Android Security: Same-ol', Same-ol' (but worse)

Jonathan Levin

http://NewAndroidBook.com

http://Technologeeks.com

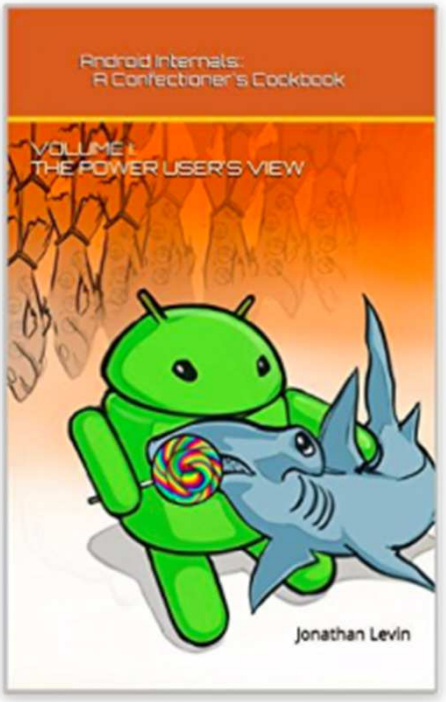
About this talk

- Provides tour of Android security, and insecurity
- Updates Last Year's talk
  - **Last year**: Features **This year:** Vulnerabilities

- http://Technologeeks.com/files/AnSec2.0.pdf

- Covered in "Android Internals: A Confectioner's Cookbook"
  - Chapter 8, to be exact

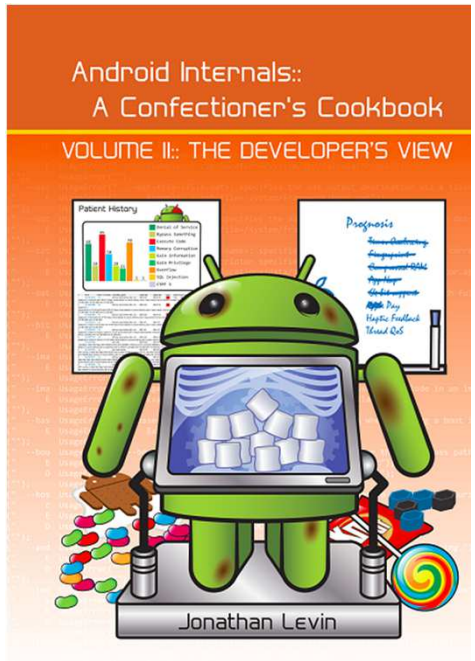
Android Internals::about



“Android Internals: A Confectioner’s Cookbook”

- **深入解析Android 操作系统 - Coming in Chinese (soon)**
- **“파워 유저 관점의 안드로이드 인터널” – In Korean (Oct!)**
- Volume I (Available now): Power User’s view
- Volume II (Available with N): Developer’s View

- Unofficial sequel to Karim Yaghmour’s “Embedded Android”, different focus:
  - More on the **how** and **why** Android frameworks and services work
  - More on Security (this talk is an excerpt from Volume I)
  - (presently) only in-depth books on the subject



- http://www.NewAndroidBook.com/ :
  - **Free** and **powerful** tools, plus bonus materials

- Android Internals & Reverse Engineering: Oct 10th-14th, NYC
  - **http://Technologeeks.com/AIRE**

Attack Surface

- Threat models for mobiles consider three main vectors:
  - Rogue applications (malware)
    - Sandbox applications
    - Enforce Strong Permissions
    - Harden OS Component Security

