The Real Deal of Android Device Security: The Third Party

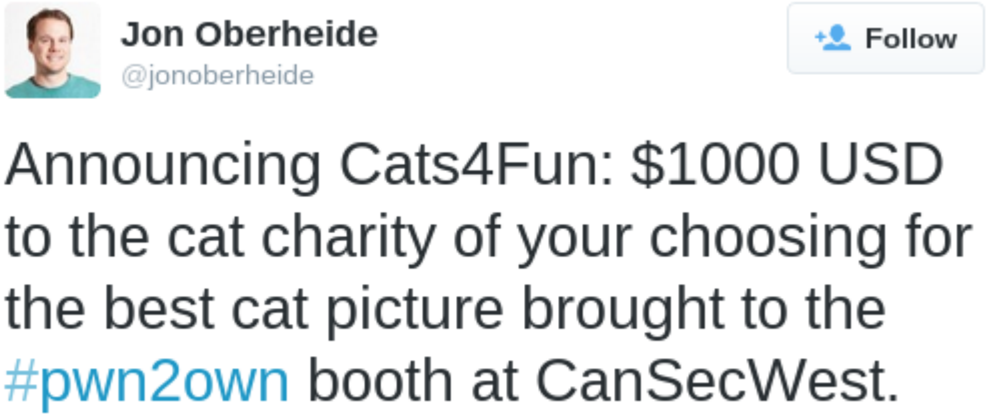
Collin Mulliner and Jon Oberheide

CanSecWest 2014

Introductions

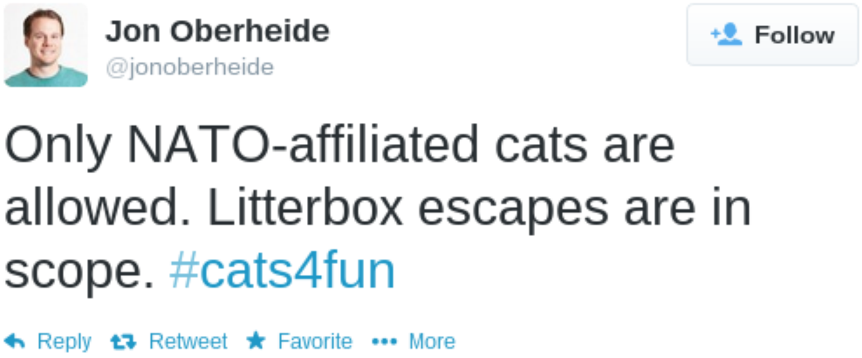
- Collin Mulliner
- Jon Oberheide

#Cats4Fun



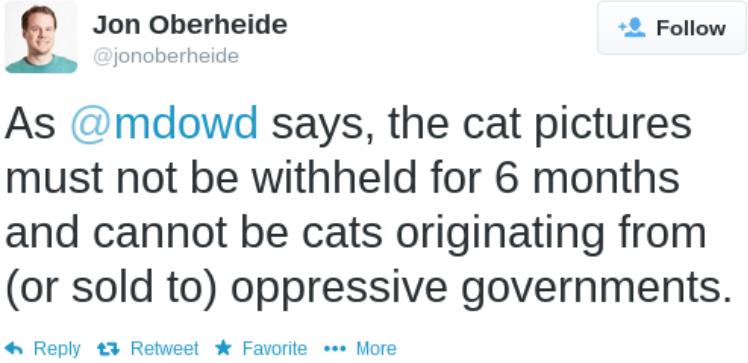
Jon Oberheide
@jonoberheide

Announcing Cats4Fun: $1000 USD to the cat charity of your choosing for the best cat picture brought to the #pwn2own booth at CanSecWest.



Jon Oberheide
@jonoberheide

Only NATO-affiliated cats are allowed. Litterbox escapes are in scope. #cats4fun



Jon Oberheide
@jonoberheide

As @mdowd says, the cat pictures must not be withheld for 6 months and cannot be cats originating from (or sold to) oppressive governments.

Thanks, Mudge!

Thanks, Mudge!

Android

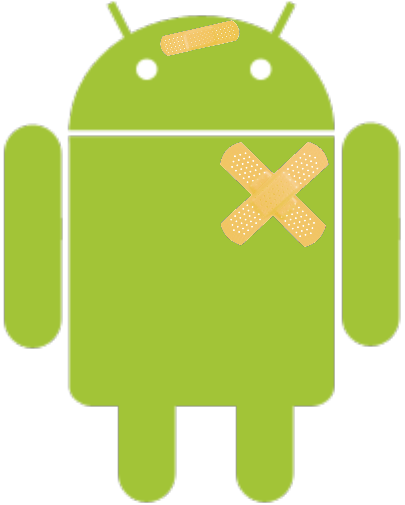
Mulliner and Oberheide, CSW 2014

Android

Most popular smartphone platform
about 1 billion devices today

This dude is in trouble

Lets patch him up!



WTF are we doing here people

- **Anti-malware**
  - 99.9%* of Android malware is bullshit toll fraud
- **MDM**
  - “Manage” your way out of an insecure platform
  - HEY I CAN SEE ALL MY VULNERABLE DEVICES, YAY!
- **Other features of mobile “security” products**
  - Find my phone (G does it), backup (G does it), …?

* I just made this up, kinda

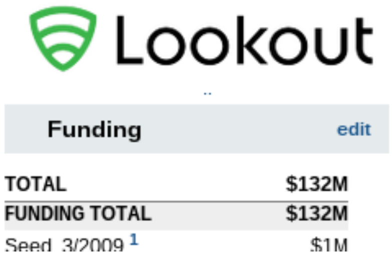
How about...

- **Maybe we try to fix the underlying issues?**
    - “Enumerating badness” always doomed to fail
    - Naw, that’s crazy talk!

- **Underlying issues (in our not-so-humble opinion)**
    - Lack of platform integrity
    - Privilege escalation vulns, large attack surface
    - Huge windows of vuln due to slow/non-existing patching practices

Our research

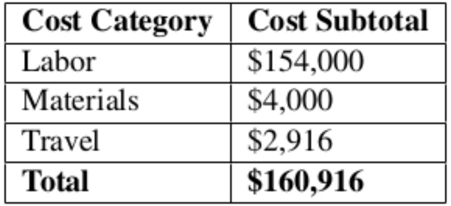
- **Investigated Android vulns and solutions**
  - Vulns in native and managed code
  - More than privesc!
- **Let's show what can be done**
  - Mostly PoC, but deployed to 100k's of real-world devices
  - If we can do this on the cheap, maybe Big Corp can do it for reals
- **"Defensive" talk, booooooooo**


Lookout

Funding edit

TOTAL	$132M
FUNDING TOTAL	$132M
Seed 3/2009 [1]	$1M

vs.



Cost Category	Cost Subtotal
Labor	$154,000
Materials	$4,000
Travel	$2,916
Total	**$160,916**

A tale of three projects

- **Vulns exist**
  - X-Ray

X-Ray
FOR ANDROID™

- **How to get rid of them**
  - PatchDroid
- **How to brick a lot of people’s phones ;-)**
  - ReKey

Ideal mobile ecosystem...HA!

- In a perfect world…
- **AOSP**: Google ships a secure base platform.
- **OEM**: Samsung and third-party suppliers don’t introduce vulns in their handsets and customizations.
- **Carrier**: T-Mobile rolls out rapid OTA updates to keep users up to date and patched.

Real-world mobile ecosystem

- In the real world…
- **AOSP**: Android improving mitigations, but slowly.
- **OEM**: Customizations by device OEMs are a primary source of vulnerabilities.
- **Carrier**: Updates are not made available for months and sometimes even years.

Mulliner and Oberheide, CSW 2014

Real-world mobile ecosystem

- In the real world…
- **AOSP**: Android improving mitigations, but slowly

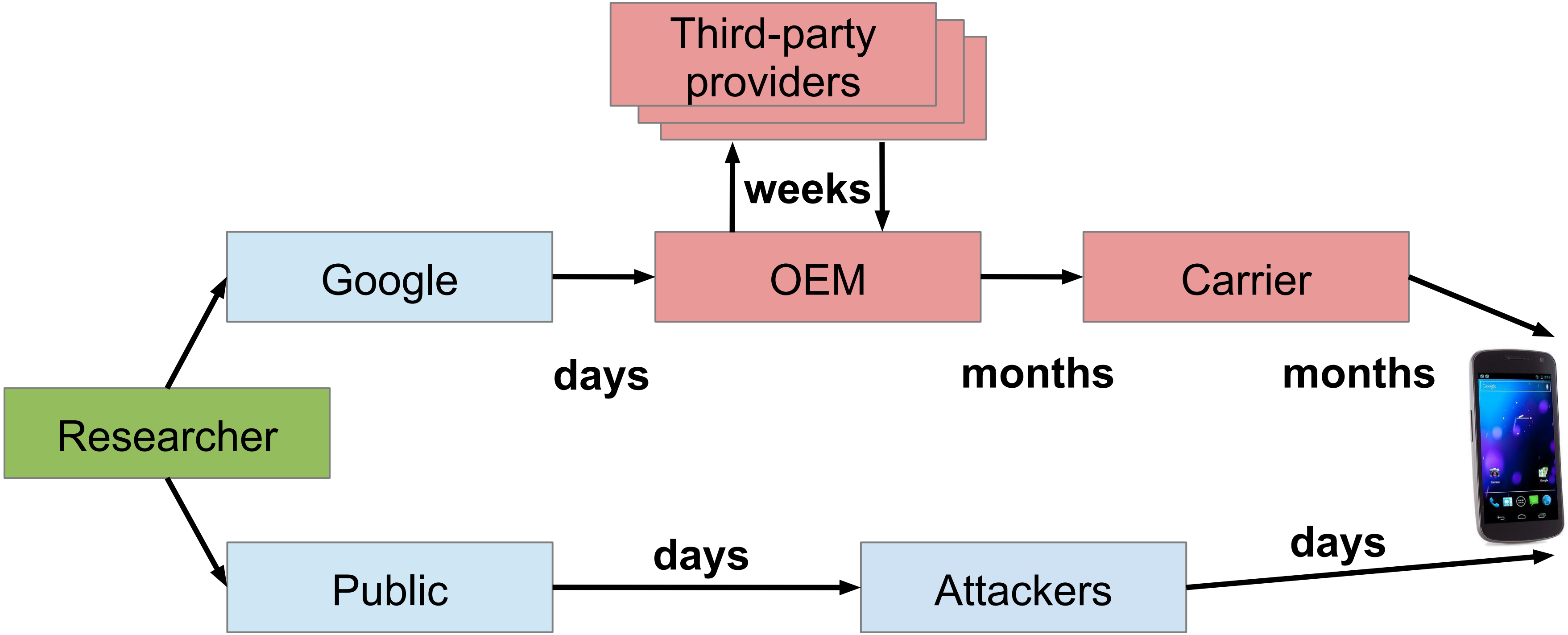
All software has vulns, mobile or otherwise.

- source of vulnerabilities.

Failing to deliver patches is the real issue.

- and sometimes even years.

