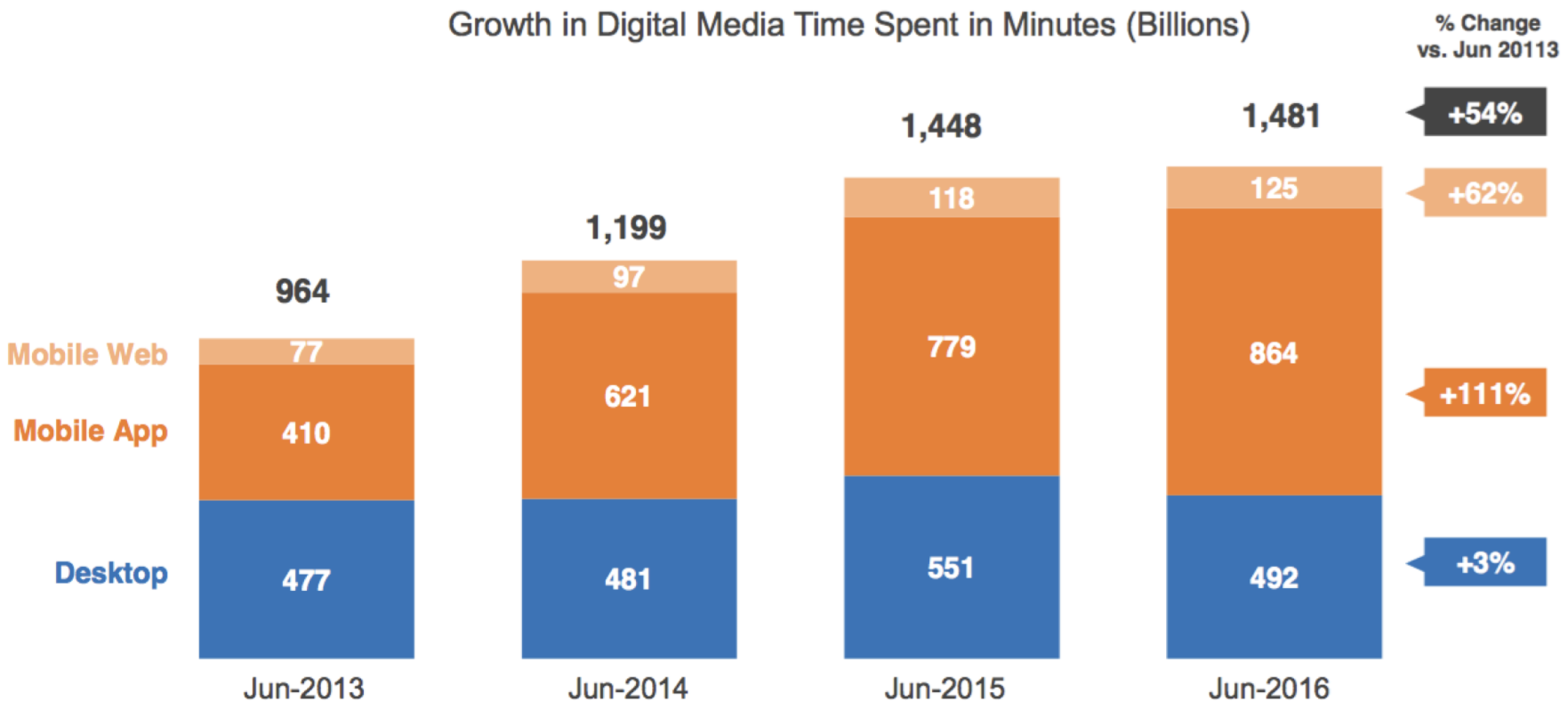
CS 5435

Mobile Security

Vitaly Shmatikov

Where Users Spend Time



comScore

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

What's Valuable on Phones?

Specific to mobile

- Identify location
- Record phone calls
- Log SMS (remember 2FA SMS?)
- Send premium SMS messages

Traditional

- Steal personal data: contact list, email, messaging, banking/financial information, private photos…
- Phishing
- Malvertising
- Join Bots

Physical Threats

Powered-off devices under complete physical control of an adversary

- Including well-resourced nation-states, police, etc.

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

Unlocked devices under control of different user (e.g. intimate partner abuse)

Devices in physical proximity to an adversary who control radio channels, including cellular, WiFi, Bluetooth, GPS, NFC

Untrusted Code

Android intentionally allows (with explicit user consent) installation of application code from arbitrary sources

- Abusing APIs supported by the OS with malicious intent, e.g. spyware
- Exploiting bugs in the OS
- 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

Network Threats

Network communication under complete control of an adversary

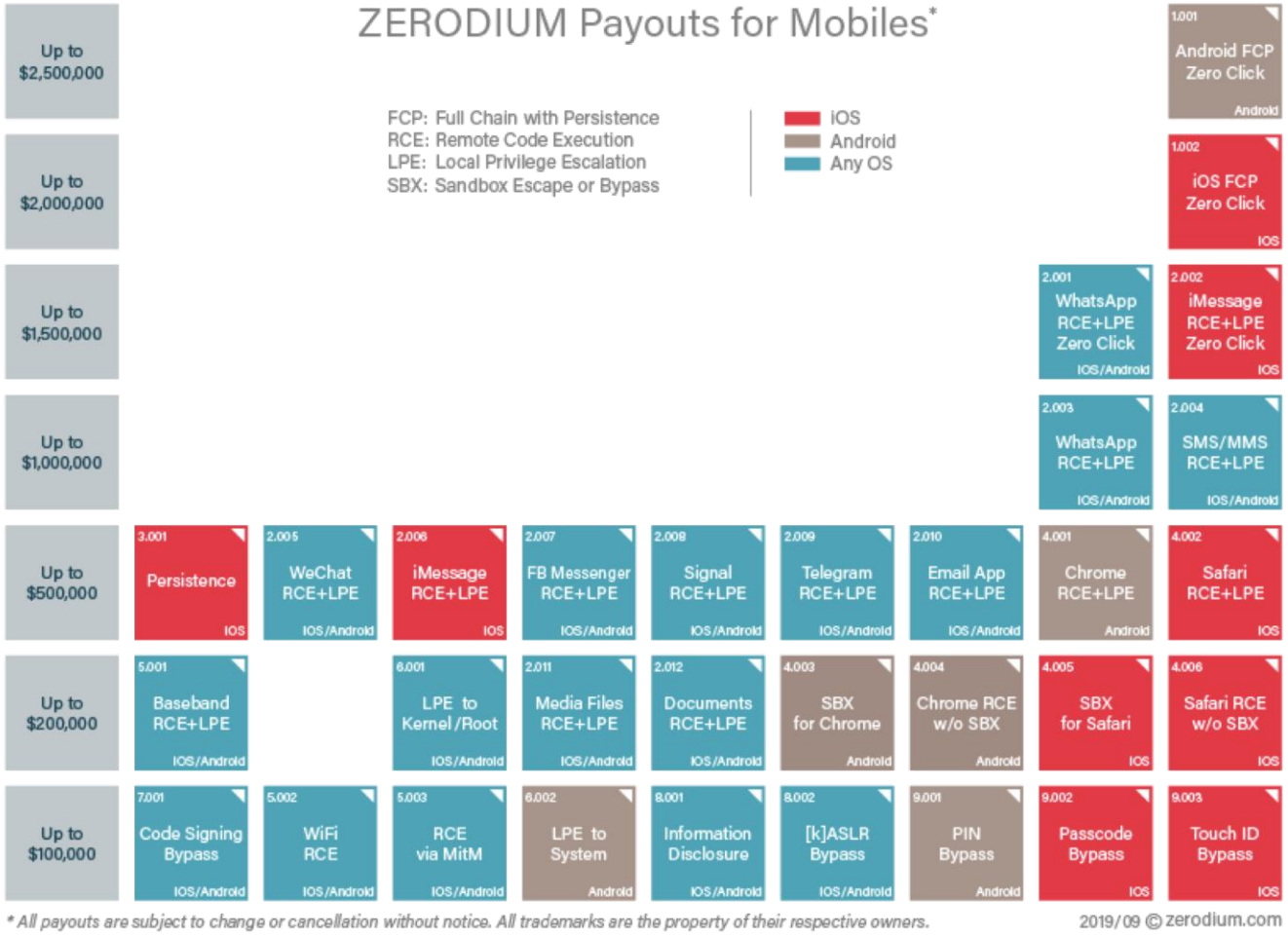
- Wi-Fi access points, possibly also routers