App Security

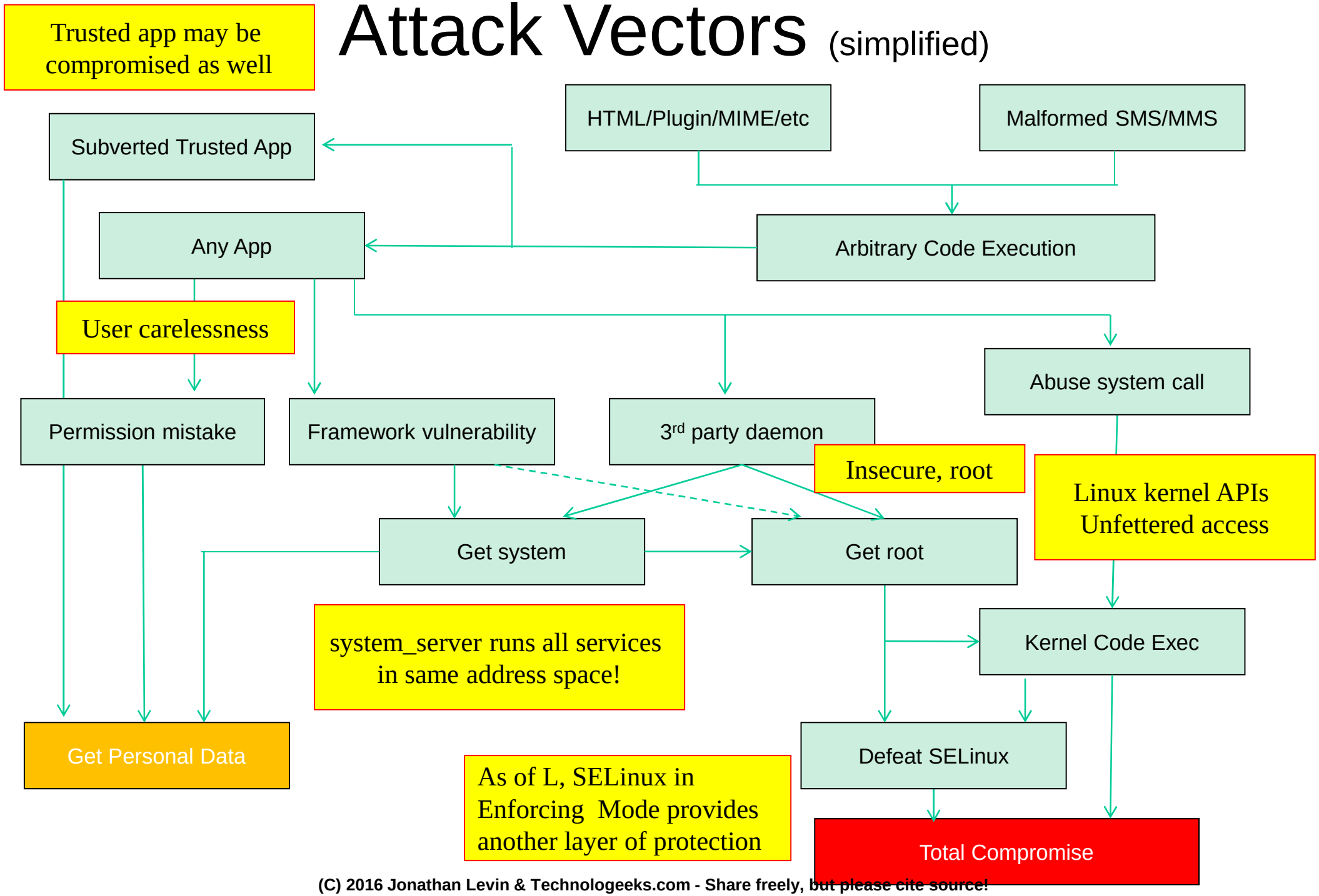
  - Drive-By/Targetted
    - Code Injection via some malicious file format

REMOTE

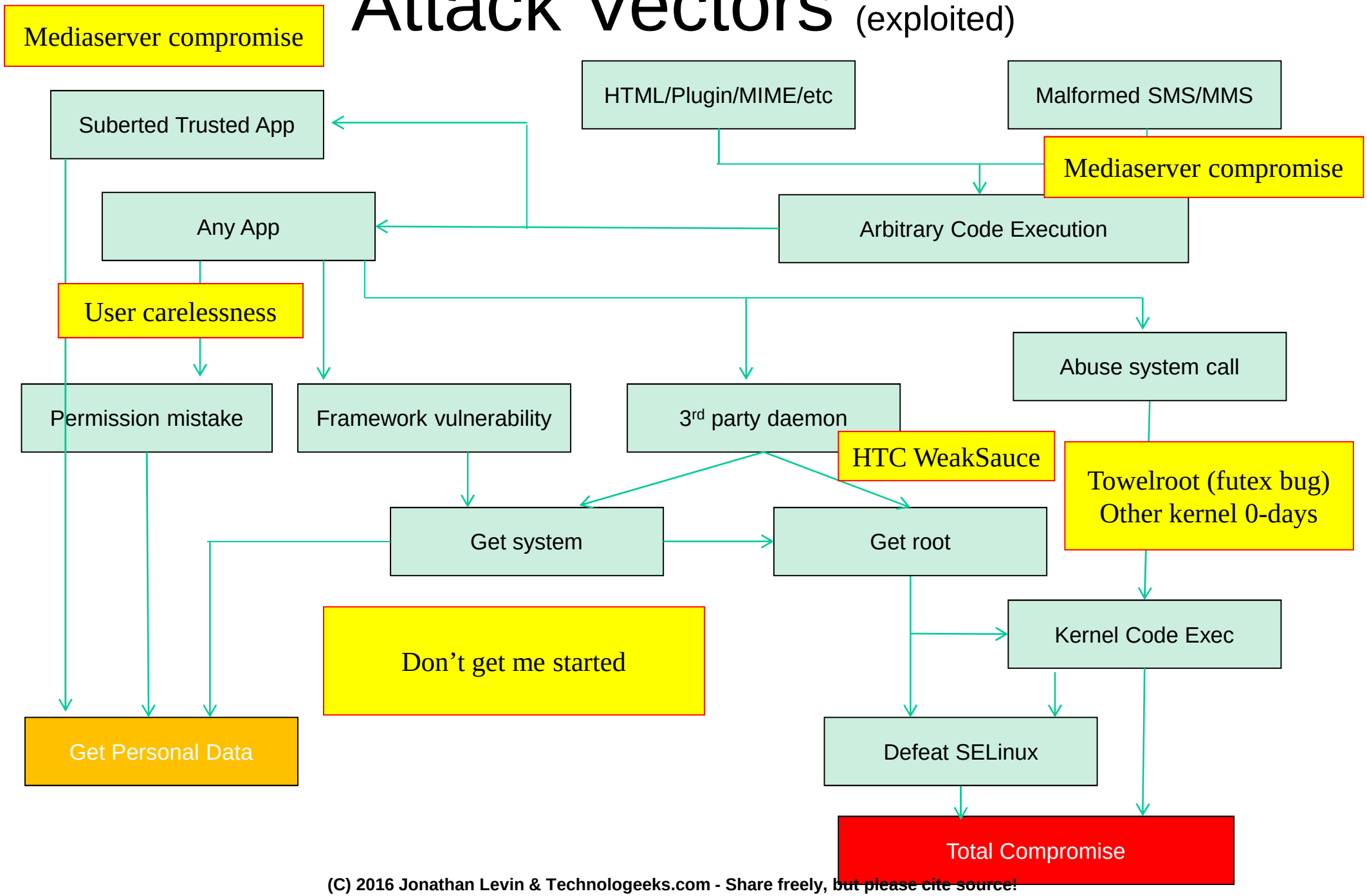
  - Rogue user (device theft, or unauthorized root)
    - Lock Screen
    - Secure Boot Process
    - Encrypt User Data

