compiler and Virtual Machine

- 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

Mobile Security

Permission Checking

Stanford | Center for Professional Development

Managed code overview

Java programming language

Bytecode execution environment

- Verifier
- Run-time checks
- Memory safety

➡ Permission checking

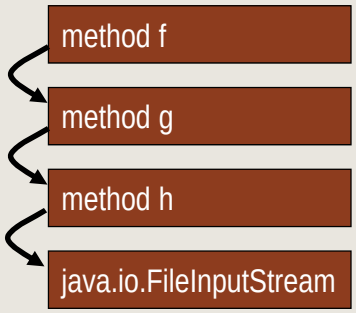
- Stack inspection

Stanford | Center for Professional Development

Stack Inspection

Permission depends on

- Permission of calling method
- Permission of all methods above it on stack
  - Up to method that is trusted and asserts this trust



  - Many details omitted here

Stories: Netscape font / passwd bug; Shockwave plug-in

Stack Inspection

Stack frames annotated with owners, set of enabled privileges

During inspection, stack frames searched from most to least recent:

- Fail if a frame belongs to owner not authorized for privilege is
- Succeed if enabled privilege is found in frame

Example: privileged printing

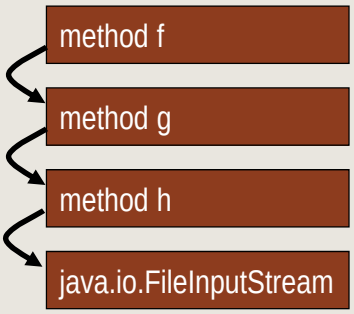
```
privPrint(f) =  (* owned by system *)
{
      checkPrivilege(PrintPriv);
      print(f);
}

foreignProg() = (* owned by Joe *)
{
      …; privPrint(file); …;
}
```


Stack Inspection

Permission depends on

- Permission of calling method
- Permission of all methods above it on stack
  - Up to method that is trusted and asserts this trust
  - Many details omitted here



Stories: Netscape font / passwd bug; Shockwave plug-in

Managed code overview

Java programming language

Bytecode execution environment

- Verifier
- Run-time checks
- Memory safety

Permission checking

- Stack inspection

Mobile Security

Android Platform Security Model

Android platform model

Architecture components

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

Permission system

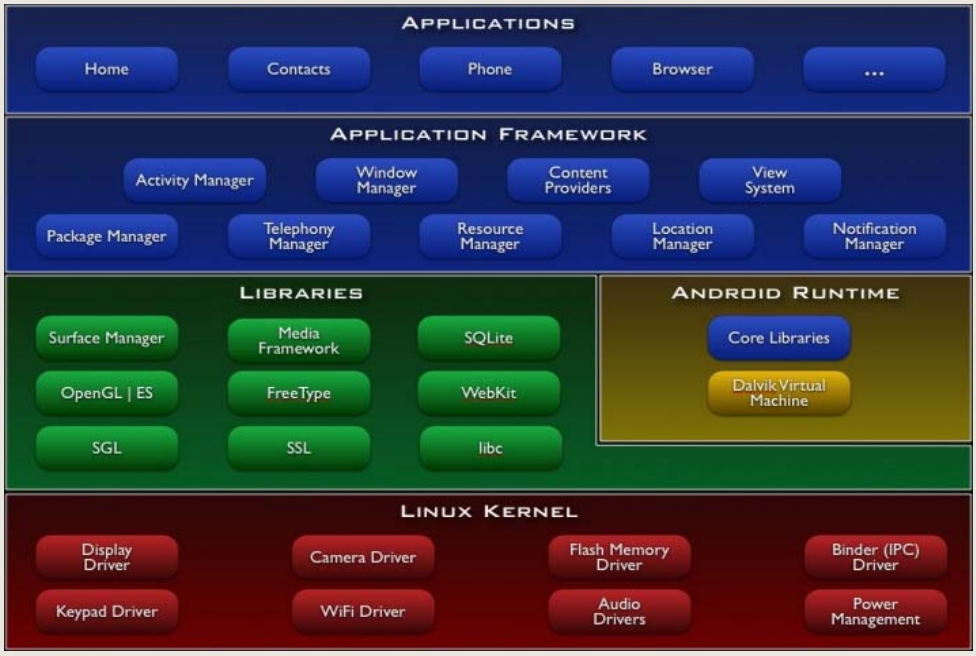
- Granted at install time
- Checked at run time

Inter-app communication

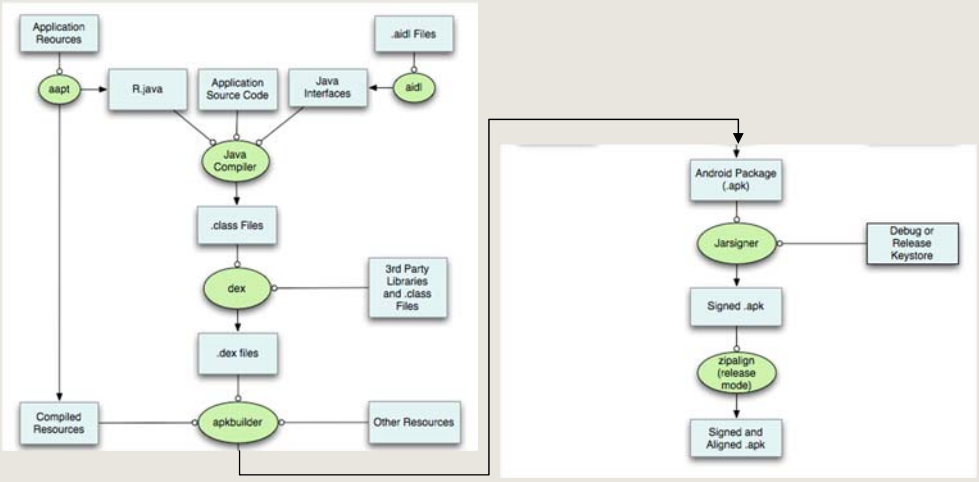
- Intent system
- Permission redelegation (intent input checking)

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)



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

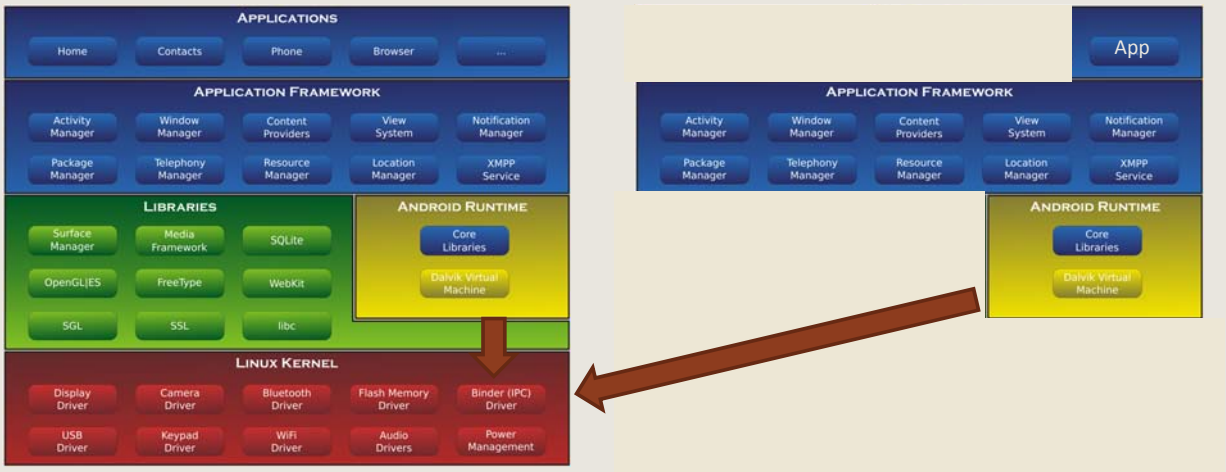
- Each application runs with its UID in its own runtime environment
  - › Provides CPU protection, memory protection
  - › Only ping, zygote (spawn another process) run as root

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



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

Mobile Security

Permission System

Stanford | Center for Professional Development

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)

Stanford | Center for Professional Development

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

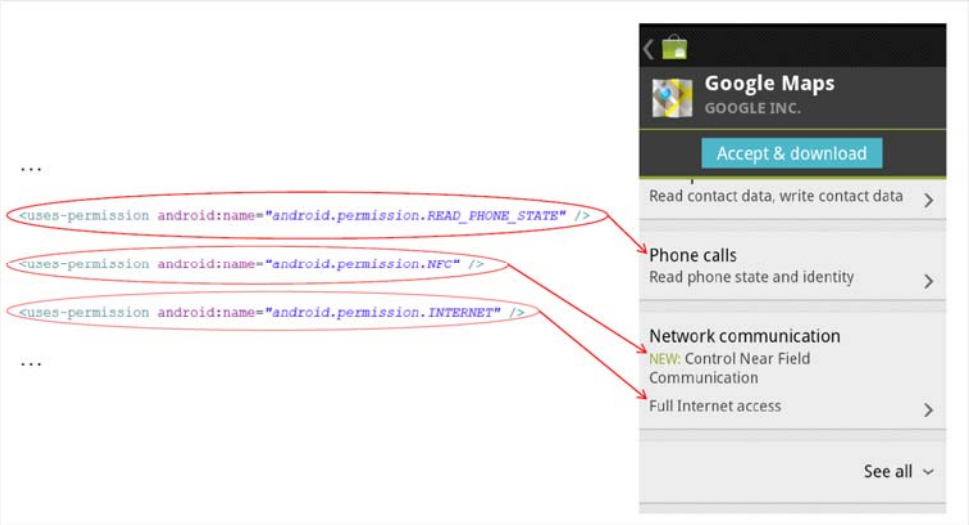
Android permissions

Example of permissions provided by Android

- “android.permission.INTERNET”
- “android.permission.READ_EXTERNAL_STORAGE
- “android.permission.SEND_SMS”
- “android.permission.BLUETOOTH”

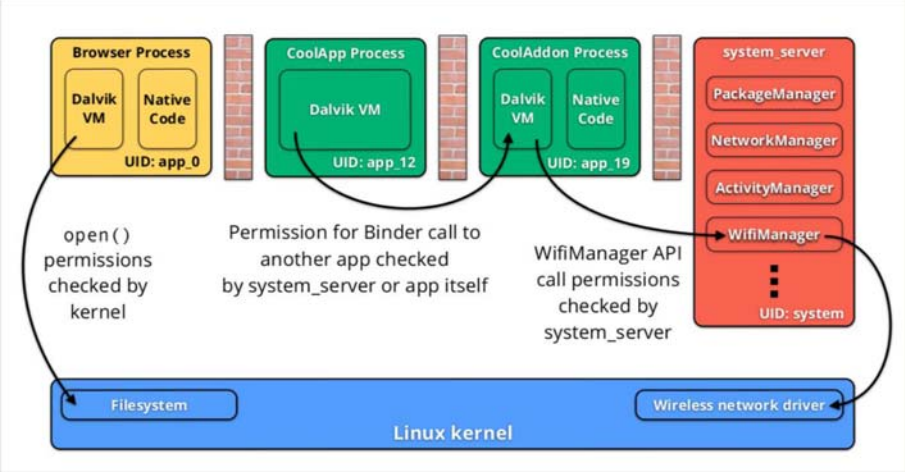
Also possible to define custom permissions