Passive eavesdropping and traffic analysis

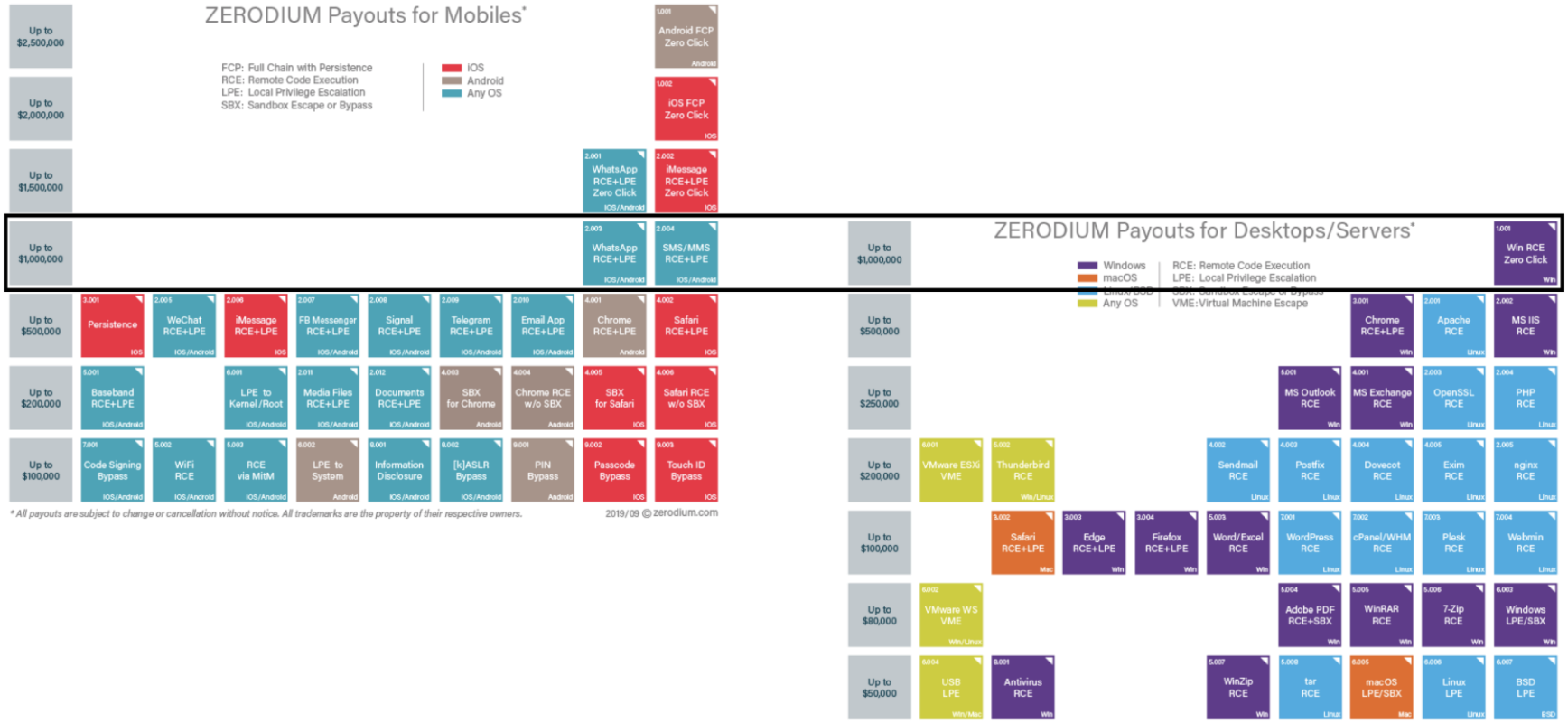
- Including tracking devices within or across networks (based on MAC addresses or other device network identifiers.)

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

Mobile Exploits Very Valuable

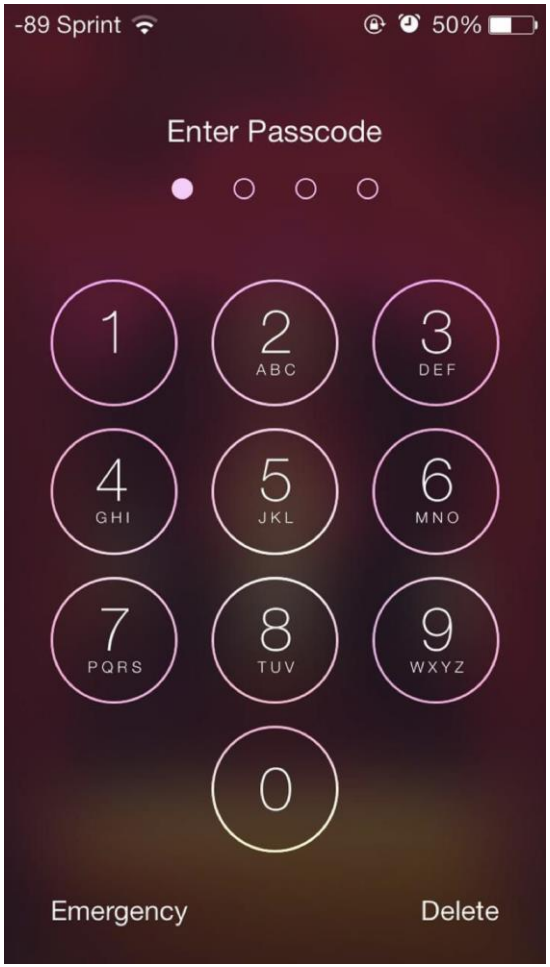


Mobile Exploits Very Valuable



Unlocking Device

PINs, patterns,
alphanumeric passwords…



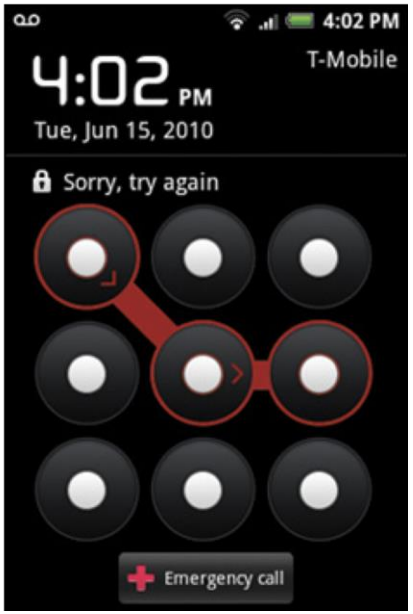
Swipe Code Problems

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 1,600 patterns with <5 strokes



iPhone Password Hashing

Goal: password hashing approach where 4-6 digits takes a very long time to crack, even if the device is physically compromised…

Additional Constraints:

- Lots of computation uses up battery (limited resource)!
- Physical access allows copying secret off the device and cracking remotely

