Remote Access + Mobile

CS155 Computer and Network Security

Stanford University

XMLHttpRequest + CORS Clarification

XMLHttpRequest

Incorrect: Website cannot make any XMLHttpRequests that cross origins unless CORS pre-flight allows.

Reality: Browsers allow sites to make XMLHttpRequests in very specific situations without a CROS pre-flight request.

XMLHttpRequest Modes

Simple Requests

If *all* five conditions are met:

- Method one of {GET, HEAD, POST}
- Only “CORS-safelisted request-header” headers are set
- Content-type is one of application/x-www-form-urlencoded, multipart/form-data, text/plain
- No event listeners are registered on any XMLHttpRequestUpload object in req
- No ReadableStream object is used

SOP applies. These are the kinds of requests that web content can already issue. No data is released unless server sends CORS header

Preflighted requests

All other requests (e.g., DELETE or application/json type.) Or, if the website explicitly requests it.

A pre-flight **OPTIONS** request is sent to the web server. If the server provides a CORS header that provides permission, *then*, the browser will allow the request through.

```
Access-Control-Allow-Origin: https://foo.bar.org
Access-Control-Allow-Methods: POST, GET, DELETE
Access-Control-Max-Age: 86400
```

Remote Access

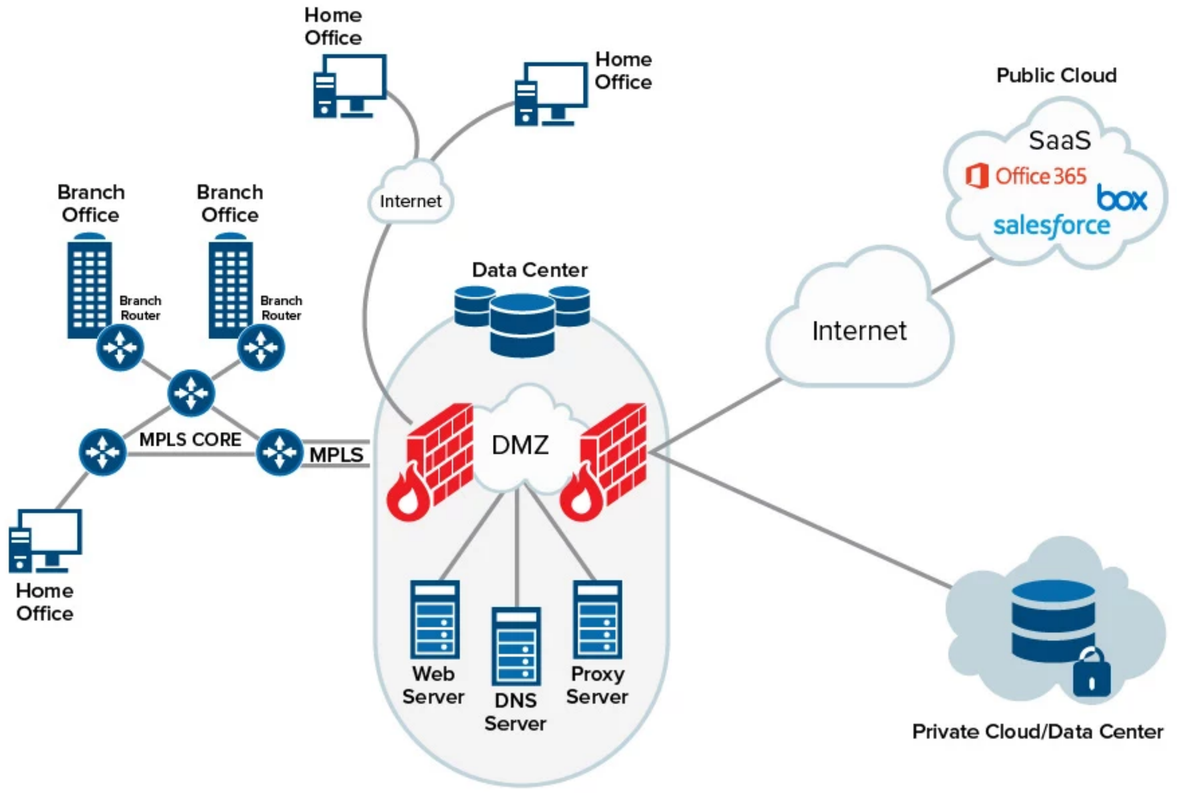
Traditional Network Model

Organization has a perimeter firewall in front of clients and servers

Some public facing servers are behind that firewall in “DMZ” (de-militarized zone)

Other servers and clients are behind a second firewall

VPN allowed remote clients to gain access behind second firewall



Virtual Private Networks (VPNs)

Problem: How do you provide secure communication for insecure protocols across the public Internet?

VPNs create a fake shared network on which traffic is encrypted

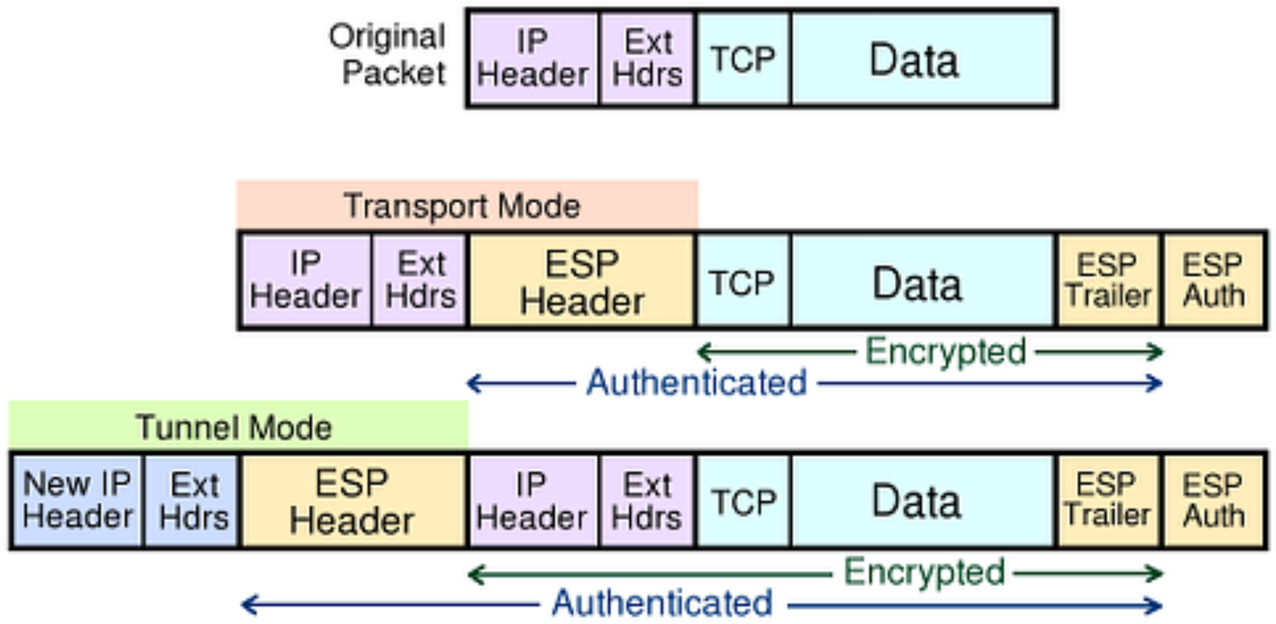
Two Broad Types:

- Remote client (e.g., traveler with laptop) to corporate network
- Connect two remote networks across Internet

IPSec

Several VPN protocols exist (PPTP, L2TP, IPsec, OpenVPN)

Most popular is IPsec. OpenVPN is open source.



Cisco AnyConnect

Stanford and many other organizations use Cisco AnyConnect

Encapsulates traffic in TLS! Initial handshake uses normal TCP-based TLS for initial handshake, HTTPS for client authentication, and then DTLS (UDP-based TLS) to transport data

Safest to build on well-known and tested cryptographic standards

WireGuard

New recently released VPN that many folks are excited about.
Much simpler than IPSEC and other protocols. Builds on modern cryptography.

Passed formal analysis of protocol

Cloudflare recently released a Rust implementation

BeyondCorp

VPNs support the idea of having a secure internal network and untrusted public Internet. Unfortunately, attacker has a ton of access once the network perimeter is breached.

Unfortunately, internal networks aren't *that* secure. Computers are compromised all the time and attackers have free reign.

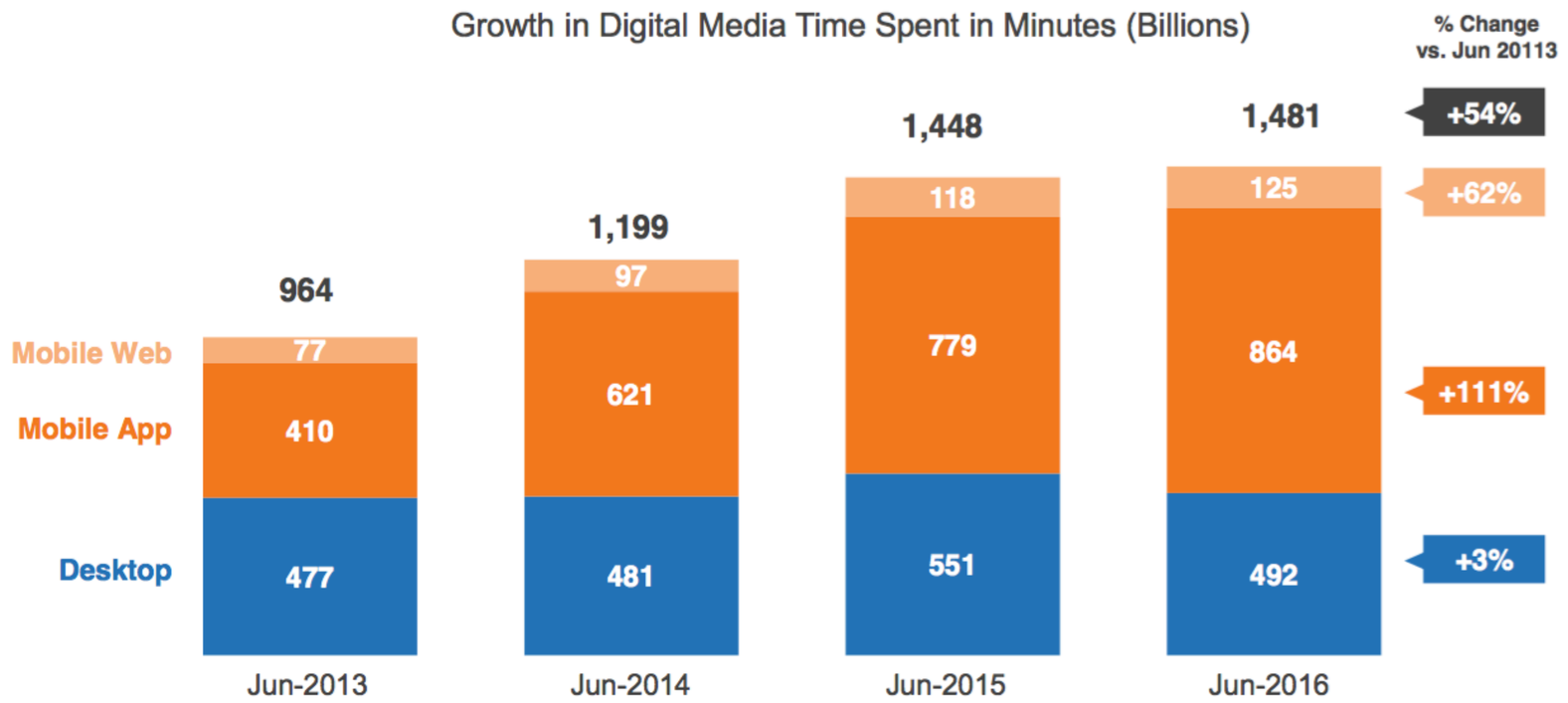
Google: assume internal network is *also* out to get you. Remove privileged intranet and put all corporate applications on the Internet.

