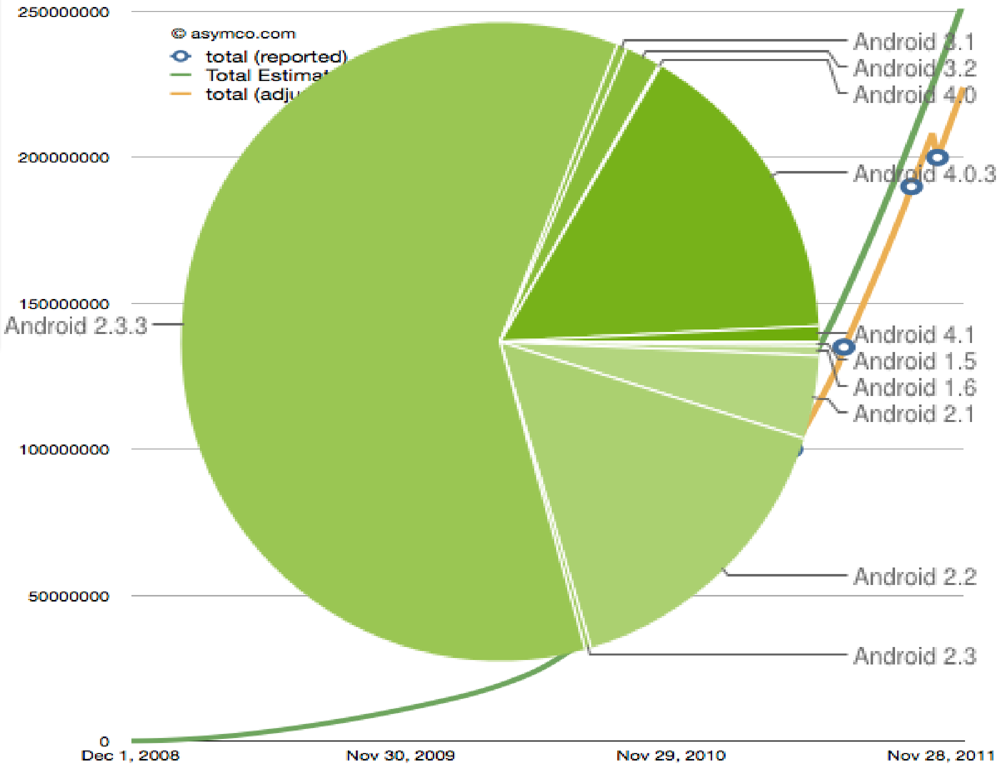
Android Security

Android

- As of December 2011, over 10 billion apps downloaded from Google Play
- Many vendors on android

Estimated Total Android Activations



Images: Google

Android History

2005	2007	2008	2009	2010	2011	2012
July: Google aquires Android, Inc.	**Nov:** Initial SDK release	**Sept:** T-Mobile announces G1 **Oct:** Source code released (some Google Apps omitted)	**Apr:** Android 1.5 (Cupcake) **Sept:** Android 1.6 (Donut) **Oct:** Android 2.0 (Éclair)	**Jan:** Android 2.1 **May:** Android 2.2 (Froyo) **Dec:** Android 2.3 (Gingerbread)	**Jan-Nov:** Android 2.2.1-2.2.3 **Feb:** Android 3.0 (Honeycomb) **Oct:** Android 4.0.1 (Ice Cream Sandwich)	**July:** Android 4.1.1 (Jelly Bean)

Android Security

- Security goal:
  - Protect user data
  - Provide application isolation
  - Protect system resources (including the network)

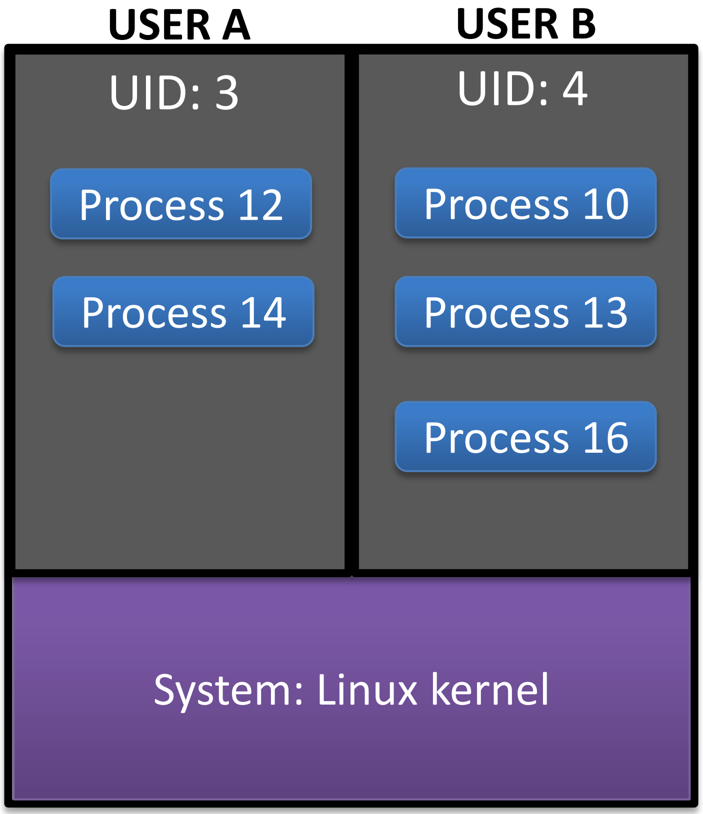
Android Security Mechanism

- Robust security at the OS level through the Linux kernel
- Mandatory application sandbox for all applications
- Secure inter-process communication
- Application signing
- Application-defined and user-granted permissions