Disclosure & patching process



Third-party providers

weeks

Google

OEM

Carrier

days

months

months

Researcher

Public

days

Attackers

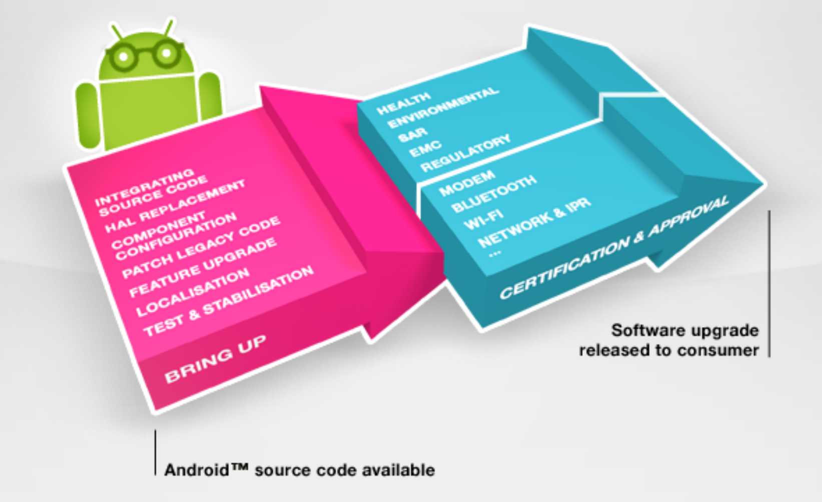
days

Mulliner and Oberheide, CSW 2014

Challenges in patching

- Why is mobile patching challenging?
  - Complicated software supply chain
  - Testing, testing, testing
  - Risk of bricking devices
  - Inverted economic incentives
- Want to patch your device's vulnerabilities?
  - Loadset controlled by carrier
  - Can't patch the device (unless rooted)

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

What the carriers say

"Patches *platforms*
to ensure *re,*
distributi &T

Mulliner and Oberheide, CSW 2014

Privilege escalation vulnerabilities

- **Android security model**
  - Permissions framework, “sandboxing” (Linux 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 jailbreak their devices
  - Public exploits reused by mobile malware to root victim's devices
- **Ooooh, fancy mobile privesc, right???**

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!

A sampling of privesc vulns

- **ASHMEM**: Android kernel mods, no mprotect check
- **Exploid**: no netlink source check, inherited from udev
- **Exynos**: third-party device driver, kmem read/write
- **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

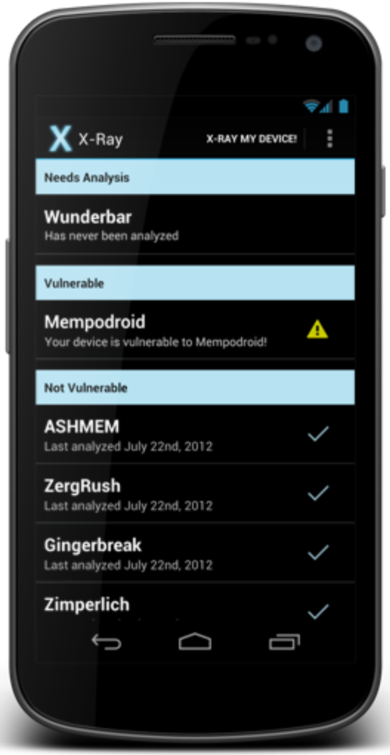
Mulliner and Oberheide, CSW 2014

X-Ray for Android

- How can we measure this problem?

- X-Ray for Android
  - DARPA CFT funded
  - Performing _actual_ vuln assessment on mobile
  - Detects most common privescs
  - Works without any special privileges or permissions

http://xray.io



Needs Analysis

Wunderbar
Has never been analyzed

Vulnerable

Mempodroid ⚠
Your device is vulnerable to Mempodroid!

Not Vulnerable

ASHMEM ✓
Last analyzed July 22nd, 2012

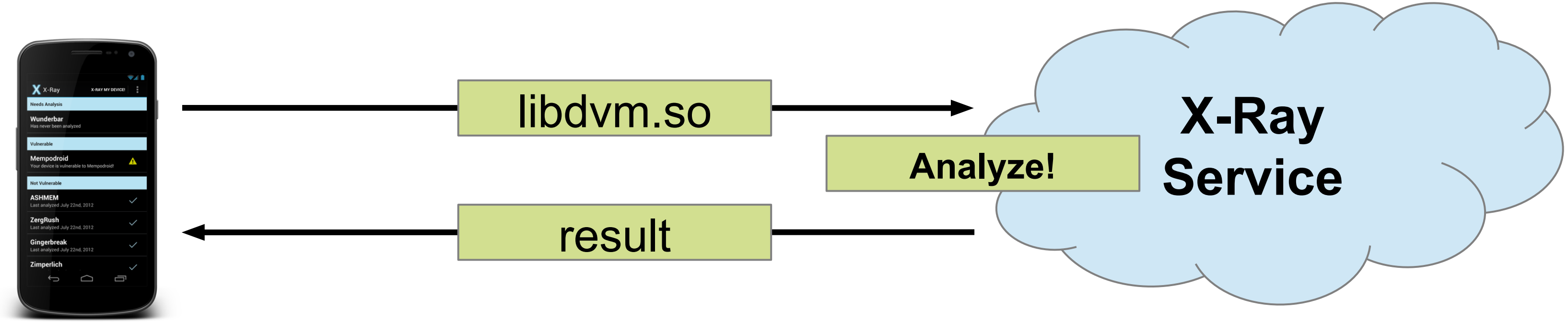
ZergRush ✓
Last analyzed July 22nd, 2012

Gingerbreak ✓
Last analyzed July 22nd, 2012

Zimperlich ✓

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



X-Ray

X-RAY MY DEVICE!

Needs Analysis

Wunderbar
Has never been analyzed

Vulnerable

Mempodroid
Your device is vulnerable to Mempodroid!

Not Vulnerable

ASHMEM
Last analyzed July 22nd, 2012

ZergRush
Last analyzed July 22nd, 2012

Gingerbreak
Last analyzed July 22nd, 2012

Zimperlich

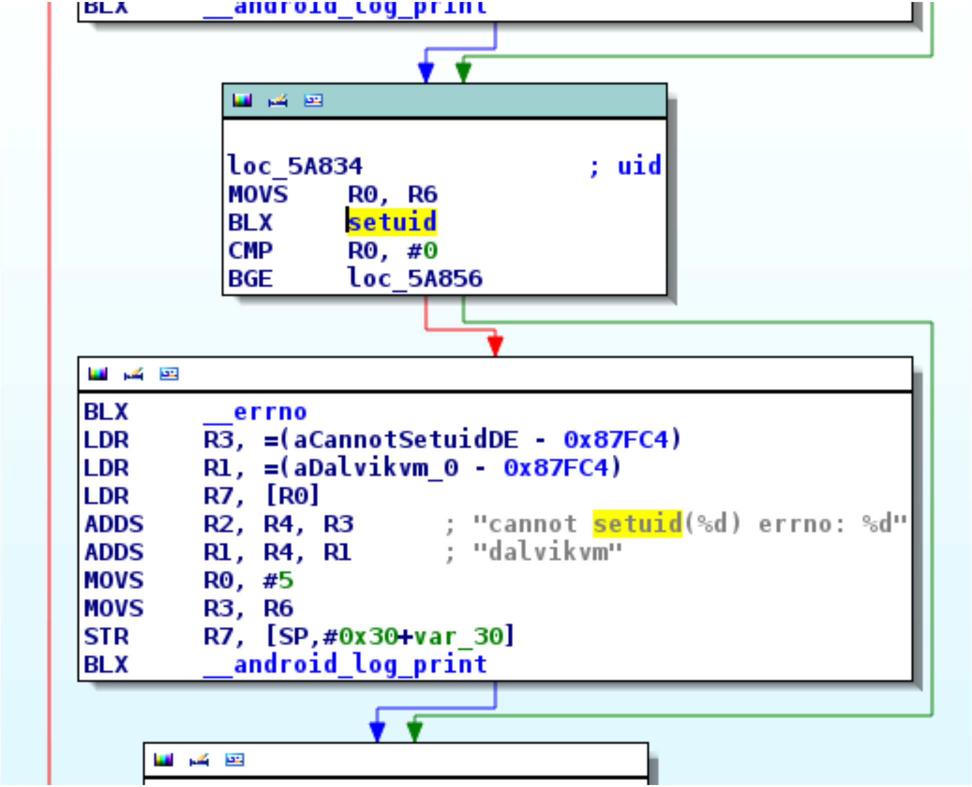
libdvm.so

Analyze!

result

X-Ray Service

Static probe example: Zimperlich



```
BLX     __android_log_print
```

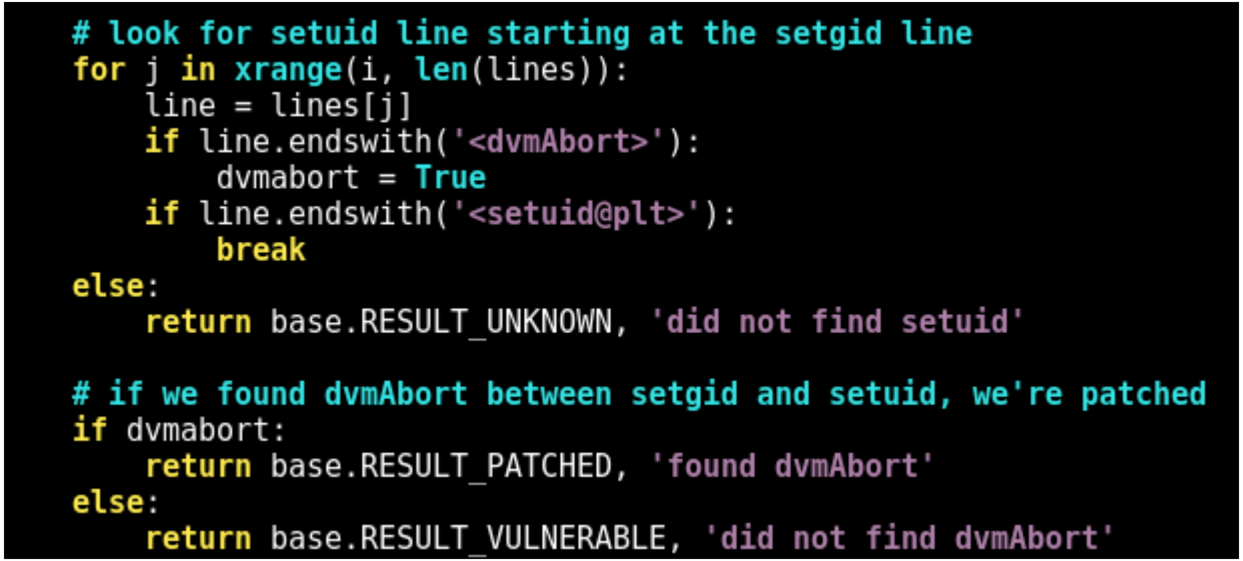
```
loc_5A834                   ; uid
MOVS    R0, R6
BLX     setuid
CMP     R0, #0
BGE     loc_5A856
```

```
BLX     __errno
LDR     R3, =(aCannotSetuidDE - 0x87FC4)
LDR     R1, =(aDalvikvm_0 - 0x87FC4)
LDR     R7, [R0]
ADDS    R2, R4, R3          ; "cannot setuid(%d) errno: %d"
ADDS    R1, R4, R1          ; "dalvikvm"
MOVS    R0, #5
MOVS    R3, R6
STR     R7, [SP,#0x30+var_30]
BLX     __android_log_print
```

Mulliner and Oberheide, CSW 2014

Ok, what does it _really_ look like?