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

Mobile Security

Inter-App Communication

Stanford | Center for Professional Development

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)

Stanford | Center for Professional Development

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

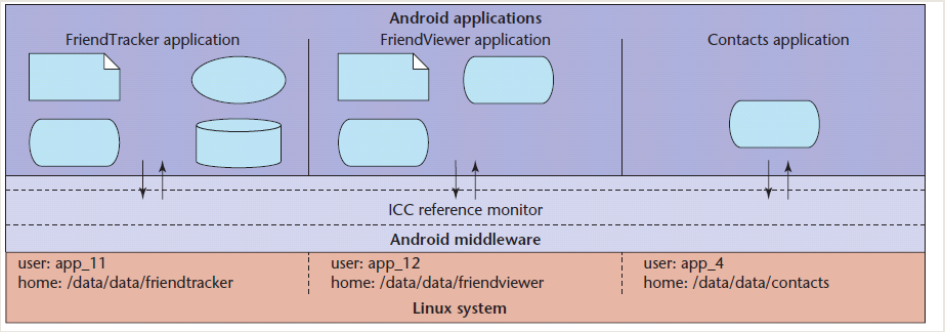
Android Intents

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

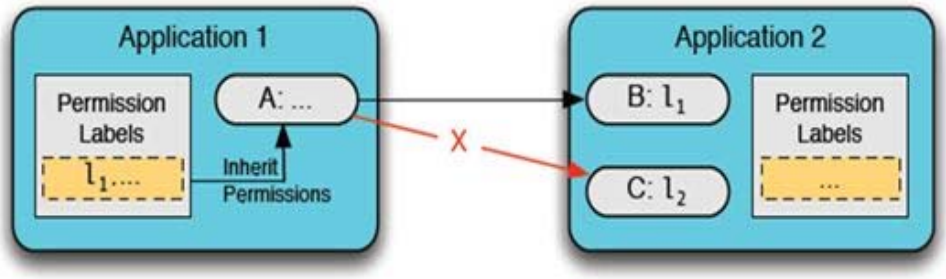
- Explicit: delivered only to a specific receiver
- Implicit: all components that have registered to receive that action will get the message



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

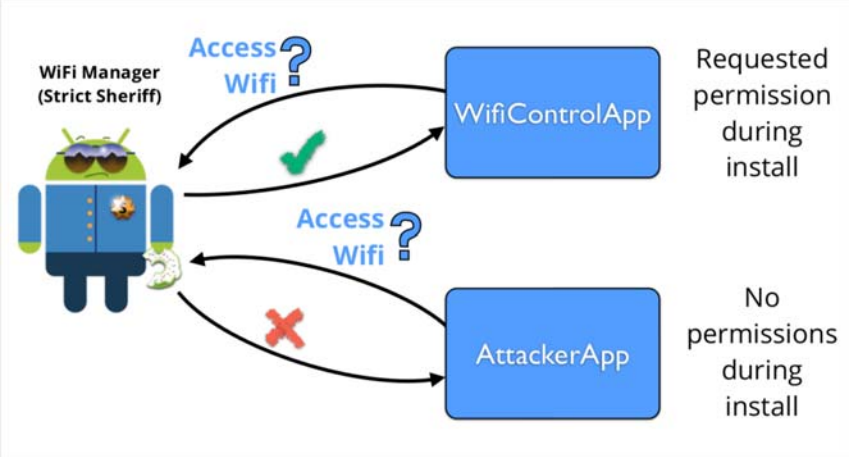
Receivers must implement appropriate input checking to handle malicious intents

Attack: Permission redelegation

Idea: an application without a permission gains additional privileges through another application

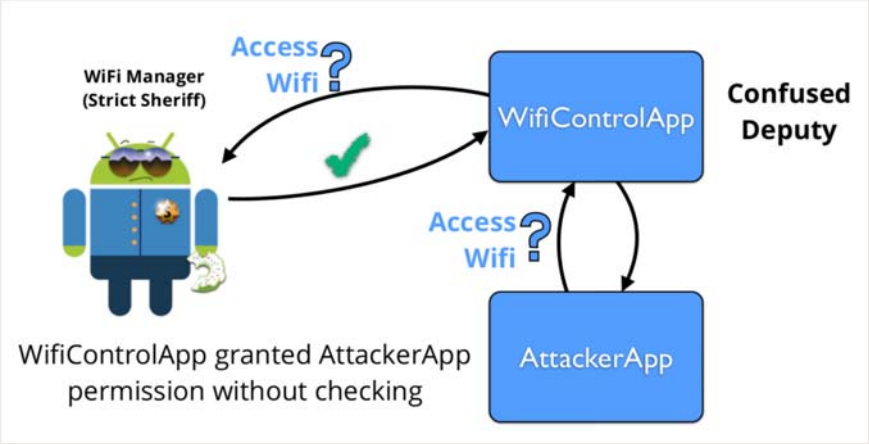
Example of the “confused deputy” problem

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

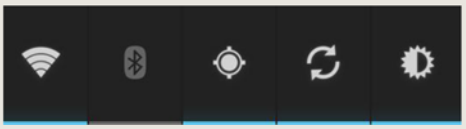
- Examine 872 apps
- 320 of these (37%) have permissions and at least one type of public component
- Construct attacks using 15 vulnerabilities in 5 apps

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%

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)

Mobile Security

Identifying Mobile Malware

Stanford | Center for Professional Development

What is “malware”

Simple answer

- An app is malware if it violates confidentiality, integrity or availability *expectations* of the user.

For example,

- If an app posts all of your contact information to Twitter, *and you installed the app for this purpose,* then the app is *not* malicious.
- But if you were told it only shows the weather, this *is* malware.

Stanford | Center for Professional Development

Detecting mobile malware

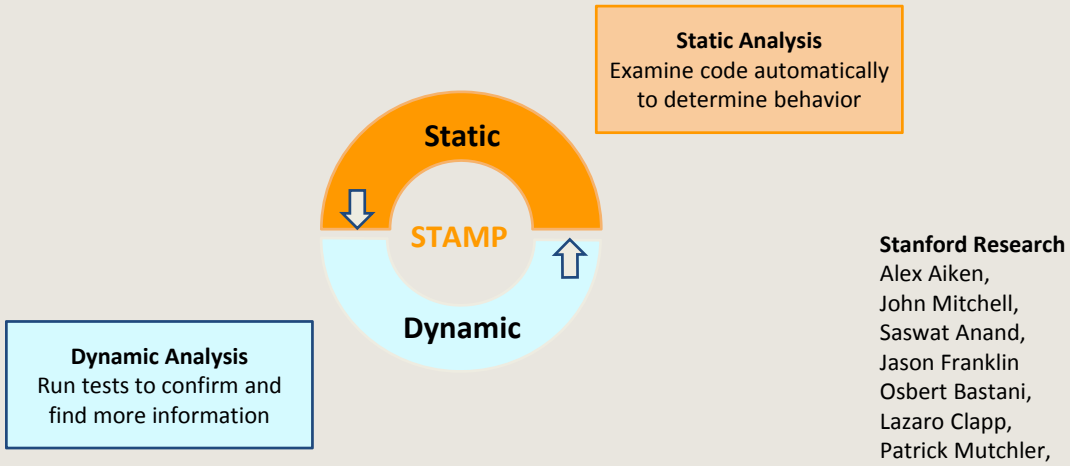
Use program information flow analysis

- Automatically analyze app bytecode to determine how information is accessed, written, or sent

Detect malicious behavior,

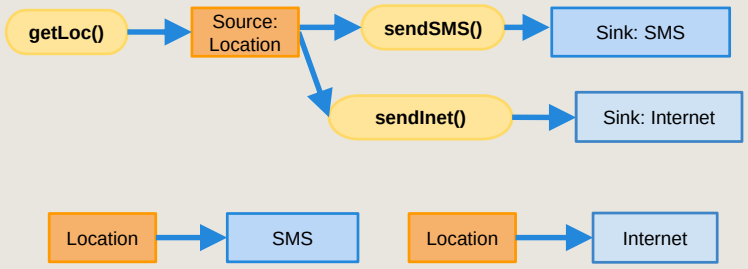
- Confidentiality: sensitive source flows to revealing sink
- Integrity: untrusted source flows to protected sink
- Availability: not generally addressed by this method

STAMP Admission System



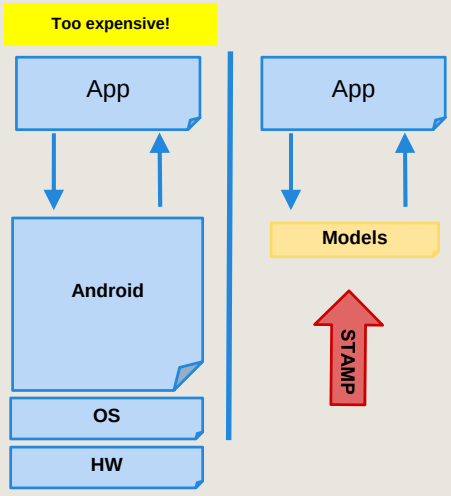
Stanford Research
Alex Aiken,
John Mitchell,
Saswat Anand,
Jason Franklin
Osbert Bastani,
Lazaro Clapp,
Patrick Mutchler,
Manolis Papadakis

Data Flow Analysis



- Source-to-sink flows
  - Sources: Location, Calendar, Contacts, Device ID etc.
  - Sinks: Internet, SMS, Disk, etc.

STAMP Approach



- Model Android/Java
  - Sources and sinks
  - Data structures
  - Callbacks
  - 500+ models

Identifying Sensitive Data

Returns device IMEI in String

Requires permission GET_PHONE_STATE

```
android.Telephony.TelephonyManager: String getDeviceId()
```

```
@STAMP(
   SRC ="$GET_PHONE_STATE.deviceid",
   SINK ="@return"
)
```


Data We Track (Sources)

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

30+ types of sensitive data

Data Destinations (Sinks)

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

10+ types of exit points

Stanford | Center for Professional Development

Currently Detectable Flow Types

396 Flow Types

Unique Flow Types = Sources x Sink

Stanford | Center for Professional Development

Example Analysis

Contact Sync for Facebook (unofficial)

Description:

This application allows you to synchronize your Facebook contacts on Android.