Traditional Linux Security Model

- Multi-user model
- A user-based permissions model
- Process isolation
- Extensible mechanism for secure IPC

- Linux:
  - Prevents user A from reading user B's files
  - Ensures that user A does not exhaust user B's memory
  - Ensures that user A does not exhaust user B's CPU resources
  - Ensures that user A does not exhaust user B's devices (e.g. telephony, GPS, bluetooth)



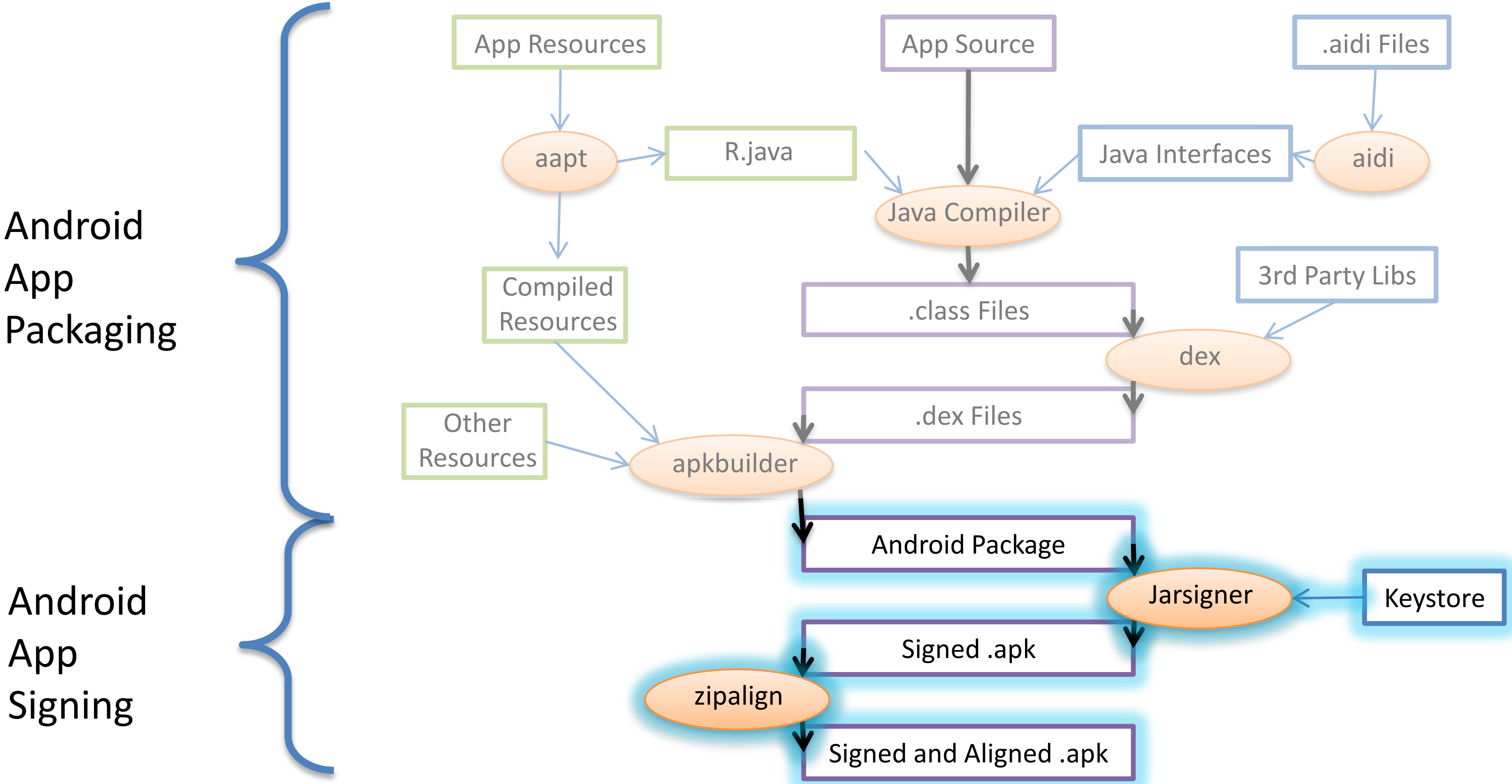
Android Security Model

- Multi-app model
- Different app installed with different UID
- Runs in a different process
- Application sandbox

Application Sandbox

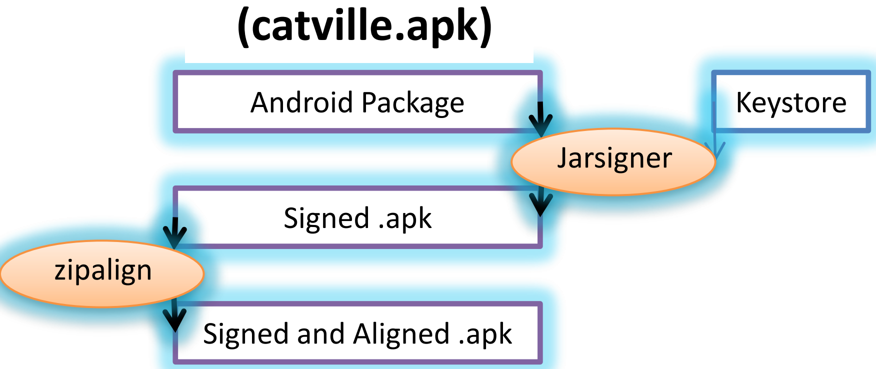
- The kernel enforces security between applications and the system at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications.
- By default, applications cannot interact with each other and applications have limited access to the operating system.