- l33t static analysis...aka ghetto objdump/python/grep



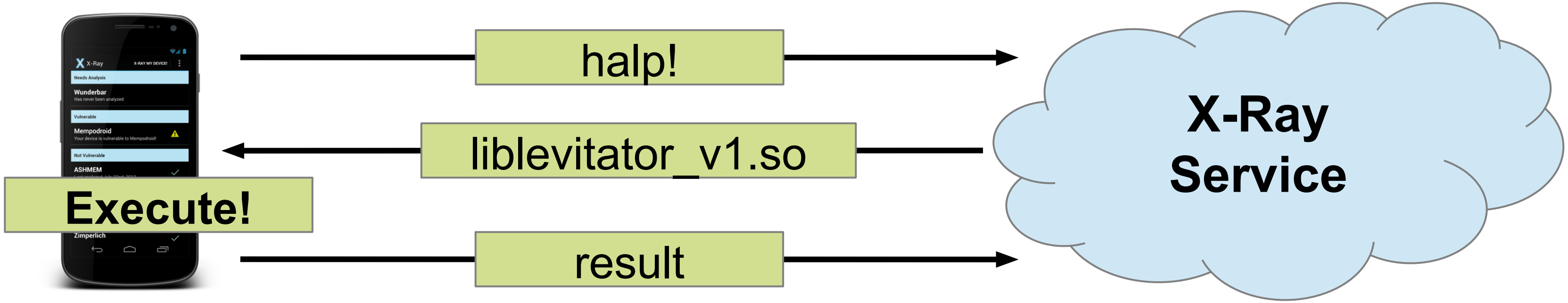
```
# look for setuid line starting at the setgid line
for j in xrange(i, len(lines)):
    line = lines[j]
    if line.endswith('<dvmAbort>'):
        dvmabort = True
    if line.endswith('<setuid@plt>'):
        break
else:
    return base.RESULT_UNKNOWN, 'did not find setuid'

# if we found dvmAbort between setgid and setuid, we're patched
if dvmabort:
    return base.RESULT_PATCHED, 'found dvmAbort'
else:
    return base.RESULT_VULNERABLE, 'did not find dvmAbort'
```

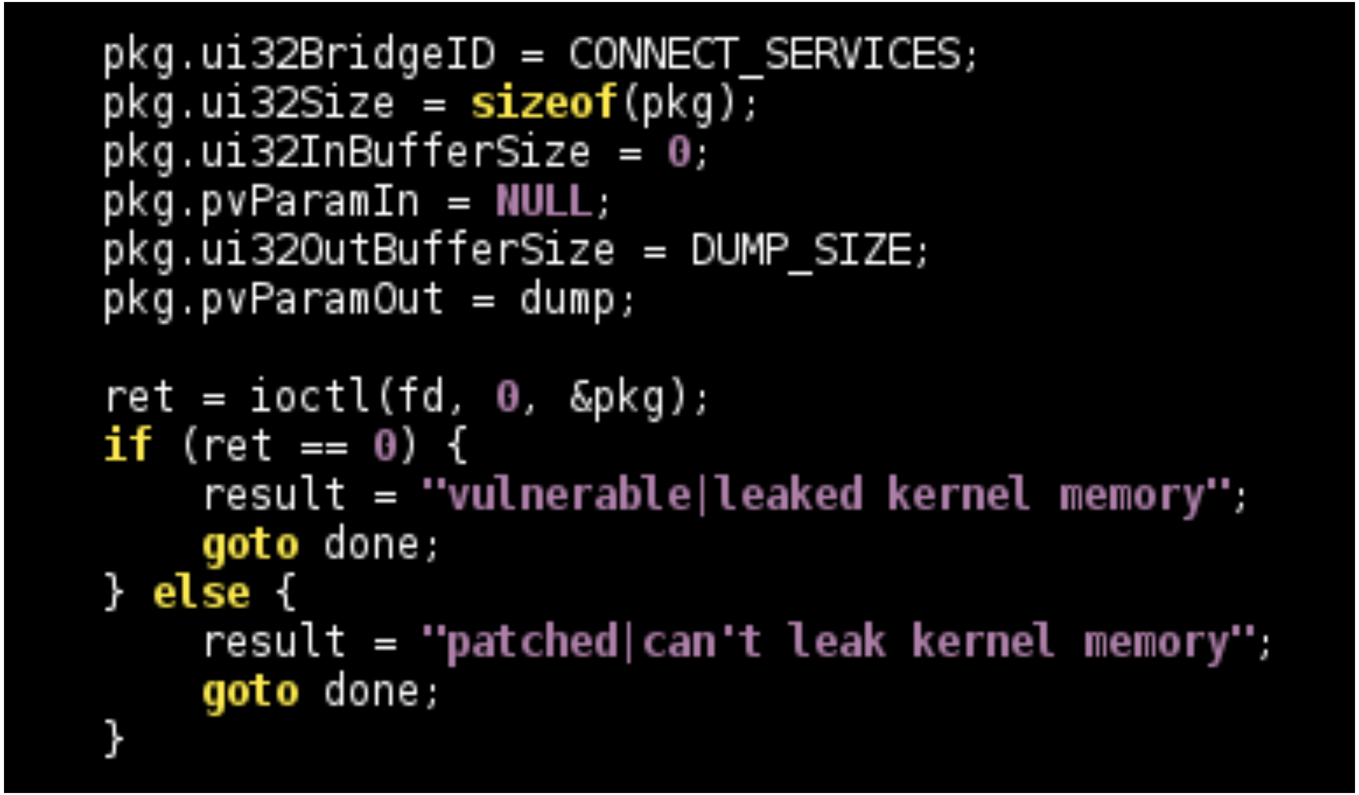
- Do we need to be that smart or perfect? Thankfully, no.

Dynamic probes (aka psuedo-exploits)

- 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!
  - Basically sad, neutered, wacky half exploits :-(



Dynamic probe example: Levitator

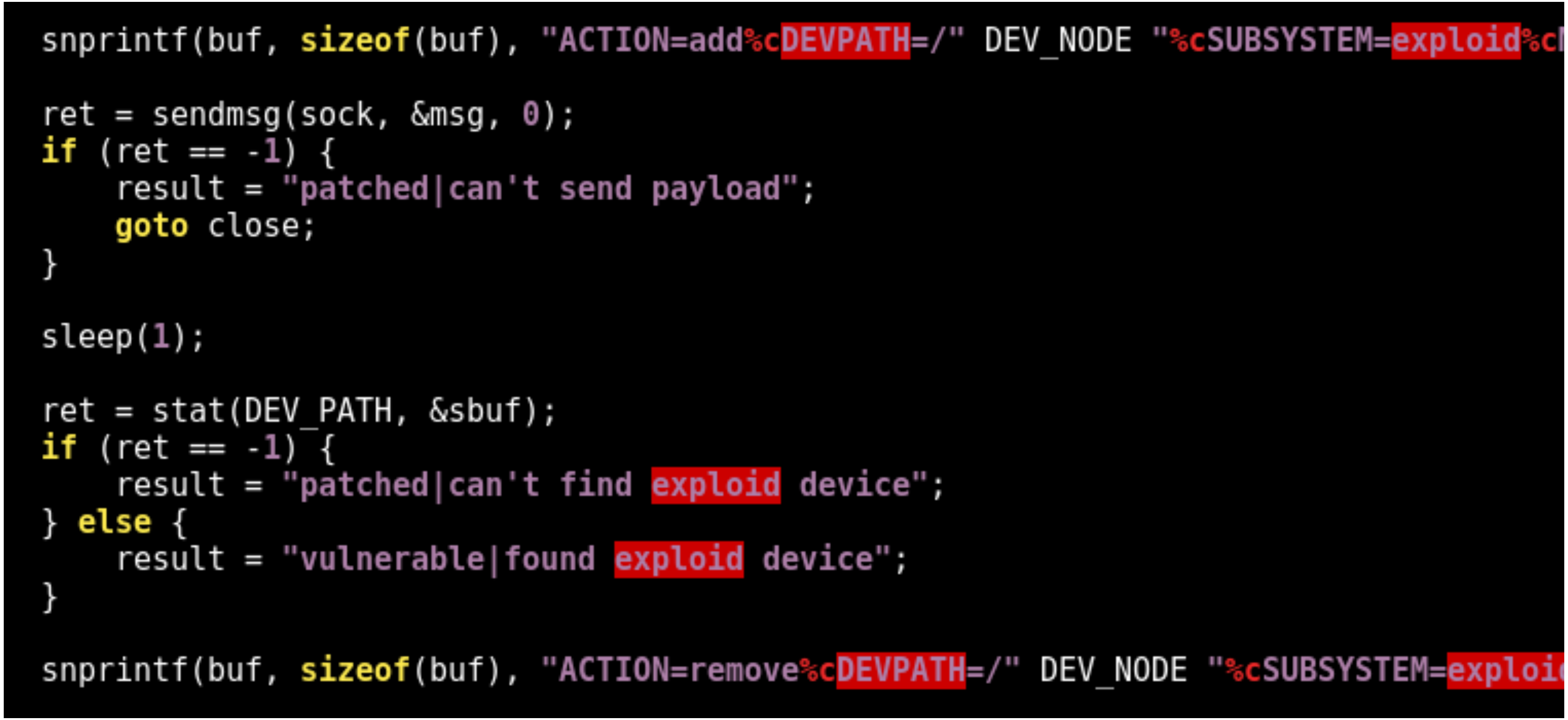


```
pkg.ui32BridgeID = CONNECT_SERVICES;
pkg.ui32Size = sizeof(pkg);
pkg.ui32InBufferSize = 0;
pkg.pvParamIn = NULL;
pkg.ui32OutBufferSize = DUMP_SIZE;
pkg.pvParamOut = dump;

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;
}
```

Mulliner and Oberheide, CSW 2014

Dynamic probe example: Exploid



```
snprintf(buf, sizeof(buf), "ACTION=add%cDEVPATH=/" DEV_NODE "%cSUBSYSTEM=exploid%c

ret = sendmsg(sock, &msg, 0);
if (ret == -1) {
    result = "patched|can't send payload";
    goto close;
}

sleep(1);

ret = stat(DEV_PATH, &sbuf);
if (ret == -1) {
    result = "patched|can't find exploid device";
} else {
    result = "vulnerable|found exploid device";
}

snprintf(buf, sizeof(buf), "ACTION=remove%cDEVPATH=/" DEV_NODE "%cSUBSYSTEM=exploi
```

Mulliner and Oberheide, CSW 2014

Probe manifests in JSON

Static probe:

```
{
    "id":             "webkit",
    "type":           "static",
    "name":           "WebKit (inactive)",
    "query_url":      "/xray/webkit/query",
    "probe_url":      "/xray/webkit/probe",
    "static_payload": "/system/lib/libwebcore.so"
}
```

Dynamic probe:

```
{
    "id":              "exynos",
    "type":            "dynamic",
    "name":            "Exynos",
    "result_url":      "/xray/exynos/result",
    "dynamic_slot":    "06",
    "dynamic_payload_armeabi":         "/xray/static/exynos/armeabi/libexynos_v1.so",
    "dynamic_signature_armeabi":       "vrX...",
    "dynamic_payload_armeabi-v7a":     "/xray/static/exynos/armeabi-v7a/libexynos_v1.so",
    "dynamic_signature_armeabi-v7a":   "mbe...",
    "dynamic_payload_mips":            "/xray/static/exynos/mips/libexynos_v1.so",
    "dynamic_signature_mips":          "F33...",
    "dynamic_payload_x86":             "/xray/static/exynos/x86/libexynos_v1.so",
    "dynamic_signature_x86":           "Lu7..."
},
```

X-Ray distribution

- Not in Google Play*, but free for download at http://xray.io

- Results collected by us (and Five Eyes) from users who ran the X-Ray app on their Android device:

74,405 devices
4,312 models
190 countries

* don't ask

Aside: Android exploitation challenges

- Android fragmentation is _real_
  - Not for app dev, but for exploit dev

- X-Ray's binary dataset
  - **3,124** unique libsysutils.so
  - **5,936** unique libdvm.so
  - **5,303** unique vold

- If only there was a way to collect all those binaries...

Scary numbers

- 6 months after the X-Ray release…
- Percent of the global Android population that are vulnerable to a privilege escalation detected by X-Ray...