Access depends solely on device and user credentials, regardless of a user's network location

Mobile Security

Mobile is Big!

Around 2B actively monthly Android users. Users spend more time on mobile than on desktops today.



comScore

Source: comScore Media Metrix Multi-Platform & Mobile Metrix, U.S.

Mobile Market Share

Android dominates global market.

Operating System	2017 Units	2017 Market Share (%)	2016 Units	2016 Market Share (%)
Android	1,320,118.1	85.9	1,268,562.7	84.8
iOS	214,924.4	14.0	216,064.0	14.4
Other OS	1,493.0	0.1	11,332.2	0.8
Total	**1,536,535.5**	**100.0**	**1,495,959.0**	**100.0**

Source: Gartner (February 2018)

Bring Your Own Device (BYOD)

Many companies are now allowing users to bring/use their own personal devices

In the past, enterprise workstations were centrally managed.

How do you handle when users want to bring their own devices?

What's Valuable on Phones?

Mobile Specific

– Identify location

– Record phone calls

– Log SMS (What about 2FA SMS?)

– Send premium SMS messages

Traditional (Similar to Desktop PCs)

- Steal personal data (e.g., contact list, email, messaging, banking/financial information, private photos)
- Phishing
- Malvertising
- Join Bots

Unique Threat Model (Physical)

Powered-off devices under complete physical control of an adversary (including nation states)

Screen locked devices under complete physical control of an adversary (e.g. thieves)

Screen unlocked devices under control of an authorized but different user (e.g. intimate partner abuse)

Devices in physical proximity to an adversary (with the assumed capability to control all available radio communication channels, including cellular, WiFi, Bluetooth, GPS, NFC, and FM)

Threat Model (Untrusted Code)

Android intentionally allows (with explicit consent by end users) installation of application code from arbitrary sources:

Abusing APIs supported by the OS with malicious intent, e.g. spyware

Exploiting bugs in the OS, e.g. kernel, drivers, or system services

Mimicking system or other app user interfaces to confuse users

Reading content from system or other application user interfaces (e.g., screen-scrape)

Injecting input events into system or other app user interfaces

Unique Threat Model (Network)

The standard assumption of network communication under complete control of an adversary certainly also holds for Android. Assume fist hop (e.g., router) is also malicious.

Passive eavesdropping and traffic analysis, including tracking devices within or across networks (e.g. based on MAC address or other device network identifiers.)

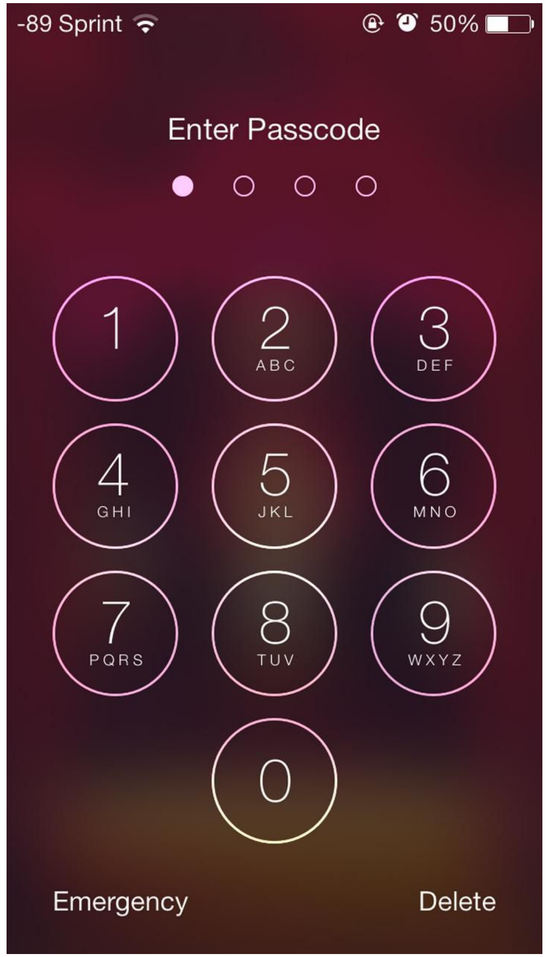
Active manipulation of network traffic (e.g. MITM on TLS.)

Physical Security

Unlocking Device

Typically: Need PIN, pattern, or alphanumeric password to unlock device

Some applications (e.g., banking apps) also require entering a PIN to access the app



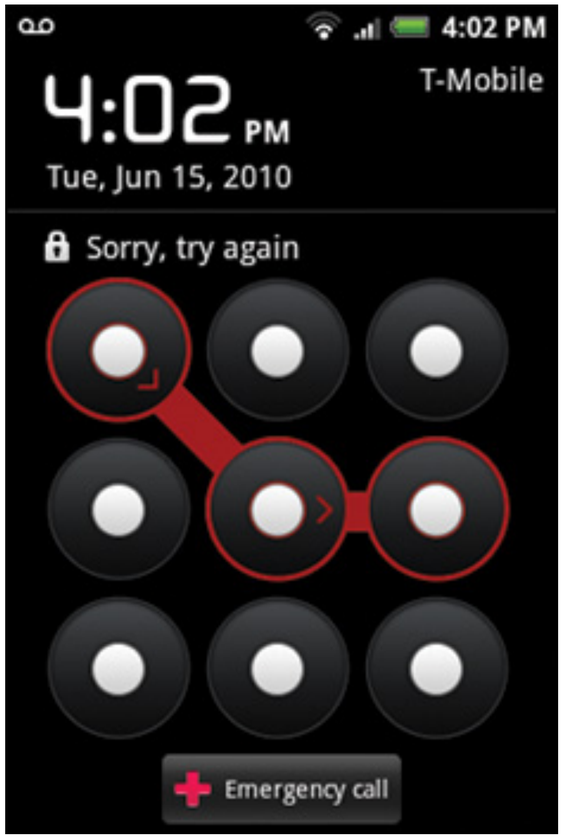
Swipe Code Problems

Smudge attacks [Aviv et al., 2010]

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

Another problem: entropy

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



Passcodes + Passwords More Secure

How do you allow only having a
4-6 digit PIN and still be secure?

Traditional Password Hashing

How are passwords typically stored? In Linux (and most web apps), you store hash of password and salt.

Offline Attack

- Steal pwd file, try hashing all passwords + salt
- Cannot reverse a hash, but can try dictionary

Online attack

- Can you try all passwords at a web site?

iPhone Unlocking (1)

Every iPhone has an additional secure processor known as the secure enclave. Memory is inaccessible to normal OS. Utilizes a secure boot process that ensures its software is signed.

Each secure enclave has an AES key burned in at manufacture. The hardware is designed such that the processor has instructions that allow encrypting and decrypting content using that key, but the key itself is never accessible (including via JTAG)

iPhone Unlocking (2)

User passcode is intertwined with AES key fused into secure enclave (known as UID). Imagine: key = $\text{Encrypt}_{\text{UID}}(\text{passcode})$.

This means that the the key to decrypt the device can only be derived on the single secure enclave on a specific phone. Not possible to take offline and brute force.

iPhone Unlocking (3)

What prevents someone from quickly secure enclave repeatedly to try different passwords?

The passcode is entangled with the device's UID many times — requires approximately 80ms per password guess.

Imagine: $\text{Encrypt}_{\text{UID}}(\text{Encrypt}_{\text{UID}}(\text{Encrypt}_{\text{UID}}(\text{passcode})\ldots))$

iPhone Unlocking (4)

At 80ms per password check…

- 5.5 years to try all 6 digits pins
- 5 failed attempts ⇒ 1min delay, 9 failures ⇒ 1 hour delay
- >10 failed attempts ⇒ erase phone

FBI–Apple Encryption Dispute

After the San Bernardino shooting in 2016, FBI tried to compel Apple to “unlock” iPhone. What were they specifically requesting?

Not possible to make password guessing any faster—innately dependent on performance of burned-in AES key

FBI–Apple Encryption Dispute

Remember…

- 5 failed attempts ⇒ 1min delay, 9 failures ⇒ 1 hour delay
- >10 failed attempts ⇒ erase phone

This is managed by code on the secure enclave, which *can* be updated by Apple, not managed in hardware.

Technical Details

The court order wanted a custom version of a secure enclave firmware that would…

1. "it will bypass or disable the auto-erase function whether or not it has been enabled" (this user-configurable feature of iOS 8 automatically deletes keys needed to read encrypted data after ten consecutive incorrect attempts)

2. "it will enable the FBI to submit passcodes to the SUBJECT DEVICE for testing electronically via the physical device port, Bluetooth, Wi-Fi, or other protocol"

3. "it will ensure that when the FBI submits passcodes to the SUBJECT DEVICE, software running on the device will not purposefully introduce any additional delay between passcode attempts beyond what is incurred by Apple hardware”

What happened?