IMPORTANT:

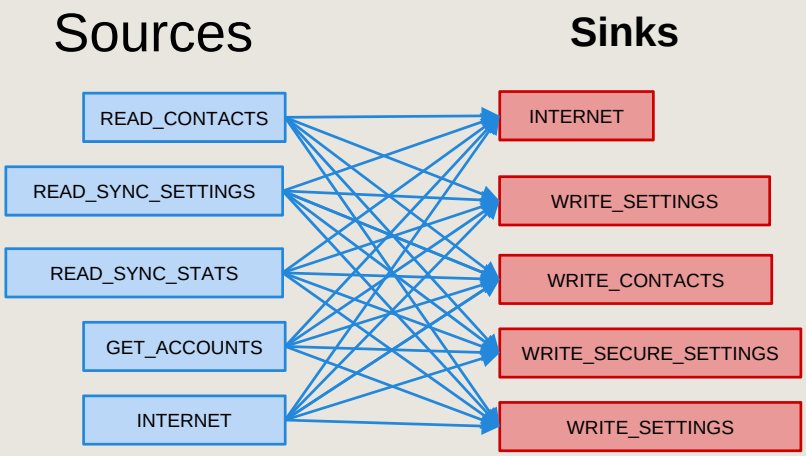
* "Facebook does not allow [sic] to export phone numbers or emails. Only names, pictures and statuses are synced."
* "Facebook users have the option to block one or all apps. If they opt for that, they will be EXCLUDED from your friends list."

Privacy Policy: (page not found)

Contact Sync Permissions

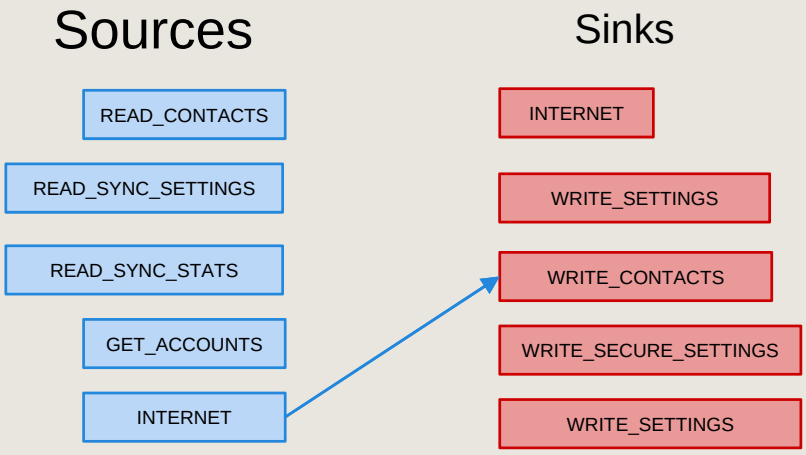
Category	Permission	Description
Your Accounts	AUTHENTICATE_ACCOUNTS	Act as an account authenticator
	MANAGE_ACCOUNTS	Manage accounts list
	USE_CREDENTIALS	Use authentication credentials
Network Communication	**INTERNET**	**Full Internet access**
	ACCESS_NETWORK_STATE	View network state
Your Personal 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

Possible Flows from Permissions



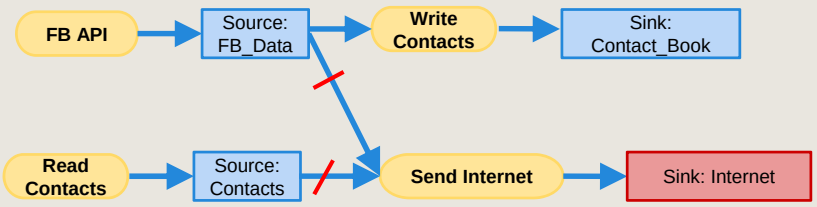
Stanford | Center for Professional Development

Expected Flows



Stanford | Center for Professional Development

Observed Flows



Detecting mobile malware

Use program information flow analysis

- Automatically analyze app bytecode to determine how information is accessed, written, or sent

Detect malicious behavior,

- Confidentiality: sensitive source flows to revealing sink
- Integrity: untrusted source flows to protected sink
- Availability: not generally addressed by this method

Mobile Security

Mobile Web Apps

Stanford | Center for Professional Development

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
- …

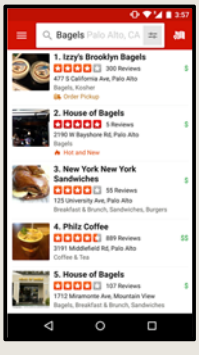
Stanford | Center for Professional Development

Mobile Web Apps

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

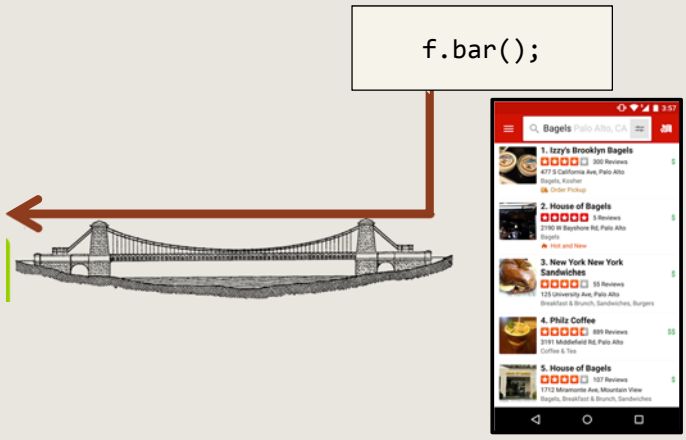
JavaScript Bridge

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



JavaScript

JavaScript Bridge



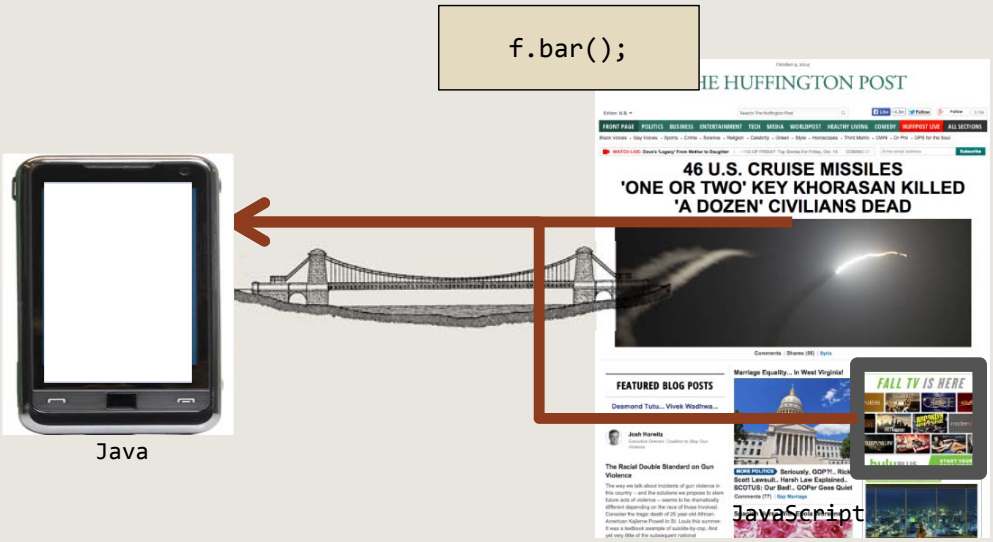
Security Concerns

Who can access the bridge?

- Everyone

Isolated in Browser

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

Mobile Security

Security Problems with URLs

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.

Mobile Security

Security Problems with SSL Errors

Handling SSL Errors

`onReceivedSslError`

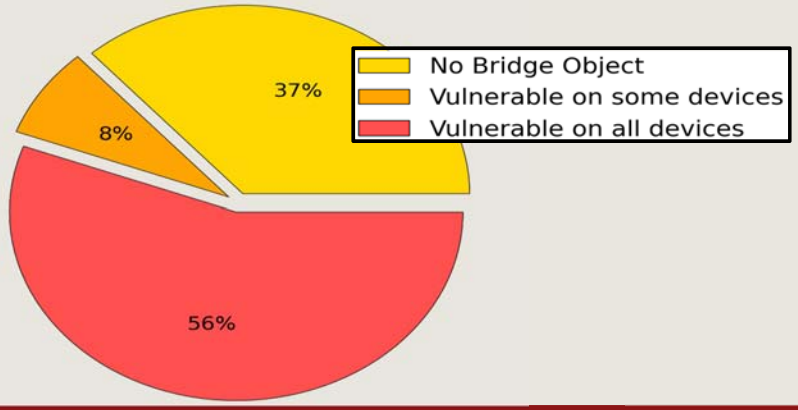
1. `handler.proceed()`
2. `handler.cancel()`
3. `view.loadUrl(...)`

Stanford | Center for Professional Development

Mishandling SSL Errors

117,974 apps implement `onReceivedSslError`

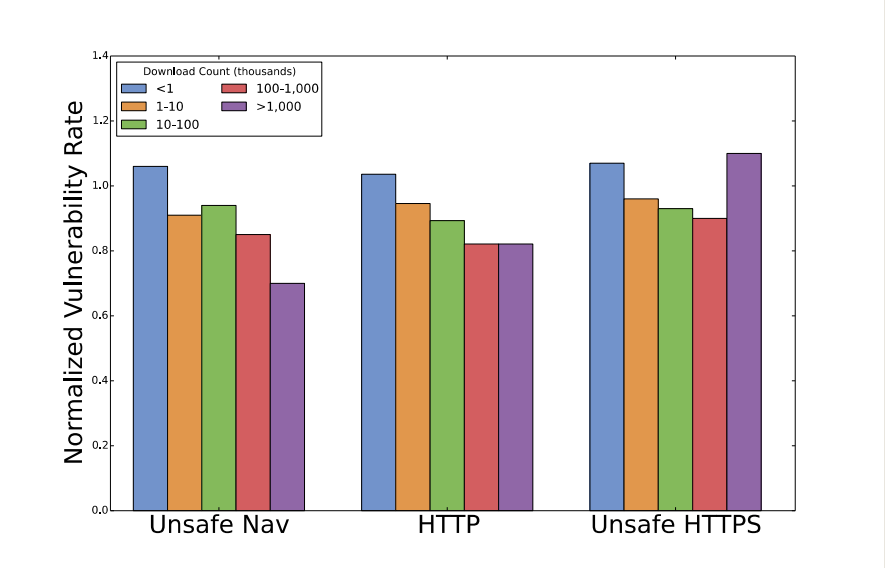
29,652 apps (25%) **must** ignore errors