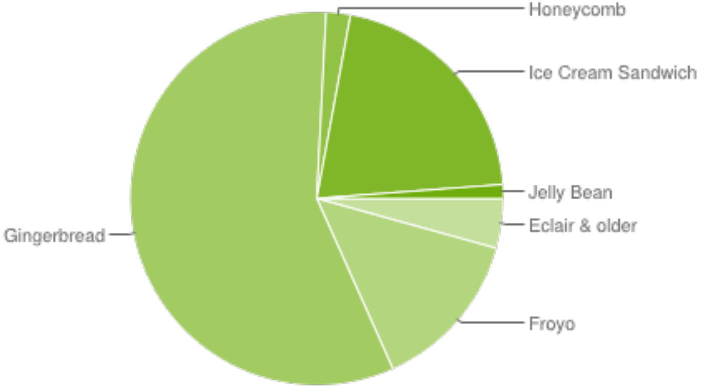
60.6% vulnerable

Mulliner and Oberheide, CSW 2014

Methodology

- How to extrapolate out to global Android population?
  - Selection bias?

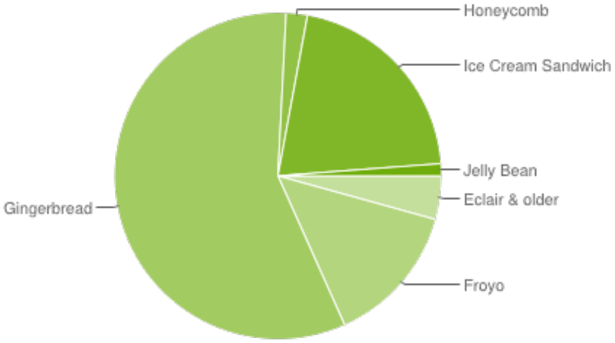
- Google provides stats on Android versions →



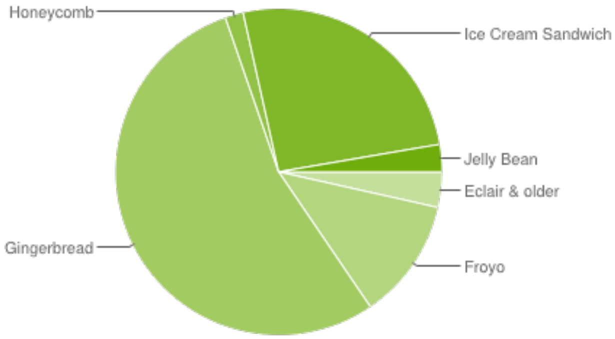
- If we saw 98.8% of 2.2 devices were vulnerable, and 2.2 makes up 15.5% of Android globally, that contributes 15.3% to the total % of vulnerable Android devices.

Death of an Android vuln

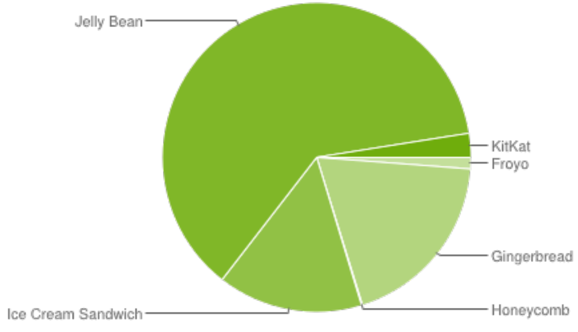
Changes over time



Late 2012



Early 2013



Early 2014

60.6% vulnerable

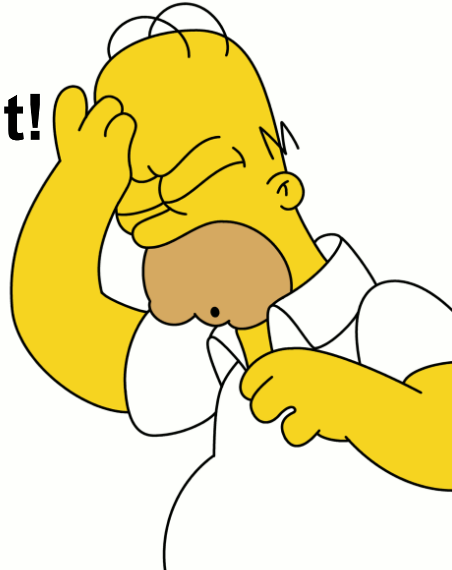
41.2% vulnerable

13.4% vulnerable

Looks like OK progress, but...
Only measuring those original 8 ancient privesc vulns from X-Ray 1.0, not any new ones!

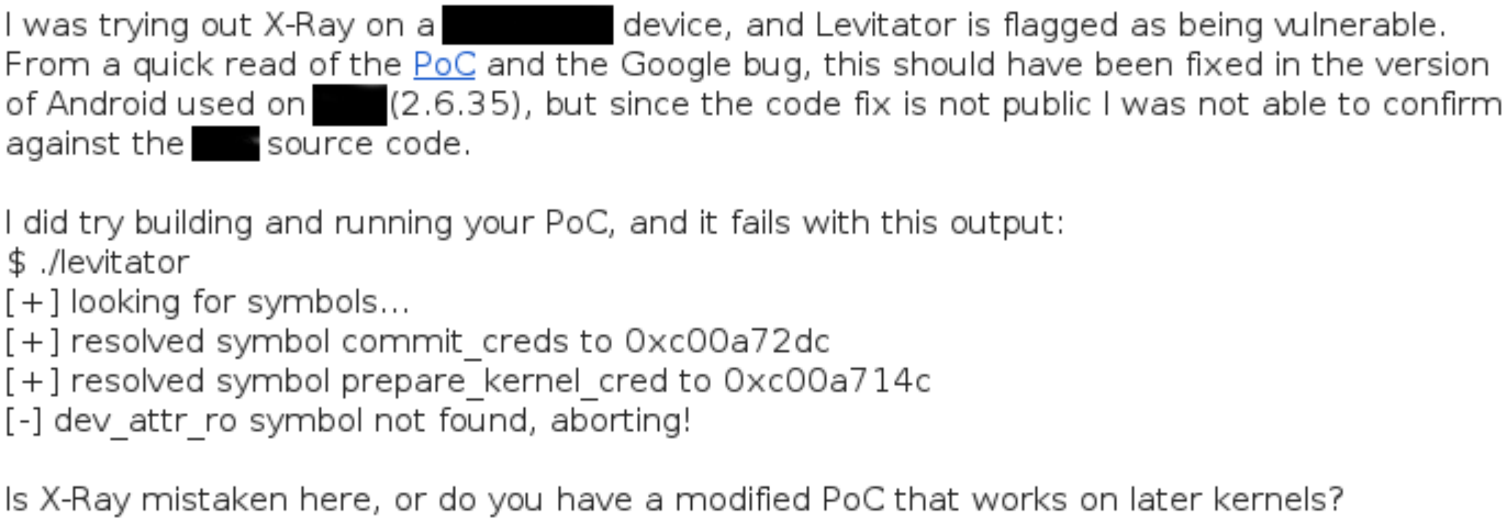
OEM vendor fuckups

- **Versions that shouldn't be patched, but are!**
  - Version 2.3.2, but not vuln to gingerbreak
  - Backports without version bumps
- **Versions that should be patched, but aren't!**
  - Version 4.1, but still vuln to mempodroid
  - Incomplete patching, regressions
- **OEM vendors relying on public exploits to do vuln assessment**



Failed exploit != patched

- **OEM vendor inquiry:**



I was trying out X-Ray on a [redacted] device, and Levitator is flagged as being vulnerable. From a quick read of the PoC and the Google bug, this should have been fixed in the version of Android used on [redacted] (2.6.35), but since the code fix is not public I was not able to confirm against the [redacted] source code.

I did try building and running your PoC, and it fails with this output:

```
$ ./levitator
[+] looking for symbols...
[+] resolved symbol commit_creds to 0xc00a72dc
[+] resolved symbol prepare_kernel_cred to 0xc00a714c
[-] dev_attr_ro symbol not found, aborting!
```

Is X-Ray mistaken here, or do you have a modified PoC that works on later kernels?

- **SORRY. I WRITE CRAPPY EXPLOITS.**

Database of vulnerable models

“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 |
+------------------------+
```