Apple planned to fight the order, “*The United States government has demanded that Apple take an unprecedented step which threatens the security of our customers. We oppose this order, which has implications far beyond the legal case at hand. This moment calls for public discussion, and we want our customers and people around the country to understand what is at stake.*”

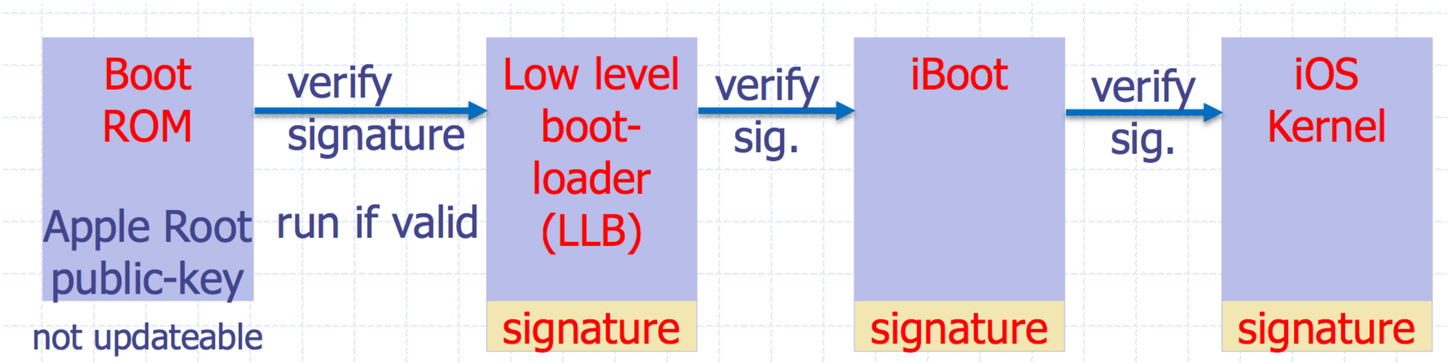
One day before hearing, FBI dropped the request, saying a third party had demonstrated a possible way to unlock the iPhone in question. No precent set re *all writs* act.

Secure Boot Chain

Why couldn't the FBI just upload their own firmware onto the secure enclave?

When an iOS device is turned on, it executes code from read-only memory known as Boot ROM. This immutable code, known as the hardware root of trust, is laid down during chip fabrication, and is implicitly trusted.

The Boot ROM code contains the Apple Root CA public key, which is used to verify that the bootloader is signed by Apple. This is the first step in the chain of trust where each step ensures that the next is signed by Apple.



Software Updates

To prevent devices from being *downgraded* to older versions that lack the security updates, iOS uses *System Software Authorization*.

Device connects to Apple with cryptographic descriptors of each component update (e.g., boot loader, kernel, and OS image), current versions, a random nonce, and device specific Exclusive Chip ID (ECID).

Apple signs device-personalized message allowing update, which boot loader verifies.

Rooting

Allows user to run applications with root privileges, e.g., modify/delete system files and app, CPU, network management

Done by exploiting vulnerability in firmware to install a custom OS or firmware image

Double-edged sword… lots of malware only affects rooted devices

FaceID/TouchID

Files are encrypted through a hierarchy of encryption keys

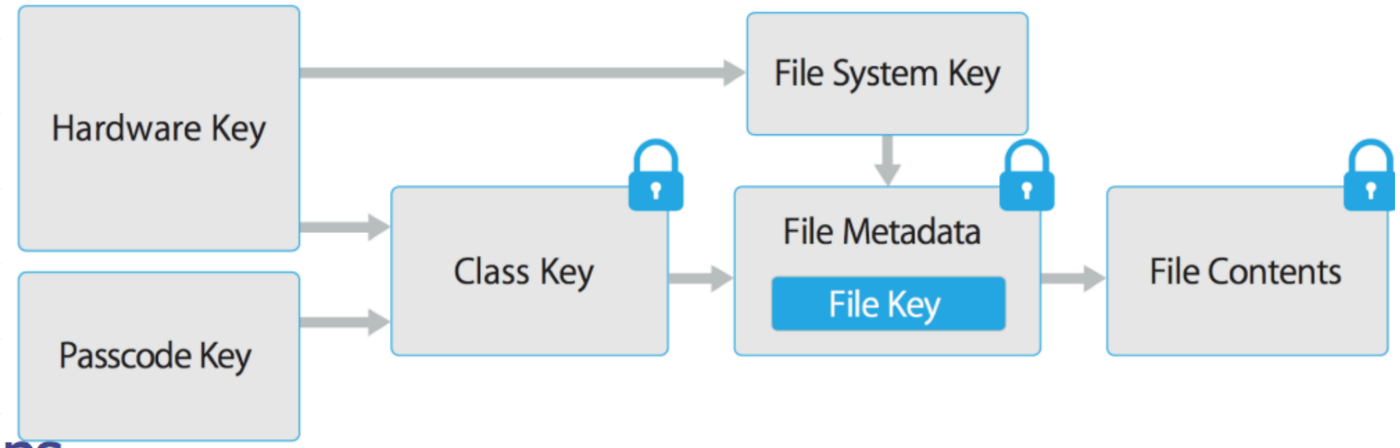
Application files written to Flash are encrypted:

- 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 (no passcode)

Resetting device deletes file-system key

All key enc/dec takes place inside the secure enclave

⇒ key never visible to apps



FaceID/TouchID

Files are encrypted through a hierarchy of encryption keys

By default (no FaceID, TouchID), class encryption keys are erased from memory of secure enclave whenever the device is locked or powered off

When TouchID/FaceID is enabled, the class keys are kept around and the hardware sensor sends fingerprint image to secure enclave. All ML/analysis is performed within the secure enclave.

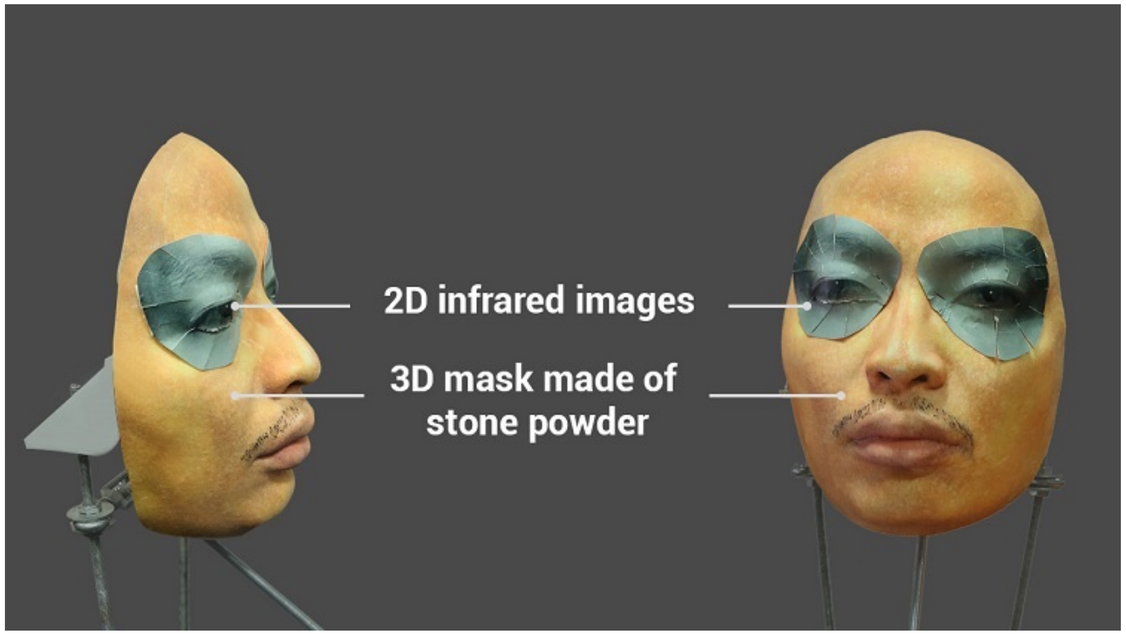
How Secure is TouchID?

Easy to build a fake finger if you have someone's fingerprint

- Several demos on YouTube. ~20 min
- Similar work on FaceID

The problem: fingerprints are not secret. Cannot replace.

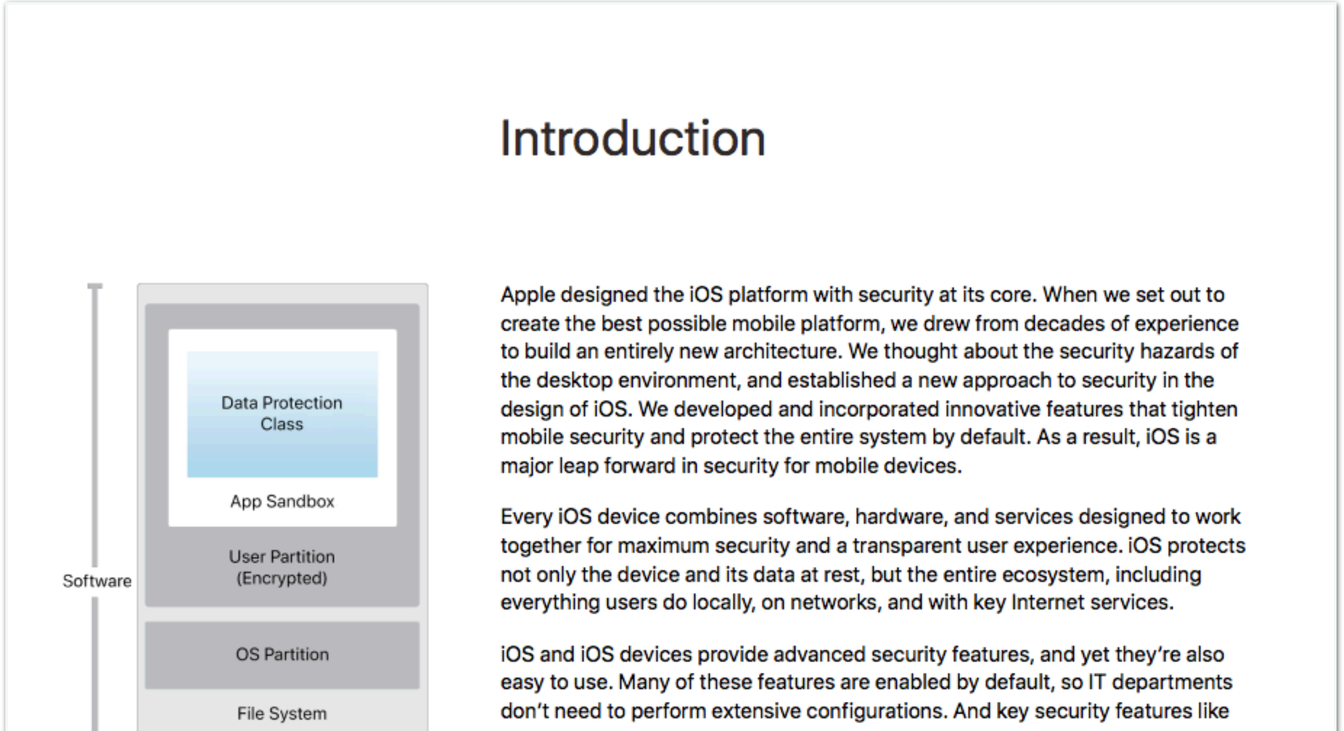
Convenient, but more secure solutions exist, e.g., unlock phone via bluetooth using a wearable device



More Information

iOS Security

https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf



Introduction

Apple designed the iOS platform with security at its core. When we set out to create the best possible mobile platform, we drew from decades of experience to build an entirely new architecture. We thought about the security hazards of the desktop environment, and established a new approach to security in the design of iOS. We developed and incorporated innovative features that tighten mobile security and protect the entire system by default. As a result, iOS is a major leap forward in security for mobile devices.

