SEPTEMBER 12 – 14, 2012
GRAND HYATT, SAN FRANCISCO

Truth, Lies and Decisions

Moving Forward in an Insecure World

Mobile Vulnerability Assessment: There's an App for That!

Jon Oberheide
CTO, Duo Security
jono@duosecurity.com

Organized by RAPID7

Introduction

- Jon Oberheide
- CTO, Duo Security

- Today
  - Fun with Android security and mobile vulnerability assessment

Agenda

- **Mobile vulnerability assessment**
- X-Ray for Android
- Preliminary X-Ray Results
- Wrap-up

Brief History of Vuln Assessment

1992: Chris Klaus creates the
Internet Scanner at Georgia Tech

Modern day VA

nexpose®

BIGFIX®

QUALYS®

eEye Digital Security®

Present day: Many forms of vulnerability assessment, management, mitigation

What about mobile?

Mobile?

Ideal mobile security

Why do we need mobile vulnerability assessment?

- In a perfect world...
- *ASOP*: Google ships a secure base platform
- *OEM*: Samsung doesn't introduce any vulnerabilities in its customization of Android
- *Carrier*: T-Mobile rolls out rapid OTA updates to keep users up to date and patched.

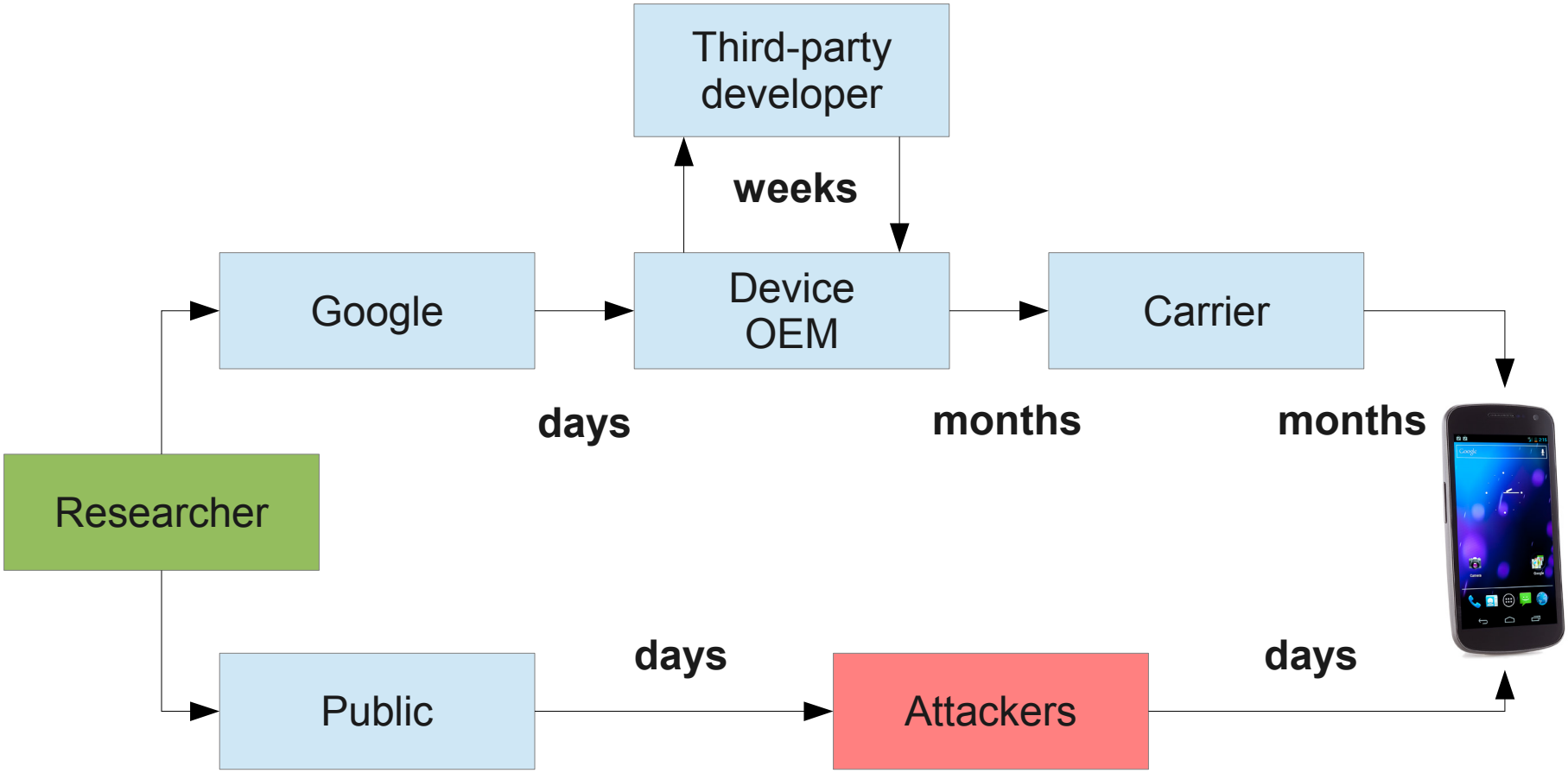
The real world

- In reality:
- *ASOP*: Android base platform is far from perfect
- *OEM*: Customizations by device OEMs are a large source of vulnerabilities.
- *Carrier*: Updates are not made available for months and sometimes even years.

All software has bugs, mobile or otherwise.

Here's where mobile differs from PC world.

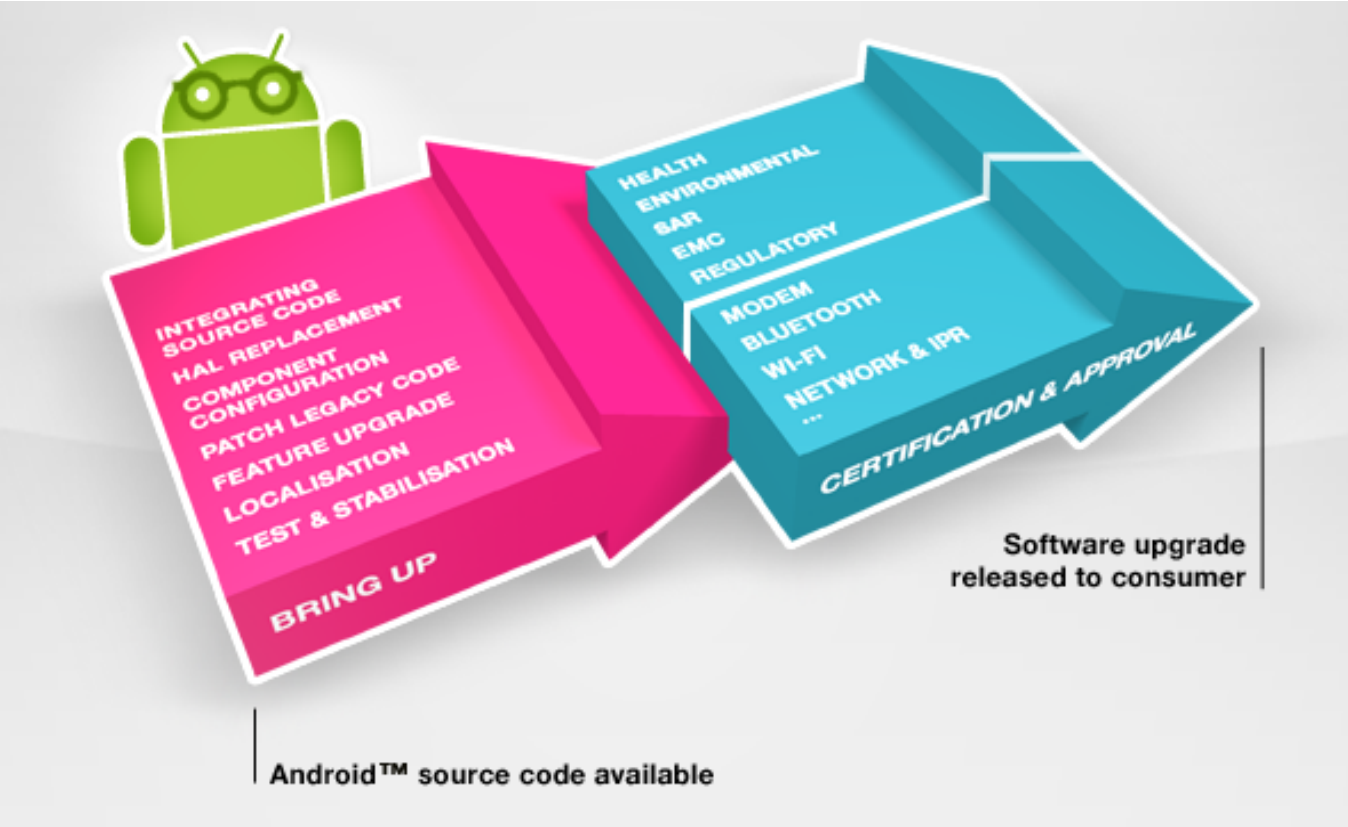
Disclosure & patching process



Challenges in patching

- Why is mobile patching challenging?
  - Complicated software supply chain
  - Testing, testing, testing
  - Risk of bricking devices
  - Inverted economic incentives