App Signing

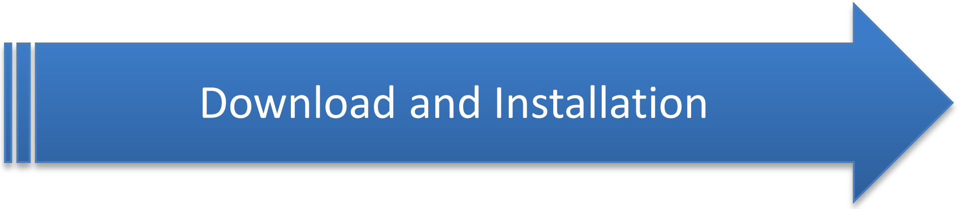


Android App Packaging
Android App Signing
App Resources
App Source
.aidi Files
aapt
R.java
Java Interfaces
aidi
Java Compiler
Compiled Resources
.class Files
3rd Party Libs
dex
.dex Files
Other Resources
apkbuilder
Android Package
Jarsigner
Keystore
Signed .apk
zipalign
Signed and Aligned .apk

App Signing



(public key A)



Apps Signed with different public keys
⇒ Apps have different UID

If two apps have the same public key, the two apps could have the same UID.

Memory-safely Enhancements

Android 1.5+

- ProPolice to prevent stack buffer overruns (-fstack-protector)
- safe_iop to reduce integer overflows
- Extensions to OpenBSD dlmalloc to prevent double free() vulnerabilities and to prevent chunk consolidation attacks. Chunk consolidation attacks are a common way to exploit heap corruption.
- OpenBSD calloc to prevent integer overflows during memory allocation

Android 2.3+

- Format string vulnerability protections (-Wformat-security -Werror=format-security)
- Hardware-based No eXecute (NX) to prevent code execution on the stack and heap
- Linux mmap_min_addr to mitigate null pointer dereference privilege escalation (further enhanced in Android 4.1)

Android 4.0+

- Address Space Layout Randomization (ASLR) to randomize key locations in memory

Android 4.1+

- PIE (Position Independent Executable) support
- Read-only relocations / immediate binding (-Wl,-z,relro -Wl,-z,now)
- dmesg_restrict enabled (avoid leaking kernel addresses)
- kptr_restrict enabled (avoid leaking kernel addresses)

Permissions

- Different types of permissions:
  - Camera functions
  - Location data (GPS)
  - Bluetooth functions
  - Telephony functions
  - SMS/MMS functions
  - Network/data connections
- Different from file permissions
- User-defined permissions

Android Application Security

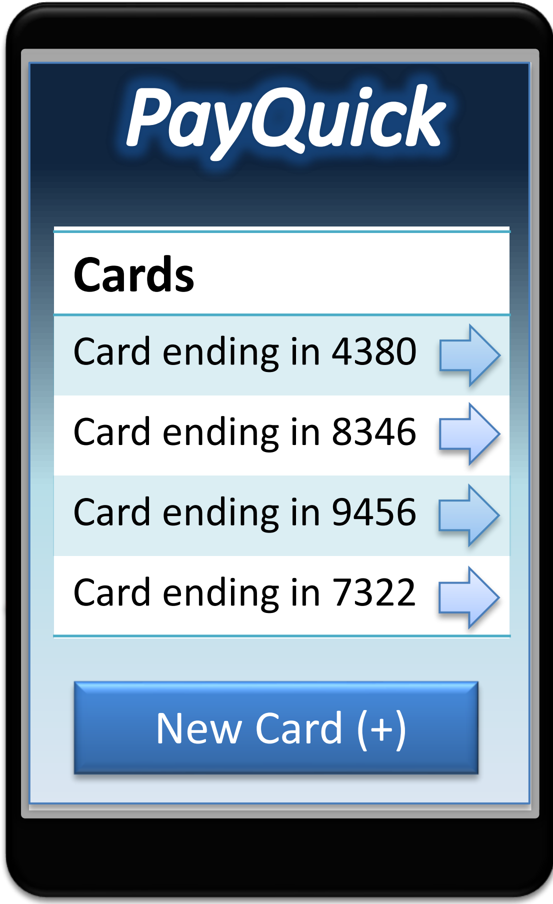
Each application is divided into components