Every iOS device combines software, hardware, and services designed to work together for maximum security and a transparent user experience. iOS protects not only the device and its data at rest, but the entire ecosystem, including everything users do locally, on networks, and with key Internet services.

iOS and iOS devices provide advanced security features, and yet they're also easy to use. Many of these features are enabled by default, so IT departments don't need to perform extensive configurations. And key security features like

Mobile Device Management

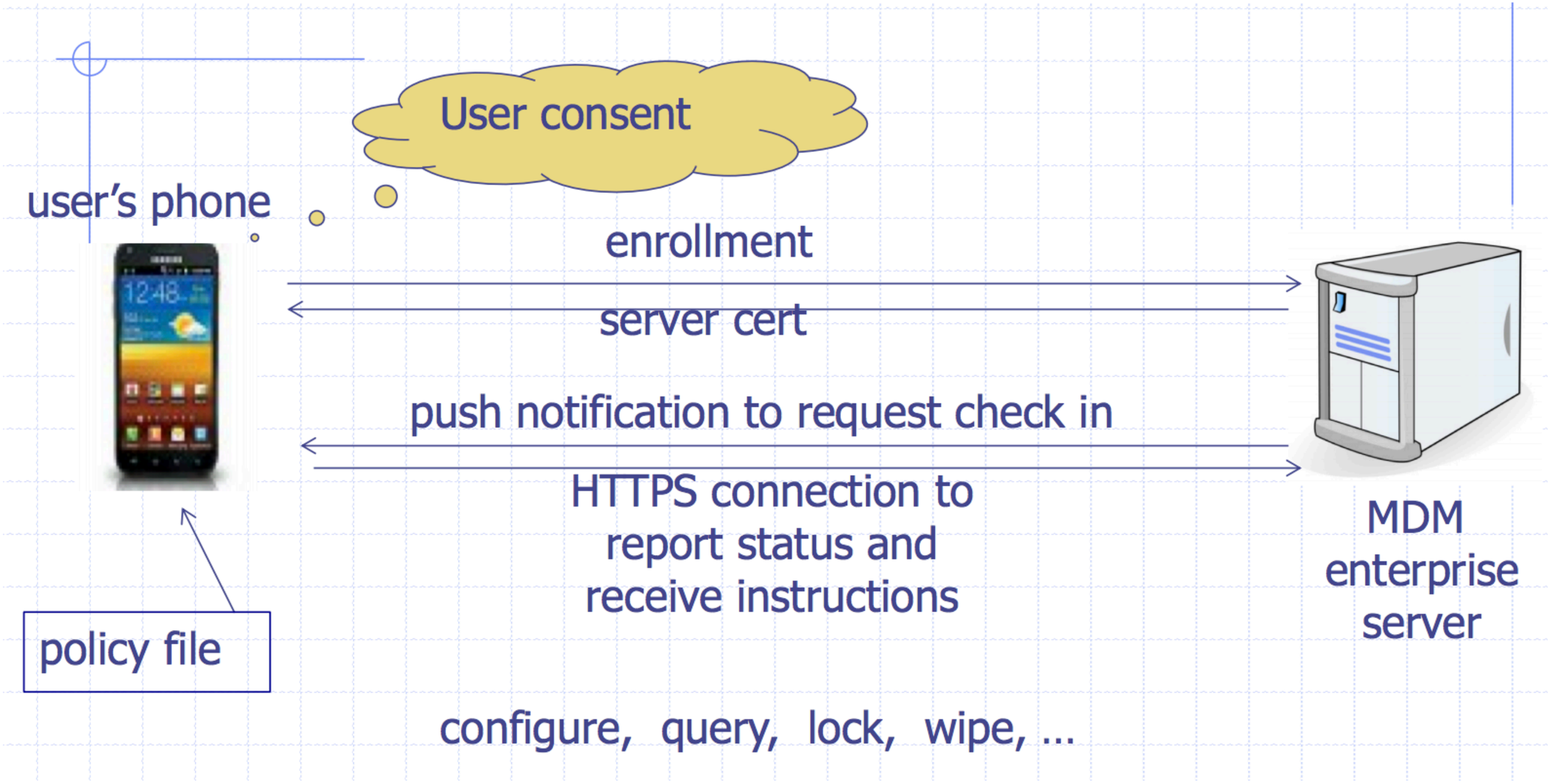
Manage mobile devices across organization

Consists of central server and client-side software. Now part of many mobile OSes too.

Allows:

- Diagnostics, repair, and update
- Backup and restore
- Policy enforcement (e.g. only allowed apps)
- Remote lock and wipe
- GPS Tracking

Sample MDM Enrollment



Mobile Malware

What's Different?

Applications are isolated

- Each runs in a separate execution context
- No default access to file system, devices, etc.
- Different than traditional OSes where multiple applications run with the same user permissions!

Applications are installed via App Store (and malware spreads)

- Market: Vendor controlled (Apple) / open (Android)
- User approval of permissions

Android Isolation

Based on Linux with Application sandboxes (using SE Linux)

- Applications run as separate UIDs, in separate processes.
- Memory corruption errors only lead to arbitrary code execution in the context of the particular application, not complete system compromise!
- Can still escape sandbox – but must compromise Linux kernel to do so

Examples of Malware

DroidDream (Android)

- Over 58 apps uploaded to Google app market
- Conducts data theft; send credentials to attackers

Attacked vulnerability in Android itself

Zitmo (Symbian, BlackBerry, Windows, Android)

- Poses as mobile banking application
- Captures info from SMS – steal banking 2FA codes
- Works with Zeus botnet

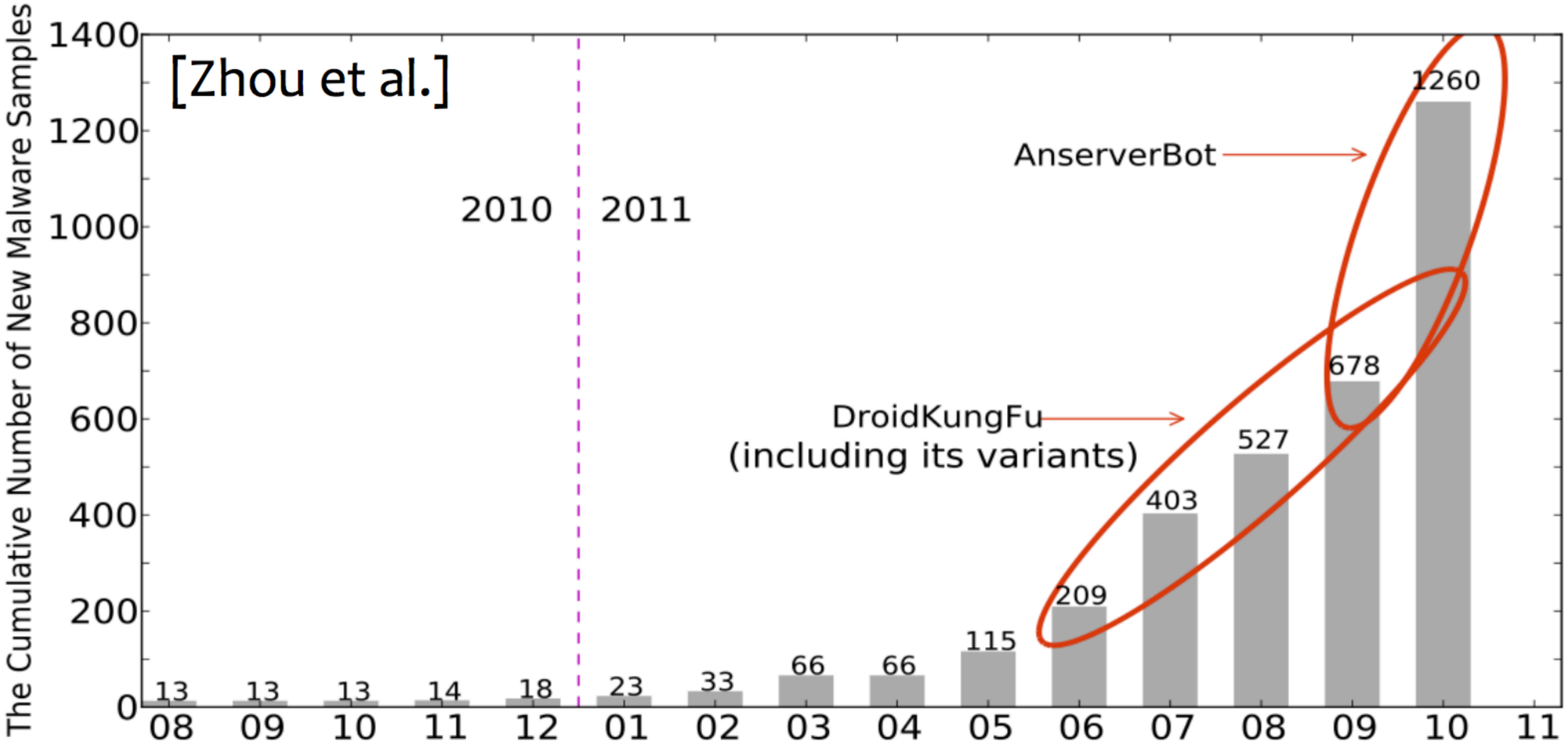
Malicious application that tricked users

Ikee (iOS)

- Worm capabilities (targeted default ssh password)
- Worked only on jailbroken phones with ssh installed

Attacked vulnerability in rooted iPhones

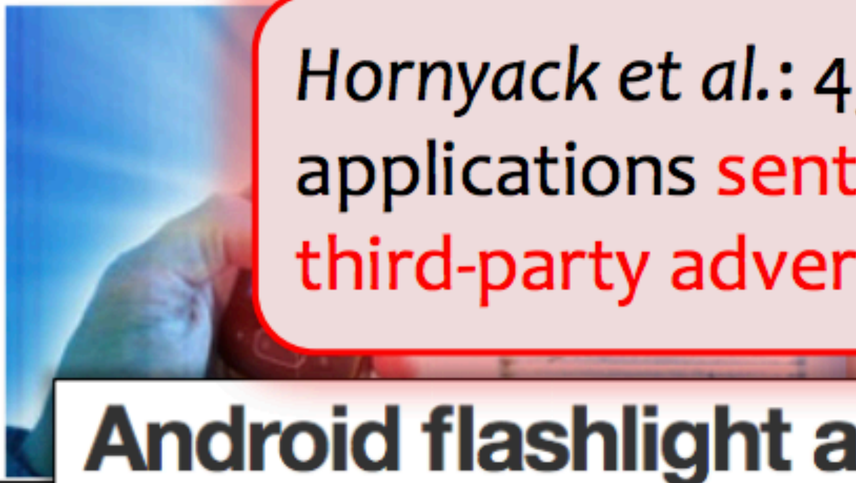
Large Target for Attackers



Legitimate Apps Too…

Top Mobile Apps Overwhelmingly Leak Private Data: Study

By Robert Lemos | Posted 2013-07-31 Email Print



Hornyack et al.: 43 of 110 Android applications sent location or phone ID to third-party advertising/analytics servers.

paid apps
pplication-

isk more often
more likely to
applications,
ons

Android flashlight app tracks users via GPS, FTC says hold on

By Michael Kassner in IT Security, December 11, 2013, 9:49 PM PST

Challenges with Isolated Apps

So mobile platforms isolate applications for security, but….