Secure Enclave

Secure enclave: additional secure processor inside every iPhone

- Memory inaccessible to normal OS
- Utilizes a secure boot process that ensures its software is signed
- AES key burned in at manufacture

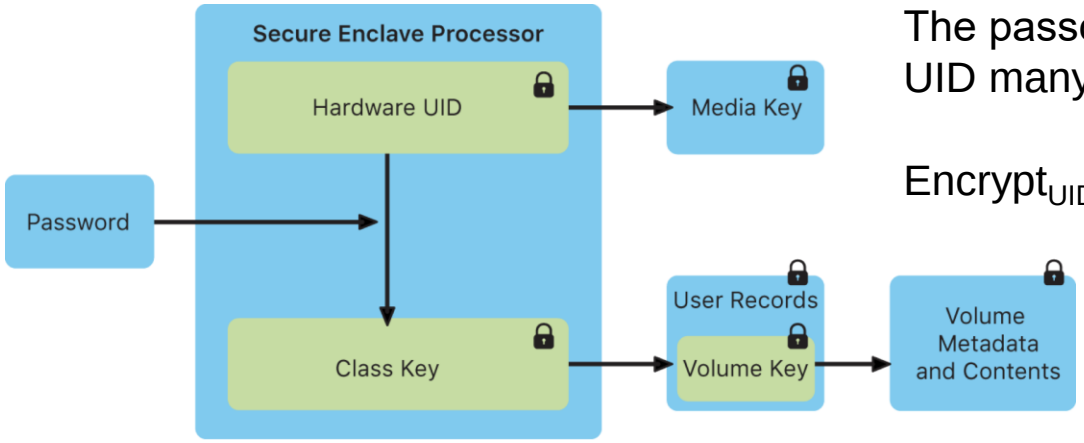
Secure enclave has instructions that allow encrypting and decrypting content using the key, but the key itself is never accessible (incl. via JTAG)

iPhone Unlocking

User passcode is intertwined with AES key fused into secure enclave (known as UID)

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



The passcode is entangled with the device's UID many times: ~ 80ms per password guess

$Encrypt_{UID}(Encrypt_{UID}(Encrypt_{UID}(passcode)\ldots))$

iPhone Unlocking

Approx. 80ms per password check

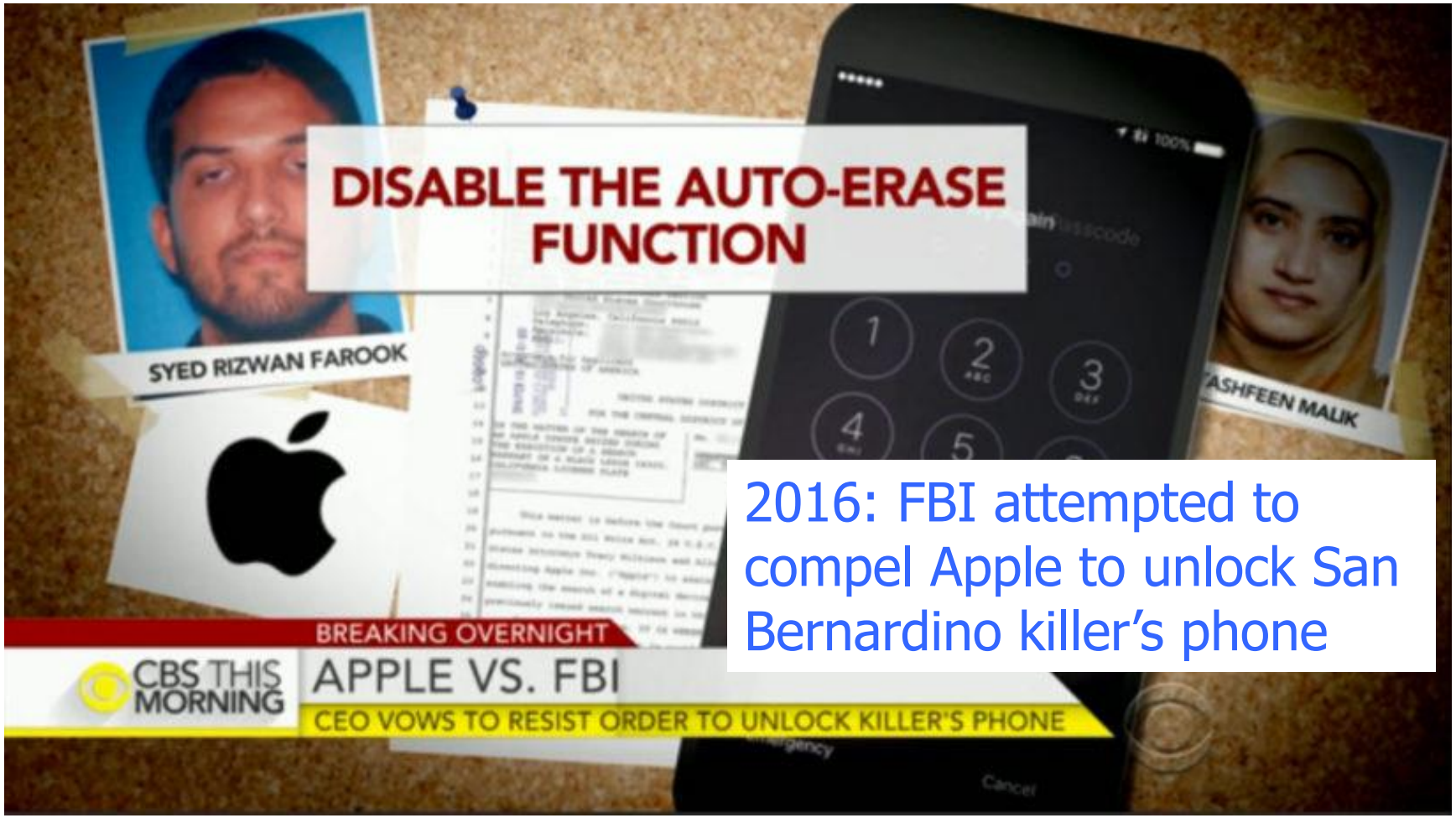
5 failed attempts ⇒ 1min delay

9 failures ⇒ 1 hour delay

10 failed attempts ⇒ erase phone

All of this enforced by firmware on the secure enclave itself — cannot be changed by any malware that controls iOS

Apple-FBI Dispute



Technical Details

The court order wanted a custom version of the 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"

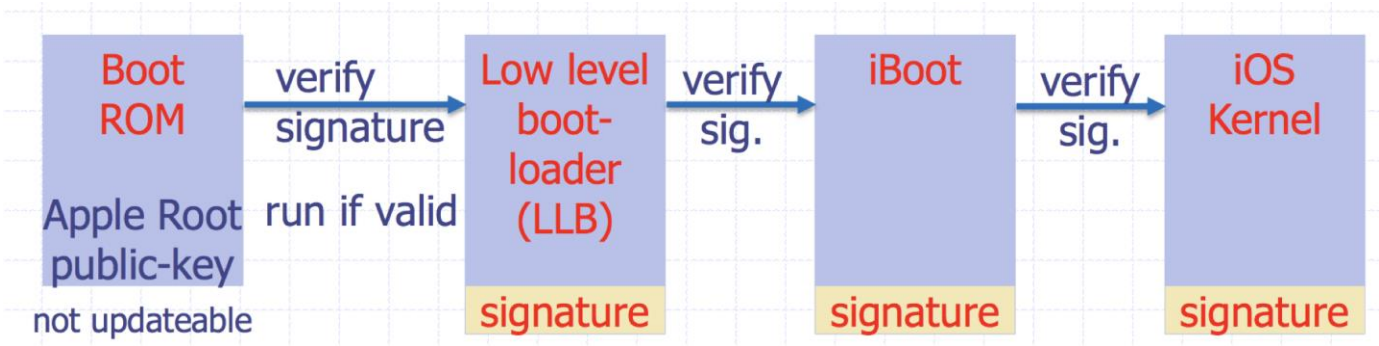
3."it will ensure that when the FBI submits passcodes to the SUBJECT DEVICE, software running on the device will not purposefully introduce any additional delay between passcode attempts beyond what is incurred by Apple hardware”

Secure Boot Chain

Why couldn't FBI upload their own firmware?