Activities

Services

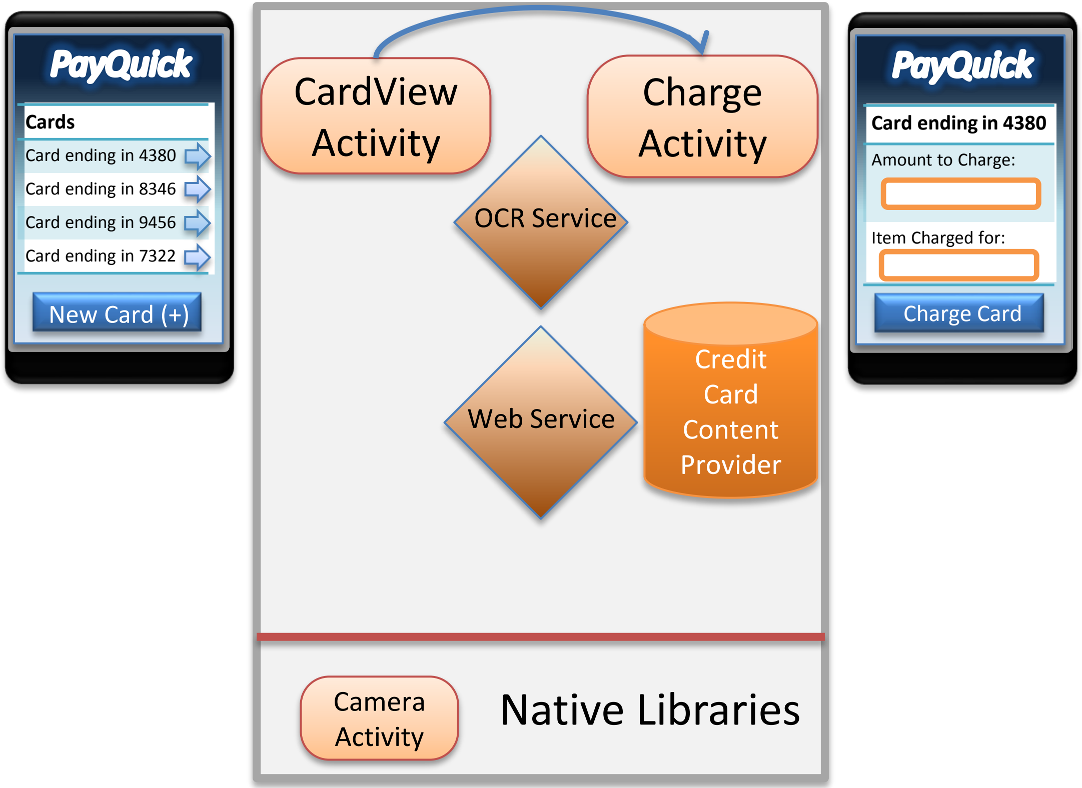
Broadcast Receivers

Content Providers



Intents

Explicit Intent for Charge Activity: viewMakeCharge with cardA



Intent Filter for Charge Activity:

.

.

.

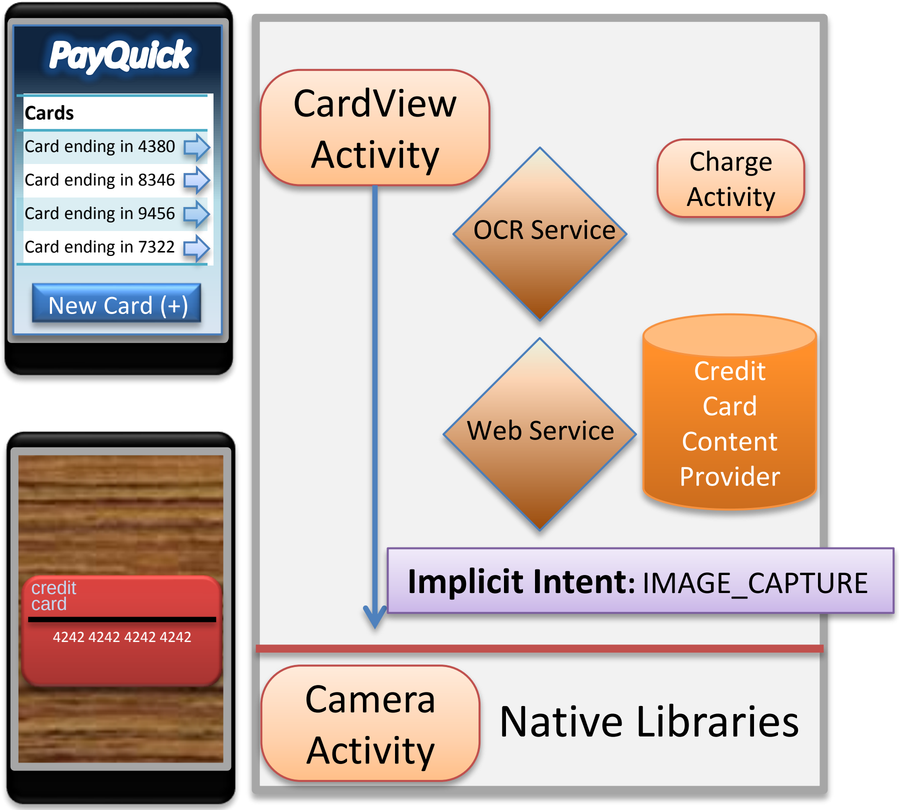
viewMakeCharge

.

.

.

