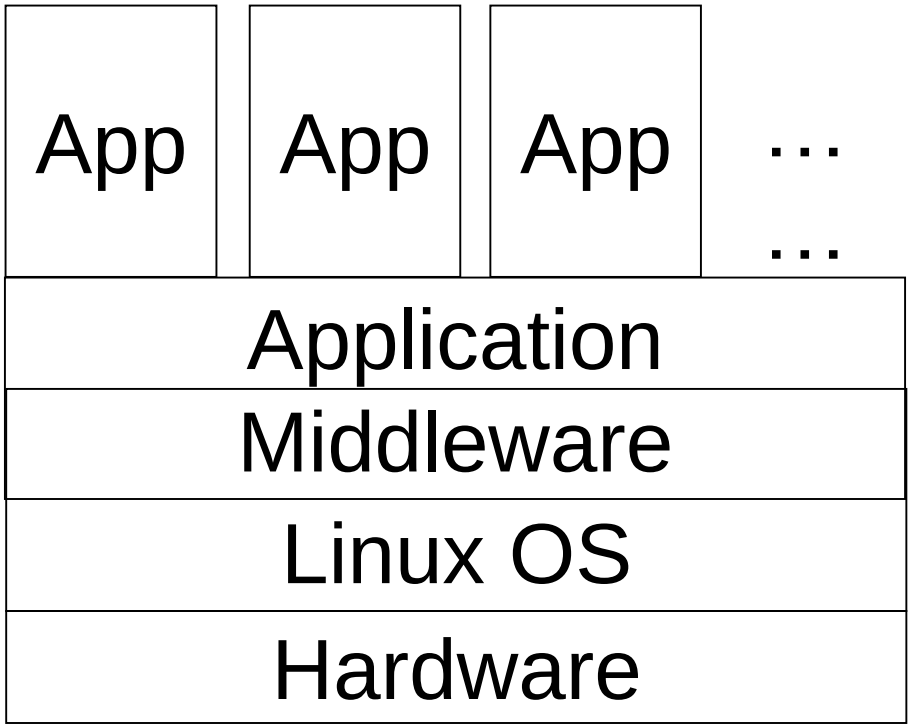
IA 313: Operating Systems Security

Lecture 6: Android System Security

Android System Basics

- An open-source operating system for mobile devices (AOSP, led by Google)
  - Consists of a base operating system (Linux), an application middleware, and a Java SDK.
  - Available since 2007; first Android phone debut in 2008.
- Currently the most popular smart-phone platform

Android System Overview





Security Considerations

- A smart phone is “mostly” a single-user system
  - The security goal is primarily to protect the user and his/her data
  - The security goal also includes that critical functions of the phone shall not be hampered
- The apps a phone runs are from manufacturers and app stores
  - The origin of the apps could be untrustworthy – apps could be vulnerable or outright malicious

Basics Protections

- Each application runs as a user process in the Linux OS with a unique user ID
  - At install time, Android gives each package a distinct Linux user ID
  - Two apps can run under the same user ID if they are signed by the same key
  - This creates a sandbox for each app, so that Linux isolates applications from each other and from the system.

Android Security Architecture

- Android operating system security objectives
  - Protect user data
  - Protect system resources (including the network)
  - Provide application isolation
- To achieve these objectives, Android provides the following security features
  - Robust security at the OS level through the Linux kernel
  - Application sandbox for all apps (application isolation)
  - Secure interprocess communication
  - Application signing
  - Application-defined and user-granted permissions

Linux Security Objectives

- Linux kernel is the foundation of the Android platform
- Linux is a multiuser operating system, with the main security objective to mutually isolate different users
  - Prevents user A from reading user B's file
  - Ensures that user A does not exhaust user B's memory/CPU resources
- Central to Linux and Android security are concepts of
  - Discretionary Access Control (DAC) – used to isolate apps (Linux)
  - Mandatory Access Control (MAC) – used to mediate the establishment of inter-component/application communication (Android middleware)

Android software stack

- Each component assumes that the components below are properly secured.
- All code above the Linux Kernel is restricted by the Application Sandbox
- Linux kernel is responsible sandboxing application
  - "mutually distrusting principals"
  - Default access to only its own data
- The app Sandbox apps can talk to other apps only via Intents (message) , IPC, and Content Providers
- To escape sandbox, permissions is needed

Security at the Linux kernel

- A user-based permissions model
- Process isolation: Each application has its sandbox based on separation of processes: to protect user resources from each another; each runs in its own Linux process to secure Inter-Process communication (IPC)

  Ex:

  – Prevents user A from reading user B's files
  – Ensures that user A does not access user B's CPU, memory resources
  – Ensures that user A does not access user B's devices (e.g. telephony, GPS, Bluetooth)

Permissions and Encryption

- **Permissions**

  In Android, each application runs as its own user. Unless the developer explicitly exposes files to other applications, files created by one application cannot be read or altered by another application.