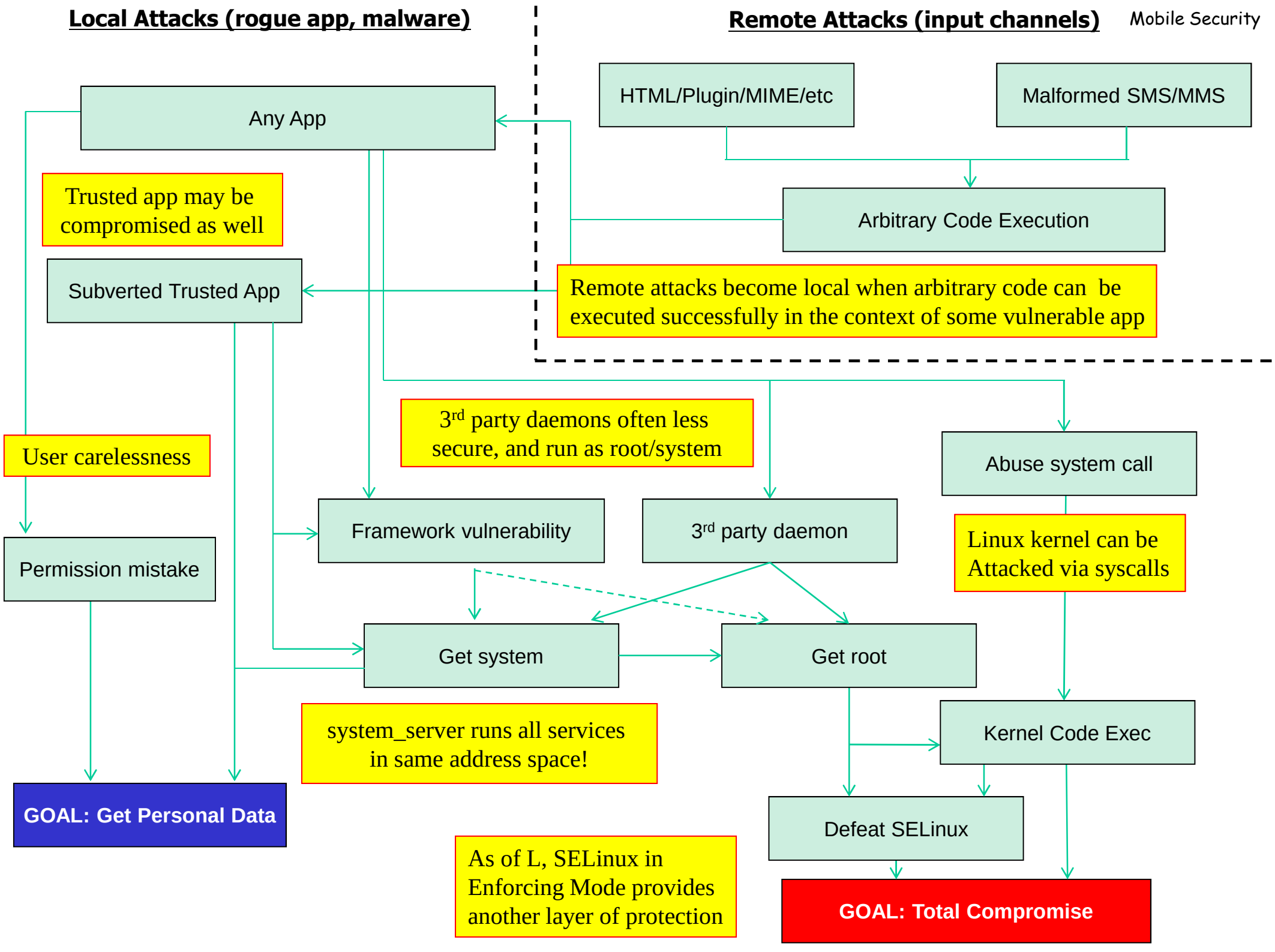
Device Security

Attack Vectors (simplified)