Intents



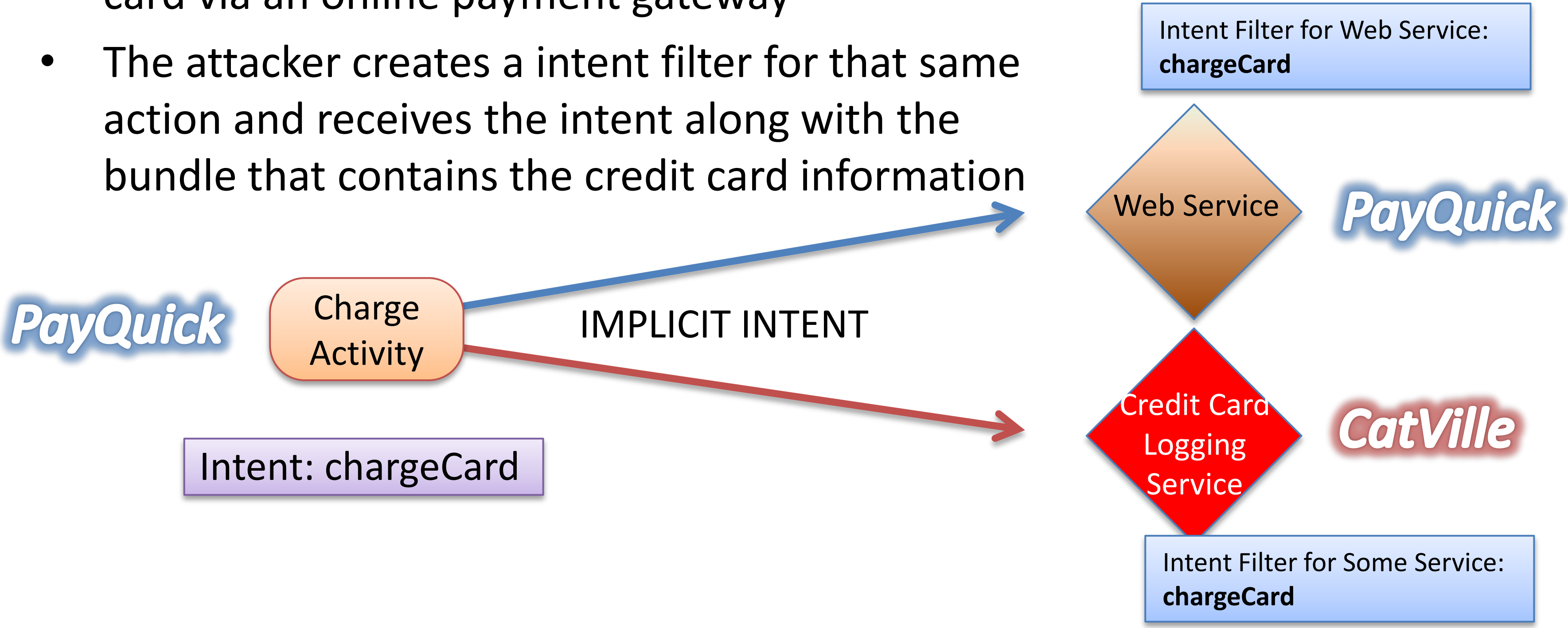
("android.media.action.IMAGE_CAPTURE")

Android App Vulnerability

- Intent
- Capability leaks
- Permission misusage
- Insecure use of system resources