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

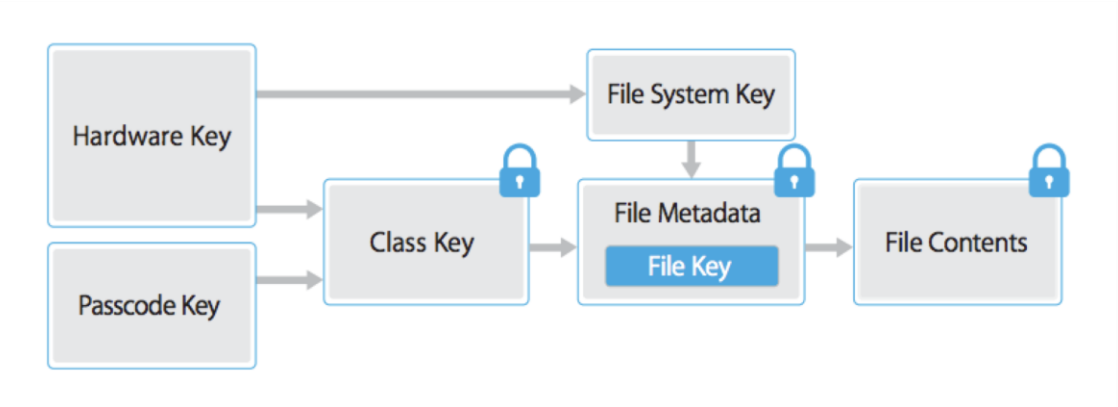
- Both for main processor and secure enclave

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



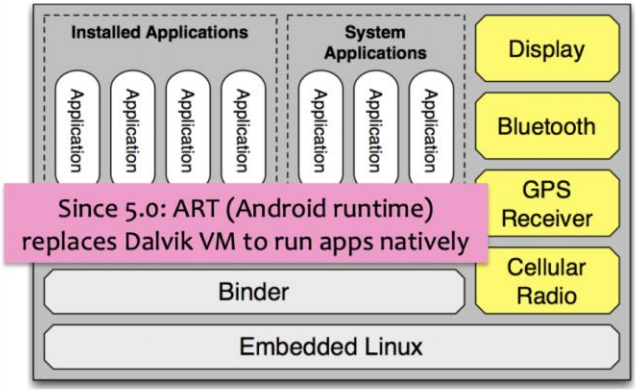
By default (no FaceID, TouchID), class encryption keys are erased from memory of secure enclave whenever the device is locked or powered off

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

Android Isolation

Based on Linux with sandboxes (SE Linux)

- Appls run as separate UIDs, in separate processes.
- Memory corruption errors only lead to arbitrary code execution in application, not complete system compromise!
- Can still escape sandbox – must compromise Linux kernel



Rooting

Allows user to run applications with root privileges

- Modify/delete system files and apps, 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

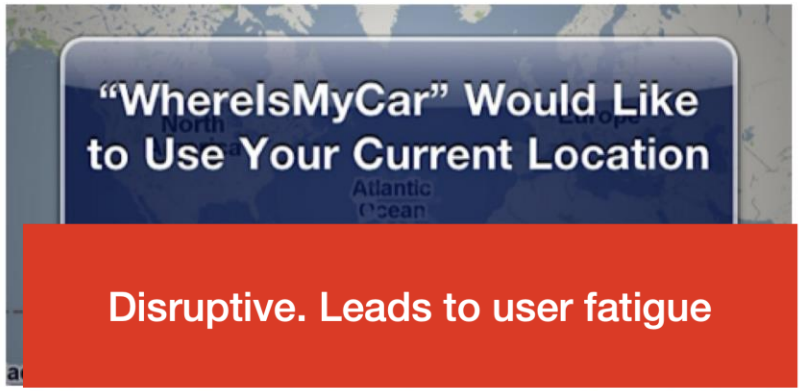
Challenges of Isolated Apps

Permissions: How can applications access sensitive resources?

Communication: How can applications communicate with each other?

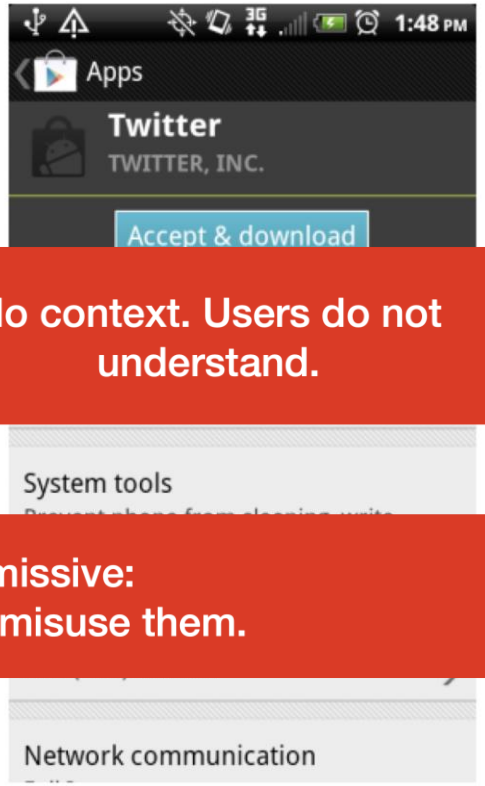
Permissions

Prompts (time-of-use)



Disruptive. Leads to user fatigue

Manifests (install-time)



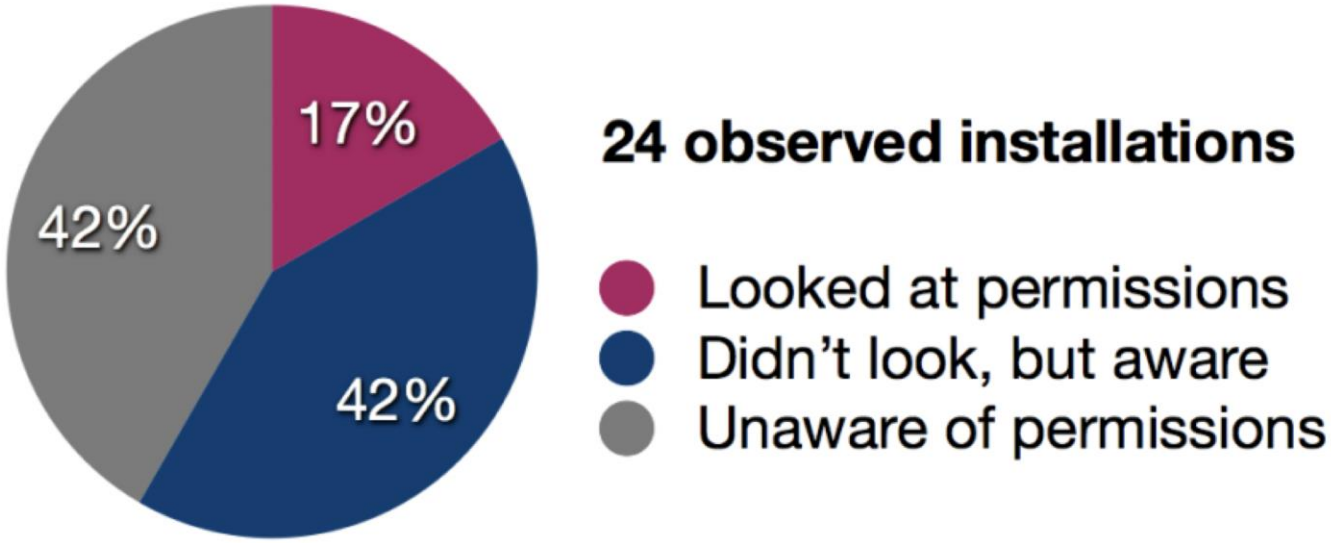
No context. Users do not understand.