Attack Vectors (exploited)




Local Attacks (rogue app, malware)
Remote Attacks (input channels)
HTML/Plugin/MIME/etc
Malformed SMS/MMS
Any App
Trusted app may be compromised as well
Arbitrary Code Execution
Subverted Trusted App
Remote attacks become local when arbitrary code can be executed successfully in the context of some vulnerable app
3rd party daemons often less secure, and run as root/system
User carelessness
Abuse system call
Framework vulnerability
3rd party daemon
Permission mistake
Linux kernel can be Attacked via syscalls
Get system
Get root
system_server runs all services in same address space!
Kernel Code Exec
GOAL: Get Personal Data
Defeat SELinux
As of L, SELinux in Enforcing Mode provides another layer of protection
GOAL: Total Compromise

CVEs

- “Common Vulnerability Exposure” index
- Publishes and standardized security vulnerabilities
  - Goal: Uniquely define particular and specific bugs
- Main database is at http://cve.mitre.org *
- Searchable database is at http://www.cvedetails.com/

https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

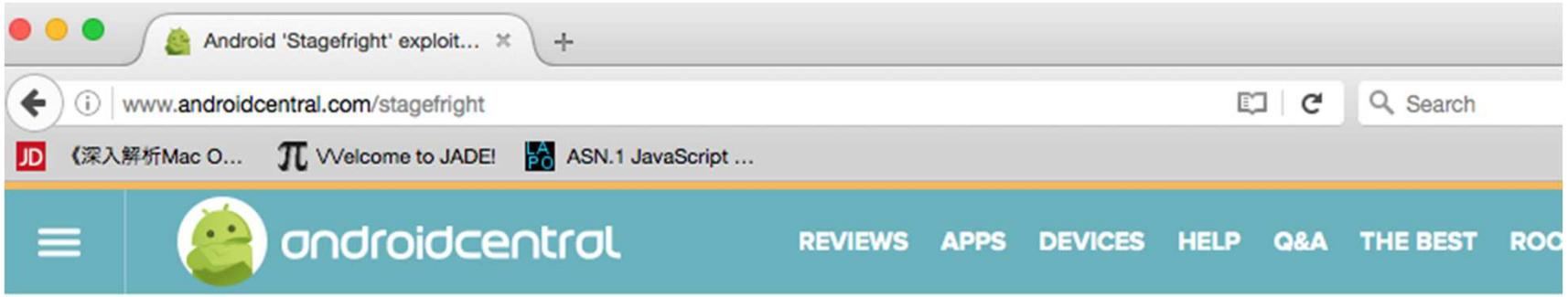
* - (pronounced: might-er)

A little history

Table 8-1: Some of the most significant vulnerabilities in Android's History

CVE/Name	Affects	Specificity	Vector	Impact
CVE-2012-6636	1.0-4.1	3rd Party	Remote	The WebView component insecurely processes Javascript, compromising the default Android Browser as well as application WebViews
CVE-2014-7911	1.0-4.x	Android	Local	luni/src/main/java/java/io/ObjectInputStream.java improper deserialization results in local code execution as `system`
CVE-2014-4322	4.x	Vendor (msm)	Local	Insecure `ioctl(2)` in Qualcommm's QSEECOM driver yields arbitrary kernel memory overwrite
CVE-2014-3153	4.x	Linux	Local	The `futex(2)` system call has a bug leading to arbitray kernel memory overwrite (Towelroot)
WeakSauce	4.x	Vendor (HTC)	Local	The proprietary `dmagent` insecurely copies files, enabling local privilege escalation (see book site)
CVE-2015-1805	4.x and up	Linux	Local	Arbitrary memory overwrite (exploited in the wild and only patched by Google on March 2016!)
CVE-2015-3636	4.x-6.0	Linux	Local	A use-after-free bug in the kernel leads to arbitrary memory overwrite (used by pingpong root).
CVE-2015-3824	4.x - 5.x	Android	Remote	A vulnerability in the `stagefright` framework allows arbitrary code execution in `mediaserver` through a malformed MMS or video file
CVE-2016-0728	4.4 and up	Linux	Local	An integer overflow in keyrings facility enables a privilege escalation (mitigated by SELinux in 5.0)
CVE-2016-0819 CVE-2016-0819	4.4-6.0.1	Vendor (msm)	Local	Vulnerabilities in Qualcomm specific drivers yield arbitrary kernel memory overwrite