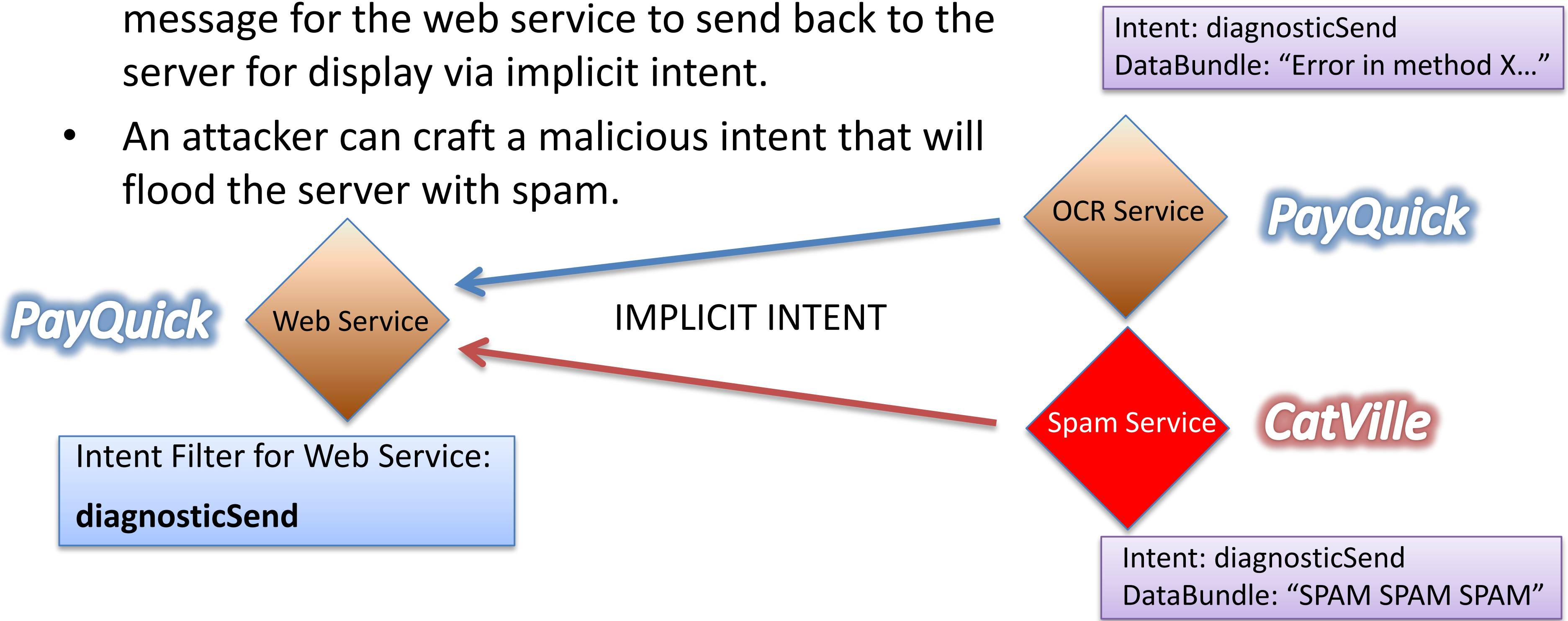
Unauthorized Intent Receipt

- After processing the card image, OCR fires an implicit intent to the Web service to charge the card via an online payment gateway
- The attacker creates a intent filter for that same action and receives the intent along with the bundle that contains the credit card information



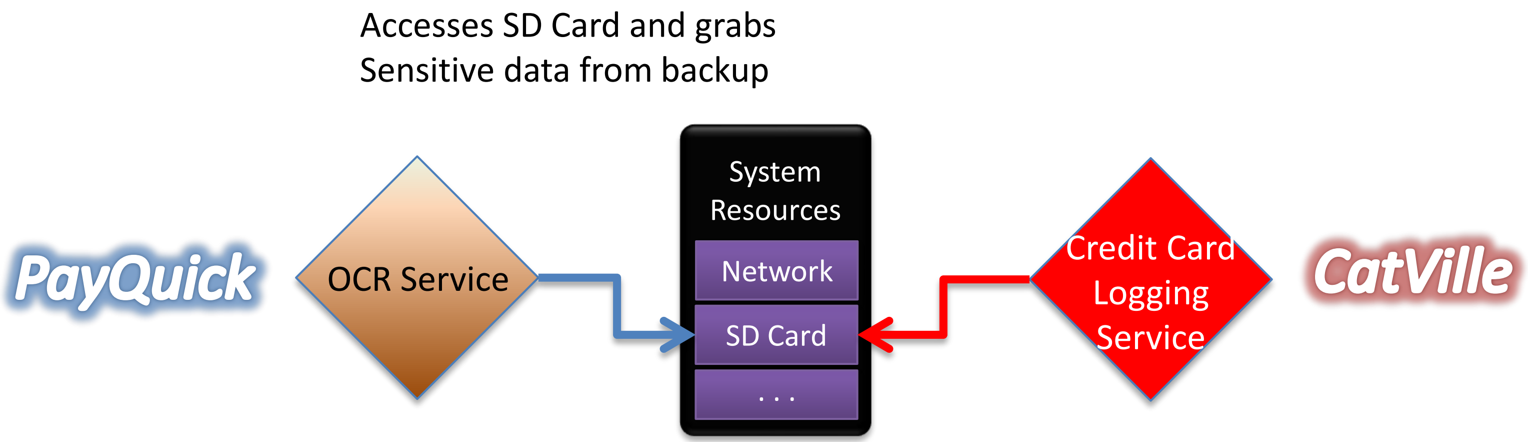
Intent Spoofing

- PayQuick's OCR service can send a diagnostic message for the web service to send back to the server for display via implicit intent.
- An attacker can craft a malicious intent that will flood the server with spam.



