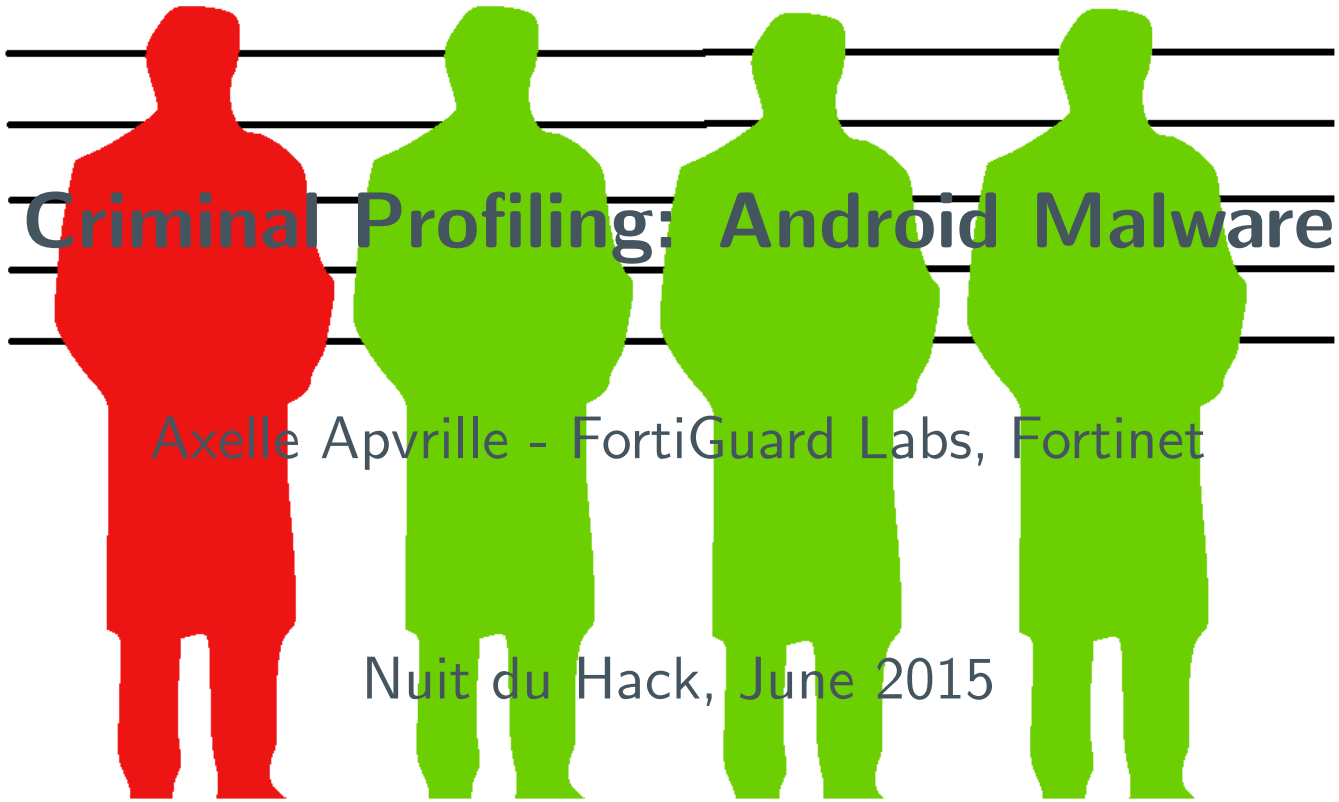


Criminal Profiling: Android Malware

Axelle Apvrille - FortiGuard Labs, Fortinet

Nuit du Hack, June 2015

FORTINET

Criminal Profiling

Kind warning

Plenty of **stats**

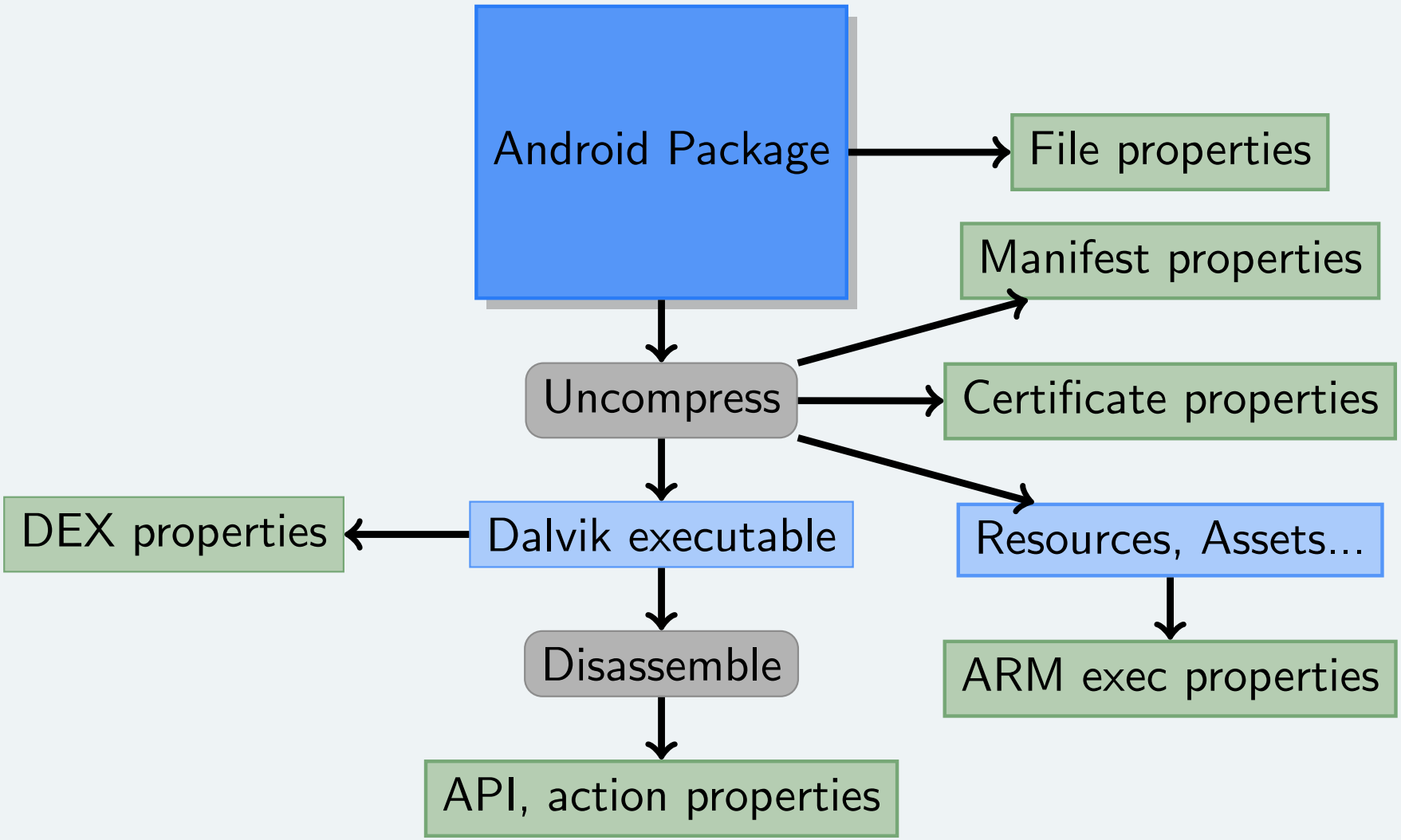
Feel free to    (or else)

Please tweet stats correctly though :)

Whenever possible, include how stats were computed: it matters (very much)

Want to re-use? Sure - please credit (fair, isn't it?)

How are stats computed?



How are stats computed?