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.

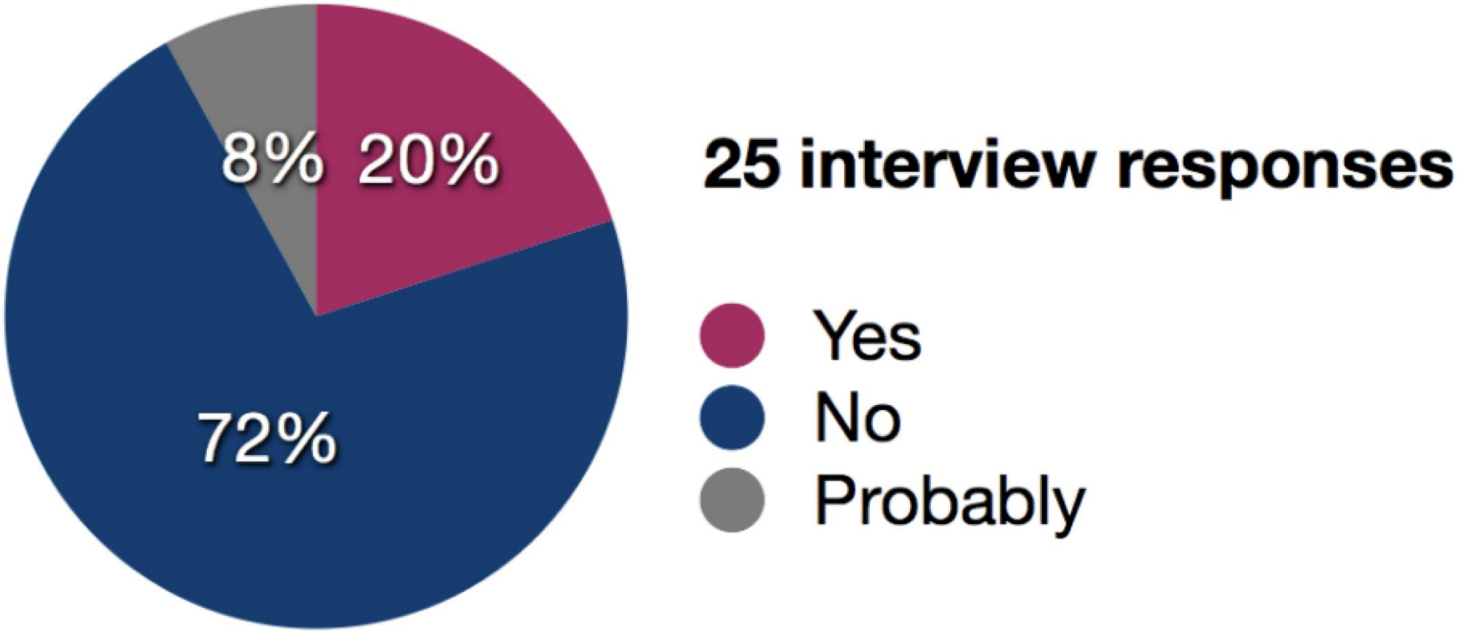
Do users understand the warnings?

	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%

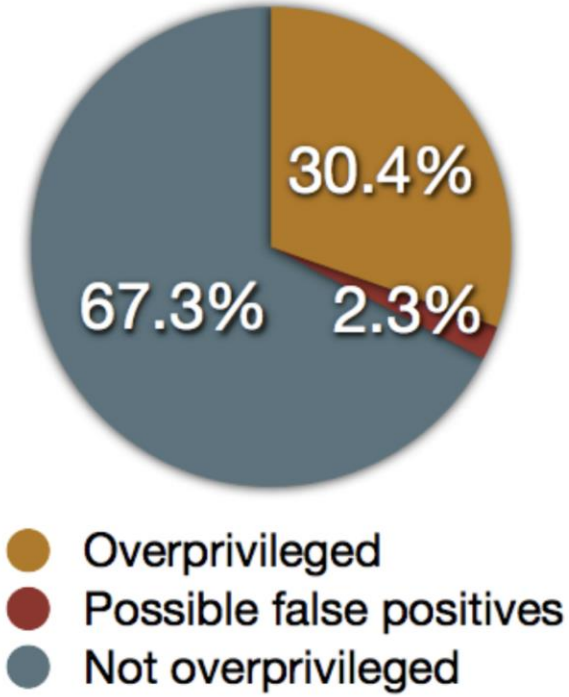
Table 4: The number of people who correctly answered a question. Questions are grouped by the number of correct choices. n is the number of respondents. (Internet Survey, $n = 302$)

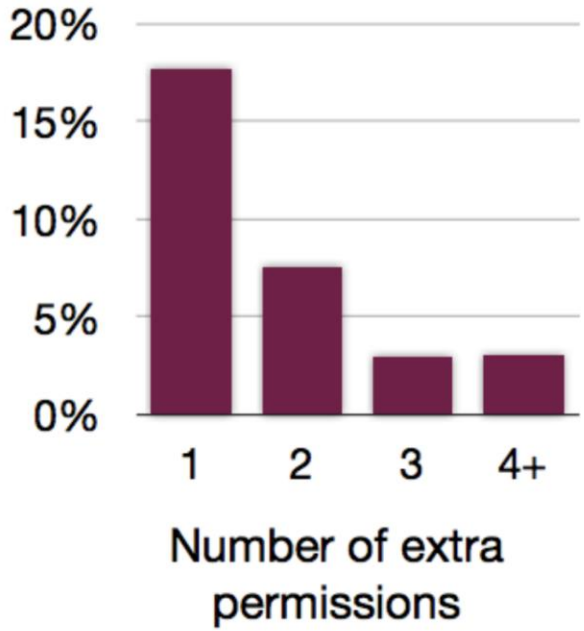
Do users act on permission information?