Primary results

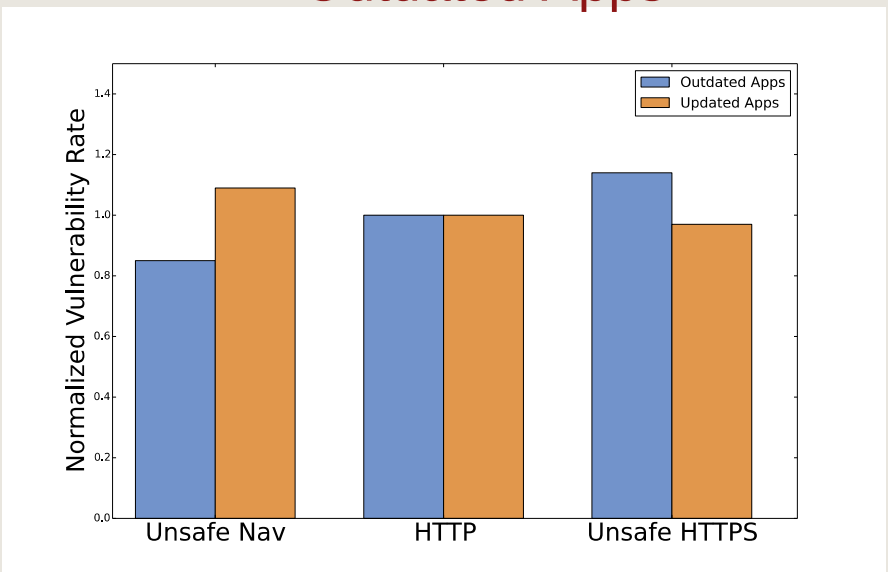
Vulnerability	% Relevant	% Vulnerable
Unsafe Nav	15	34
HTTP	40	56
Unsafe HTTPS	27	29

Popularity



Stanford | Center for Professional Development

Outdated Apps



Stanford | Center for Professional Development

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

Mobile Security

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

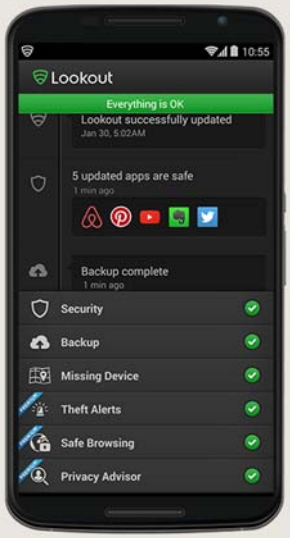
Lock and wipe

- Web interface to remotely remove data

Data backup

- Store and retrieve from cloud

https://www.lookout.com/android



Mobile Security

iOS Security

Topics for this section

iOS Security architecture:

- boot process,
- hardware security features,
- unlock process

iOS cryptography:

- key management and the iOS crypto framework

The state of iOS security

- **Find and call** (2012): accesses user's contacts and spams friends
- **Jekyll-and-Hyde** (2013): a benign app that turns malicious after it passes Apple's review process
  App can post tweets, take photos, send email and SMS, etc.
- **Xsser mRat** (2014): steal information from jailbroken iOS devices
- **WireLurker** (2014): infects iOS through USB to OSX machines
- **Xagent** (2015): Spyware. Steals texts, contacts, pictures, …
- **AceDeceiver** (2016): infects by exploiting vuln. in Fairplay (DRM)

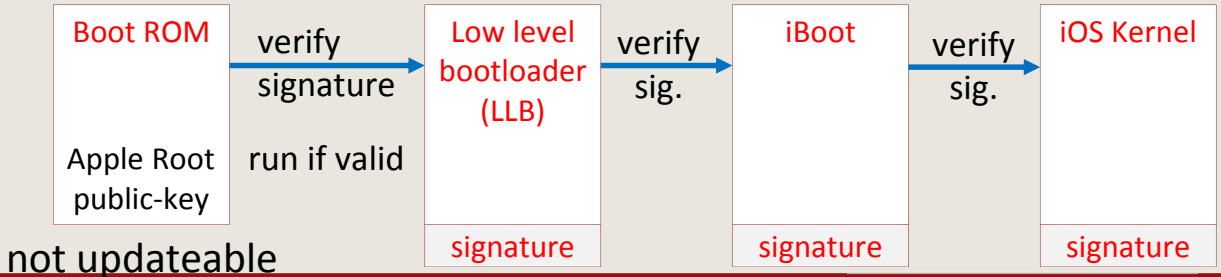
Mobile Security

iOS Security Architecture

Secure boot chain

Every layer ensures that the next layer is properly signed

Root of trust: boot ROM, installed during fabrication



not updateable

Secure boot chain

Ensures only authorized iOS code can boot

Jailbreaking works by exploiting bugs in the chain

- Disables verification down the line

Note: bugs in the boot ROM are especially damaging

- Boot ROM cannot be updated

Mobile Security

Software Update

Software update

All iOS software updates are signed by Apple

- Signature from Apple's software update server covers:
  hash of update code,
  device unique ID (ECID) and nonce from device

⇒ Apple keeps track of which devices (ECID) updated to what

Why sign nonce and device ID? (harder for Apple to distribute patch)

- Cannot copy update across devices ⇒ Apple can track updates
- Nonce ensures device always gets latest version of update

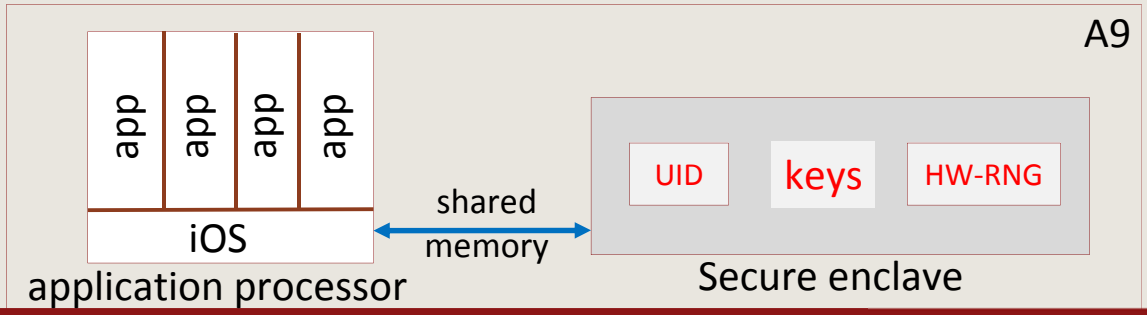
Mobile Security

Secure Enclave

Secure enclave (Apple A7 and later)

Coprocessor fabricated in the Apple A7, A8, …

- All writes to memory and disk are encrypted with a random key generated in the enclave
- Used for device unlock, ApplePay, … (more on this later)



Mobile Security

Protecting Application Data

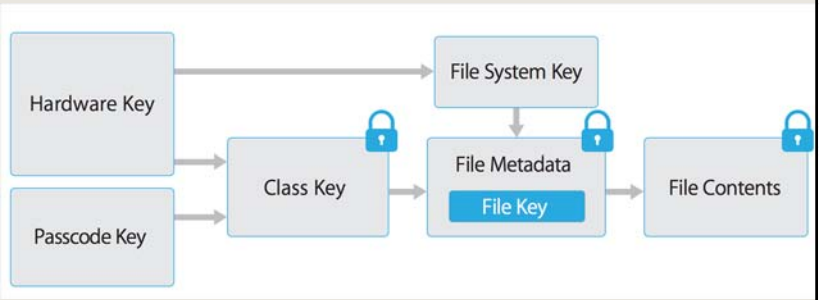
Data protection: protecting application data

Application files written to Flash are encrypted:

- Per-file key: encrypts all file contents (AES-XTS)
- Class key: encrypts per-file key (ciphertext stored in metadata)
- File-system key: encrypts file metadata

Resetting device deletes file-system key

All key enc/dec takes place inside the secure enclave
⇒ key never visible to apps



Mobile Security

Class key: Data Protection Classes

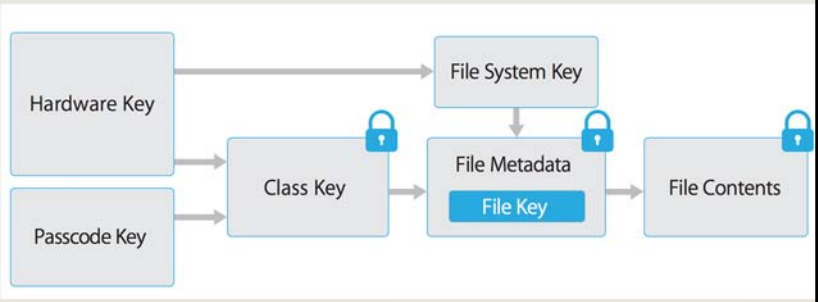
Data protection: protecting application data

Application files written to Flash are encrypted:

- Per-file key: encrypts all file contents (AES-XTS)
- Class key: encrypts per-file key (ciphertext stored in metadata)
- File-system key: encrypts file metadata

Resetting device deletes file-system key

All key enc/dec takes place inside the secure enclave
⇒ key never visible to apps



Class key: data protection classes

Every file has a class: determines when file can be accessed

- **Complete protection**: (best) [no access when device is locked]
  -- class key derived from user's passcode and device UID
  -- key is erased 10s after device lock (unless TouchID enabled)
  -- class key always erased on reboot
- **Protected Until First User Authentication:** (default)
  -- same as complete protection, but key is available when locked
- **No protection**: class key is never erased, until a device wipe
- **Protected Unless Open:**
  -- open file can be written to while device is locked
  -- new files can be created

Setting Data Protection level: NSFileManager example

```
// Build protection attribute dictionary
NSDictionary *attr =
    [NSDictionary    dictionaryWithObject:NSFileProtectionComplete
                     forKey: NSFileProtectionKey];

// Set protection attribute on a specific file
[[NSFileManager defaultManager]
                     setAttributes: attr          ← protection attribute
                     ofItemPath: filePath         ← for file path
                     error: &error];
```


The DataProectionClass entitlement

If app does not need to read/write files when device is locked:

- Add NSFileProtectionComplete entitlement to project
- Affects all files managed with NSFileManager, NSData, SQLite

Mobile Security

Key Chain

Stanford | Center for Professional Development

Key chain

SQLite database used to manage passwords and keys

- Items can be shared among apps from same developer
  - Enforced by code signing

Access classes: **Complete, UntilFirstUserAuth, NoProtection**

- ThisDeviceOnly ×3: backup can only be restored to this device

Access control:

- Can specify conditions when item readable: pass code entered, using TouchID
- ACL is enforced by secure enclave

Key chain

Name	Value
WiFi password	lhs456mnb
iCloud token	p98y98y34
Bluetooth key (non-migratory)	W%Ssg3$#

Stanford | Center for Professional Development

Example: adding an item to keychain
(incomplete)

General API: SecItem**Add**, SecItem**Update**, SecItem**CopyMatching**, SecItem**Delete**

```
NSMutableDictionary *dict = [NSMutableDictionary dictionary];

[dict setObject:kSecClassGenericPassword forKey:kSecClass];          ← type
[dict setObject:@"example"               forKey:kSecLabel];
[dict secObject:@"dabo"                  forKey:kSecAttrAccount];
[dict secObject:@"8sdfg876wa"            forKey:kSecValueData];      ← value
[dict secObject:kSecAttrAccessibleWhenUnlocked forKey:kSecAttrAccessible];

OSStatus error = SecItemAdd(dict, NULL);
...
```

when is item accessible (must specify)

Backup to iCloud

Data backup: encrypted data sent from device to iCloud

- Exception: not applied to data of class NoProtection

Class keys: backed up protected by "iCloud keys" (for device migration)

keychain class keys:

- Non-migratory class keys: wrapped with a UID-derived key

  ⇒ Can only be restored on current device

- App-created items: not synced to iCloud by default

  [dict secObject:kCFBooleanTrue forKey:kSec**AttrSynchronizable**];

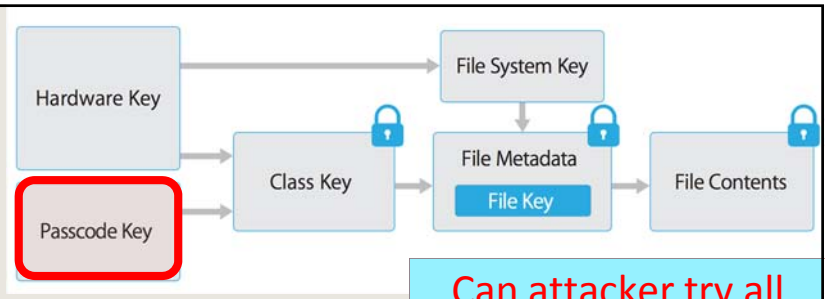
Mobile Security

Device Unlock

Stanford | Center for Professional Development

Device unlock

Passcode key:
derived by hashing
passcode and device ID



Can attacker try all 6-digit passcodes?