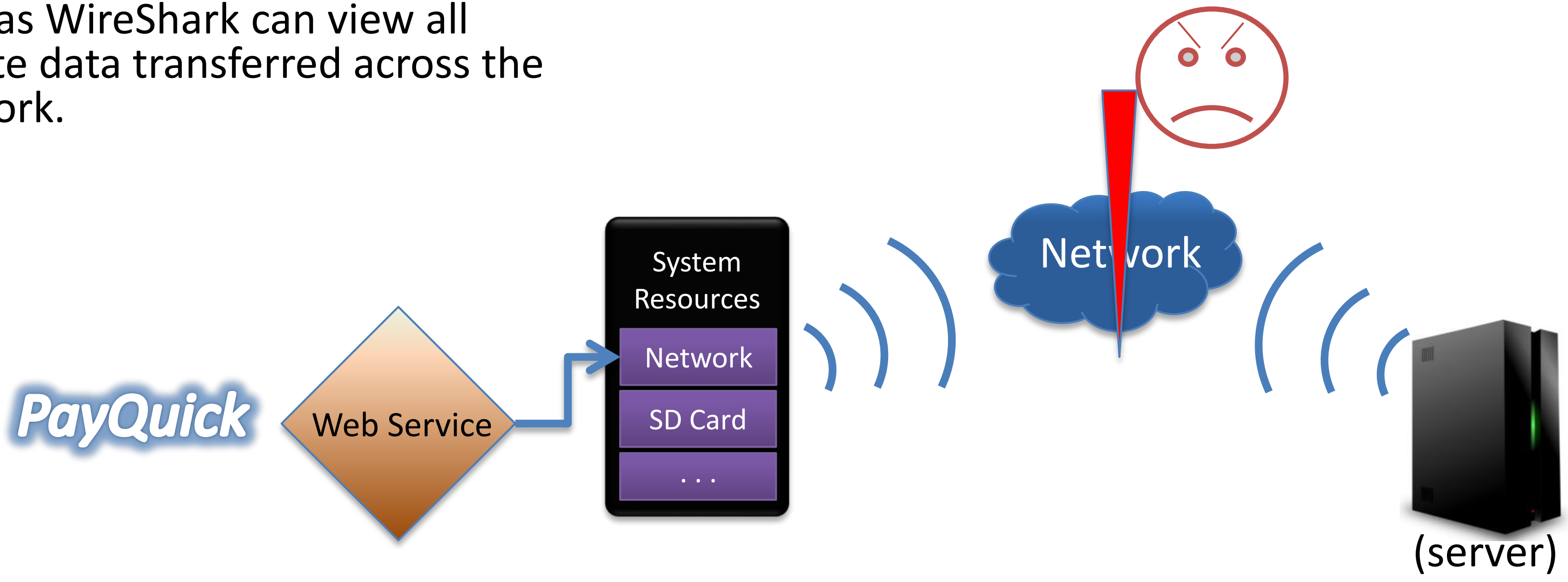
Insecure Storage

- PayQuick stores a backup of all credit card on SD card
- All of this data is readable by any application
  - Catville can access the SD card, and the data is not encrypted.



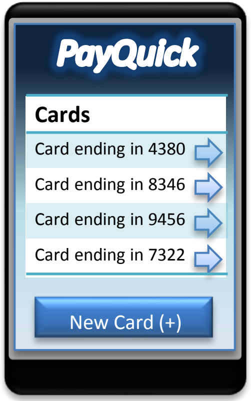
Insecure Network Communication

- PayQuick uses HTTP instead of HTTPS to make online charges.
- A network attacker with a sniffer such as WireShark can view all private data transferred across the network.



Overprivileged Application

- The PayQuick App is grossly over-privileged.

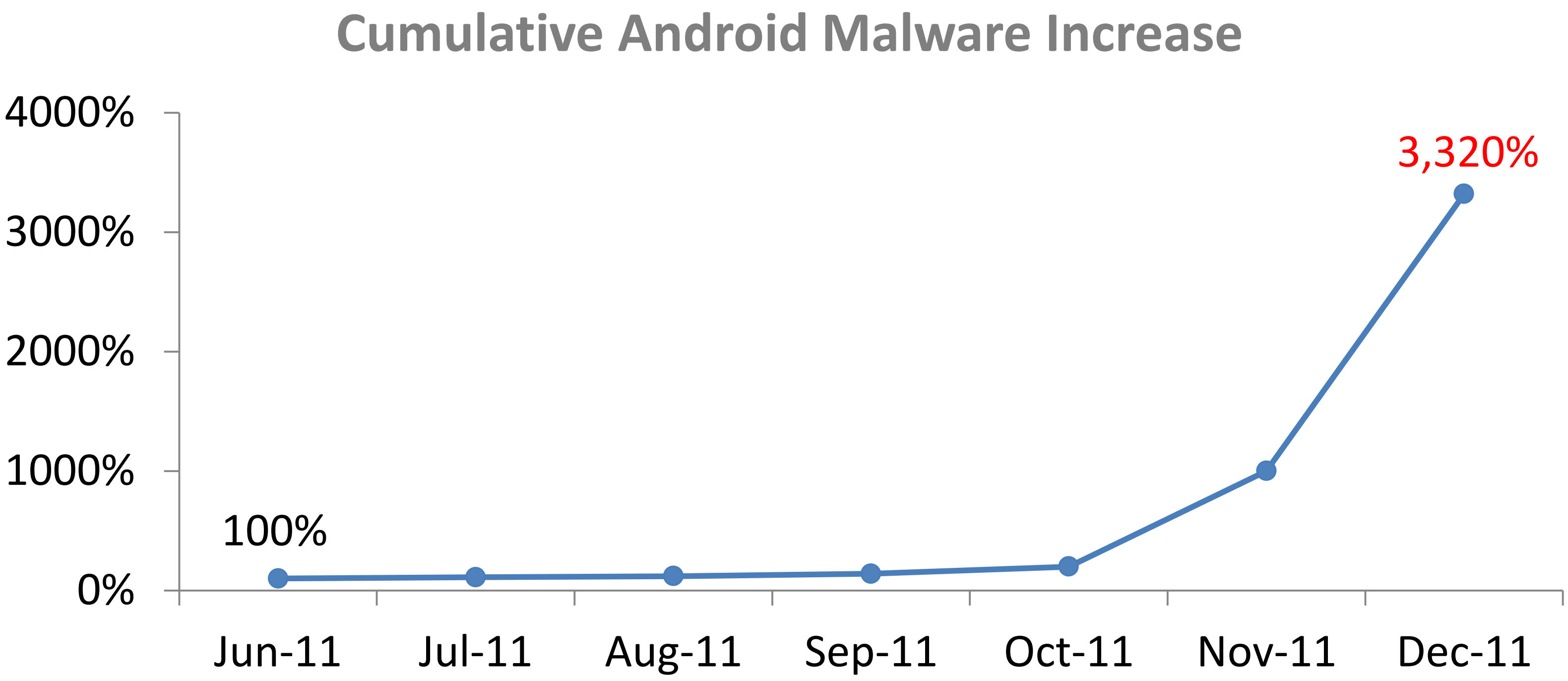


Privileges:
- Location data (GPS)
- Camera Functions
- Bluetooth functions
- Telephony functions
- SMS/MMS functions
- Network/data connections

Extra Privileges

- This violates the principle of least privilege, and should the attacker infect PayQuick, this gives the attacker more privileges.