```
mysql> SELECT DISTINCT(model)
FROM results
WHERE probe='levitator'
AND result='vulnerable'
AND model LIKE '%Kindle%';
+-------------+
| model       |
+-------------+
| Kindle Fire |
+-------------+
```

It’s like PRISM...for Android!

TOP SECRET//COMINT//REL USA, FVEY

XRAY Project Results

- (S//SI//REL) Covert platform for mobile TAO implants
  - Highly successful (~75,000 active implants worldwide)
- (S//SI) Metadata selector types
  - Device ID, manufacturer, model, version, carrier, country, IP address, vulnerability state
- (S//SI) Integrates with POOPCHUTE and BLAMEVUPEN
  - Palm Pilot support in development

TOP SECRET//COMINT//REL TO USA, FVEY//20230108

Lessons learned from X-Ray

- Man, OEMs and carriers sure suck at patching.
- If only there was some way to patch these vulns ourselves!
- BRING OUT THE GERMAN!

Use Bug to Gain Root to Patch Bug

Use Bug to Gain Root to Patch Bug

Introducing

PatchDroid

Use Bug to Gain Root to Patch Bug

Introducing

PatchDroid

...but we actually have users root their devices

Challenges

- **No access to source code**
  - AOSP != code running on devices
  - modifications by OEMs
- **Can’t modify system files and/or partitions**
  - patched binaries might brick device
  - cannot replace signed partitions or files on them
- **Scalability and testing**
  - too many different devices and OS versions
  - patches need to be decoupled form source code

PatchDroid

- **Third-party security patches for Android**
  - includes: attack detection and warning mechanism

- **Independent of device and Android version**
  - support for Dalvik bytecode and native code

PatchDroid cont.

- **Scalable**
  - only develop patch once, patch any device
  - test patches in the field

- **Practical**
  - almost no overhead (user won’t notice any)
  - we don’t need source code
    - not everything of Android is open source

PatchDroid - The System

- **In-memory patching at runtime**
  - need to patch processes at startup
    - before process executes vulnerable code
    - monitor system for new processes
  - no need to modify system files or system partitions
    - important!

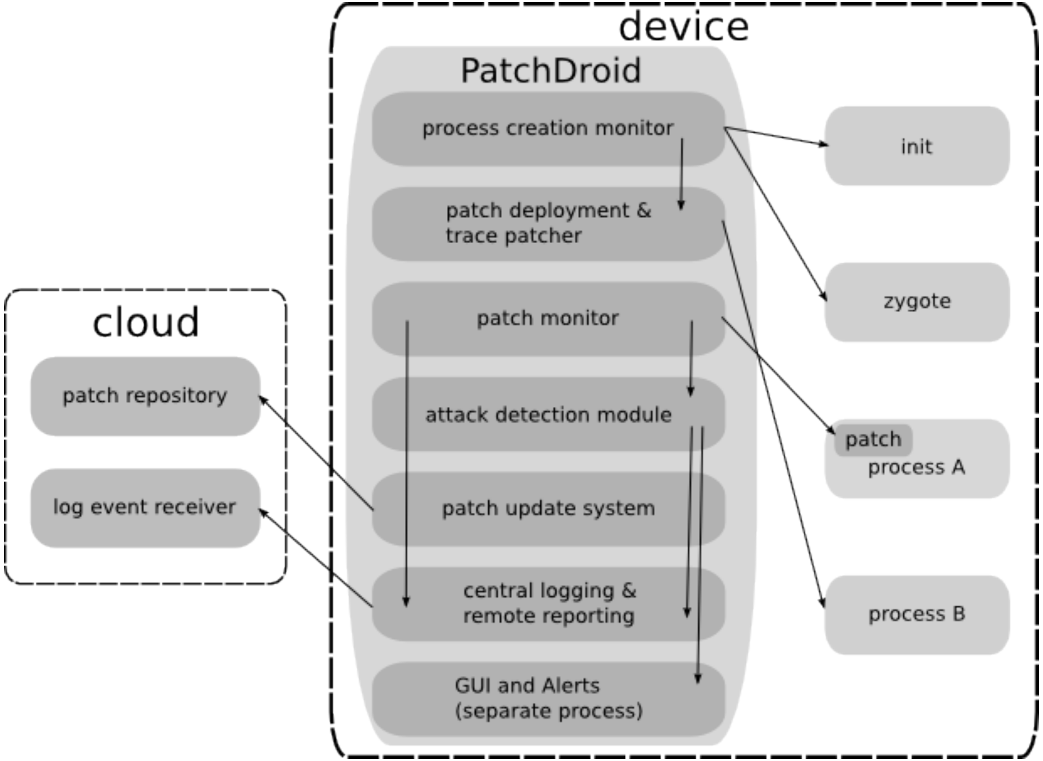
PatchDroid - The System cont.

- **Patches as independent code**
  - self-contained shared library
  - patching via function hooking
  - no access to original source code required
  - scale across different OS versions

Overview

- PatchDroid system architecture
- Patches in our system
  - creating a patch
- Technical insights
- ReKey!