What the carriers say



"*Patches must be integrated and tested for different platforms to ensure the best possible user experience. Therefore, distribution varies by manufacturer and device.*" - AT&T

Challenges in mobile VA

- Mobile device software is ***different***
  - Diverse set of software, hardware, configuration
  - No control of software loadset
- Want to assess your device's vulnerabilities?
  - Rely on version numbers? Rely on exploitation?
  - Diverse set of devices, many device-specific vulnerabilities
- Want to patch your device's vulnerabilities?
  - Can't patch the device, unless rooted

Privilege escalation vulnerabilities

Why are privilege escalation vulnerabilities important?

- Android security model
  - Permissions framework, “sandboxing” (uid/gid)
  - Compromise of browser (or other app) != full control of device
- Privilege escalation vulnerabilities
  - Unprivileged code execution → Privileged code execution
  - Publicly released to allow users to jaibreak their devices
  - Public exploits reused by mobile malware to root victim's devices
  - Private exploits available, but little need currently

Quick trivia

- What's wrong with the following code?

```
/* Code intended to run with elevated privileges */
do_stuff_as_privileged();

/* Drop privileges to unprivileged user */
setuid(uid);

/* Code intended to run with lower privileges */
do_stuff_as_unprivileged();
```

- Assuming a uid/euid=0 process dropping privileges...

Zimperlich vulnerability

- Return value not checked! setuid(2) can fail:

```
EAGAIN The uid does not match the current
       uid and uid brings process over its
       RLIMIT_NPROC resource limit.
```

- Android's zygote does fail if setuid does:

```
err = setuid(uid);
if (err < 0) {
    LOGW("cannot setuid(%d): %s", uid, strerror(errno));
}
```

- Fork until limit, when setuid fails, app runs as uid 0!

Android privesc vulns at a glance

- ASHMEM: Android kernel mods, no mprotect check
- Exploid: no netlink source check, inherited from udev
- Gingerbreak: no netlink source check, GOT overwrite
- Levitator: My_First_Kernel_Module.ko, kmem read/write
- Mempodroid: inherited from upstream Linux kernel
- RageAgainstTheCage: no setuid retval check
- Wunderbar: inherited from upstream Linux kernel
- Zimperlich: no setuid retval check
- ZergRush: UAF in libsysutils

Mobile vulns are nothing special!
Same old mistakes, different platform!