Google Response



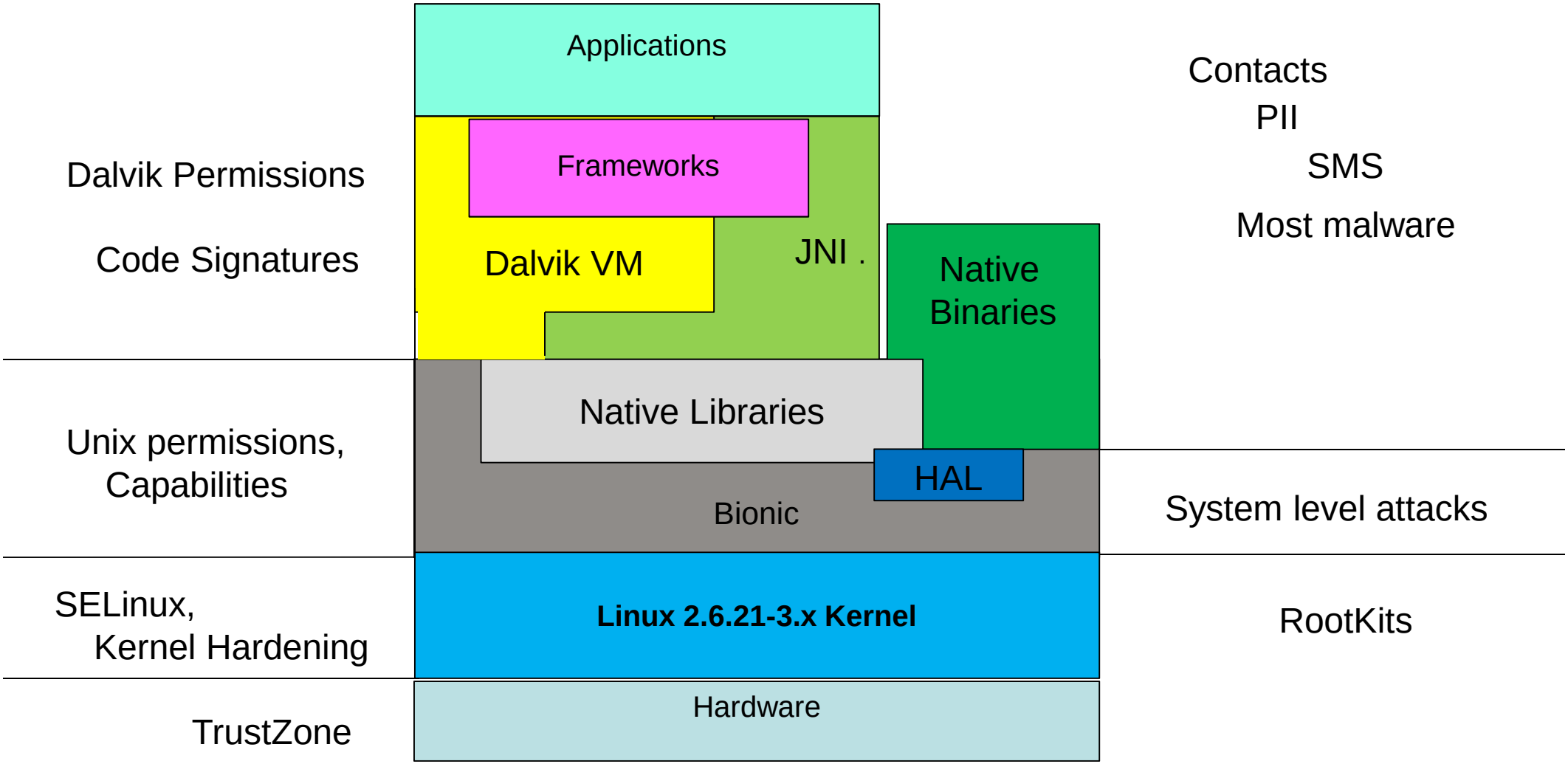
And for its part, Google in July reiterated to *Android Central* that there are multiple mechanisms in place to protect users.

> We thank Joshua Drake for his contributions. The security of Android users is extremely important to us and so we responded quickly and patches have already been provided to partners that can be applied to any device.
>
> Most Android devices, including all newer devices, have multiple technologies that are designed to make exploitation more difficult. Android devices also include an application sandbox designed to protect user data and other applications on the device.