1) **Permissions:** How can applications access sensitive resources?

2) **Communication:** How can applications communicate with each other?

(1) Permission Granting Problem

Smartphones (and other modern OSes) try to prevent such attacks by limiting applications' default access to:

– System Resources (clipboard, file system)

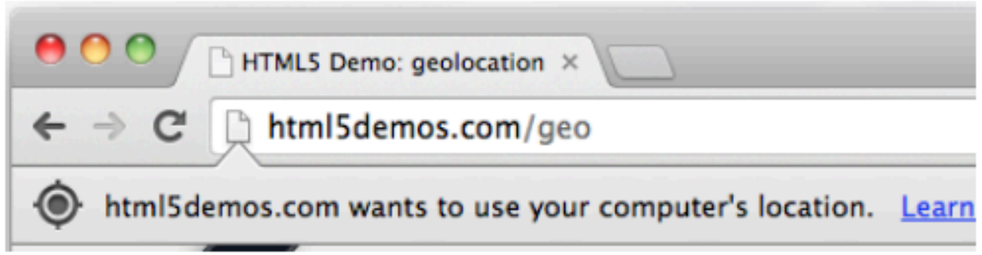
– Devices (e.g., camera, GPS, phone, …)

How should operating system grant permissions to applications?

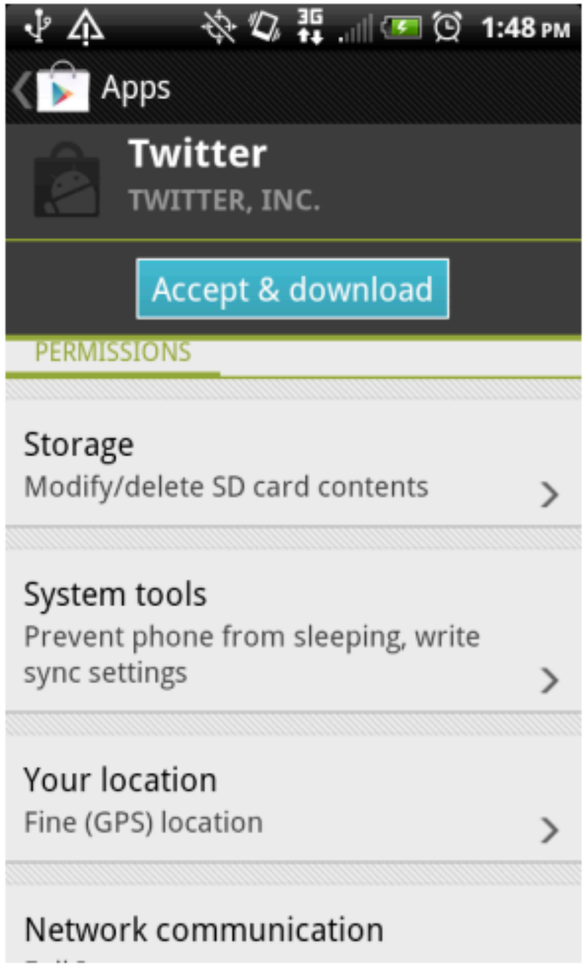
Standard approach: Ask the user.

State of the Art

Prompts (time-of-use)

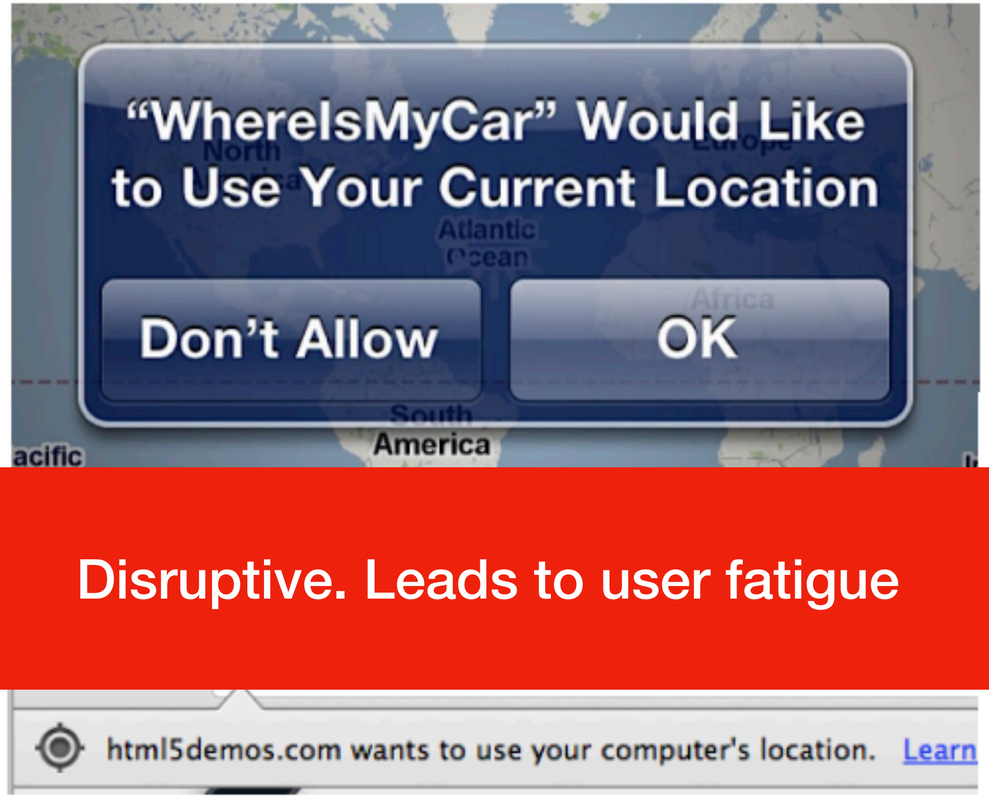


Manifests (install-time)



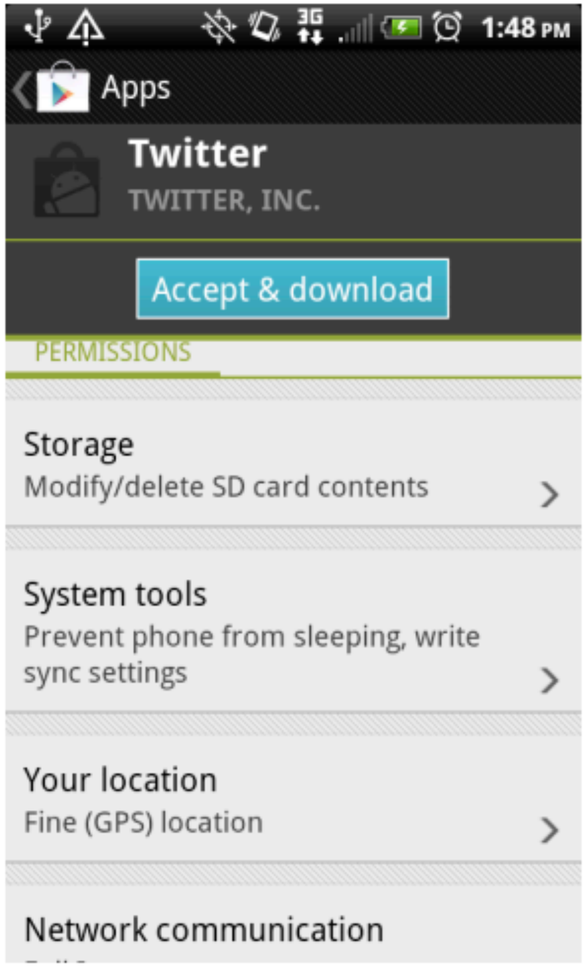
State of the Art

Prompts (time-of-use)



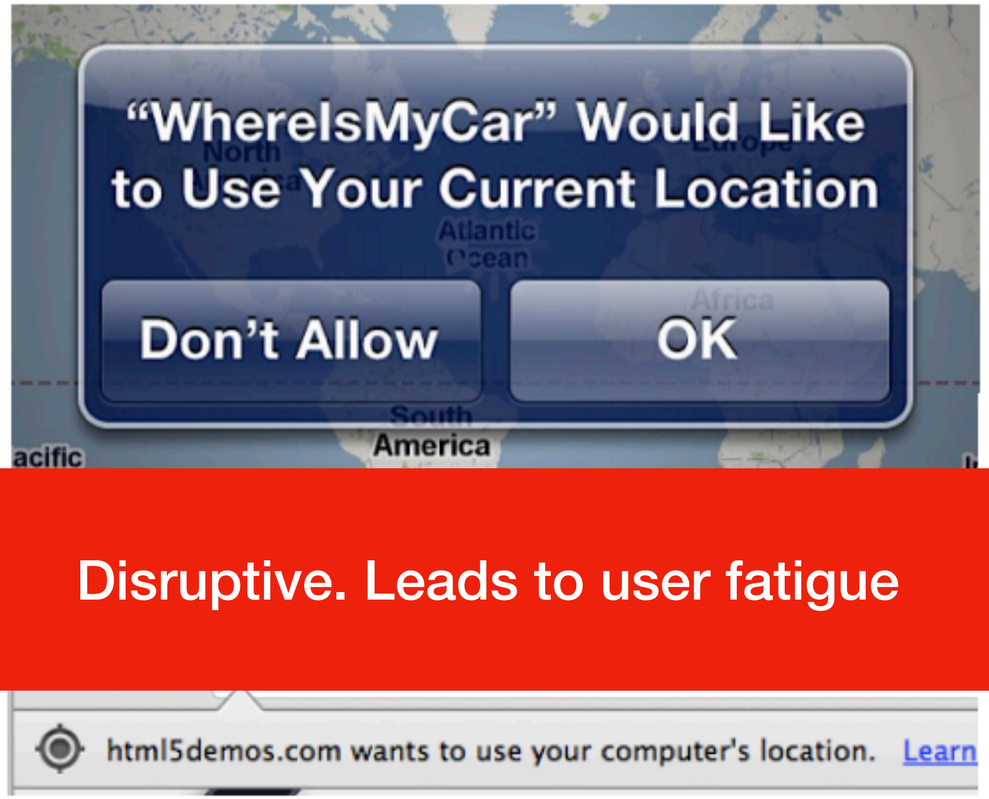
Disruptive. Leads to user fatigue

Manifests (install-time)