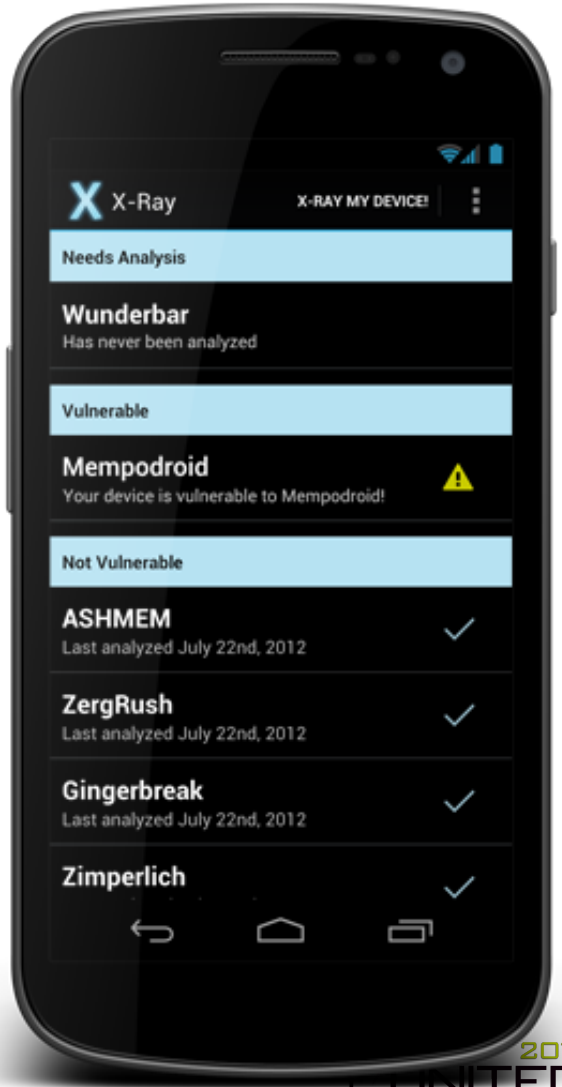
Agenda

- Mobile vulnerability assessment
- **X-Ray for Android**
- Preliminary X-Ray Results
- Wrap-up

X-Ray for Android

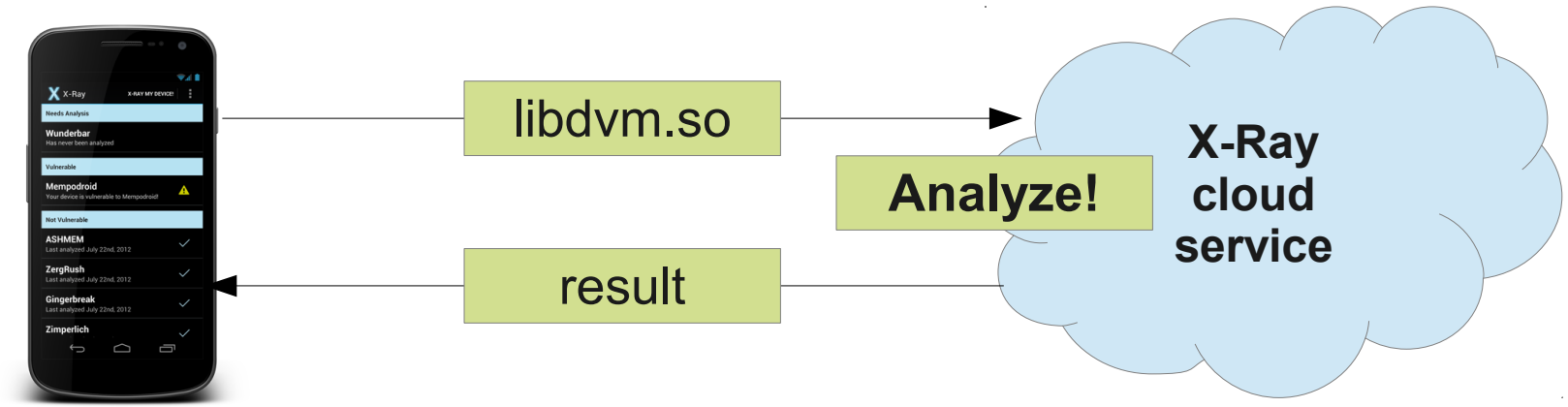
- X-Ray for Android
  - First app to perform _actual_ vulnerability assessment on mobile
  - Detects 8 of the most common Android privilege escalation vulnerabilities
  - Works without any special privileges or permissions
  - Freely available for end users to run