  - a public release of PatchDroid
- Demo

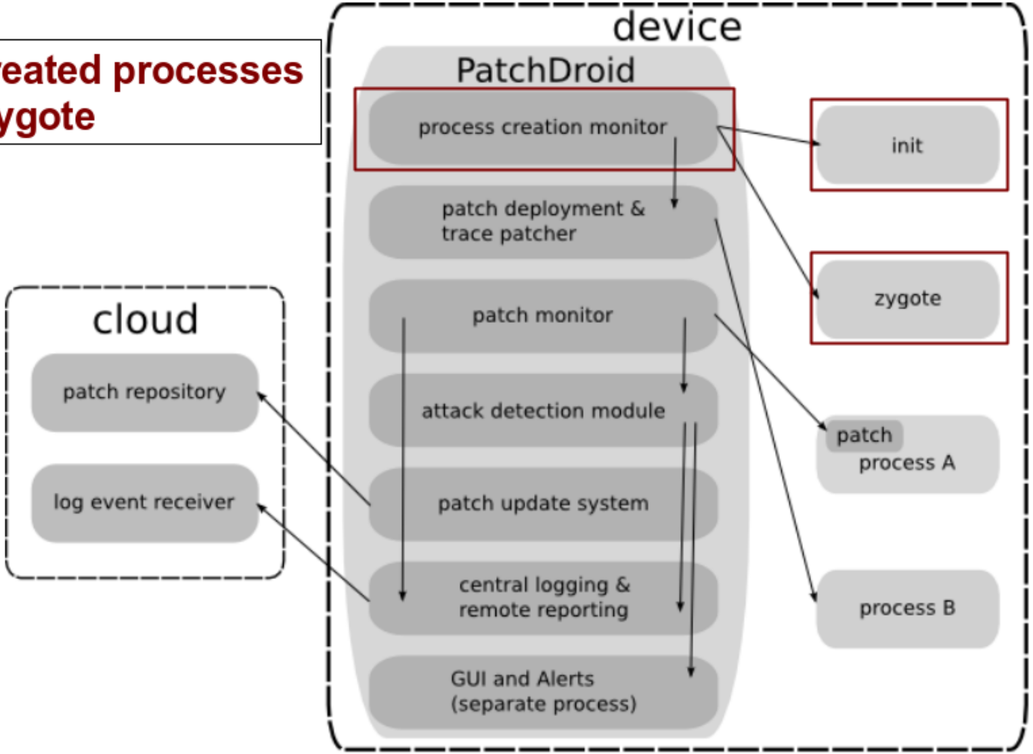
Architecture



Mulliner and Oberheide, CSW 2014

Architecture

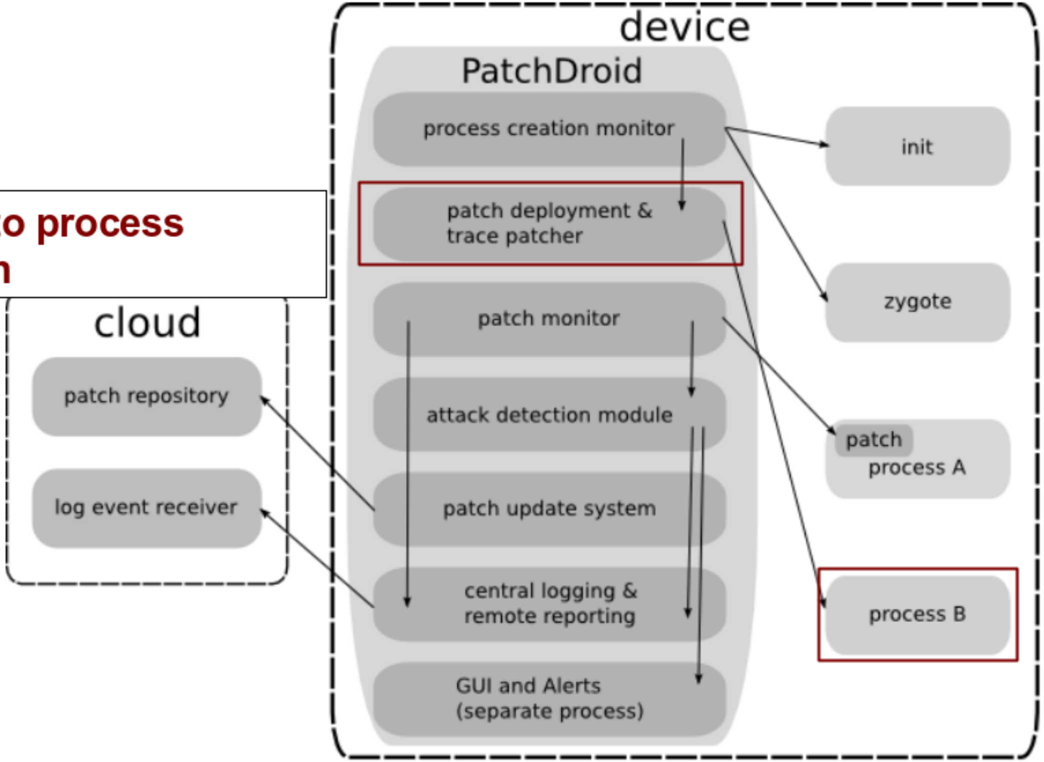
Identify newly created processes - trace init and zygote



Mulliner and Oberheide, CSW 2014

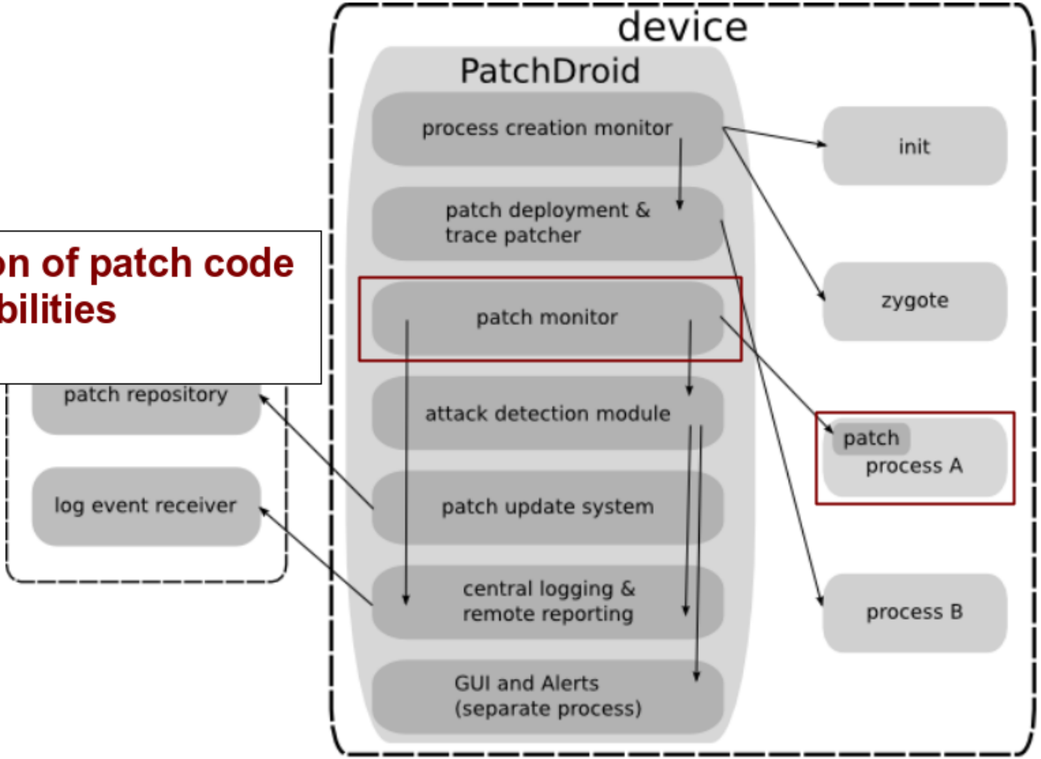
Architecture

Deploy patch into process
- library injection



Mulliner and Oberheide, CSW 2014

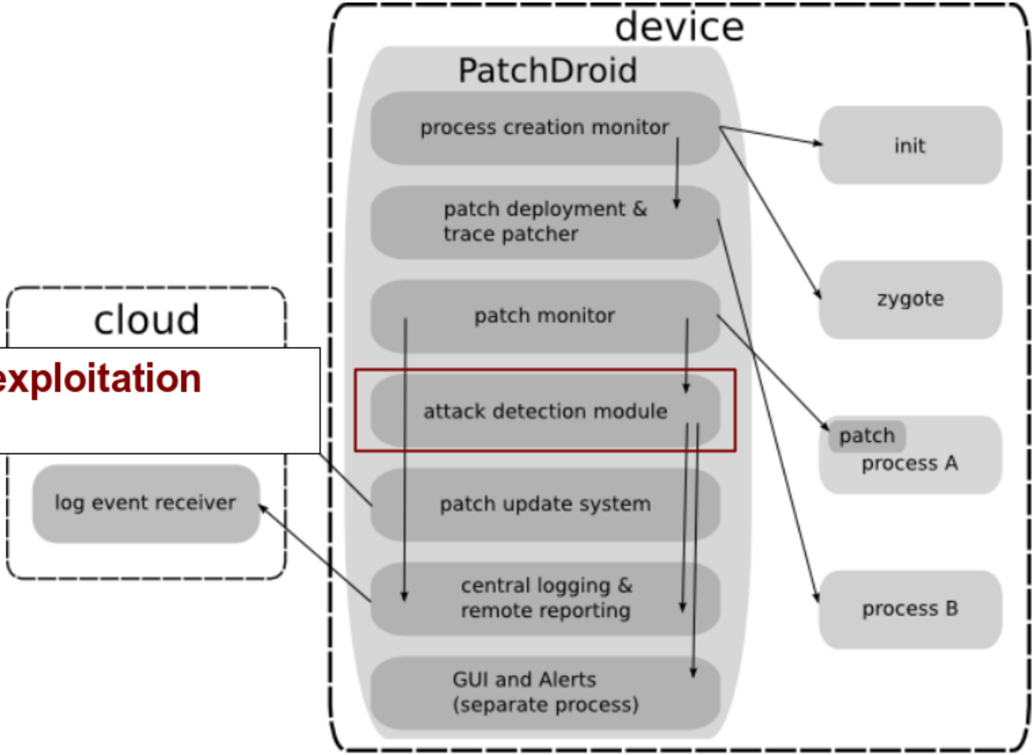
Architecture



Monitor execution of patch code
- check for instabilities
- collect logs

Mulliner and Oberheide, CSW 2014

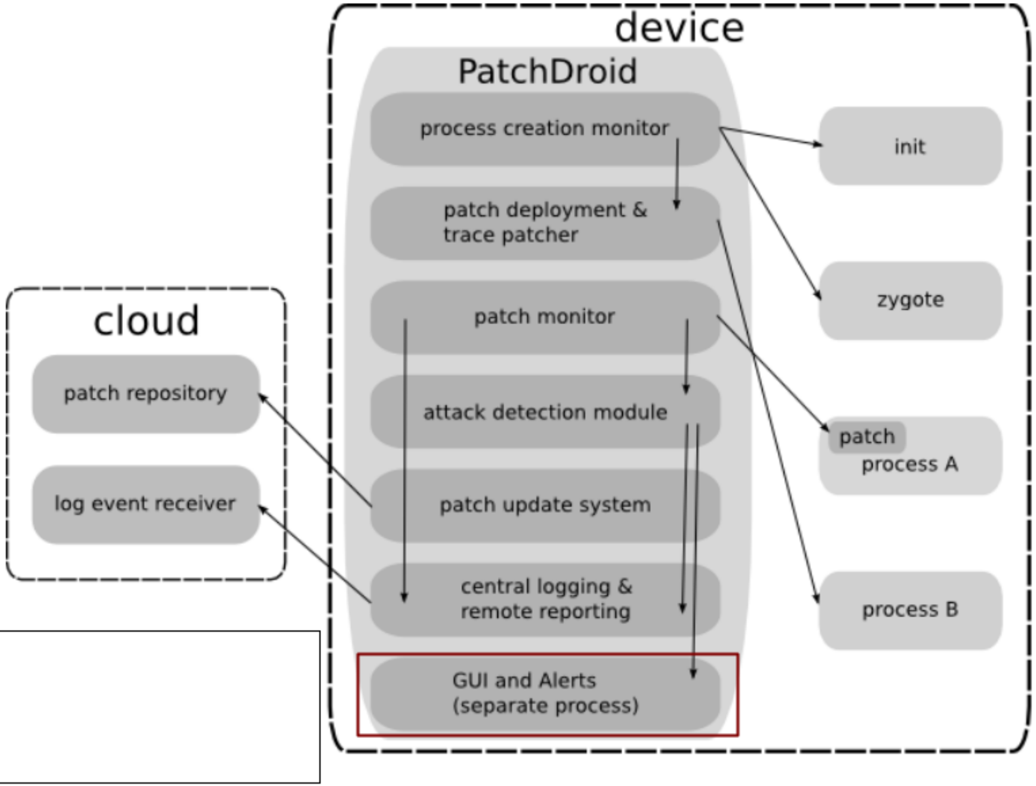
Architecture



Analyze log for exploitation attempt

Mulliner and Oberheide, CSW 2014

Architecture

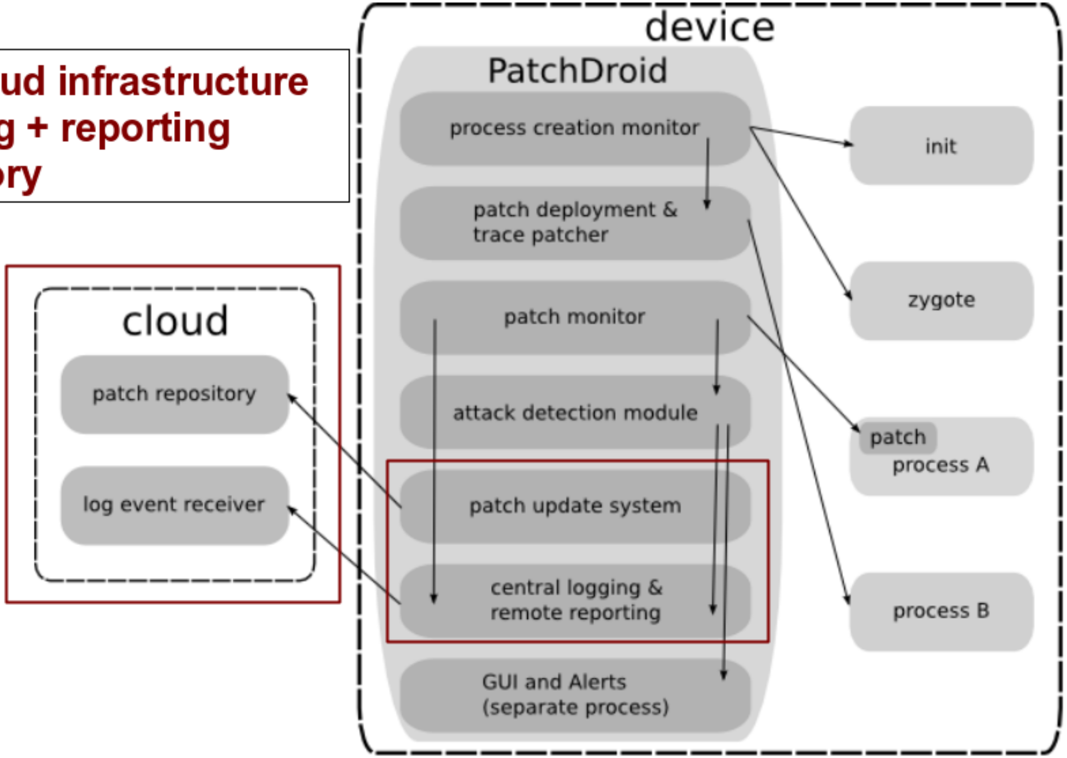


PatchDroid App
-GUI
-display alerts

Mulliner and Oberheide, CSW 2011

Architecture

PatchDroid cloud infrastructure
-central logging + reporting
-patch repository



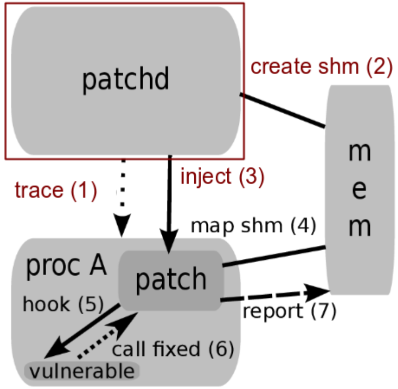
Mulliner and Oberheide, CSW 2014

Anatomy of a Patch

- **Replacement for vulnerable <u>function</u>**
    - equivalent code without vulnerability
    - wrapper that adds input/output sanitization
- **Install**
    - hook vulnerable function
        - keep original function usable, we will need it later
- **Communication link**
    - read config parameters
    - write log messages, report attacks

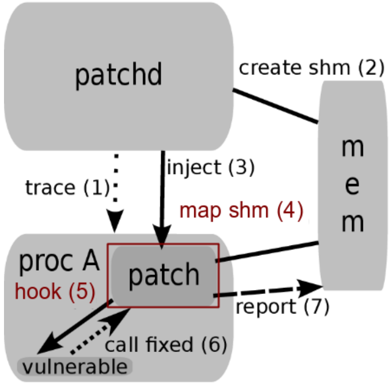
Lifetime of a Patch

- Deployment
  - trace target process
  - setup communication
  - inject patch library



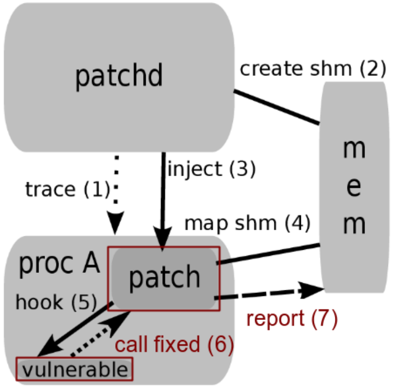
Lifetime of a Patch

- Installation
  - connect communication
  - hook function(s)



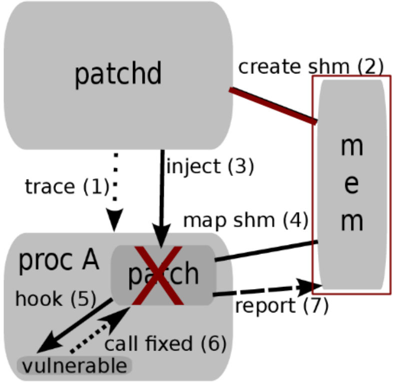
Lifetime of a Patch

- Fixed function is called
  - log (and report attack)
  - collect telemetry
  - (call original function)



Lifetime of a Patch

- Patch failure
  - detected using telemetry
  - failing patch is removed



- This is tricky
  - works only to certain extend
  - but enables some kind of field testing

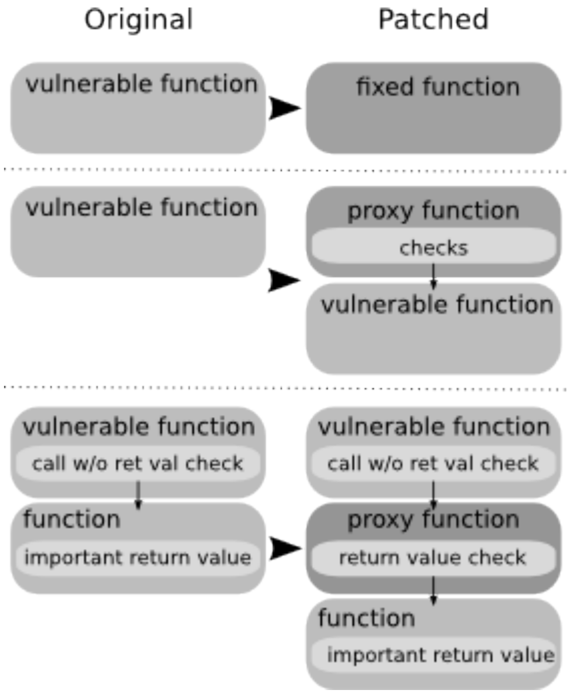
Creating a Patch

- Extract patch from source, **transform** to PatchDroid patch
  - apply patch strategy best suited for vulnerability
  - sources: e.g., AOSP, Cyanogen, etc...

- Develop custom patch
  - vulnerability known, but no patch available

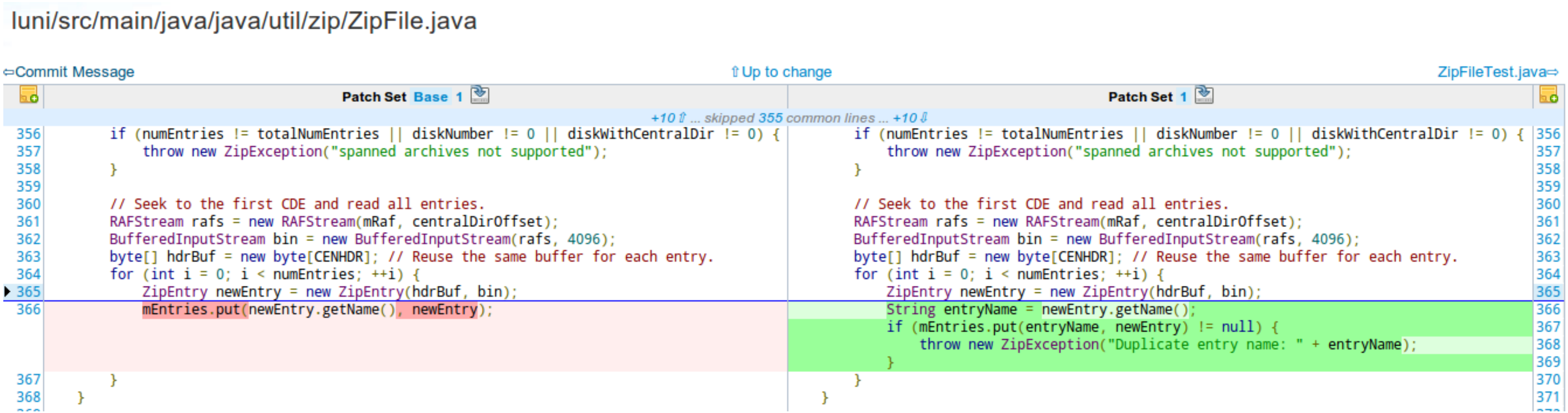
Patching Strategies

- replace
- proxy
- add return value check



Mulliner and Oberheide, CSW 2014

Source Patch -> PatchDroid Patch



luni/src/main/java/java/util/zip/ZipFile.java