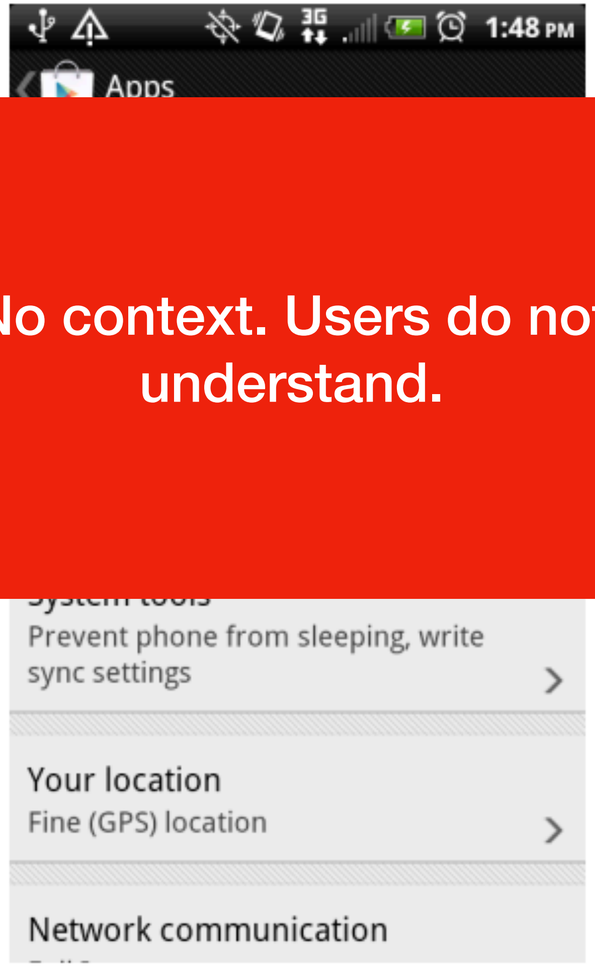
State of the Art

Prompts (time-of-use)



Disruptive. Leads to user fatigue

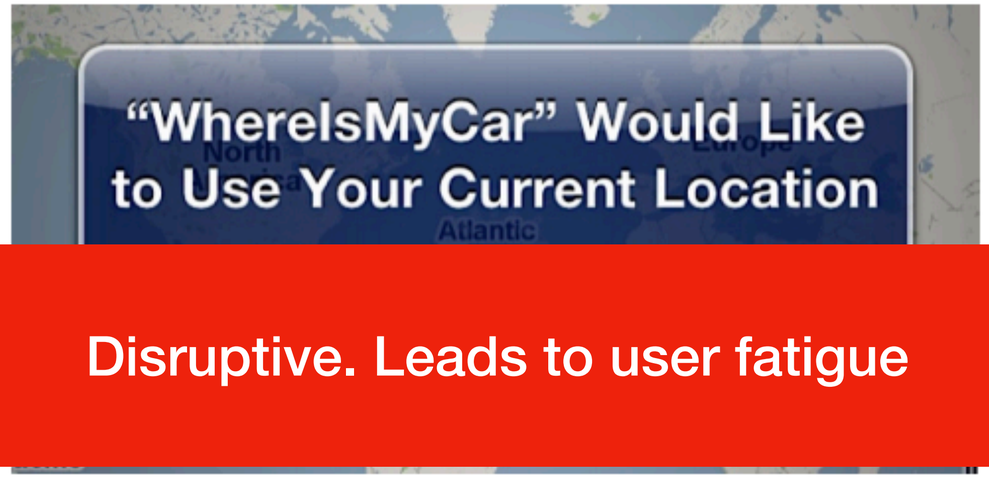
Manifests (install-time)



No context. Users do not understand.

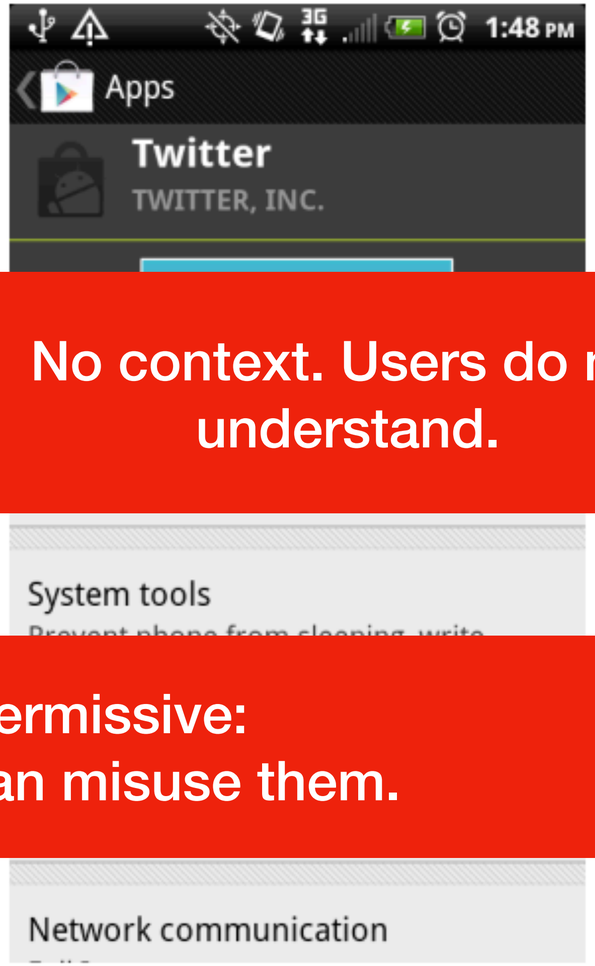
State of the Art

Prompts (time-of-use)



Disruptive. Leads to user fatigue

Manifests (install-time)

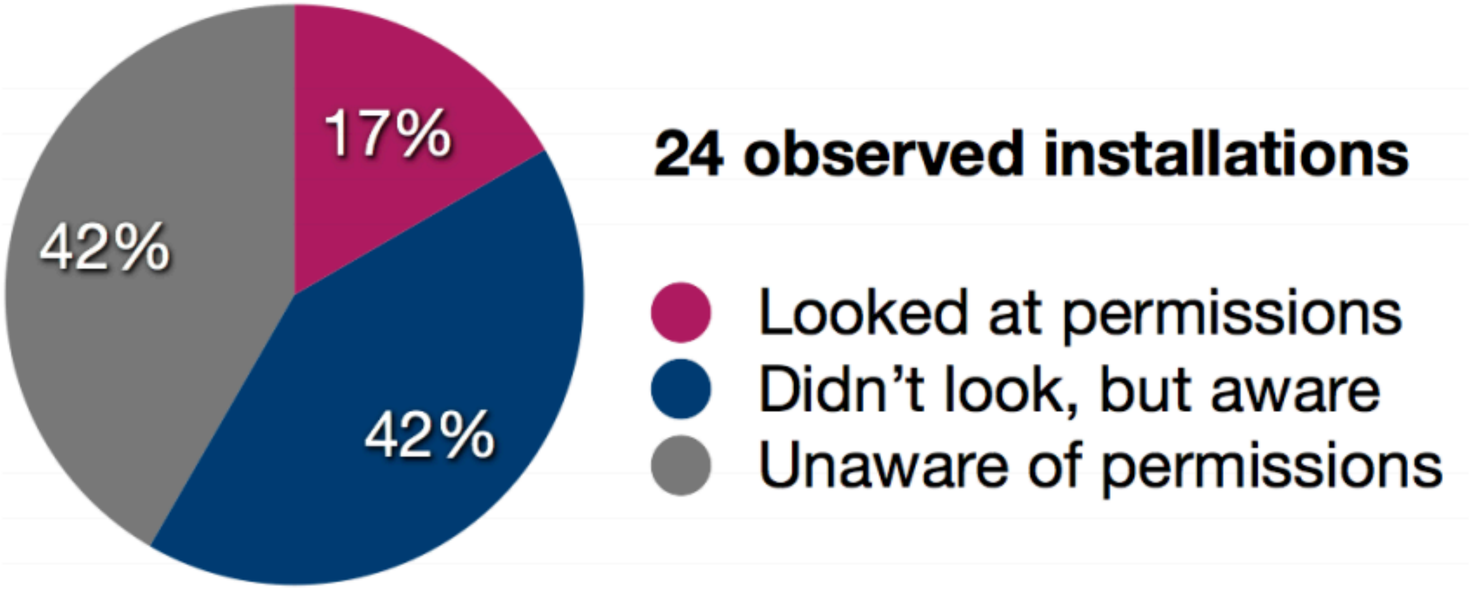


No context. Users do not understand.

In practice, both are overly permissive:
Once granted permissions, apps can misuse them.

Are Manifests Usable? (Felt et al)

Do users pay attention to permissions?



… but 88% of users looked at reviews.