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



Developers Ask for Too Much





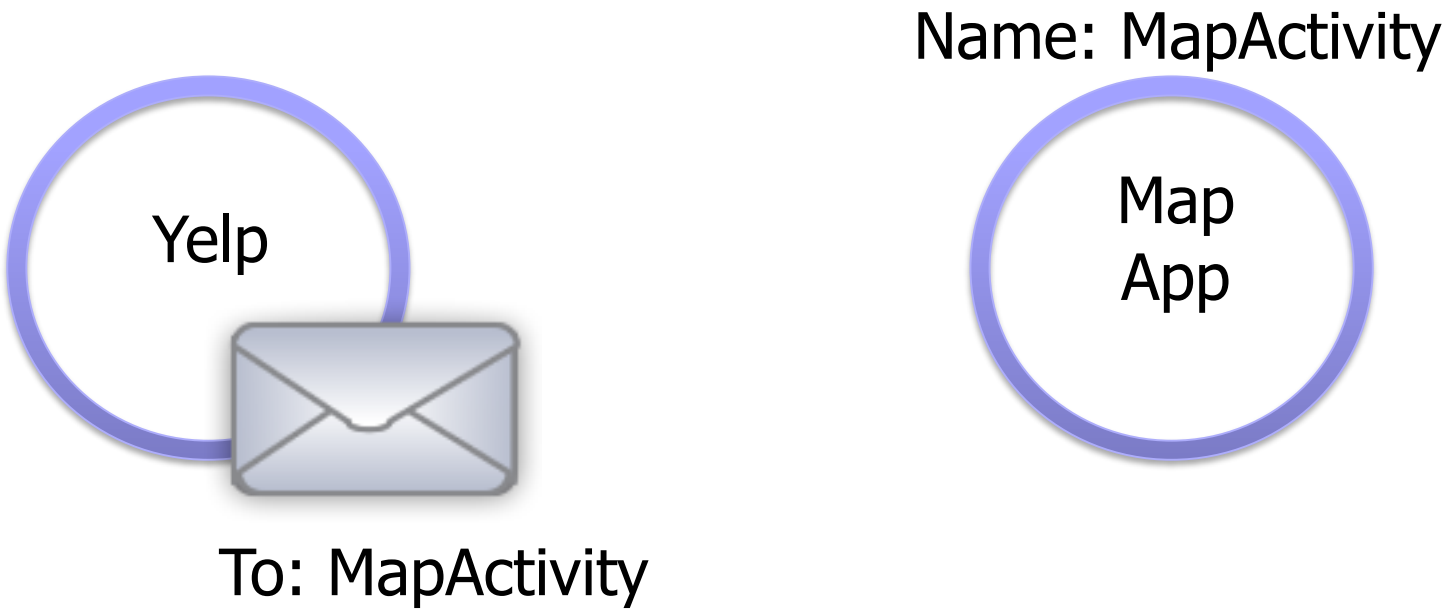
Structure of Android Applications

Applications include multiple components

- Activities: user interface
- Services: background processing
- Content providers: data storage
- Broadcast receivers for messages from other apps

Intent: primary messaging mechanism for interaction between components

Explicit Intents



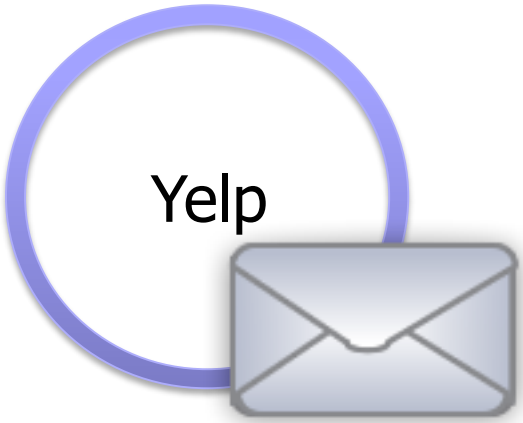
Only the specified destination receives this message

Implicit Intents

Handles Action: VIEW

Map App

Yelp



Implicit Intent
Action: VIEW

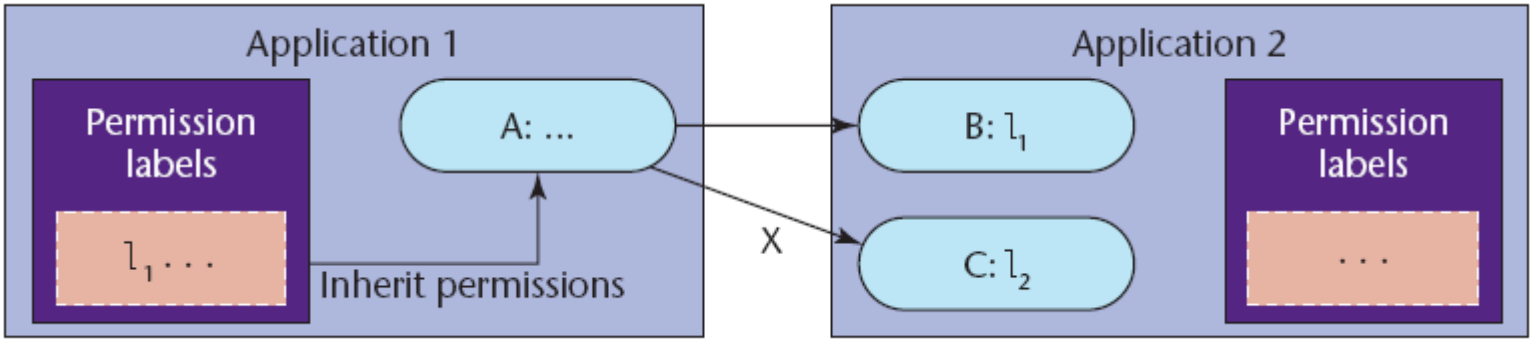
Handles Action: VIEW

Browser App

Android Security Model

Access permitted if labels assigned to the invoked component are in the collection of invoking component

Based on permission labels assigned to applications and components



Every app runs as a separate user

- Underlying Unix OS provides system-level isolation

Reference monitor in Android middleware mediates inter-component communication

Mandatory Access Control

- Permission labels are set (via manifest) when app is installed and cannot be changed
- Permission labels only restrict access to components, they do not control information flow – means what?
- Apps may contain "private" components that should never be accessed by another app (example?)
- If a public component doesn't have explicit permissions listed, it can be accessed by any app

System API Access

System functionality (eg, camera, networking) is accessed via Android API, not system components

App must declare the corresponding permission label in its manifest + user must approve at the time of app installation

Signature permissions are used to restrict access only to certain developers

- Ex: Only Google apps can directly use telephony API

Refinements

Permission labels on broadcast intents

- Prevents unauthorized apps from receiving these intents – why is this important?

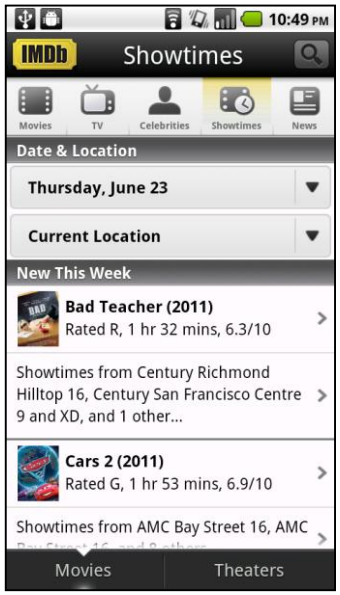
Pending intents

- Instead of directly performing an action via intent, create an object that can be passed to another app, thus enabling it to execute the action
- Invocation involves RPC to the original app
- Introduces delegation into Android's MAC system

Unique Action Strings

common developer pattern

IMDb App



Handles Actions:
willUpdateShowtimes,
showtimesNoLocationError

Implicit Intent
Action: willUpdateShowtimes

Eavesdropping

[Felt et al. "Analyzing Inter-Application Communication in Android". Mobisys 2011]

IMDb App

Showtime Search

Implicit Intent
Action: willUpdateShowtimes

Eavesdropping App

Handles Action:
willUpdateShowtimes,
showtimesNoLocationError

Malicious Receiver

Intent Spoofing

[Felt et al.]