Hashing uses secret UID on secure enclave
⇒ deriving passcode key requires the secure enclave

Secure enclave enforces 80ms delay per evaluation:

- 5.5 years to try all 6 digits pins
- 5 failed attempts ⇒ 1min delay, 9 failed attempts ⇒ 1 hour delay
- >10 failed attempts ⇒ erase phone. Counter on secure enclave.

Stanford | Center for Professional Development

Smudge attacks [AGMBS 2010]

Typing on screen leaves a "smudge"

- Keys can be read with proper lighting
- Significantly reduces brute force guessing time

Proposed defense:

- Force user to move finger around keypad after every authentication
  - … otherwise, device stays locked

Unlocking with Touch ID

Passcode can always be used instead of Touch ID

Passcode required after:

- Reboot, five unsuccessful Touch ID attempts, …

How does it work? How does device get passcode key?

Other uses (beyond unlock):

- Enable access to keychain items
- Apple Pay
- Can be used by applications

How does it work?

Touch ID: sends fingerprint image to secure enclave (encrypted)

- Enclave stores skeleton encrypted with secure enclave key

Recall: with Touch ID off, upon lock, class-key **Complete** is deleted

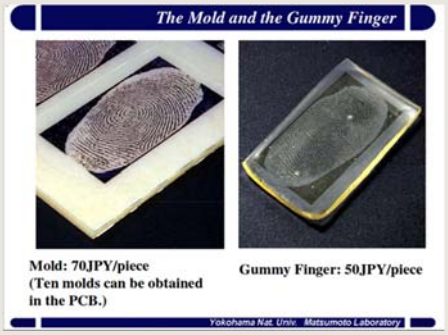
⇒ no data access when device is locked

With Touch ID on: class-key is stored encrypted by secure enclave

- Decrypted when authorized fingerprint is recognized
- Deleted upon reboot, 48 hours of inactivity, or five failed attempts

Stanford | Center for Professional Development

How secure is it?



Building a fake finger from a fingerprint:

- Several demos on YouTube:
  about 20 mins of work
- The problem: fingerprints are not secret
  No way to reset once stolen

Convenient, but more secure solutions exist:

- Unlock phone via bluetooth using a wearable device
  ⇒ phone locks as soon as device is out of range
- Enable support for both a passcode **and** a fingerprint

Stanford | Center for Professional Development

Mobile Security

App Security

Stanford | Center for Professional Development

Runtime process security

All 3rd party apps are sandboxed:

- run as the non-privileged user “mobile”
  access limited by underlying OS access control
- each app has a unique home directory for its files
  randomly assigned when the app is installed
- Accessing other info: by mediated services provided by iOS

Stanford | Center for Professional Development

App exploit mitigation: XN and ASLR

- **XN bit** (eXecute Never): [a.k.a NX bit]
  Mark stack and heap memory pages as non-execute, enforced by CPU
- **ASLR** (address space layout randomization):
  at app startup: randomize location of executable, heap, stack
  at boot time: randomize location of shared libraries

Makes it harder to exploit memory corruption vulnerabilities
(see writing secure code course)

Stanford | Center for Professional Development

Jailbreak detection

Jailbreaking: enables installing apps outside 3rd party sandbox

- Apps installed in /Applications (not in "mobile" home dir.)

Goal: app wants to detect if device is jailbroken and not run if so

- e.g., banking apps

Some methods:

- _dyld_get_image_name(): check names of loaded dynamic libs
- _dyld_get_image_header(): inspect location in memory

Can be easily bypassed: jailbreak detection is brittle

- e.g., using Xcon tool (part of Cydia)

Stanford | Center for Professional Development

Mobile Security

App Code Signing

Stanford | Center for Professional Development

App code signing

All executable code must be signed by an Apple certificate

- Applies to: native apps, 3rd party apps (signed after Apple review), and dynamic libraries
  - App can link against any dynamic library with the same TeamID (10-char string)
  - Example: an ad network library

Not perfect: Charlie Miller's **InstaStock** app

- stock ticker program: passed Apple review
- After launch: downloads "data" from remote site, stores it in non-XN region, executes it ⇒ app becomes malicious
- Why is there a non-XN region? Needed for Safari JIT.

Stanford | Center for Professional Development

Mobile Security

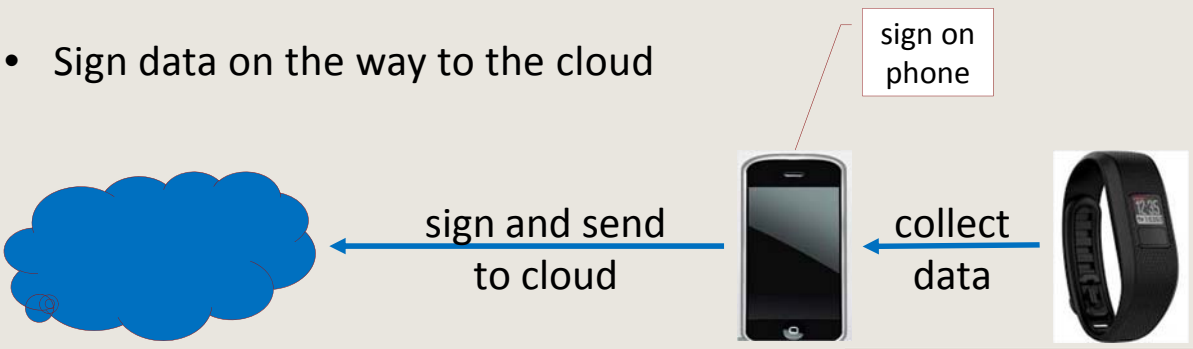
App Layer Crypto

Stanford | Center for Professional Development

Application layer crypto: CCCrypt

Some applications need to use crypto directly:

- Encrypt/decrypt to ensure end-to-end security
- Sign data on the way to the cloud



Stanford | Center for Professional Development

Application layer crypto: CCCrypt

iOS provides CommonCrypto framework (CCCrypt)

- Implements many crypto algorithms (including bad ones, e.g. DES)
  - For encryption/decryption, use **RNCrypto** framework in CCCrypt

Warnings:

- Many pitfalls in using crypto algorithms (see crypto course)
- Generating entropy (e.g. random IV)

  int result = **SecRandomCopyBytes**(NULL, length, &buf);

Further reading

- Apple iOS Security Guide:
  www.apple.com/business/docs/iOS_Security_Guide.pdf
- iOS application security, by David Thiel, 2016

Mobile Security

App Security- iMessage

Stanford | Center for Professional Development

Topics for this section

Application security

- iMessage security
- Using TLS and WebView securely
- Preventing data leaks

Geolocation services: how to

- Phone sensors: unintended consequences

Stanford | Center for Professional Development

iMessage security

iMessage: Apple's messaging system

- Support end-to-end encryption (as in WhatsApp, Signal, …)

Setup: when iMessage first runs

- Device generates two public/private key pairs

  encryption keys: $\mathbf{pk_e/sk_e}$ signing keys: $\mathbf{pk_s/sk_s}$

- Sends $\mathbf{pk_e}$, $\mathbf{pk_s}$ to Apple's directory service (IDS)
- At IDS: store keys from all user's devices under user's APN address

iMessage security

User sends an iMessage:

- All device keys belonging to recipient are fetched from IDS
- Message encrypted under all recipient's encryption keys and signed under sender's key (see crypto course)
  - Crypto bugs in 2016 version [Usenix Security 2016]
- Ciphertext sent to APN (push notification service) for delivery
  - Queued for 30 days for offline devices

Note: metadata (timestamp, recipients) is readable by Apple

Mobile Security

App Security- Using TLS

Stanford | Center for Professional Development

Using TLS in apps

Talking to server via HTTP: **NSURLConnection** or **NSURLSession**

- Servers must support TLS 1.2 with valid certs
- TLS automatically applied to apps that are compiled for iOS 9
  - Invalid server cert results in a hard failure and no connection

Unfortunately, app developer can override App Transport Security

- App's **Info.plist**: list URLs to access over HTTP
- Using deliberate code (not shown here on purpose)

Don't do this!

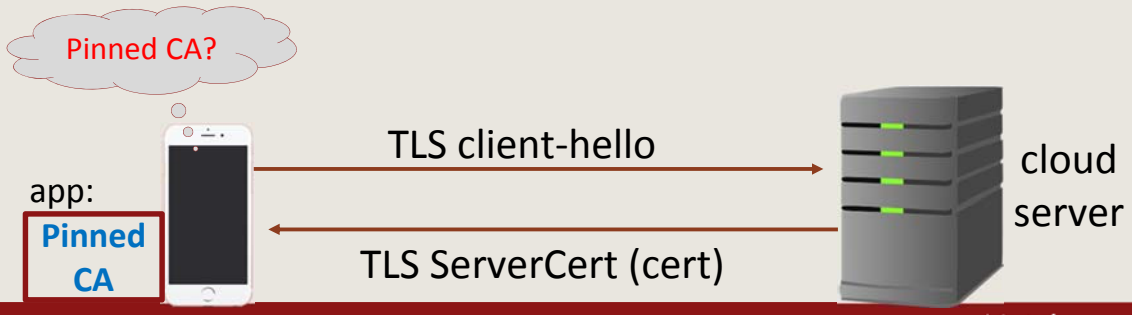
Stanford | Center for Professional Development

Certificate pinning

Many apps only talk to their home cloud:

- Should pin to the CA that issued the certs to cloud servers
- Reject all other certs and fail to connect

  ⇒ ensures a mistake by another CA does not compromise app



Stanford | Center for Professional Development

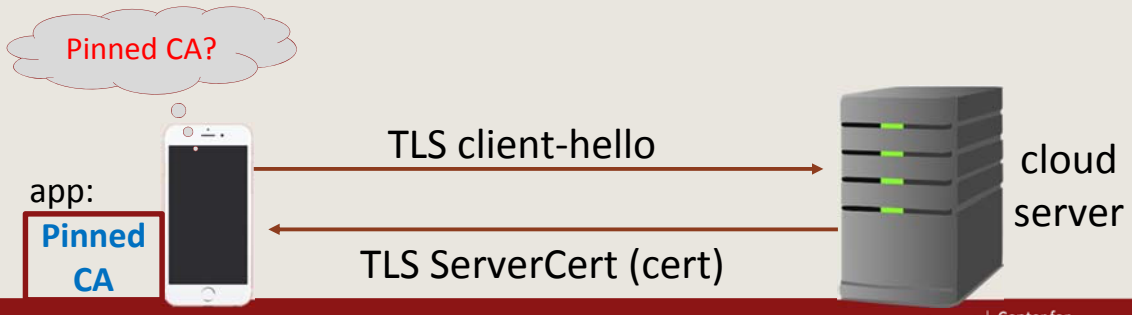
Certificate pinning: how?