Android Malware



Malware Characterization

- Installation methods
- Activation mechanisms
- Malicious payloads

Malware Installation

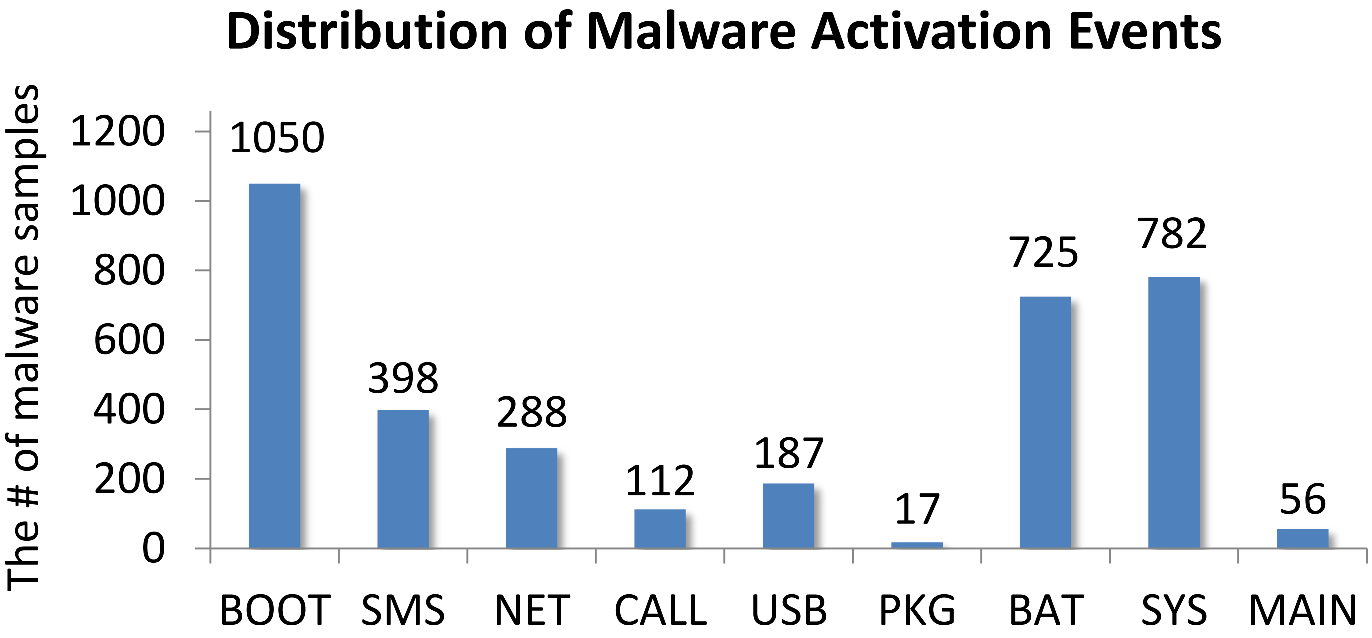
- Users tend not to install malware intentionally
- Attackers trick users into installing malware
  - Repackaging
  - Update attack
  - Drive-by download

Cool App!

CatVille

Activation Mechanisms

- By listening to various system events
- By hijacking the main activity



Malicious Payloads

- Privilege escalation
- Remote control
- Financial charges
- Information collection
- Example malicious behaviors
  - Exploit vulnerabilities in Android kernel & platform
  - Exploit vulnerabilities in other apps
  - Steal users' data
  - Send paid SMS
  - Botnets: download malicious payload & launch other malicious activities

What You Have Learned in CS161

- Software security/Secure coding
- Secure architecture principles/OS security
- Applied crypto basics
- Network security & malicious code
- Web security
- Mobile security

Principles

- Secure design & architecture
- Secure code
- Defending against attacks
- General practice

Secure Design & Architecture Principles

- Isolation
- Least authority/privilege
  - Capabilities
  - Privilege separation
- Policy & enforcement
  - Reference monitor
- Reduce attack surface & TCB
- Auto-update

Secure Code

- Simplicity & modularity
- Auditability
  - Secure code should make it easier to audit
    - E.g., components are side-effect free
- Do not mix code and data
  - Minimize attacker’s control

Input validation

- Make implicit assumptions explicit & enforce it with checks
- Examples
  - Buffer overflow
  - XSS, SQL
  - Server-side validation checks in web apps
- Other issues:
  - Sufficient checks
  - TOCTTOU
  - Authorization checks

Defense

- Defense in depth
- Prevention, detection, remediation, recovery
- Defense should be resilient against evasion
  - Anti-virus
  - If it’s easy to evade, attackers will
  - Use white list instead of black list
  - A proper cost-benefit analysis
- Accountability
  - Audit
  - Provenance

Do not re-invent secure procedures

- Do not invent your own ciphers
- Do not invent your own white list
- Do not invent your own secure communication protocols

Holistic View

- Usability
- Economics