Malicious Injection App

Malicious Component

Action:
showtimesNoLocationError

IMDb App

Handles Action:
willUpdateShowtimes,
showtimesNoLocationError

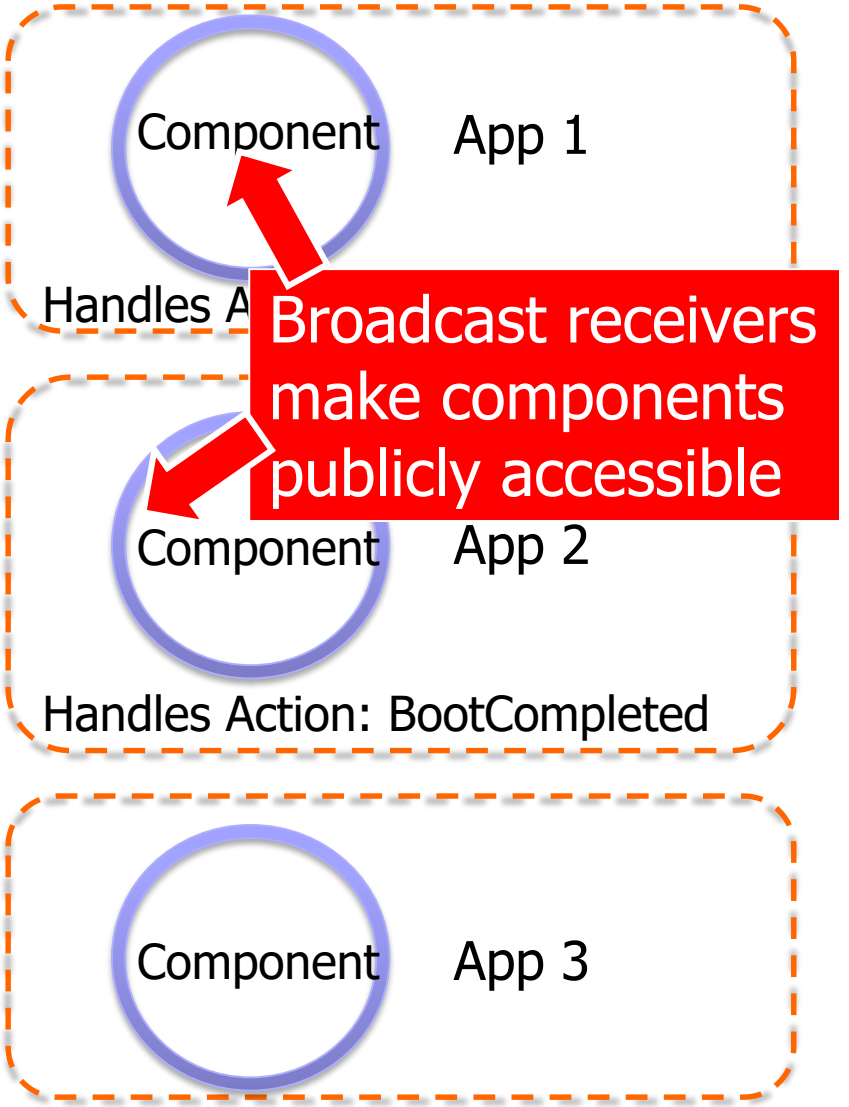
Results UI

Also man-in-the-middle

System Broadcast

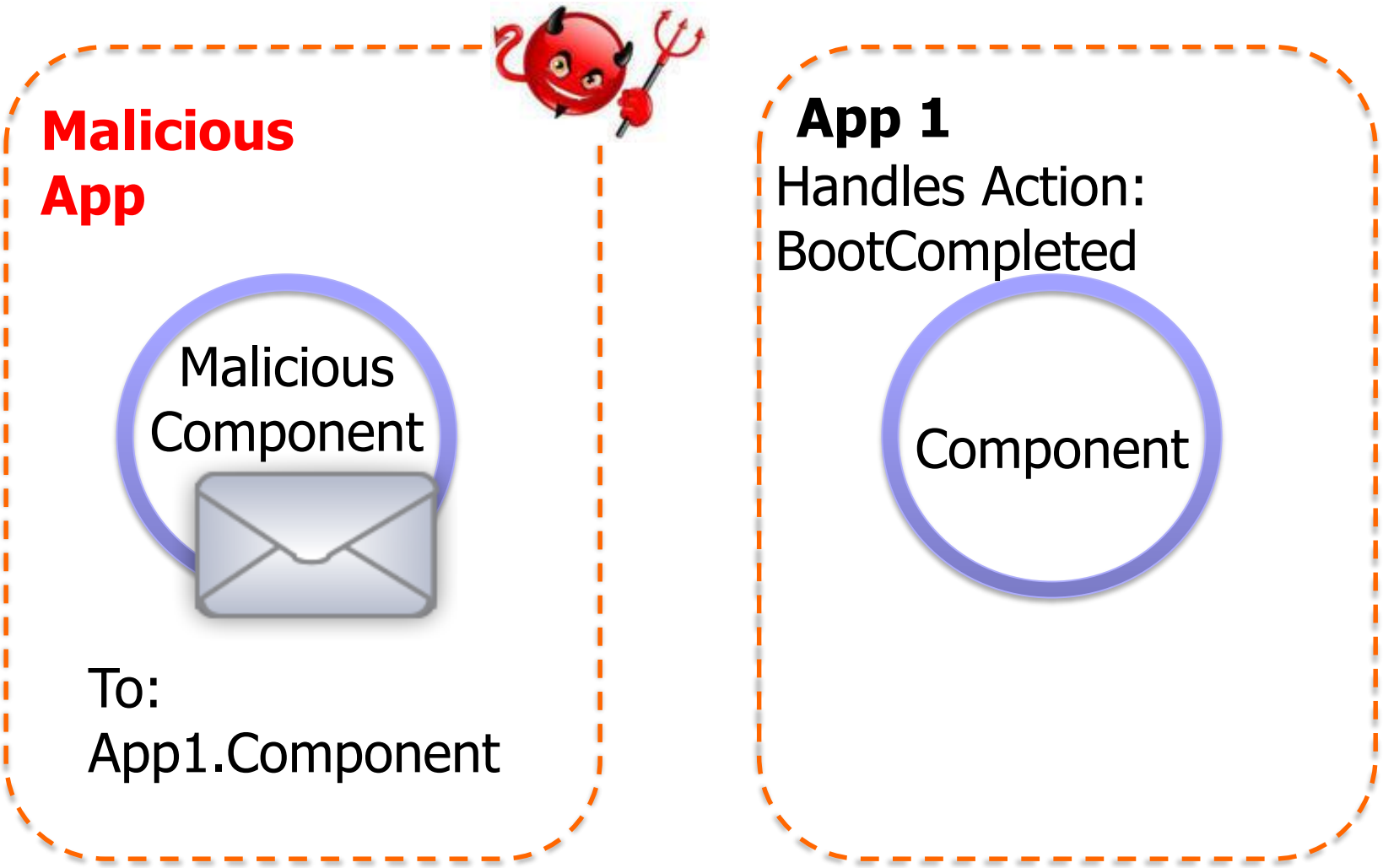
[Felt et al.]

Event notifications broadcast by the system (can't be spoofed)



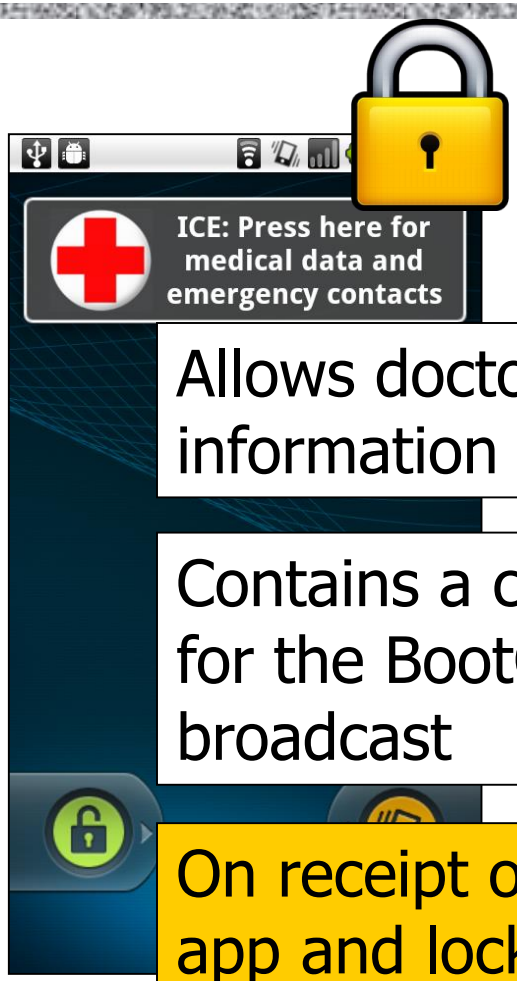
Exploiting Broadcast Receivers

[Felt et al.]