- **Password Protection**

  Android can require a user-supplied password prior to providing access to a device. In addition to preventing unauthorized use of the device, this password protects the cryptographic key for full file system encryption.

Encryption

- Encryption

  Android 3.0+ provides full filesystem encryption, so all user data can be encrypted in the kernel

- For a lost or stolen device, full filesystem encryption on Android devices uses the device password to protect the encryption key, so modifying the bootloader or operating system is not sufficient to access user data without the user’s device password.

Android Application Security (Application framework)

- Almost all Android applications are written in the Java and run in the Dalvik virtual machine. However, applications can also be written in native code.

  .java -> .class -> .jar -> .dex

  Android application deployed in a single .apk file.
- Android middleware is based on the Linux kernel. It provides several native libraries and a Dalvik virtual machine (DVM) instead of Java virtual machine (JVM) for its applications' runtime environment where application isolation is enforced.
- The Java written Android middleware provides development APIs, the system service, all basic phone device functionalities

Android Permission Model

- Permissions are the core concepts in the Android security to control the access from one application component to another.
- All permissions are set at installation time and can't change until the application is reinstalled.
- Android's permission only restricts components from accessing resources
  - E.g. an application needs the READ_CONTACTS permission to read the user's address book
- If a public component doesn't explicitly declare any access permission, Android permits any application to access it.

Android Permission Levels

- Android permissions fall into four levels:
  - Normal – cannot impart real harm to the user (e.g. change the wallpaper); apps need to request them, but automatically granted
  - Dangerous – can impart real harm (e.g. call numbers, open Internet connections) and apps need to request them with user confirmation.
  - Signature – these are automatically granted to a requesting app if that app is signed by the same certificate as that which declared the permission
  - Signature/System – same as Signature, except that the system image gets the permissions automatically as well (for use by device manufacturers only)

Security Weaknesses of Android Permissions

- Confused deputy attacks
  - If you do not have a right permission, ask your neighbor
  - Example: *MalwareApp* with no permissions granted, invokes *BrowserApp* (having *INTERNET* permission) to download malicious files
- Attacks by colluding applications
  - Divide necessary permissions among two or more apps
  - Example: *MalwareApp-1* has access to user LOCATION, *MalwareApp-2* has *INTERNET* permission and the two malware apps communicate (e.g., Intents, file sharing, etc.)

Android Inter-Component Communication (ICC)

- Each Android app contains one or more components of the following types:
  - Activity: A GUI screen
  - Service: background processing
  - Broadcast Receiver: receives messages from other component
  - Content Provider: store and share data

- Each component runs as a separate thread in the OS

Security Protection for ICC

- An ICC interface is protected in two ways:
  - A private component cannot be accessed outside of the app
  - A component can be protected by permissions
- A permission is a label defined by the system or app developer to protect an Android component

OS protected APIs

- Cost-Sensitive APIs
  - Telephony
  - SMS/MMS
  - Network/Data connections
  - In-App Billing
  - NFC Access
- Sensitive Data Input Devices
  - Location data (GPS)
  - Camera functions
  - microphone
- Bluetooth functions
- Personal Information

Interprocess Communication

- Standard IPC
  - file system, local sockets, or signals.
  - Linux permissions still apply.
- New IPC mechanisms:
  - Binder: RPC mechanism for in-process and cross-process calls. Via a custom Linux driver.
  - Services: interfaces directly accessible using binder.
  - Intents: A message object that represents an "intention" to do something.
  - ContentProviders: A data storehouse

Application Signing

- Why self signing?
  - Market ties identity to developer account
  - CAs have had major problems with fidelity in the past
  - No applications are trusted. No "magic key"
- What does signing determine?
  - Shared UID for shared keys
  - Self-updates

Application Signing

- All .apk files must be signed with a certificate
  - identifies the author of the application.
  - does not need to be signed by a certificate authority
- allows the system to grant or deny applications
  - access to signature-level permissions
  - request to be given the same Linux identity as another application.
- If the public key matches the key used to sign any other APK, the new APK may request to share a UID with the other APK.