Example: //github.com/iSECPartners/ssl-conservatory/tree/master/ios

In **willSendRequestForAuthenticationChallenge :**

verifyPinnedCertificateForTrust:

checks that server cert matches pinning DB



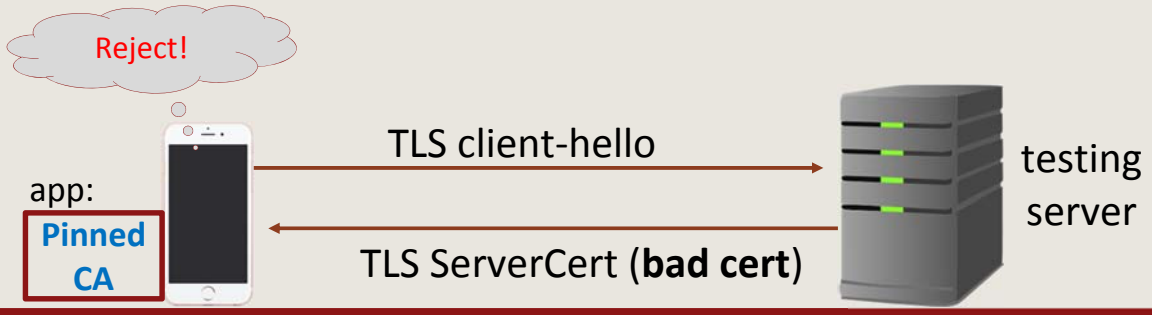
Stanford | Center for Professional Development

Certificate pinning: testing

Make sure to test pinning:

- Setup a server with a invalid certs or certs from other CAs
- Check that app refuses to connect!

[See //developer.apple.com/library/ios/technotes/tn2326]



Stanford | Center for Professional Development

Deleting private state in NSURLSession

NSURLSessionConfiguration *configure =

[NSURLSessionConfiguration **ephemeralSessionConfiguration**];

⇒ private data is never written to disk: (stored in RAM)
caches, cookies, or credentials

⇒ deleted when app invalidates session or is terminated

[configure setTLS**MinumumSupportedProtocol** = kTLSProtocol12];

⇒ Force TLS 1.2 (or above) [relevant once TLS1.3 is out]

Stanford | Center for Professional Development

WebView

An application can launch a web view on part or all of the screen

- Content is read from a given URL and presented in WebView
- Interface to use: **WKWebView**

WKWebView security preferences: WKPreferences *p

- [p **setJavaScriptEnabled**: NO]; ← no JavaScript in WebView
- After page loads: **hasOnlySecureContent** flag
  True only if all content was loaded over HTTPS
  App should close WebView if not

Mobile Security

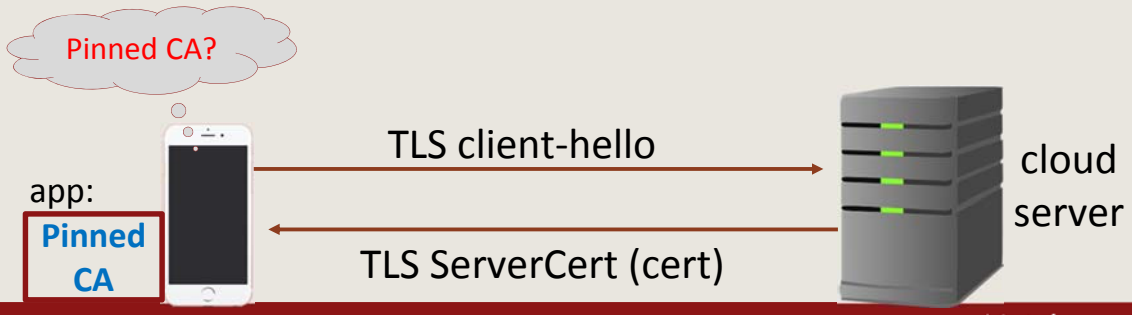
App Security- Certificate Pinning

Certificate pinning

Many apps only talk to their home cloud:

- Should pin to the CA that issued the certs to cloud servers
- Reject all other certs and fail to connect

  ⇒ ensures a mistake by another CA does not compromise app



Stanford | Center for Professional Development

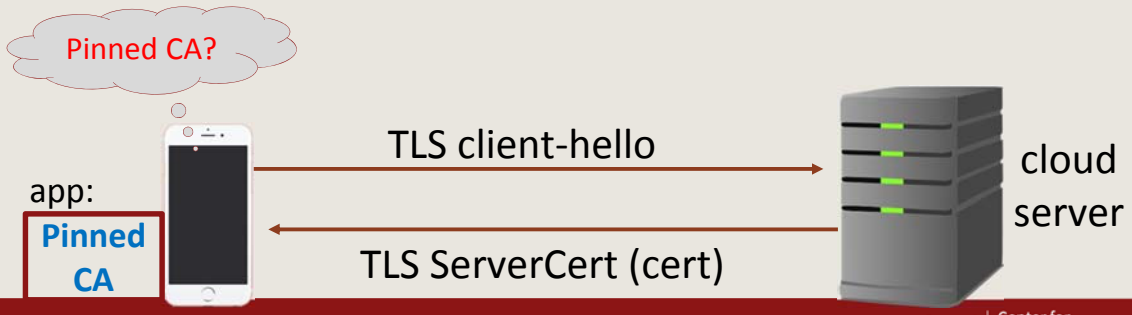
Certificate pinning: how?

Example: //github.com/iSECPartners/ssl-conservatory/tree/master/ios

In **willSendRequestForAuthenticationChallenge :**

verifyPinnedCertificateForTrust:

checks that server cert matches pinning DB



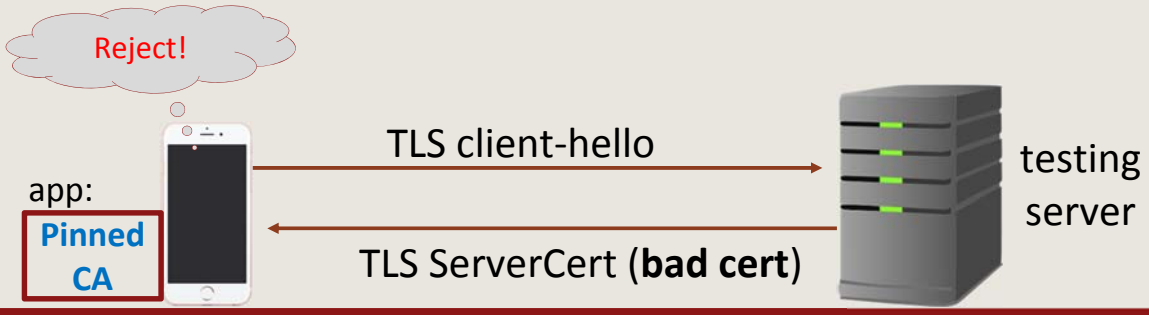
Stanford | Center for Professional Development

Certificate pinning: testing

Make sure to test pinning:

- Setup a server with a invalid certs or certs from other CAs
- Check that app refuses to connect!

[See //developer.apple.com/library/ios/technotes/tn2326]



Stanford | Center for Professional Development

Mobile Security

App Security- Deleting Private State

Stanford | Center for Professional Development

Deleting private state in NSURLSession

NSURLSessionConfiguration *configure =
[NSURLSessionConfiguration **ephemeralSessionConfiguration**];

⇒ private data is never written to disk: (stored in RAM)
caches, cookies, or credentials

⇒ deleted when app invalidates session or is terminated

[configure setTLS**MinumumSupportedProtocol** = kTLSProtocol12];

⇒ Force TLS 1.2 (or above) [relevant once TLS1.3 is out]

Mobile Security

App Security- Web View

WebView

An application can launch a web view on part or all of the screen

- Content is read from a given URL and presented in WebView
- Interface to use: **WKWebView**

WKWebView security preferences: WKPreferences *p

- [p **setJavaScriptEnabled**: NO]; ← no JavaScript in WebView
- After page loads: **hasOnlySecureContent** flag
  True only if all content was loaded over HTTPS
  App should close WebView if not

Mobile Security

Data Leaks

How data can leak out of an application

Pasteboard: user makes use of copy/cut

- Data is pasted to system pasteboard … readable by every app

  ⇒ prevent copy/cut of sensitive content (using **canPerformAction**)

- UIPasteboardSniffer-iOS: an app that sniffs the system pasteboard

Snapshots: before sending app to background

- iOS takes screen snapshot … stores on disk
- Example: incoming phone call while user is entering a password

  ⇒ password ends up on disk

Handling snapshots: splash screen

The following function is called when application is about to be suspended:

splash image name

```
(void) applicationDidEnterBackground … {
    self.splash = ... get screen object ...
    [self.splash setImage:[UIImage imageNamed:@"splash.png"]];
    ...
}
```

Remove splash screen using **applicationWillEnterForegroud**

Mobile Security

Geolocation on iOS

Stanford | Center for Professional Development

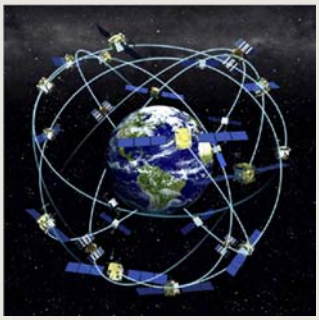
How Geolocation Works

Method 1: Visible WiFi hotspot SSIDs detected by phone

- Phone queries central service to obtain location
- Fast and low-power, but inaccurate location

Method 2: GPS

- Accurate, but requires signal from four satellites
- Often unavailable
- Requires lots of power



Stanford | Center for Professional Development

Querying the Geolocation API

[[self locationManager] setDesiredAccuracy: **accuracy**];

Accuracy: (use lowest possible)

- kCLLocationAccuracy**BestForNavigation**
- kCLLocationAccuracy**Best**
- kCLLocationAccuracy**NearestTenMeters** ← IFTT approach home
- kCLLocationAccuracy**HundredMeters**
- kCLLocationAccuracy**Kilometer** ← find closest gas station