Real World Example: ICE

[Felt et al.]

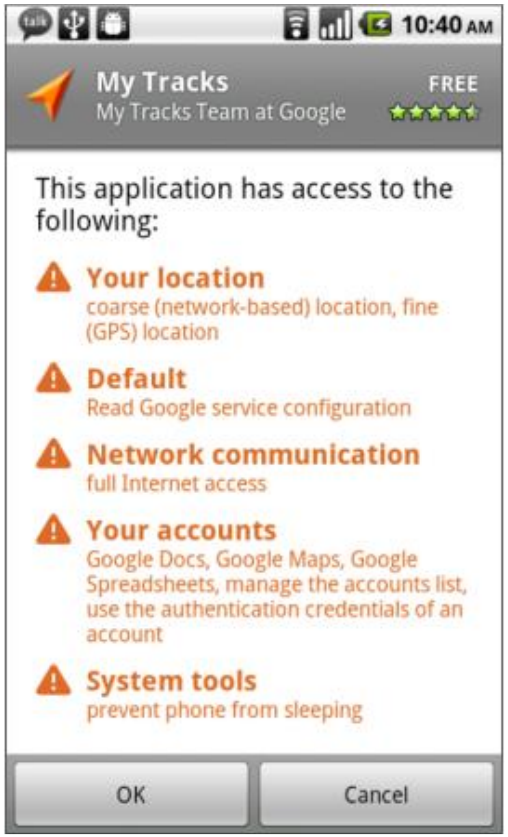


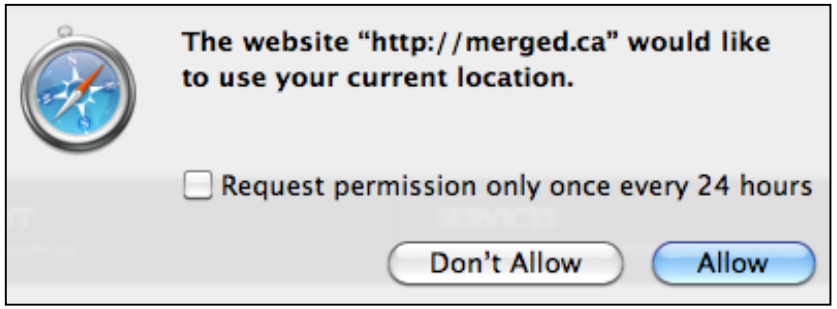
Allows doctors access to medical information on phones

Contains a component that listens for the BootCompleted system broadcast

On receipt of this intent, exits the app and locks the screen

Permissions: Not Just Android



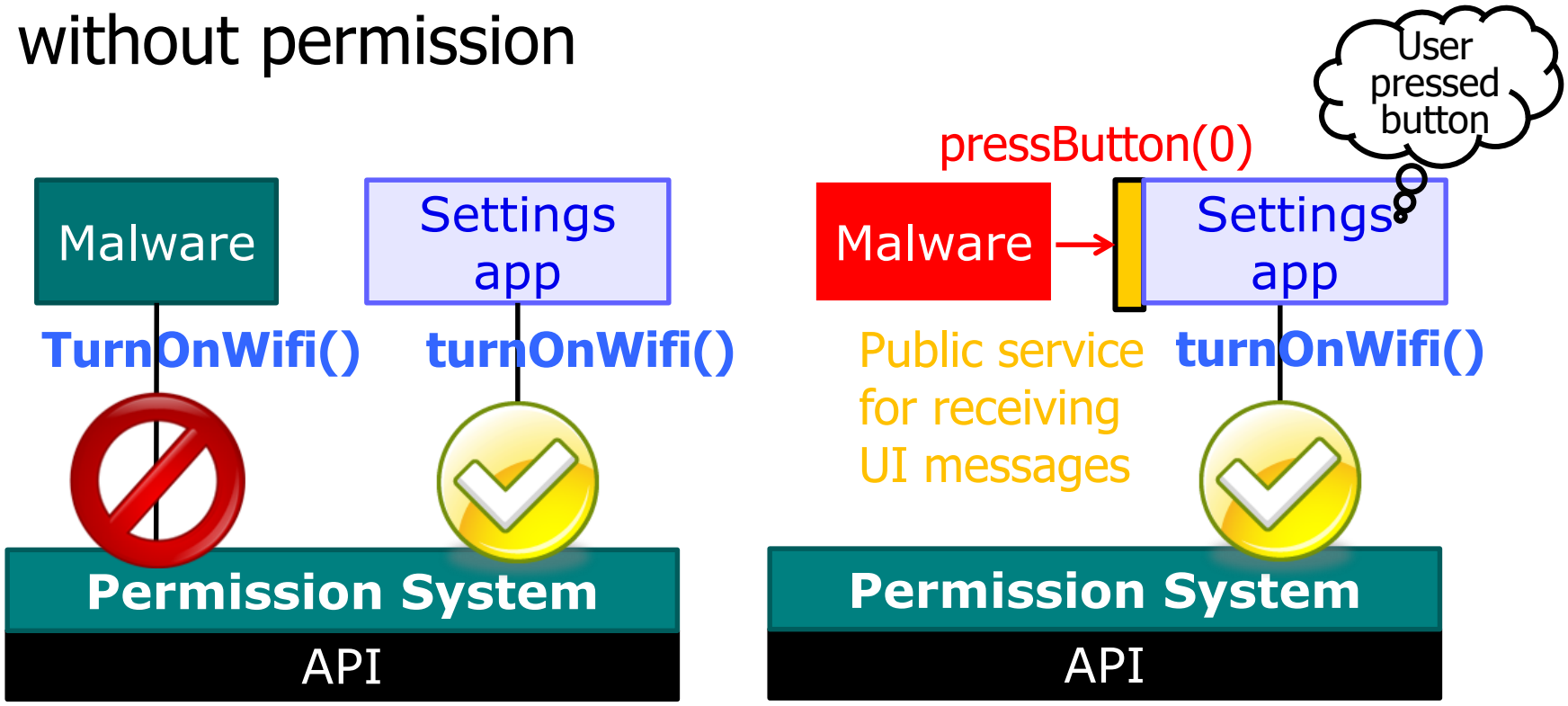


All mobile OSes, HTML5 apps, browser extensions…

Permission Re-Delegation

[Felt et al. "Permission Re-Delegation: Attacks and Defenses". USENIX Security 2011]

An application with a permission performs a privileged task on behalf of an application without permission



Examples of Re-Delegation

[Felt et al.]

Permission re-delegation is an example of a "confused deputy" problem

The "deputy" app may accidentally expose privileged functionality…

… or intentionally expose it, but the attacker invokes it in a surprising context

- Example: broadcast receivers in Android

… or intentionally expose it and attempt to reduce the invoker's authority, but do it incorrectly

- Remember postMessage origin checks?