User IDs and File Access

- Each package is assigned a distinct Linux UID
  - there is no /etc/passwd however
  - different device => may have a different UID
  - files created by pkg are owned by this “user”
- Shared UID feature
  - Two applications are signed ⇒ can share UIDs
  - More interactivity

File system Encryption

- Full file system encryption
  - Android 3.0 and later
  - AES128
  - Password + random salt

SIM Card Access

- Low level access to the SIM card is not available to third-party apps.
- The OS handles all communications with the SIM card including access to personal information (contacts) on the SIM card memory.
- Applications also cannot access AT commands, as these are managed exclusively by the Radio Interface Layer (RIL). The RIL provides no high level APIs for these commands.

SMS Vulnerabilities

- SMS
  - Short Messaging System
  - Very commonly used protocol
  - Used to send "Text Messages"
- GSM uses 2 signal bands, 1 for "control", the other for "data".
- SMS operates entirely on the "control" band.
- High volume text messaging can disable the "control" band, which also disables voice calls.
- Can render entire city 911 services unresponsive.

MMS Vulnerabilities

- MMS
  - Unsecure data protocol for GSM
  - Extends SMS, allows for WAP connectivity
- Exploit of MMS can drain battery 22x faster
  - Multiple UDP requests are sent concurrently, draining the battery as it responds to request
- Does not expose data
- Does make phone useless

Bluetooth Vulnerabilities

- Bluetooth
  - Short range wireless communication protocol
  - Used in many personal electronic devices
  - Requires no authentication
- An attack, if close enough, could take over Bluetooth device.
- Attack would have access to all data on the Bluetooth enabled device
- Practice known as bluesnarfing

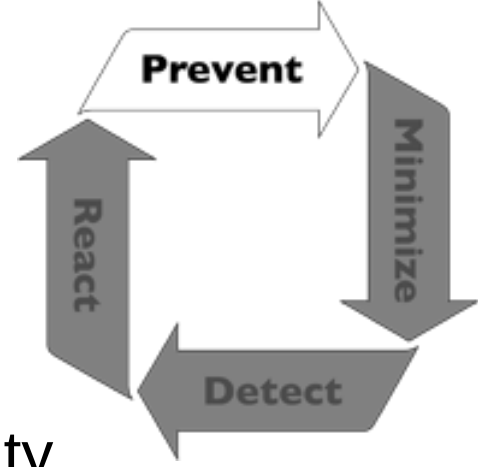
Information Misuse by Apps

- Phone identifiers: phone number, IMEI (device identifier), IMSI (subscriber identifier), and ICC-ID (SIM card serial number).
- Phone identifiers are frequently leaked through plaintext requests.
- Phone identifiers are used as device fingerprints.
- Phone identifiers, specifically the IMEI, are used to track individual users.
- Not all phone identifier use leads to exfiltration.
- Phone identifiers are sent to advertisement and analytics servers.

Cornerstones of Android security

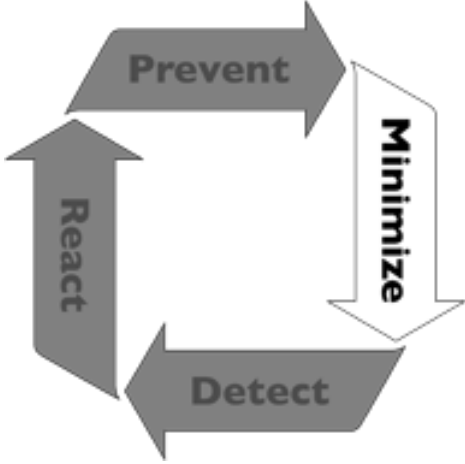
- Prevention
- Minimization
- Detection
- Reaction