http://xray.io

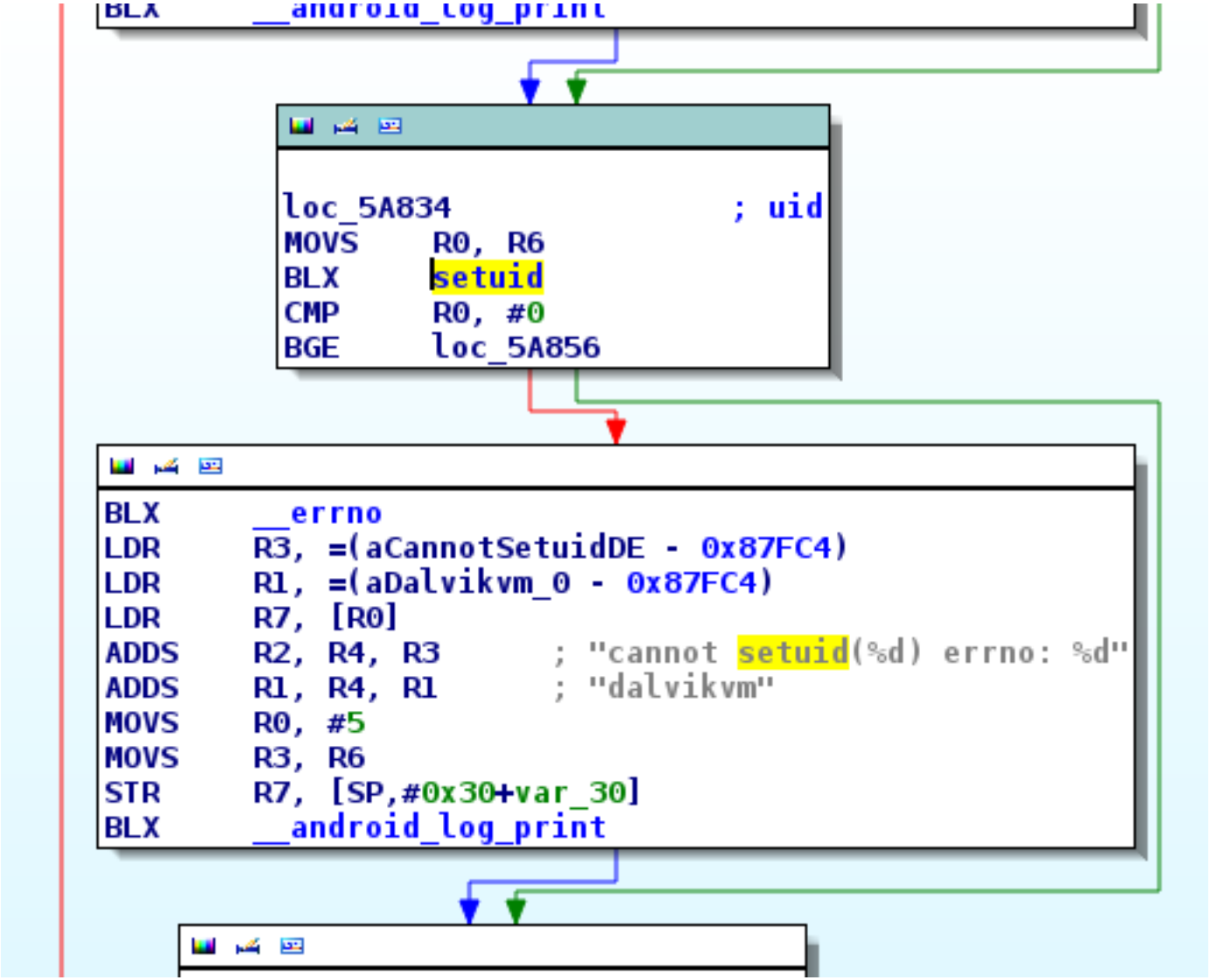


Static probes

- Static probes
  - Can identify vulnerabilities using static analysis
  - Send up vulnerable component (eg. binary, library) to service
  - Disassemble and look for patched/vulnerable code paths

