Upgrading Your Android, Elevating My Malware: Privilege Escalation Through Mobile OS Updating

Bader AlBassam
bader@mail.usf.edu
University of South Florida

2015/November/09
1. Introduction
   • Background

2. Update Process

3. Package Manager Service

4. What can go wrong

5. SecUP

Bader AlBassam | Upgrading Your Android, Elevating My Malware:
Presenting a paper written by Xing Et al. Published at Microsoft Research
Previously attacks involving updates involved exploiting unpatched systems using knowledge obtained from patches.
Android, with such a large market share, evolves quickly. There are frequent updates and full revisions to the system approximately every 3 months*. 

*Approximately every 3 months refers to the release cycle of major updates for the Android OS.
Android is a fast evolving system

Android, with such a large market share, evolves quickly. There are frequent updates and full revisions to the system approximately every 3 months*. Also, due to the many versions co-existing, Android is very fragmented.
Definition (Pileup)

*Privilege Escalation through updating.*
Android update process

1. User clicks “update” from OTA.
2. Gets bootloader.img and other system files.
3. Reboot into recovery mode (files are verified here then replaced).
4. Reboot into new OS.
Package Manager Service

Installs, upgrades, configures, and removes packages.
/system/ system apps
/data/app/ all 3rd party apps
PMS registers
- permissions
- shared UID
- activities
- intent filters
- actions
- services
- ...

Upgrading Your Android, Elevating My Malware:
PMS needs to decide what to do in case of a conflict in the case of duplicated attributes and properties. For this, PMS has a data structure `mSettings` that records all the information about existing apps on the old OS.
PMS needs to decide what to do in case of a conflict in the case of duplicated attributes and properties. For this, PMS has a data structure mSettings that records all the information about existing apps on the old OS. Contains \{mPackages, mUserIds, mSahredUsers, mPermissions...\}
PMS has to perform conflict resolution upon upgrades to carefully combine apps together. It uses the same program logic for upgrading as it does for installing new apps.
PMS has to perform conflict resolution upon upgrades to carefully combine apps together. It uses the same program logic for upgrading as it does for installing new apps. Android refers to privileges, resources, and data by names.
Permission Harvesting

Each app has its own permissions defined in `manifest.xml` before installation. An app can define its own permissions. If an app defined a permission not defined in the current system, but defined in an upcoming update, then the app now controls that permission.
Permission Harvesting

Each app has its own permissions defined in `manifest.xml` before installation. An app can define its own permissions. If an app defined a permission not defined in the current system, but defined in an upcoming update, then the app now controls that permission. This includes system permissions.
Shared UID Grabbing

android:sharedUserID ← used to declare shared UID between apps.
Data contamination

Android keeps data for apps under /data/data/PackageName which is owned by a Linux UID. During update, PMS compares UID in pkgSetting with the package name of the new system app being installed. If they match, directory is the directory is kept.
Android keeps data for apps under /data/data/PackageName which is owned by a Linux UID. During update, PMS compares UID in pkgSetting with the package name of the new system app being installed. If they match, directory is the directory is kept. Example attack: Injecting browser database file.
Android doesn’t allow the existence of two permissions with matching names. If a permission is covered in mPermissionTrees, then it cannot be declared.
Android doesn’t allow the existence of two permissions with matching names. If a permission is covered in mPermissionTrees, then it cannot be declared.

Android can still declare permissions UP a tree.
The authors developed a tool called *SecUP* for finding Pileup flaws.
Fig. 2. Architecture of SecUP
Fig. 3. Framework of Vulnerability Detector
Fig. 8. Exploit Opportunities Affected by Vendor Customization