Prevent



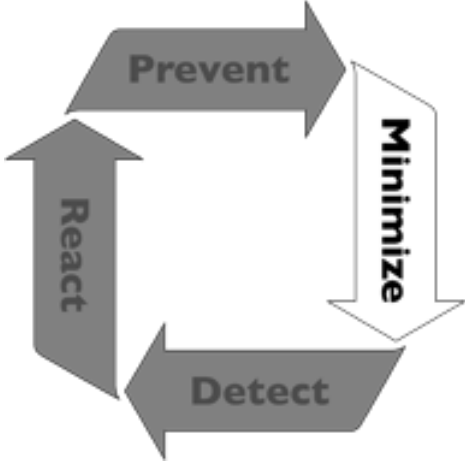
- 5 million new lines of code
- Uses almost 100 open source libraries
- Android is open source ⇒ can't rely on obscurity
- Teamed up with security experts from
  - Google Security Team
  - iSEC Partners
  - n.runs
- Concentrated on high risk areas
  - Remote attacks
  - Media codecs
  - New/custom security features
- Low-effort/high-benefit features
  - ProPolice stack overflow protection
  - Heap protection in dlmalloc

Minimize



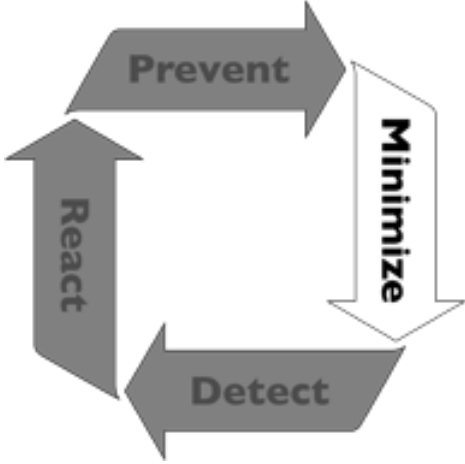
- We cannot rely on prevention alone
  - Vulnerabilities happen
- Users will install malware
- Code will be buggy
- How can we minimize the impact of a security issue?
- My webmail cannot access my banking web app
  - Same origin policy
- Why can malware access my browser? my banking info?
- Extend the web security model to the OS

Minimize …



- Traditional operating system security
  - Host based
  - User separation
- Mobile OSes are for single users
- User separation is like a "same user policy"
- Run each application in its own UID is like a "same application policy"
  - Privilege separation
- Make privilege separation relatively transparent to the developer

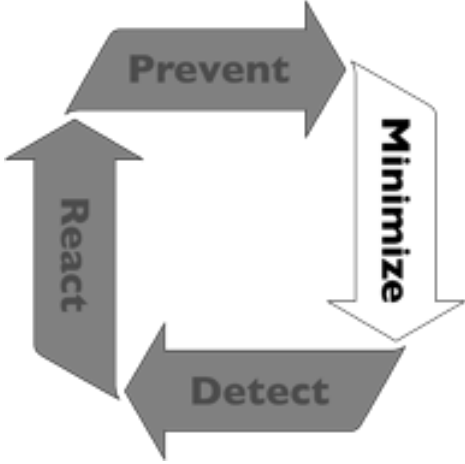
Application Sandbox

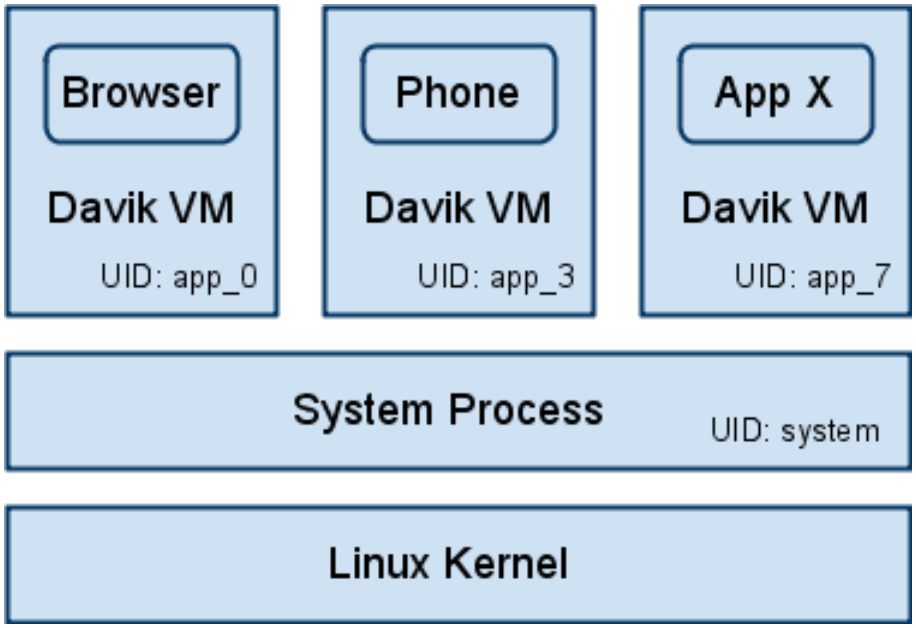


- The sandbox is based on separation of
  - processes
  - file permissions
  - Authenticated IPC
- Each application
  - is a different “user”; its own UID
  - runs in its own Linux process
  - its own Dalvik VM
- Sandboxes native code and sys applications