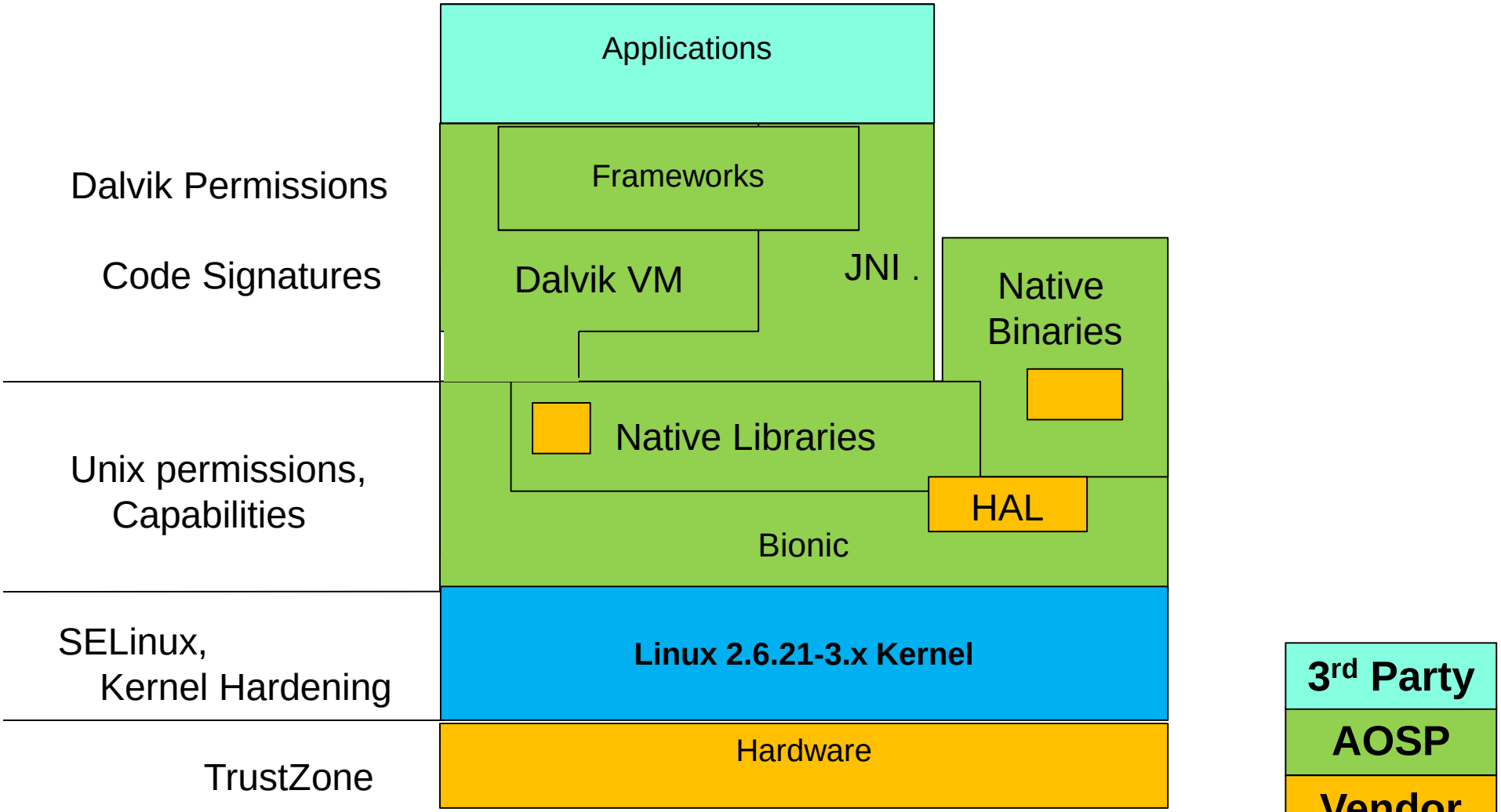
Android Application Security Model

- Android's security is derived from that of Linux and Java
- Linux inheritance:
  - Applications run as separate UIDs
  - Kernel supports miscellaneous tweaks
  - Network access filtered in kernel by UserID
  - SELinux ("SEAndroid") policies used extensively
- Java Inheritance:
  - Dalvik VM provides sandbox for applications
  - Declarative security model for operations

Android Architecture

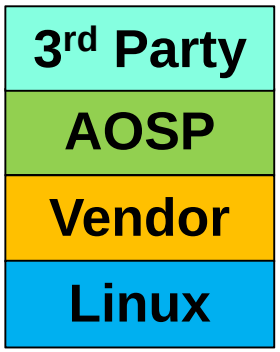


Android Code Bases



Scope of Vulnerabilities

- **3rd Party** : Affects only devices with installed App
- **AOSP** : Affects ALL Android devices (global)
- **Vendor** : Affects device vendor **or chipset vendor**
- **Linux** : Universal (also desktops, servers)
  - Recommended: Monoculture on back of envelope (Geer, Usenix 2005)



Updates (or lack thereof)

- Android is becoming Windows of the mobile world

  Microsoft , 2004 : Google : 2016

- Android's Update Policy is simply horrendous
  - Getting better with N (finally)
  - Still problematic due to existing fragmentation
  - Exacerbated by vendor, carrier policies

- Impact: 70-80% devices left vulnerable, unpatched

Tour of Android Vulnerabilities

Prelude: Vectors for Code Injection

- Buffer Overflows (stack: rare, heap: uncommon)
  - Example: char *c = malloc(10); strcpy (c, str...);
- Integer Overflows (common)
  - Example: int a; int b; int c = a+b
  - Lethal when used as basis for malloc()
- Use-After-Free (ubiquitous)
  - Example: char *c = malloc(..); free (c); *c =....;

CounterMeasures for Code Injection

- isolation/sandboxing, pray SELinux works
- ASLR (ELF PIE + randomize_va_space)
- DEP
- Stack Canaries and compiler level protections.
- PXN (Privileged eXecute Never)
- Restrict `dmesg` and kernel pointers (via sysctl)
  - kernel.kptr_restrict = 2