```java
        if (numEntries != totalNumEntries || diskNumber != 0 || diskWithCentralDir != 0) {
            throw new ZipException("spanned archives not supported");
        }

        // Seek to the first CDE and read all entries.
        RAFStream rafs = new RAFStream(mRaf, centralDirOffset);
        BufferedInputStream bin = new BufferedInputStream(rafs, 4096);
        byte[] hdrBuf = new byte[CENHDR]; // Reuse the same buffer for each entry.
        for (int i = 0; i < numEntries; ++i) {
            ZipEntry newEntry = new ZipEntry(hdrBuf, bin);
            mEntries.put(newEntry.getName(), newEntry);
        }
    }
```

```java
        if (numEntries != totalNumEntries || diskNumber != 0 || diskWithCentralDir != 0) {
            throw new ZipException("spanned archives not supported");
        }

        // Seek to the first CDE and read all entries.
        RAFStream rafs = new RAFStream(mRaf, centralDirOffset);
        BufferedInputStream bin = new BufferedInputStream(rafs, 4096);
        byte[] hdrBuf = new byte[CENHDR]; // Reuse the same buffer for each entry.
        for (int i = 0; i < numEntries; ++i) {
            ZipEntry newEntry = new ZipEntry(hdrBuf, bin);
            String entryName = newEntry.getName();
            if (mEntries.put(entryName, newEntry) != null) {
                throw new ZipException("Duplicate entry name: " + entryName);
            }
        }
    }
```

- Missing return value check
  - `mEntries.put()` returns != null,key is already used
  - dup key == multiple zip entries with same name

Transform

- Hook: `java.lang.LinkedHashMap.put()`
  - call orig method and check return value
  - throw exception if result != null
- `LinkedHashMap` is used outside of `ZipFile`
  - need to only patch behavior in `ZipFile` code
- Hook: `java.util.ZipFile.readCentralDir()`
  - install hook for `LinkedHashMap`
  - call original `readCentralDir()`
  - unhook `LinkedHashMap`

PatchDroid - Implementation

- **patchd: the patch daemon**
    - monitor system for newly created process
    - inject patches into process
    - monitor patched process

- **PatchDroid App**
    - UI
    - Helper Service
    - Attack Notification

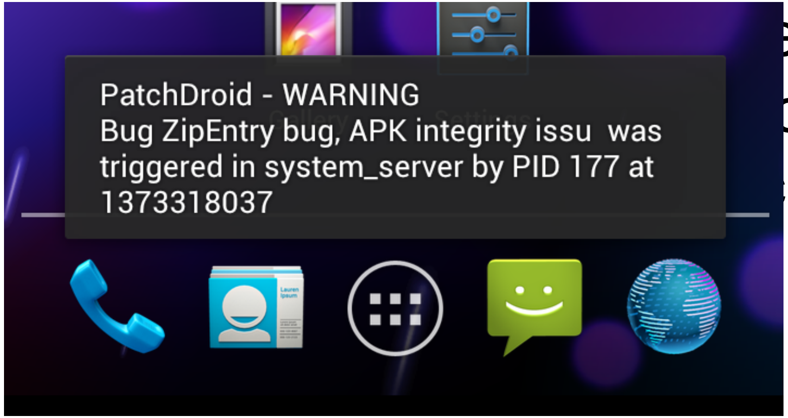
PatchDroid - Implementation

- patchd: the patch daemon

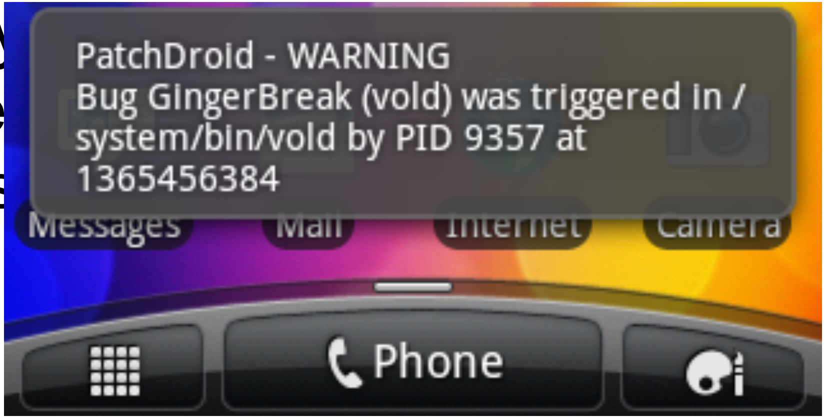
ewly

oce

ess



PatchDroid - WARNING
Bug ZipEntry bug, APK integrity issu was triggered in system_server by PID 177 at 1373318037



PatchDroid - WARNING
Bug GingerBreak (vold) was triggered in /system/bin/vold by PID 9357 at 1365456384

  - UI
  - Helper Service
  - Attack Notification

Hooking Techniques

- **Native patches based on ADBI**
  - framework for hooking native code on Android
  - http://github.com/crmulliner/adbi/

- **Dalvik patches based on DDI**
  - framework for hooking Dalvik methods
  - http://github.com/crmulliner/ddi/

Insights

- **patchd uses `ptrace()` for monitoring and injection**
  - most target processes run as root
  - patchd -> requires root

- **PatchDroid app lives in /data/data/…**
  - no need to modify ‘/system’ file system
    - often signed and checked by bootloader
  - can be installed/removed like any other app
    - we don’t want to brick devices

Patches

	Target Process
Native	
Zimperlich	zygote
GingerBreak	vold
ZergRush	vold
Dalvik	
Local SMS Spoofing	system_server
MasterKey	system_server

Patches

Native	**Target Process**
Zimperlich	zygote
GingerBreak	vold
ZergRush	vold
Dalvik	
Local SMS Spoofing	system_server
MasterKey	system_server

MasterKey Bug

- **Discovered by the guys from BlueBox**
- **Bug in handling of APK files**
  - APK can be modified without breaking its signature
- **Can be used for privilege escalation (root device)**
  - modify APK signed with platform/oem key
  - that APK roots any device from given OEM!

MasterKey Bug cont.

- **Actually multiple bugs**
- **Bugs in Java code (Dalvik bytecode)**
  - first priv esc vuln due to bug in Dalvik bytecode
- **Bug present in AOSP until version 4.3**
  - Affected almost all Android devices at that time

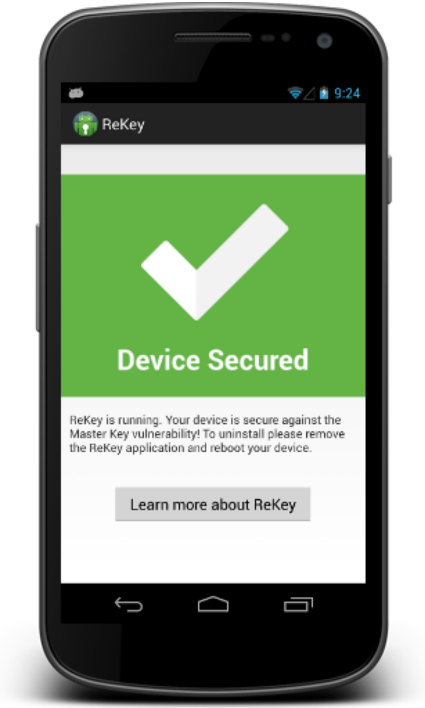
Patching MasterKey Bug(s)

- **Patching Strategies**
  - Add missing return value check