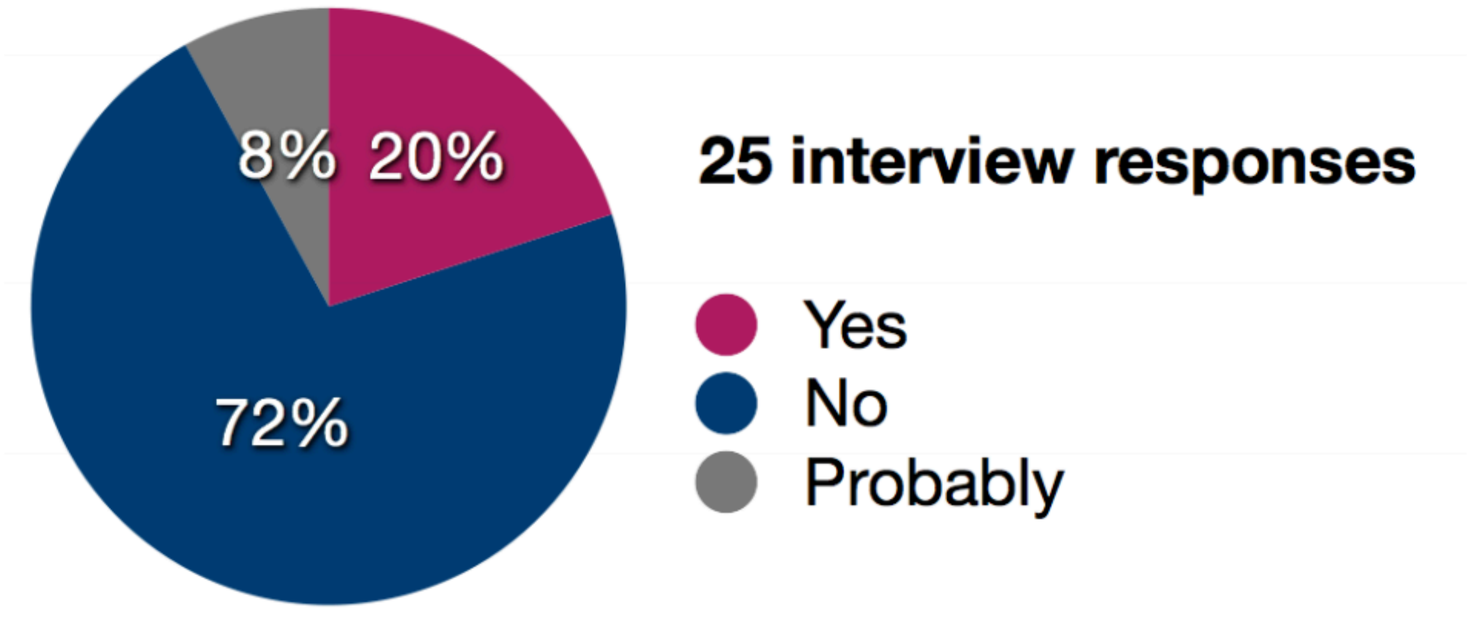
Do users understand the warnings?

	Permission	n	Correct Answers	
1 Choice	READ_CALENDAR	101	46	45.5%
	CHANGE_NETWORK_STATE	66	26	39.4%
	READ_SMS$_1$	77	24	31.2%
	CALL_PHONE	83	16	19.3%
2 Choices	WAKE_LOCK	81	27	33.3%
	WRITE_EXTERNAL_STORAGE	92	14	15.2%
	READ_CONTACTS	86	11	12.8%
	INTERNET	109	12	11.0%
	READ_PHONE_STATE	85	4	4.7%
	READ_SMS$_2$	54	12	22.2%
4	CAMERA	72	7	9.7%

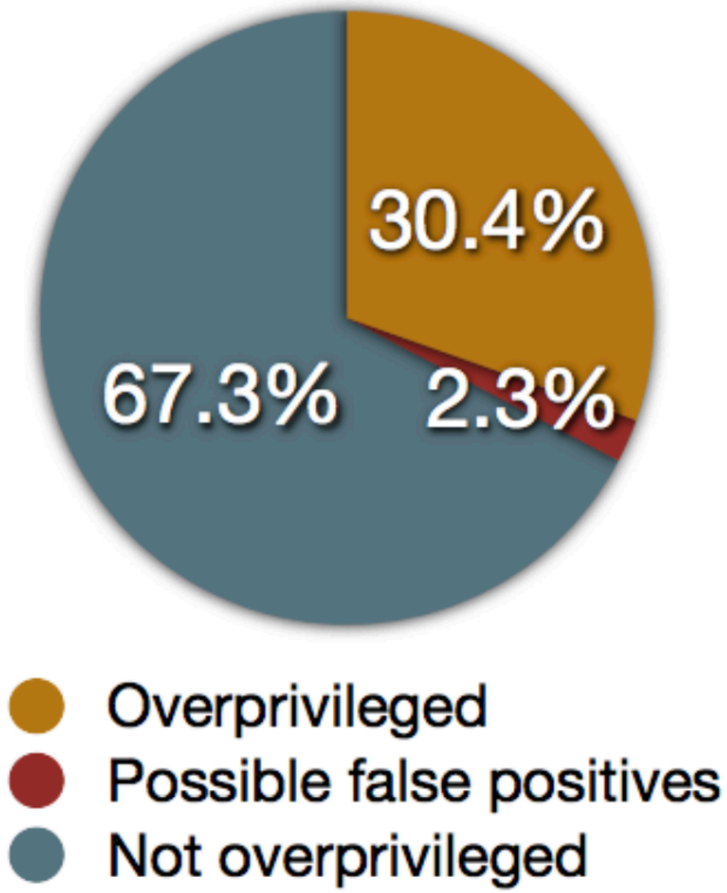
Table 4: The number of people who correctly answered a question. Questions are grouped by the number of correct choices. n is the number of respondents. (Internet Survey, $n = 302$)

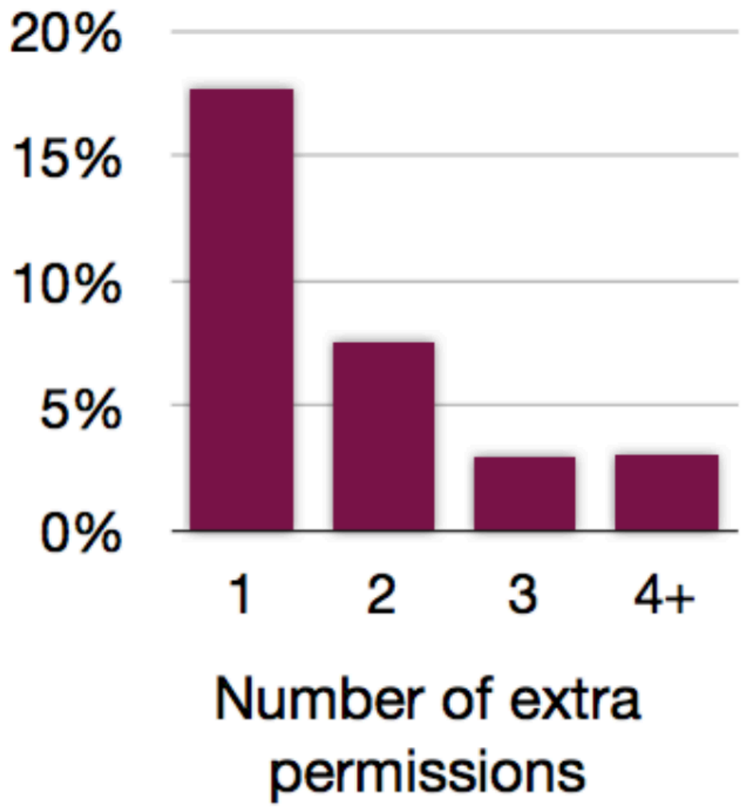
Do users act on permission information?

"Have you ever not installed an app because of permissions?"

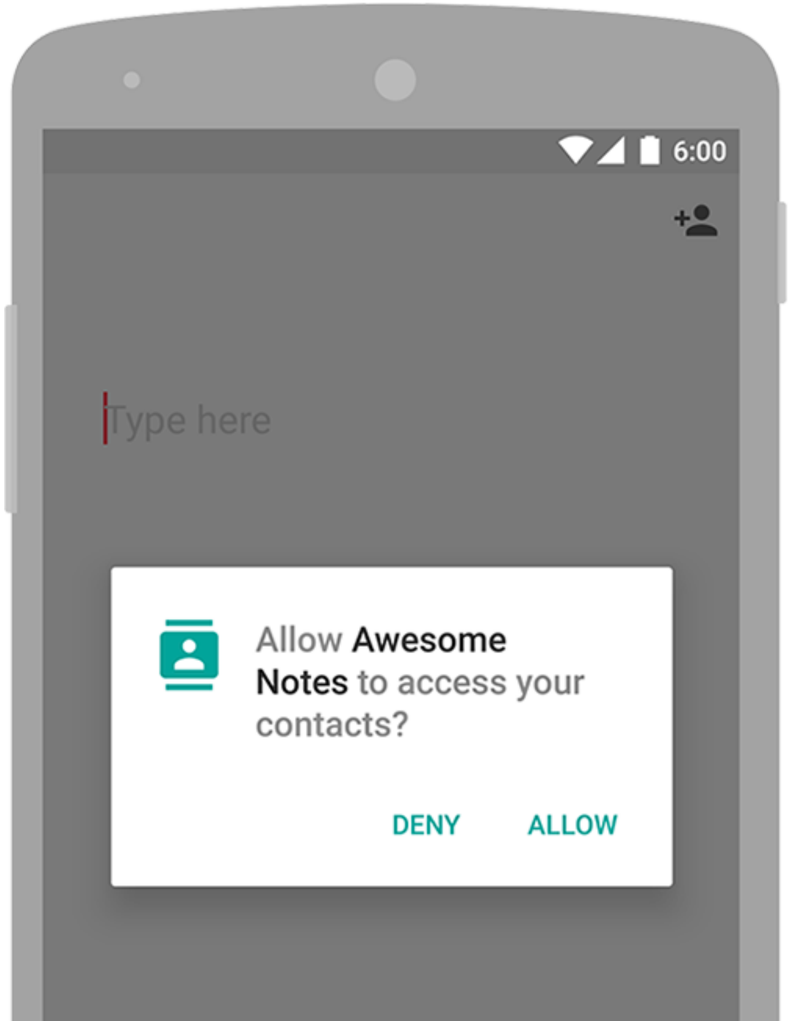


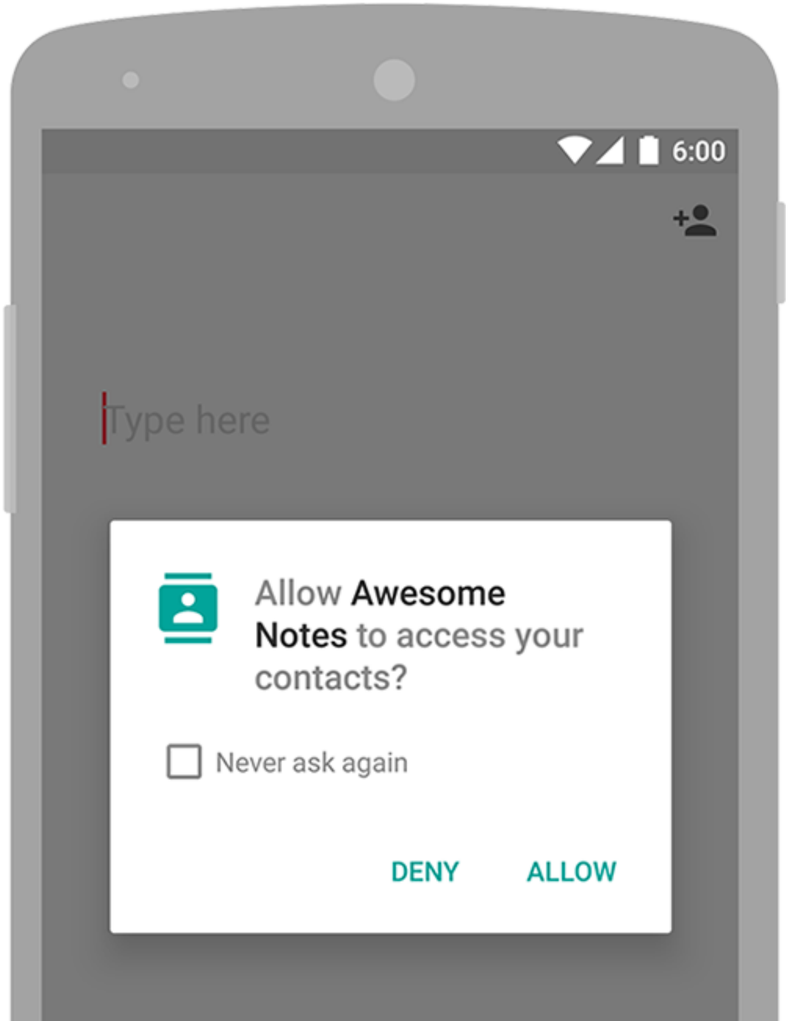
Developers Don’t know the Permissions They Need





Android Now Asks at Runtime (was not the case historically)





Manifests

In both cases, the Android app needs to request permission in its manifest—it's just up to the Operating System when it asks the user.