Counter-

CounterMeasures for Code Injection

- isolation/sandboxing, pray SELinux works — Opt-in, still not tight enough, keep praying
- ASLR (ELF PIE + randomize_va_space) — Info Leaks, Feng Shui, sprays
- DEP — Return Oriented Programming (ROP)
- Stack Canaries and compiler level protections. — Directed overwrite
- PXN (Privileged eXecute Never) — Overflow in kernel
- Restrict `dmesg` and kernel pointers (via sysctl)
  - kernel.kptr_restrict = 2 — Info Leaks, sprays in kernel

Top 3 risks - #3: File Formats

- File Formats: Codec, HTML/CSS, JS
  - Vector: Remote (and also Local)
  - Impact: Privilege Escalation – usually media/drm/system
  - Reason: overly complex formats, interpreters
    - Particularly, CSS/JS parsing, media files
  - Countermeasures:
    - Standard Code Injection Countermeasures
    - SELinux

Top 3 risks - #3: File Formats

- Case Study:
  - (Another) StageFright Bug (CVE-2015-3864)
  - http://googleprojectzero.blogspot.com/2015/09/stagefrightened.html
  - https://github.com/NorthBit/Metaphor
  - https://www.exploit-db.com/exploits/38226/

```
1886        case FOURCC('t', 'x', '3', 'g'):
1887        {
1888            uint32_t type;
1889            const void *data;
1890            size_t size = 0;
1891            if (!mLastTrack->meta->findData(
1892                    kKeyTextFormatData, &type, &data, &size)) {
1893                size = 0;
1894            }
1895
1896            uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
1897            if (buffer == NULL) {
1898                return ERROR_MALFORMED;
1899            }
1900
1901            if (size > 0) {
1902                memcpy(buffer, data, size);
1903            }
1904
1905            if ((size_t)(mDataSource->readAt(*offset, buffer + size, chunk_size))
1906                    < chunk_size) {
1907                delete[] buffer;
1908                buffer = NULL;
```

Top 3 risks - #2: Binder

- Binder: Deliberately Malformed parcels
  - Vector: Local
  - Impact: Privilege Escalation – system, likely root
  - Reason: **LOUSY NATIVE CODE, NO AIDL**
  - CounterMeasures: q.v. Code Injection

Top 3 risks - #2: Binder

- Case Study #1:
  - LibCUtils–CVE-2015-1528 (< 5.1)
  - **http://seclists.org/fulldisclosure/2015/Mar/63**
  - **https://www.blackhat.com/docs/us-15/materials/us-15-Gong-Fuzzing-Android-System-Services-By-Binder-Call-To-Escalate-Privilege-wp.pdf**

- Case Study #2:
  - “Hey, Your Parcel Looks Bad” (BlackHat Asia ‘16)

https://www.blackhat.com/docs/asia-16/materials/asia-16-He-Hey-Your-Parcel-Looks-Bad-Fuzzing-And-Exploiting-Parcelization-Vulnerabilities-In-Android.pdf

Top 3 risks - #1: Linux Kernel

- Linux Kernel: vulnerable system calls, or network stack

  - Vector: Local (usually), Remote (very rare)