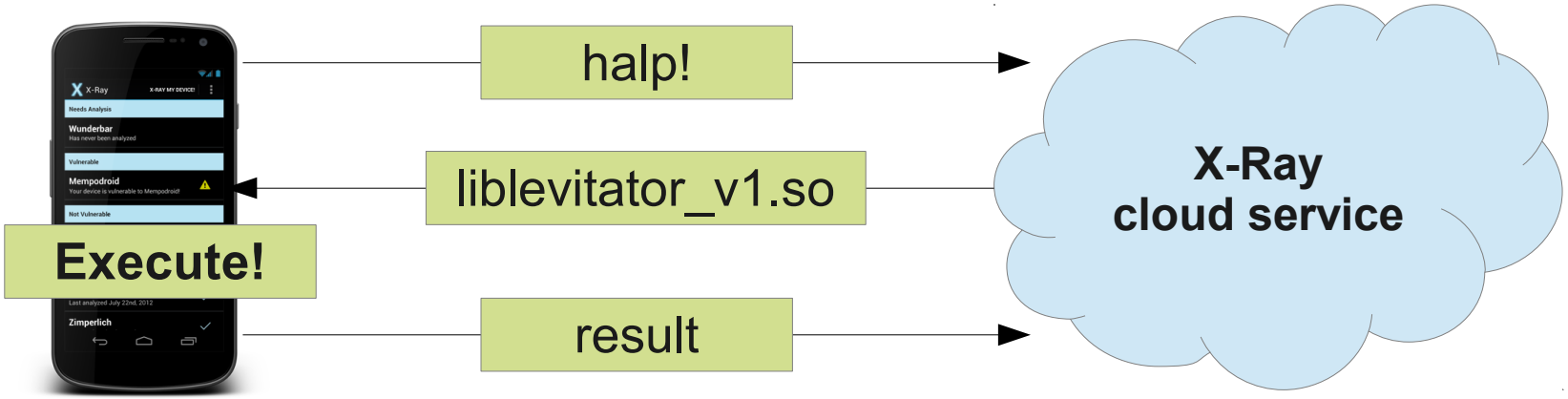


Static probe example: Zimperlich



Dynamic probes

- Dynamic probes
  - Not all vulnerabilities are in software components we can access
  - Example: kernel vulns, kernel image not accessible by X-Ray
  - Probe locally for vulnerability presence!



Dynamic probe example: Levitator

```
pkg.ui32BridgeID = CONNECT_SERVICES;
pkg.ui32Size = sizeof(pkg);
pkg.ui32InBufferSize = 0;
pkg.pvParamIn = NULL;
pkg.ui32OutBufferSize = DUMP_SIZE;
pkg.pvParamOut = dump;

ret = ioctl(fd, 0, &pkg);
if (ret == 0) {
    result = "vulnerable|leaked kernel memory";
    goto done;
} else {
    result = "patched|can't leak kernel memory";
    goto done;
}
```

Agenda

- Mobile vulnerability assessment
- X-Ray for Android
- **Preliminary X-Ray Results**
- Wrap-up

X-Ray Launch

- Launched X-Ray app publicly in late July the week of BlackHat
- Not in Google Play, but download at http://xray.io
- Scan results collected from users who downloaded and ran the X-Ray app on their Android device

26,257 devices
1,146 models
140 countries

Scary numbers

- First number spit out during our analysis

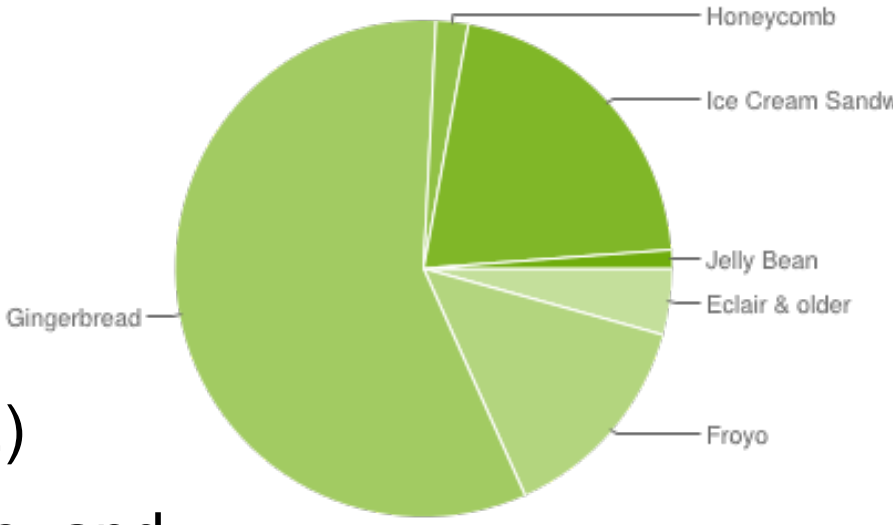
75.8% vulnerable

- WHOA.
- But we had some known false positives in the probes.
- After correcting the FPs:

59.8% vulnerable

Methodology

- How to extrapolate out to global Android population?
  - Selection bias?
  - Google provides stats on Android versions →



- If we saw 98.8% (1020 / 1032) of 2.2 devices were vulnerable, and 2.2 makes up 15.5% of Android globally, that contributes 15.3% to the total % of vulnerable Android devices.

http://developer.android.com/about/dashboards/index.html

Breakdown of data

Version bucket	Vulnerable (X-Ray)	Global population (Google)	Vulnerable globally (extrapolated)
1.5	100%	0.2%	0.2%
1.6	100%	0.5%	0.5%
2.1	96.7%	4.2%	4.1%
2.2	98.8%	15.5%	15.3%
2.3	100%	0.3%	0.3%
2.3.3	63%	60.3%	38.1%
3.1	0%	0.5%	0%
3.2	0%	1.8%	0%
4.0	21.4%	0.1%	0.02%
4.0.3	9.6%	15.8%	1.4%
4.1	0%	0.5%	0%