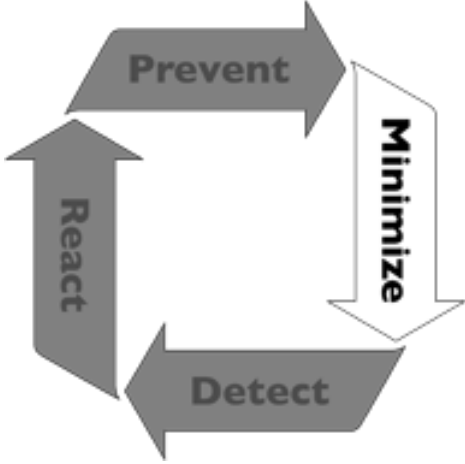
Application Sandbox …

- Each application runs within its own UID and VM
- Default privilege separation model
- Instant security features
  - Resource sharing
    - CPU, Memory
  - Data protection
    - FS permissions
  - Authenticated IPC
    - Unix domain sockets



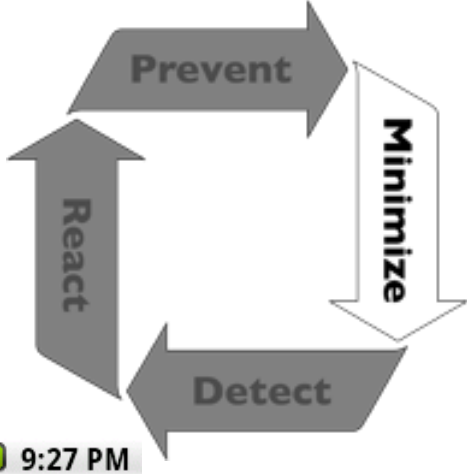


Application Sandbox …

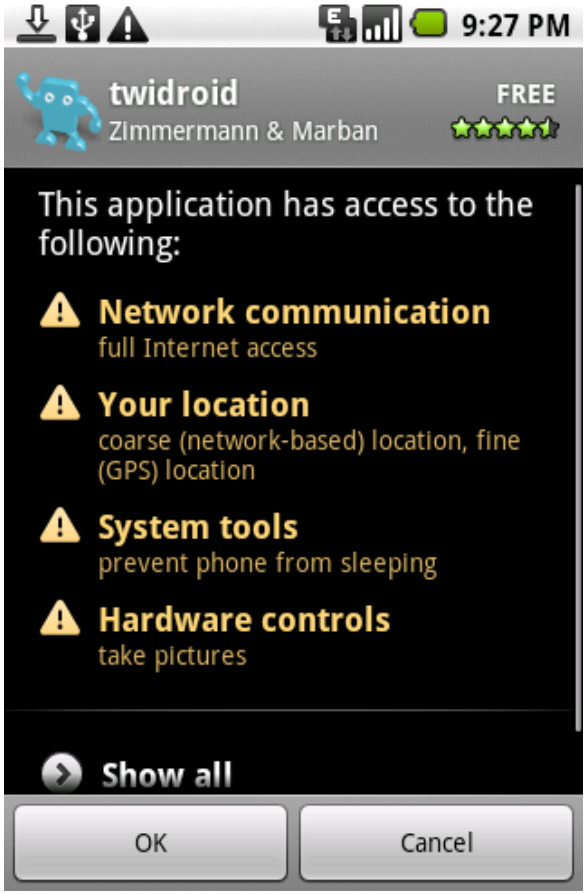


- Place access controls close to the resource, not in the VM
  - Smaller perimeter ⇒ easier to protect
- Default Linux applications have too much power
- Lock down user access for a "default" application
- Fully locked down applications limit innovation
- Relying on users making correct security decisions is tricky