  - Add input/output sanitisation (thru proxy function)

- **Fast turnaround**
  - 3 hours for initial version, coding + testing

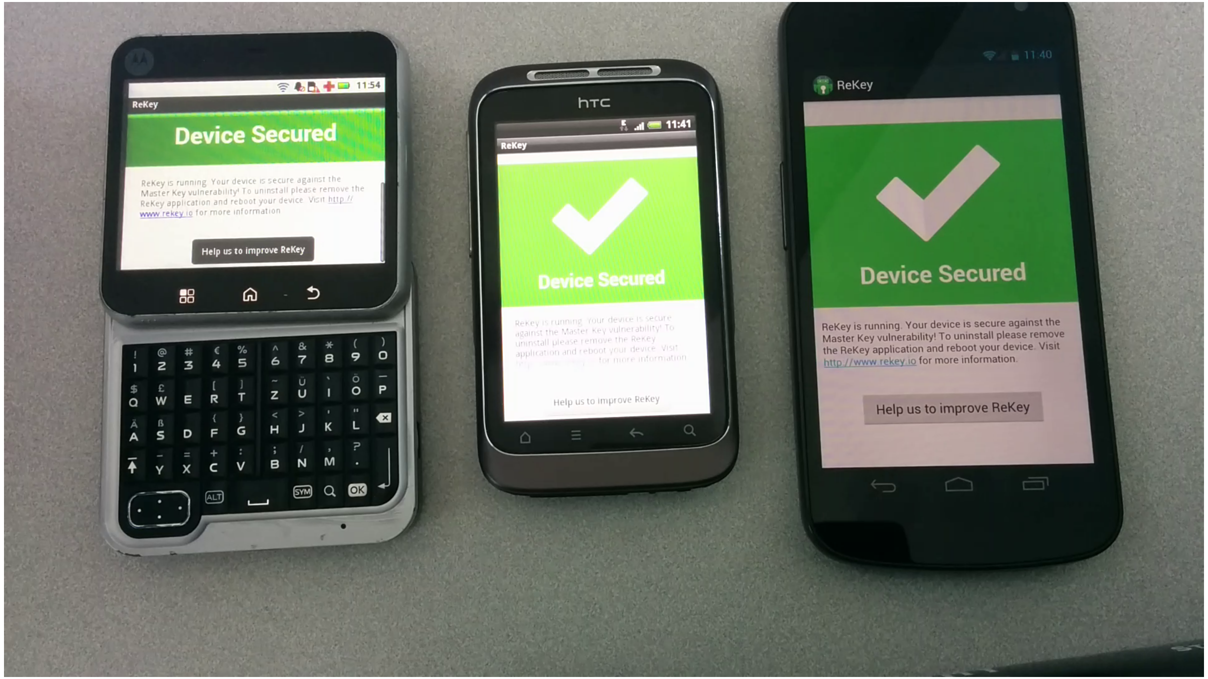
ReKey

- Special version of PatchDroid
  - **Patches for MasterKey only!**
- Released on July 16th 2013
  - Available Google Play!
- ReKey your device
  - **http://rekey.io**



ReKey is running. Your device is secure against the Master Key vulnerability! To uninstall please remove the ReKey application and reboot your device.

PatchDroid / ReKey - Demo

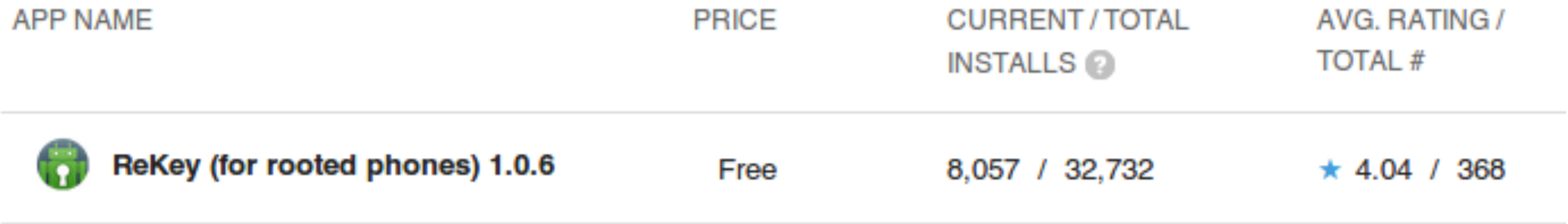

11:54
ReKey
Device Secured
ReKey is running. Your device is secure against the Master Key vulnerability! To uninstall please remove the ReKey application and reboot your device. Visit http://www.rekey.io for more information.
Help us to improve ReKey
htc
11:41
ReKey
Device Secured
ReKey is running. Your device is secure against the Master Key vulnerability! To uninstall please remove the ReKey application and reboot your device. Visit http://www.rekey.io for more information.
Help us to improve ReKey
11:40
ReKey
Device Secured
ReKey is running. Your device is secure against the Master Key vulnerability! To uninstall please remove the ReKey application and reboot your device. Visit http://www.rekey.io for more information.
Help us to improve ReKey

Data & Stats

- Google Play

- ReKey opt-in

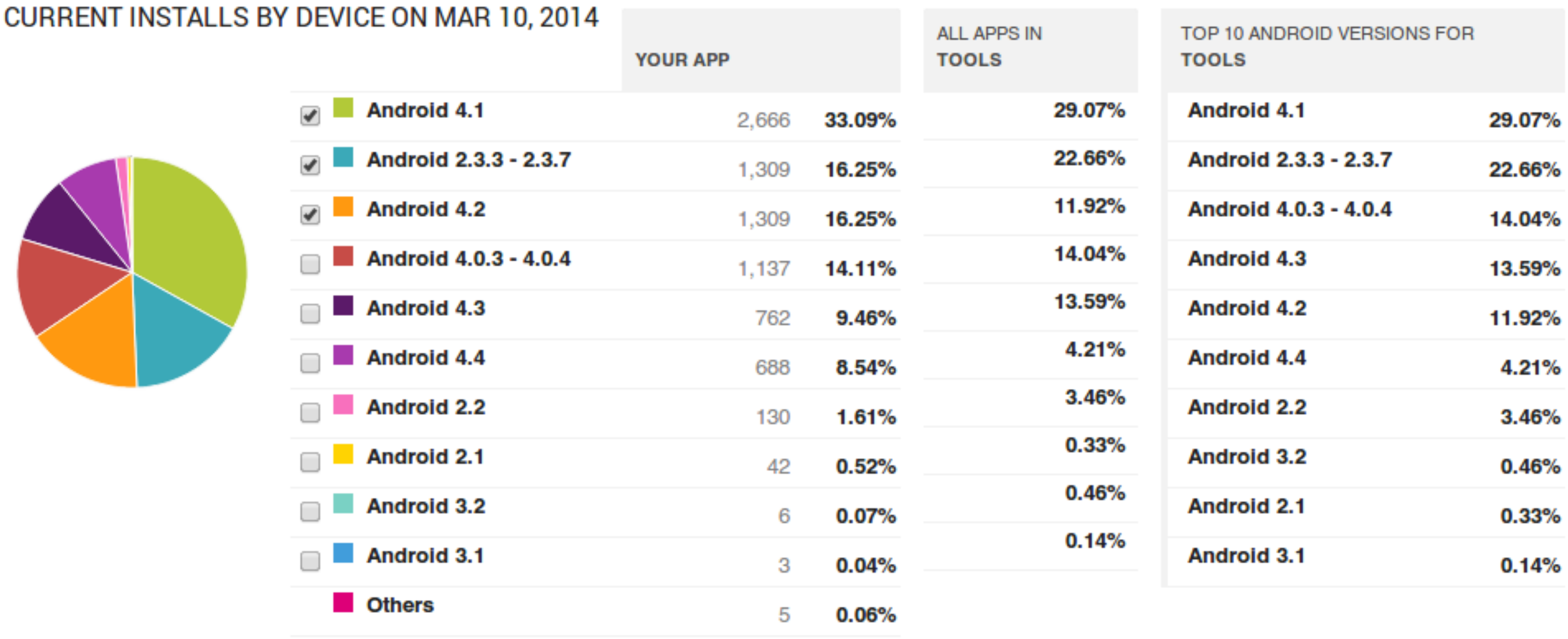
ReKey Stats - installs



APP NAME	PRICE	CURRENT / TOTAL INSTALLS	AVG. RATING / TOTAL #
ReKey (for rooted phones) 1.0.6	Free	8,057 / 32,732	★ 4.04 / 368

remember: we require a pre-rooted device

ReKey Stats - Android versions

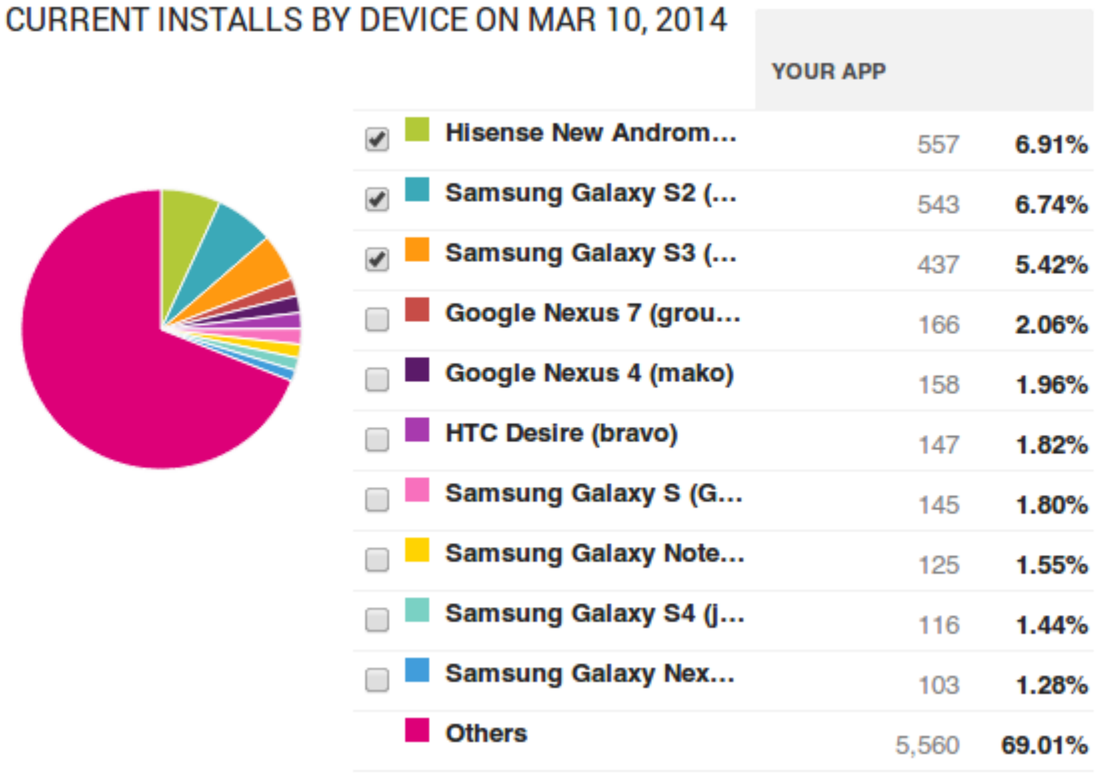


CURRENT INSTALLS BY DEVICE ON MAR 10, 2014

			YOUR APP		ALL APPS IN TOOLS
☑	■	Android 4.1	2,666	33.09%	29.07%
☑	■	Android 2.3.3 - 2.3.7	1,309	16.25%	22.66%
☑	■	Android 4.2	1,309	16.25%	11.92%
☐	■	Android 4.0.3 - 4.0.4	1,137	14.11%	14.04%
☐	■	Android 4.3	762	9.46%	13.59%
☐	■	Android 4.4	688	8.54%	4.21%
☐	■	Android 2.2	130	1.61%	3.46%
☐	■	Android 2.1	42	0.52%	0.33%
☐	■	Android 3.2	6	0.07%	0.46%
☐	■	Android 3.1	3	0.04%	0.14%
	■	Others	5	0.06%	

TOP 10 ANDROID VERSIONS FOR TOOLS	
Android 4.1	29.07%
Android 2.3.3 - 2.3.7	22.66%
Android 4.0.3 - 4.0.4	14.04%
Android 4.3	13.59%
Android 4.2	11.92%
Android 4.4	4.21%
Android 2.2	3.46%
Android 3.2	0.46%
Android 2.1	0.33%
Android 3.1	0.14%

ReKey Stats - Devices



CURRENT INSTALLS BY DEVICE ON MAR 10, 2014

			YOUR APP	
☑	■	Hisense New Androm...	557	6.91%
☑	■	Samsung Galaxy S2 (...	543	6.74%
☑	■	Samsung Galaxy S3 (...	437	5.42%
☐	■	Google Nexus 7 (grou...	166	2.06%
☐	■	Google Nexus 4 (mako)	158	1.96%
☐	■	HTC Desire (bravo)	147	1.82%
☐	■	Samsung Galaxy S (G...	145	1.80%
☐	■	Samsung Galaxy Note...	125	1.55%
☐	■	Samsung Galaxy S4 (j...	116	1.44%
☐	■	Samsung Galaxy Nex...	103	1.28%
	■	Others	5,560	69.01%

ReKey opt-in data

- 7k logs
- 942 unique device models
- Android versions
  - 1.5.1 to 4.4.2

Lessons Learned

“My ZTE Score M, is badly hacked and your software detected it, after I found obvious examples (all of which I video-taped). Help please if possible? Thank you.”

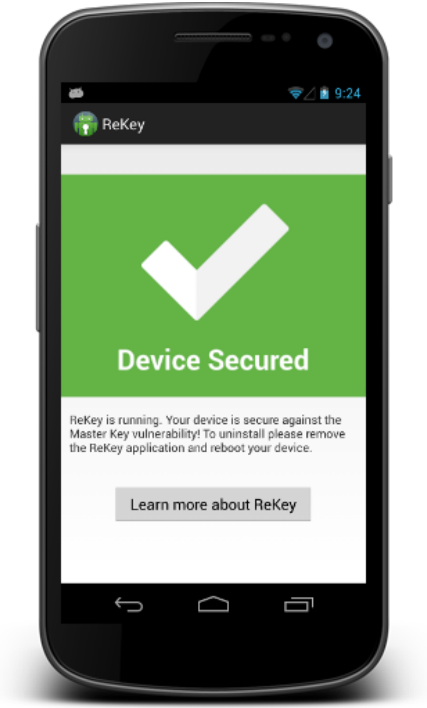
STAHP.

Conclusions

- **Android security is fucked**
- **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**

What’s Next?

- **PatchDroid / ReKey**
  - basically working but still a PoC
- **Add patches for vendor specific bugs!?**
  - that’s a lot of bugs
- **Open Source it?**
  - X-Ray probes are woefully out of date
  - Exynos, Webkit, MasterKey, etc
  - Interest in open source version for community development and new probes?



Device Secured

ReKey is running. Your device is secure against the Master Key vulnerability! To uninstall please remove the ReKey application and reboot your device.

Q & A

http://x-ray.io

http://rekey.io

http://patchdroid.com

detailed academic paper

twitter:

@collinrm @jonoberheide

Thanks & Greetz

- mudge
  - DARPA $$$
- Joshua ‘jduck’ Drake
  - heavy PatchDroid testing
- Greetz
  - zach, ben, van Hauser, i0nic, AHH crew

Alternative ‘Hotpatching’ Tools

- Xposed framework
  - made for modding Android without reflashing FW
  - replaces zygote

- Cydia Substrate
  - mode for modding Android without reflashing FW
  - complex