Delete location data as soon as possible

Mobile Security

Location Services Without Tracking

Private Location Services

Many location services possible without tracking user location:

Privately find friends close by

Private location statistics (hotspots)

Private location based advertising

Private location based search

Traffic based navigation

Proximity alerts

Detect when friends are nearby

- Naïve implementation: 24/7 user tracking by server

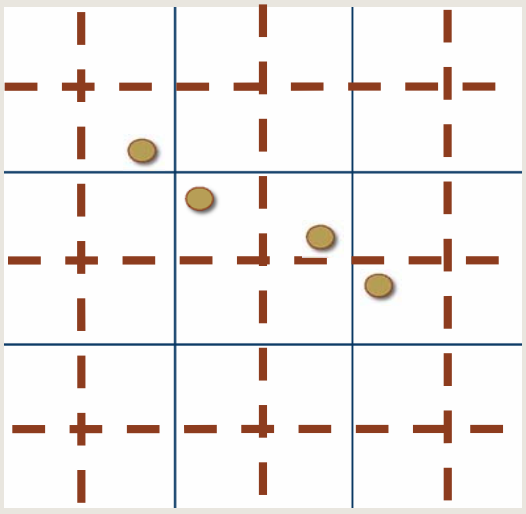
Our privacy goals:

- When not nearby, friends don't see your location
- Server never sees your location

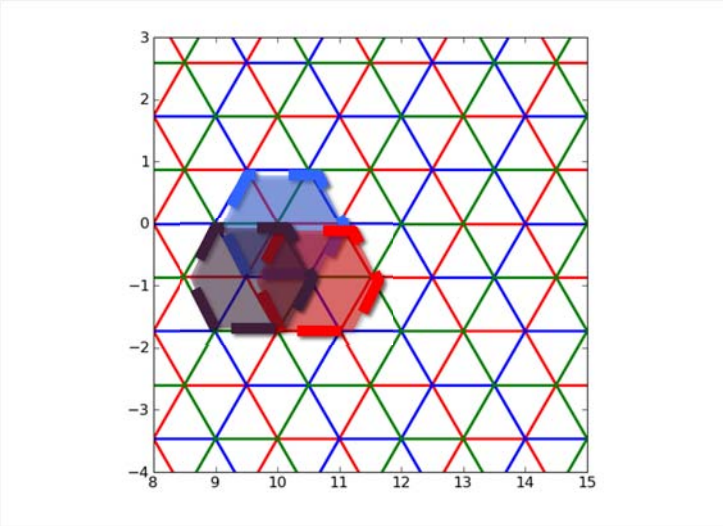
⇒ no collection is safer for user and for cloud service

Proximity alerts: applications

Reducing proximity test to equality test



Correct method: 3 hexagonal grids



Users are “close” if they match on **one** of three grids

Our approach

Trust assumption: server does not collude with your friends

r (x – y) mod p

Alice does not know r

Total traffic: 18 bytes, easy computation

Our approach

$x \in F_p$

$k_1, k_2 \in F_p$

y

k_1, k_2

choose random r

$r, \quad t = r\cdot(y+k_1) + k_2$

$s = x+k_1$

$u = t - r\cdot s$

compute:

$u - k_2 = r\cdot(x-y)$

Bob learns nothing about x

Server sees r, t, s

Alice learns $r\cdot(x-y)$

Our approach: scale

Every user has many friends

6 bytes / friend

6 bytes / friend

(when Bob moves)

$r_i (x - y_i) \bmod p$

Total traffic per phone: about 3□12□n bytes (every 5 mins)

(in reality, slight more due to engineering constraints)

Summary

Many location services possible without revealing location:

- Private proximity alerts
- Private location statistics (hotspots)

Many more examples …

Stanford | Center for Professional Development

Mobile Security

Abusing Mobile Sensors

Stanford | Center for Professional Development

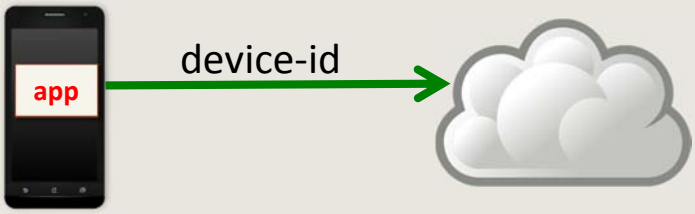
Example 1: device fingerprinting

IdentifierForVendor: returns instance of NSUUID class

- Returns same device ID for for all apps by same vendor on the device
- Backed and restored via iTunes … **but can be reset by user**

Accelerometer gives a stable device fingerprint [BBMN'14, DRXCN'14]

- App. can tell if it has been previously installed on device



Example 2: Power usage sensor

Modern phones measure power drained from battery

Enables apps to optimize power use

Repeatedly read:

```
/sys/class/power_supply/battery/voltage_now
/sys/class/power_supply/battery/current_now
```

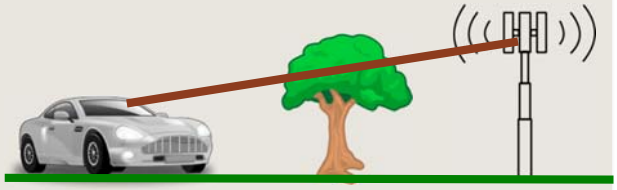
Unrestricted access.

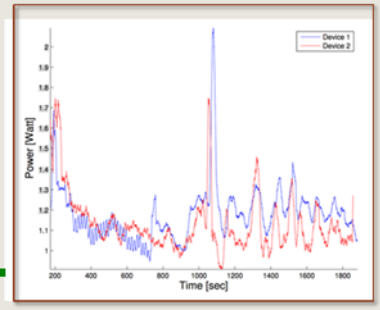
Can this be abused?

Example 2: Power usage sensor

Can this be abused? [MBSN'15]

Observation: power used by radio depends on distance and obstacles to cell tower





Stanford | Center for Professional Development

So what?

Our work: [Usenix Security 2015]

power readings + machine learning ⇒ GPS

Why? Routes in a city have unique power fingerprints

Three goals:

✓ 1. identify route car is taking among a known set of routes

✓ 2. identify car's location along a known route

✓ 3. identify car's route based on a database of pre-measured short segments

Stanford | Center for Professional Development

Lessons

Sensors can have unintended consequences

⇒ risk in giving apps direct access to sensors

Prevention:

- Always require permissions to access sensors
- Reduce data from sensors to min needed for utility or only provide abstract view of sensor data

Mobile Security

Mobile Device Management (MDM)

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

Models

HYOD (Here's Your Own Device)

CYOD (Choose Your Own Device)

BYOD (Bring Your Own Device): managed vs unmanaged

Stanford | Center for Professional Development

Requirements

- Device and OS Management
  - Configuration Management and Hardening
  - Confidentiality & Encryption
  - Remote Wipe

Stanford | Center for Professional Development

Requirements

- Device and OS Management
  - Backup / Recovery
  - Enterprise VPN & Proxy
  - Patch Management

Stanford | Center for Professional Development

Requirements

- Application Management
  - Enterprise App Stores
  - Mobile Application Scanning
  - Malware Protection
- Monitoring & Enforcement
- Reporting

Stanford | Center for Professional Development

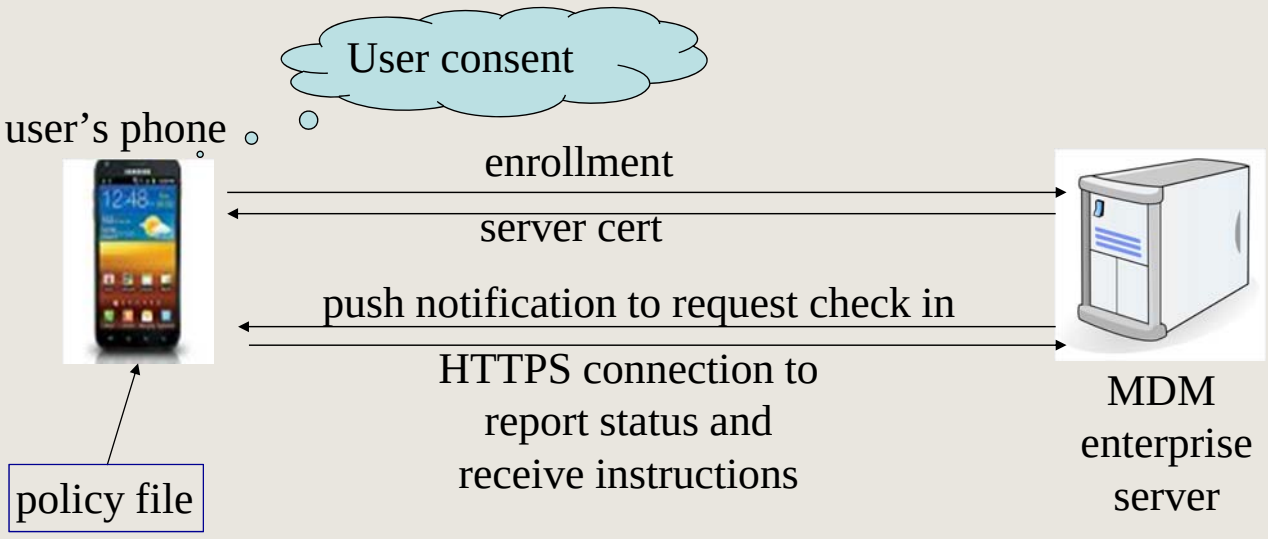
Mobile Security

MDM Architecture and Trade-offs

Stanford Advanced Computer Security Certificate

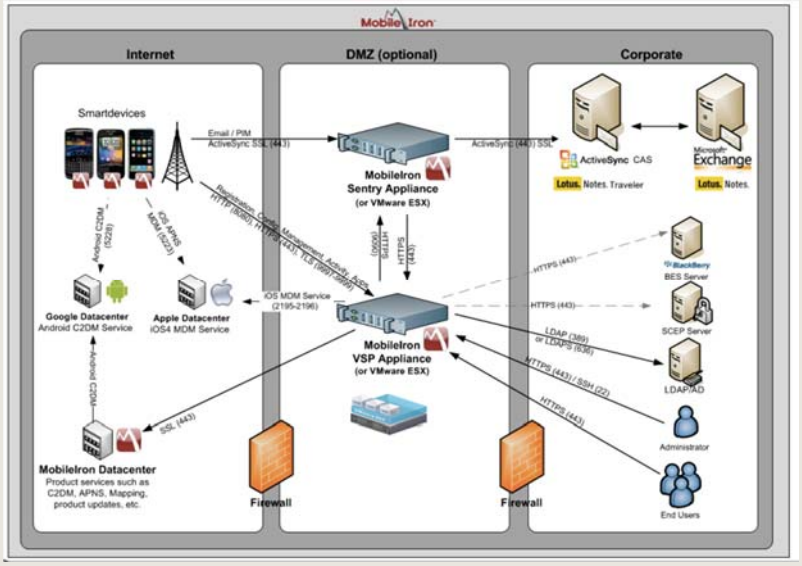
Stanford | Center for Professional Development