  - Impact: Full system compromise

  - Reason: Too many cooks, with too many features

- Solution: SELinux

  - Limited scope, **not designed for app security**

Top 3 risks - #1: Linux Kernel

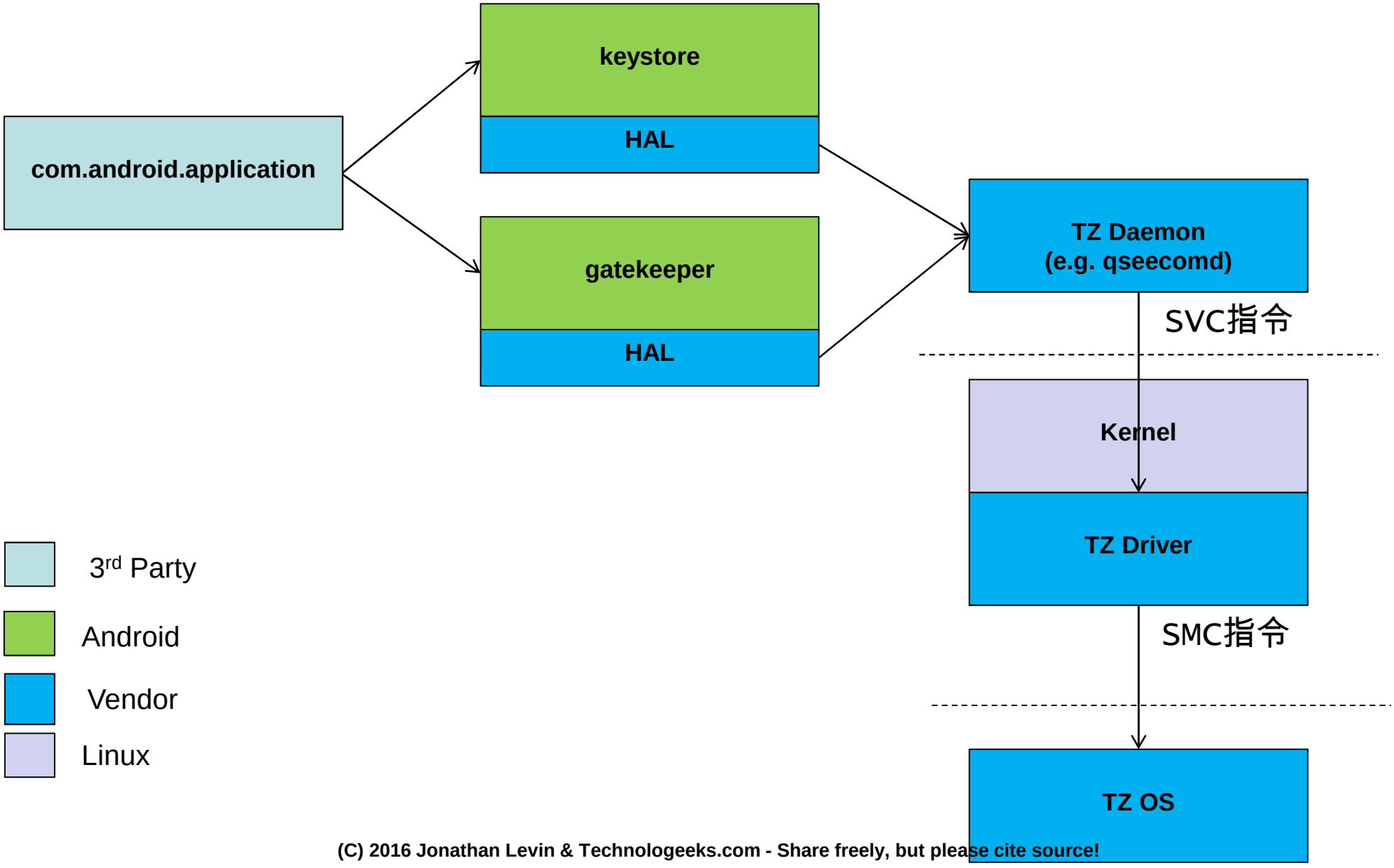
- Case Study #1:
  - TowelRoot (CVE-2014-3153) – futex bug
- Case Study #2:
  - PingPong Root (CVE-2015-3636) – ICMP sockets
- Case Study #3: (SELinux blocks this one)
  - Keyrings (CVE-2016-0728)
  - **https://www.exploit-db.com/exploits/40003/**

Bonus Risk: TrustZone

- ARMv7/v8 memory separation at bus level
- SCR separates “secure world” from non secure
  - In ARMv8, coupled with Exception Levels (ELx)

- “Secure World” runs own OS(!), providing:
  - Keystore Access (“hardware backed cryptography)
  - Gatekeeper Functionality (crypto-tokens)
  - PRNG
  - Boot/System Integrity verification (e.g. Samsung TIMA)

Android & TrustZone



TrustZone Vulnerabilities

- Problem: TZ OS is often vendor defined, closed source
  - Google trying to standardize with “Trusty OS”
  - Qualcomm (most common) has own, and **BUGGY**

- http://bits-please.blogspot.com
  - AMAZING detail of trustzone exploitation on MSM, step-by-step
  - Particularly as of /2015/03/getting-arbitrary-code-execution-in.html

Android Vulnerabilities

CVE	Bug(s)	Severity	Updated versions	Date reported
CVE-2015-6639	ANDROID-24446875*	Critical	5.0, 5.1.1, 6.0, 6.0.1	Sep 23, 2015
CVE-2015-6647	ANDROID-24441554*	Critical	5.0, 5.1.1, 6.0, 6.0.1	Sep 27, 2015

CVE	Bug(s)	Severity	Updated versions	Date reported
CVE-2016-0825	ANDROID-20860039*	High	6.0.1	Google Internal

CVE	Android bugs	Severity	Updated Nexus devices	Date reported
CVE-2016-2431	24968809*	Critical	Nexus 5, Nexus 6, Nexus 7 (2013), Android One	Oct 15, 2015
CVE-2016-2432	25913059*	Critical	Nexus 6, Android One	Nov 28, 2015

Hindsight is 20/20

- All the CVEs discussed are obvious, in retrospect:

> “Reports that say that something hasn't happened are always interesting to me, because as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns – the ones we don't know we don't know. And if one looks throughout the history of our country and other free countries, it is the latter category that tend to be the difficult ones.
>
> D. Rumsfeld, 2002,
> http://archive.defense.gov/Transcripts/Transcript.aspx?TranscriptID=2636

- **Known knowns = CVEs, Past Vulnerabilities**
- **Known unknowns = Vulnerabilities we suspect**
- **Unknown unknowns = 0-days in the wild**

So, overall..

- Sad Truth: Android “spitballs” Linux features together
- Sometimes it sticks. More often than not.. It doesn’t.

Resources

- The Book website: http://NewAndroidBook.com
- Technologeeks.com: http://Technologeeks.com/
- Android Internals: http://Technologeeks.com/AIRE