Interesting tidbits: version numbers

- So, should we use version numbers for VA?
  - Data says....NO!
- Not many earlier version numbers that shouldn't be vulnerable, but are.
- A lot more later version numbers that ***should be patched, but are still vulnerable***!
- Patch regressions, bad third-party ROMs, etc

Interesting tidbit: affected devices

```
/*
 * levitator.c
...
 *      The vulnerability affects Android devices with the PowerVR SGX chipset
 *      which includes popular models like the Nexus S and Galaxy S series. The
 *      vulnerability was patched in the Android 2.3.6 OTA update.
 */
```

```
mysql> SELECT COUNT(DISTINCT(model))
FROM results WHERE probe='levitator'
AND result='vulnerable';
+------------------------+
| COUNT(DISTINCT(model)) |
+------------------------+
|                    136 |
+------------------------+
1 row in set (0.41 sec)
```

```
mysql> SELECT DISTINCT(model) FROM results
WHERE probe='levitator' AND
result='vulnerable' AND model LIKE '%Kindle%'
+-------------+
| model       |
+-------------+
| Kindle Fire |
+-------------+
1 row in set (0.43 sec)
```

X-Ray gives global visibility into affected device models

Agenda

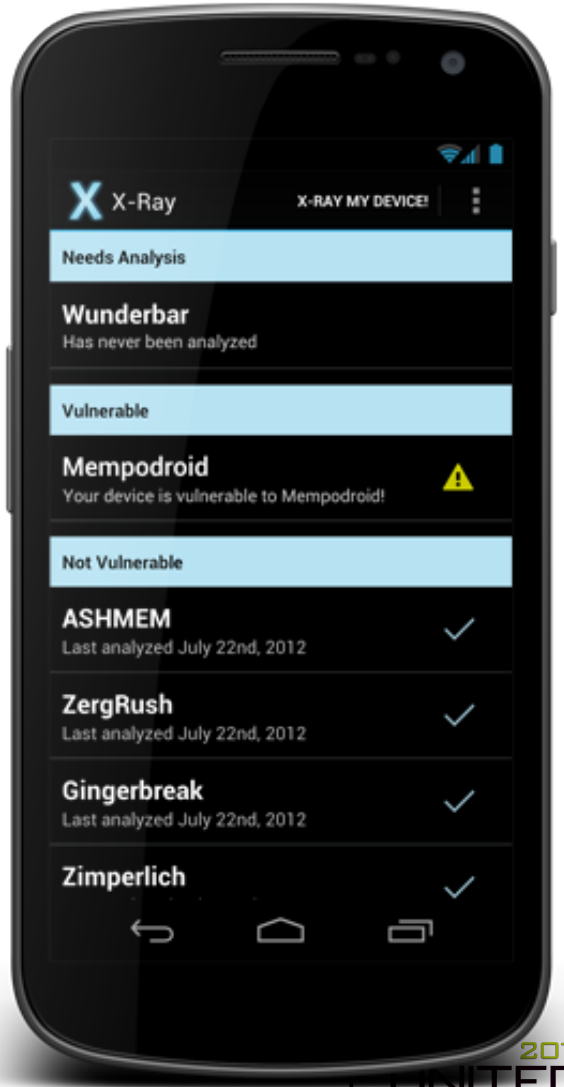
- Mobile vulnerability assessment
- X-Ray for Android
- Preliminary X-Ray Results
- **Wrap-up**

Lessons learned from X-Ray

- Slow patching is as big as a problem as many suspected
  - X-Ray demonstrates the problem with hard data
- Mobile VA techniques can't rely on version numbers
  - Positive identification based on the actual code
- Mobile patching practices need to change *somehow*
  - Centralized? Third-party ecosystem?

What's next for X-Ray?

- Additional vulnerability probes
  - Including non-privesc vulns
    (I'M LOOKING AT YOU WEBKIT)
- Long-term vulnerable population tracking
  - Update stats as Google updates
    the version distribution data
  - Is mobile patching improving over time?
- Patching?!?
  - Exploit the vuln to gain privilege to patch it!
  - Some challenges: stability, warranties, etc



Next steps

This is the biggest problem in mobile security today.

- More public pressure on the responsible parties
  - Top-down from Google
  - Bottom-up from users and companies
- Open up platform security to third-parties?
  - Allow enterprises, third-parties to offload patching responsibility
- Better platform security in general, less vulns to patch

Q&A

Contact Information:

Jon Oberheide
jono@duosecurity.com
@jonoberheide