Strawman MDM Architecture



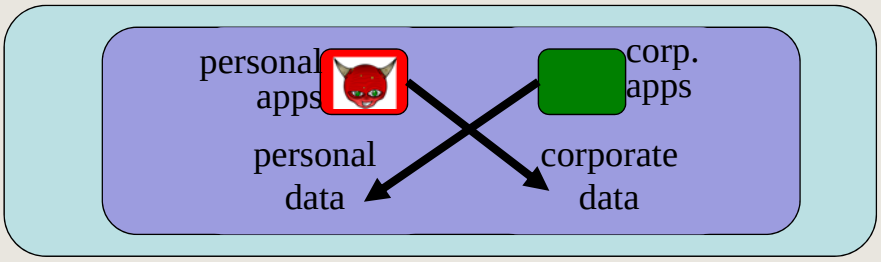
Stanford | Center for Professional Development

Representative Production MDM Architecture



Courtesy:
MobileIron

MDM Key Problem



MDM Architectures / Approaches

Device Policy Client:
Enterproid / Android for Work, deployed as an app

Virtual Machines (VMs) / Mobile Virtual Platform (MVP):
Separate VMs for personal and corporate (VMWare)

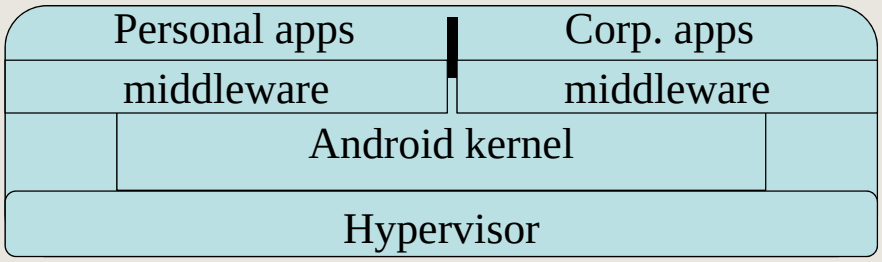
Virtual Mobile Device:
Turns your mobile into a dumb terminal / VDI for mobile
(Avast / Remotium)

Mobile Security

MDM Approach #1: Device Policy Content

Stanford Advanced Computer Security Certificate

Enterproid / Google for Work



https://static.googleusercontent.com/media/www.google.com/en/US/work/android/files/andro

Android Device Administration API

- BIND_DEVICE_ADMIN permission
- DeviceAdminInfo
  - USES_POLICY_LIMIT_PASSWORD USES_POLICY_WATCH_LO
  - USES_POLICY_RESET_PASSWORD
  - USES_POLICY_FORCE_LOCK
  - USES_POLICY_WIPE_DATA
  - USES_POLICY_SETS_GLOBAL_PROXY
  - USER_POLICY_EXPIRE_PASSWORD
  - USES_POLICY_ENCRYPTED_STORAGE
  - USES_POLICY_ENCRYPTED_STORAGE
  - USES_POLICY_DISABLE_CAMERA
  - USES_POLICY_DISABLE_KEYGUARD_FEATURES

Android Keyguard Unlock Methods

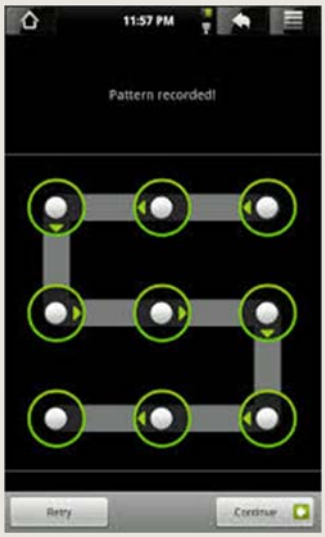
Slide

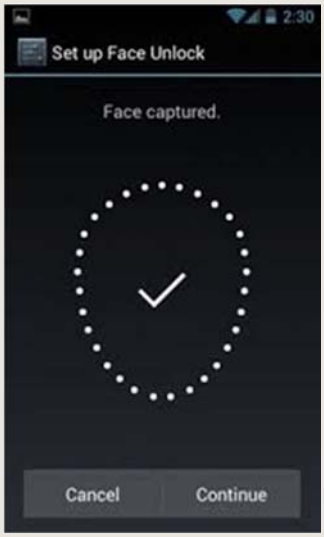
Face Unlock

Pattern

PIN

Password

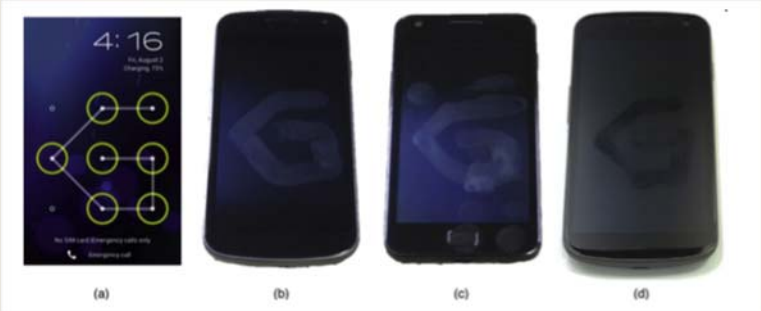




Attacks against Pattern Unlock

Smudge attacks [Aviv et al., 2010]

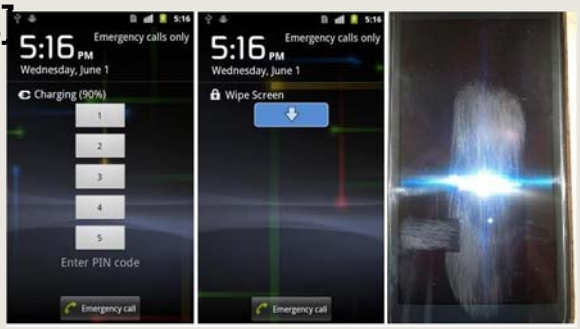
- Entering pattern leaves smudge that can be detected with proper lighting
- Smudge survives incidental contact with clothing



Attacks against Pattern Unlock

Potential defense [Moxie 2011]

- After entering pattern, require user to swipe across

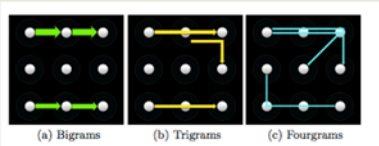


Another problem: entropy

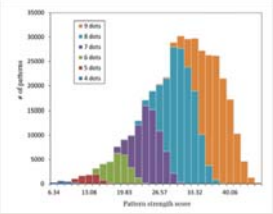
- People choose simple patterns – few strokes
- At most 1600 patterns with <5 strokes

Pattern Unlock Improvements

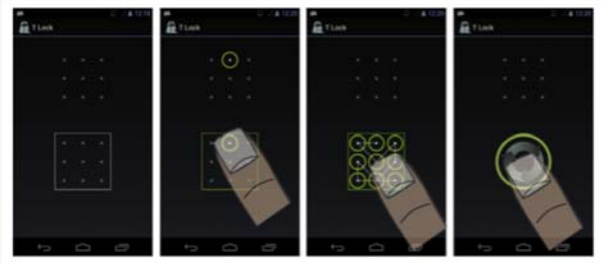
Most common patterns
[Andriotis et al 2013]



Strength Score [Sun et al., 2014]



TinyLock [Kwon et al., 2014]



Google Play for Work

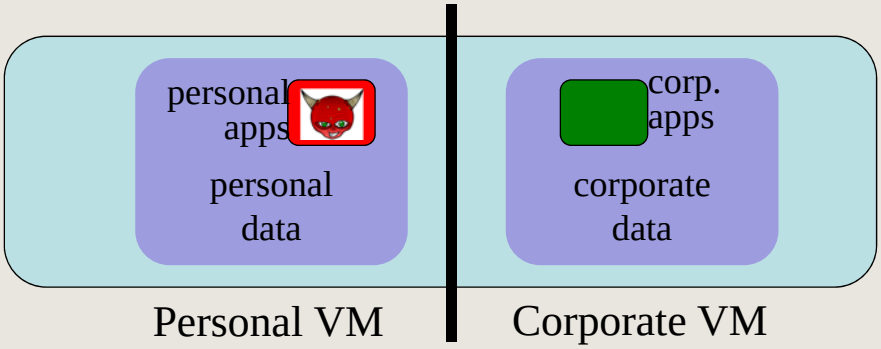
- Remotely install or remove apps
- Define which users should be able to see which apps
- Administrators can see which users have which apps installed

Mobile Security

MDM Approach #2: VM-Based

Stanford Advanced Computer Security Certificate

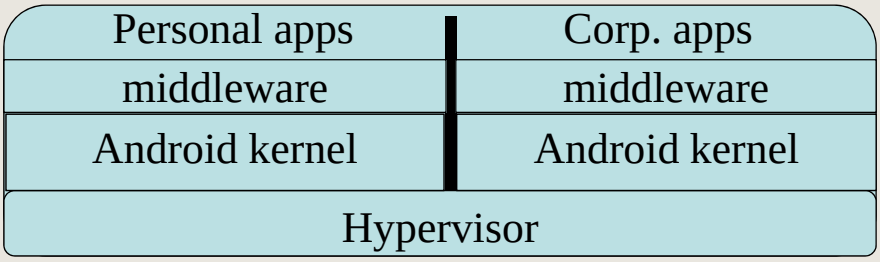
Virtual-machine-based approach



Virtual-machine-based approach

Corporate VM managed by corporate policy:

- Corporate app store, backup, management



Problems:

- Battery life time issues: two running versions of Android
- UI problem: which VM has focus? user can be confused
- Coarse: user needs two email progs, contact lists, etc.

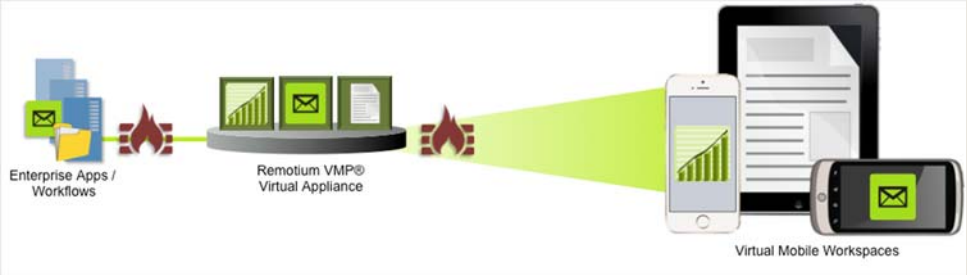
Mobile Security

MDM Approach #3: Virtual Mobile Device

Stanford Advanced Computer Security Certificate

Stanford | Center for Professional Development

Virtual Mobile Device



Personal apps Corp. app

Android kernel

Stanford | Center for Professional Development

Summary: Key questions to ask about MDM?

What is the architecture?

How does it achieve MDM requirements?

What are its pros / cons (enterprise manage-ability, impact on client, battery life, privacy implications)?

Stanford | Center for Professional Development