The OS might also just grant the right it doesn't seem dangerous

Manifest also defines what endpoints *other* endpoints can access. Whole class of malware that takes advantage of this of misconfiguration.

Inter-Process Communication

Primary mechanism for IPC between application components in Android:
Intents

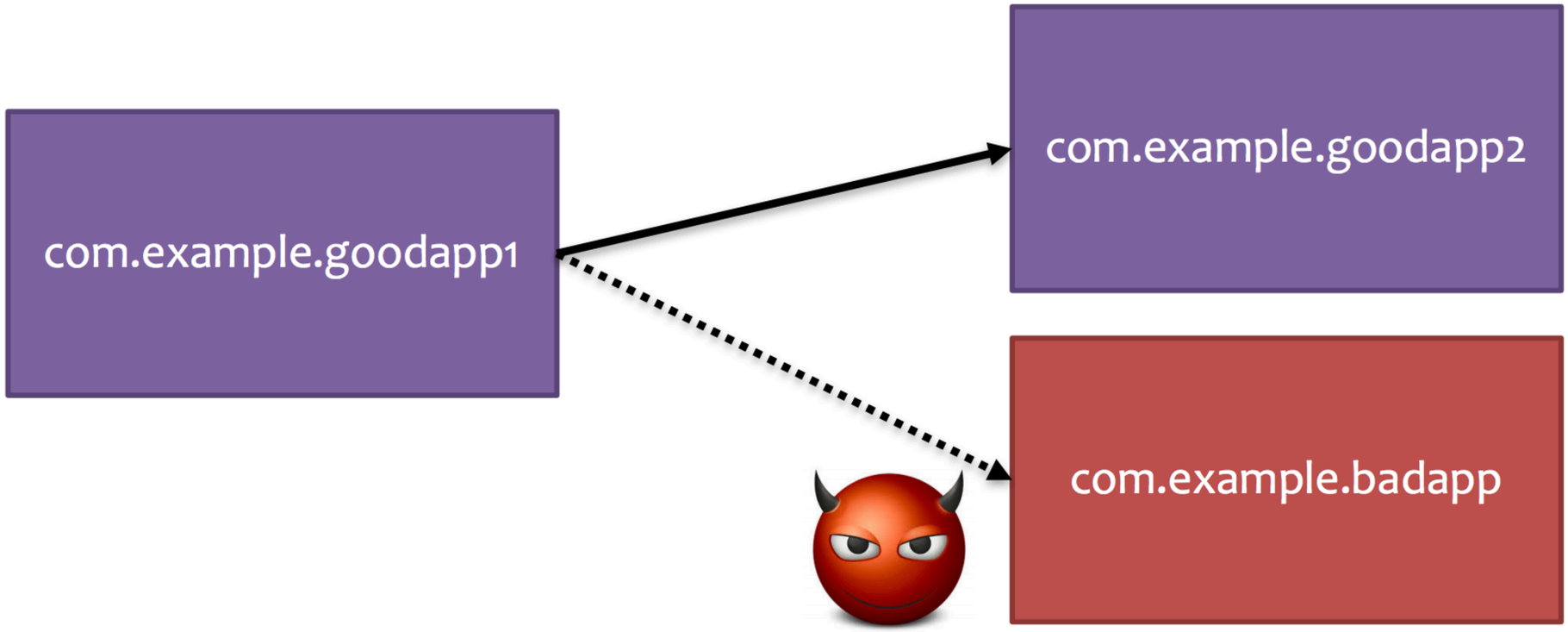
Explicit: specify name: e.g., com.example.testApp.MainActivity

Implicit: Specify action (e.g., ACTION_VIEW) and/or data (URI & MIME type)

An implicit intent specifies an action that can invoke any app on the device able to perform the action. Using an implicit intent is useful when your app cannot perform the action, but other apps probably can and you'd like the user to pick which app to use.

Intent Eavesdropping

Attack #1: Eavesdropping / Broadcast Theft



Unauthorized Intent Receipt

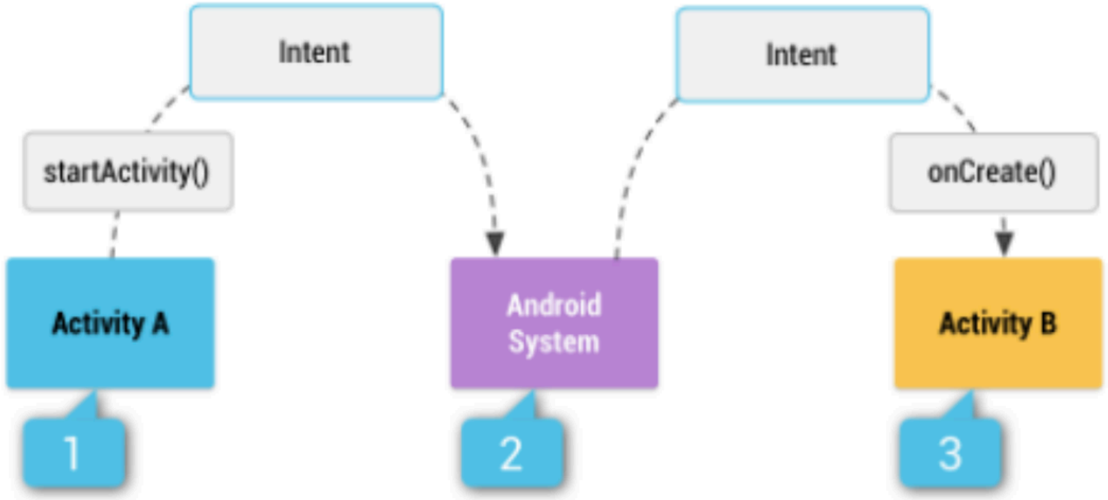
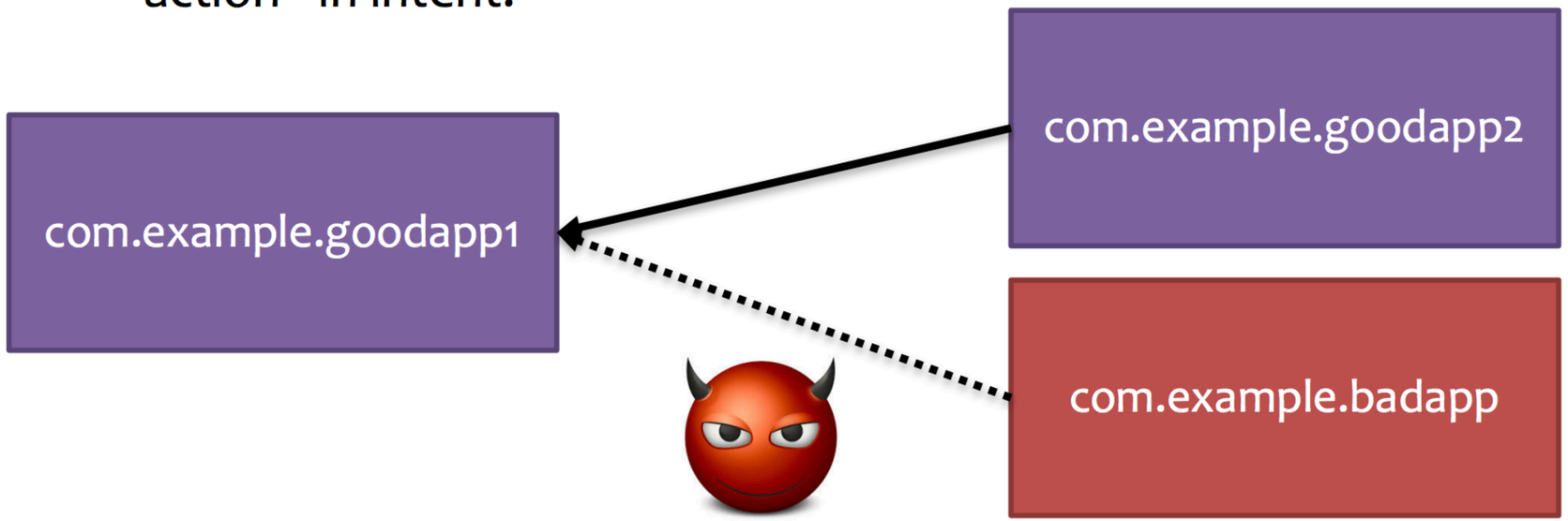


Figure 1. How an implicit intent is delivered through the system to start another activity: **[1]** *Activity A* creates an `Intent` with an action description and passes it to `startActivity()`. **[2]** The Android System searches all apps for an intent filter that matches the intent. When a match is found, **[3]** the system starts the matching activity (*Activity B*) by invoking its `onCreate()` method and passing it the `Intent`.

"Caution: To ensure that your app is secure, always use an explicit intent when starting a Servier. Using an implicit intent to start a service is a security hazard because you can't be certain what service will respond to the intent, and the user can't see which service starts."

Intent Spoofing

- Attack #1: General intent spoofing
  - Receiving implicit intents makes component public.
  - Allows data injection.
- Attack #2: System intent spoofing
  - Can't directly spoof, but victim apps often don't check specific "action" in intent.



Intent + Malware

Malware often times takes advantage of improperly filtered intents to gain access to the permissions in other applications