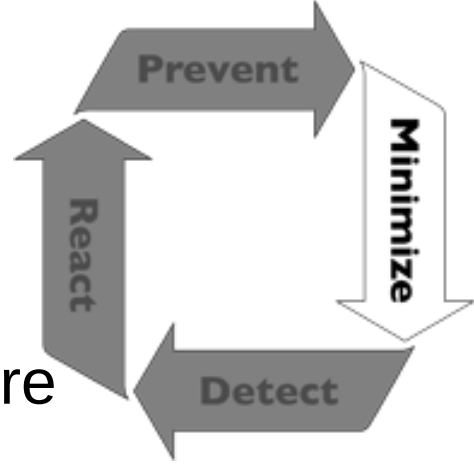
Permissions



- Whitelist model
  1. Allow minimal access by default
  2. Allow for user accepted access to resources
- Ask users less questions
- Make questions more understandable
- 194 permissions
  - More ⇒ granularity
  - Less ⇒ understandability

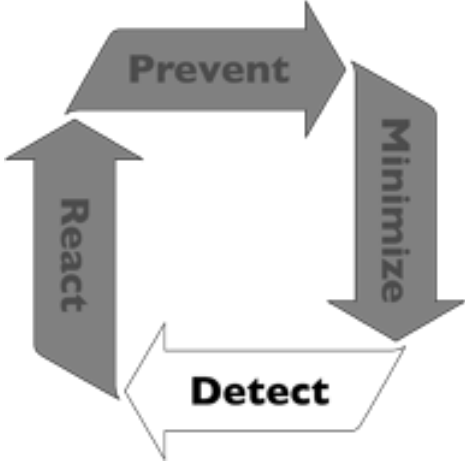


More Privilege Separation



- Media codecs are very complex ⇒ very insecure
- Won't find all the issues media libraries
- Banish OpenCore media library to a lesser privileged process
  - mediaserver
- Immediately paid off
  - Charlie Miller reported a vulnerability in our MP3 parsing
  - oCERT-2009-002

Detect



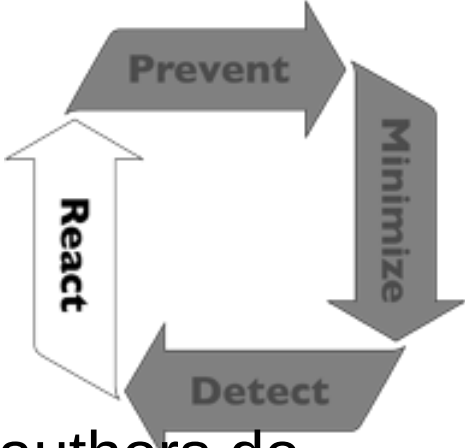
- A lesser-impact security issue is still a security issue
- Internal detection processes
  - Developer education
  - Code audits
  - Fuzzing
  - Honeypot
- Everyone wants security ⇒ allow everyone to detect issues
  - Users
  - Developers
  - Security Researchers

React



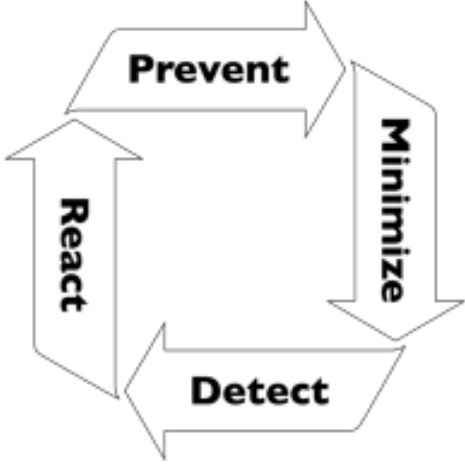
- Autoupdaters are the best security tool since Diffie-Hellman
- Every modern operating system should be responsible for:
  - Automatically updating itself
  - Providing a central update system for third-party applications
- Android's Over-The-Air update system (OTA)
  - User interaction is optional
  - No additional computer or cable is required
  - Very high update rate

Shared UID Regression



- Shared UID feature
  - Malware does not hurt computers, malware authors do
  - Two applications are signed ⇒ can share UIDs
  - More interactivity
- Panasonic reported that shared UID was broken
  - If the user installs malware, then the attacker could share UIDs with an existing installed app, like the browser
  - Breaks Application Sandbox

Security Philosophy



- Finite time and resources
- Humans have difficulty understanding risk
- Safer to assume that
  - Most developers do not understand security
  - Most users do not understand security

- Security philosophy cornerstones
  - Need to **prevent** security breaches from occurring
  - Need to **minimize** the impact of a security breach
  - Need to **detect** vulnerabilities and security breaches
  - Need to **react** to vulnerabilities and security breaches swiftly