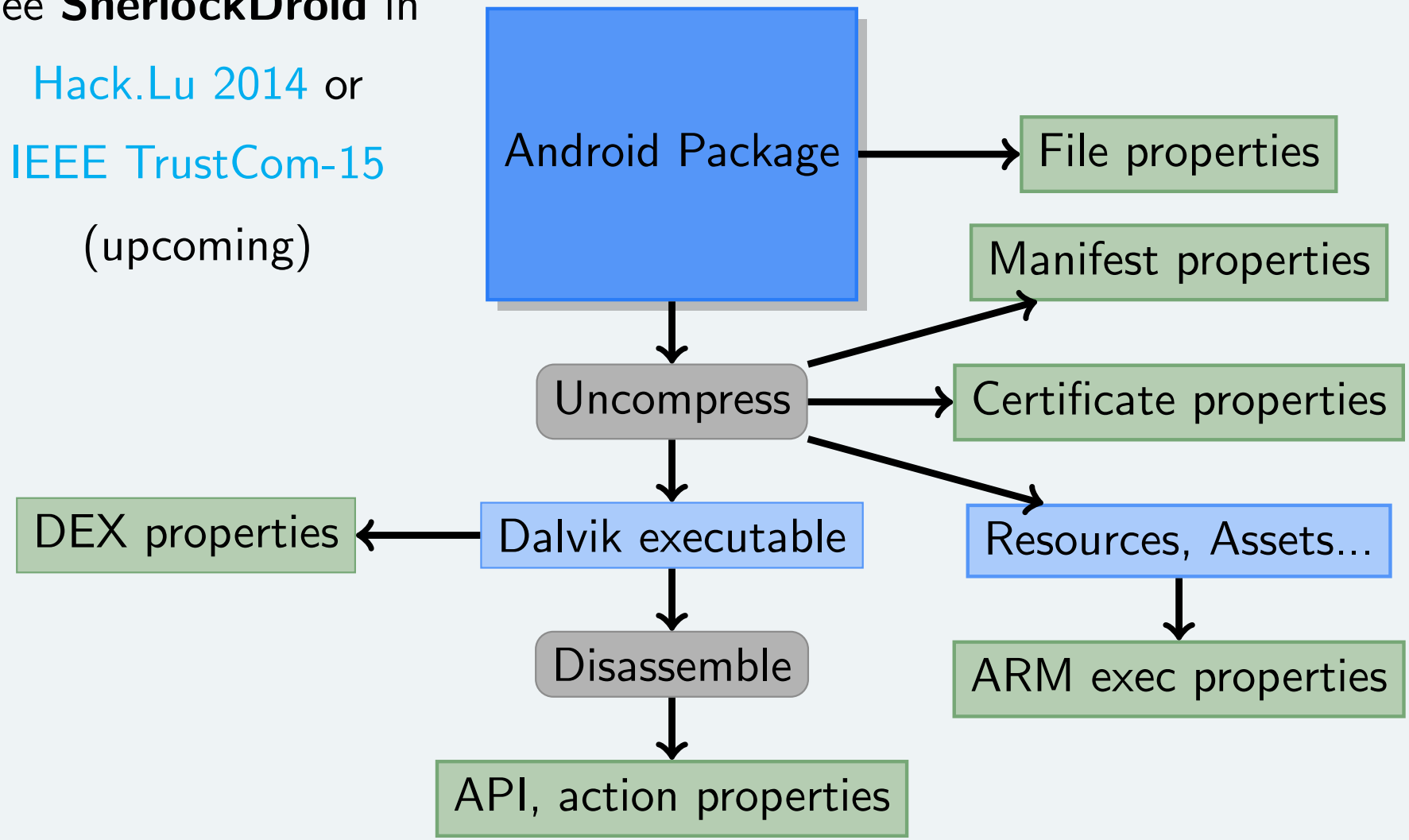
289 **static** properties

See **SherlockDroid** in

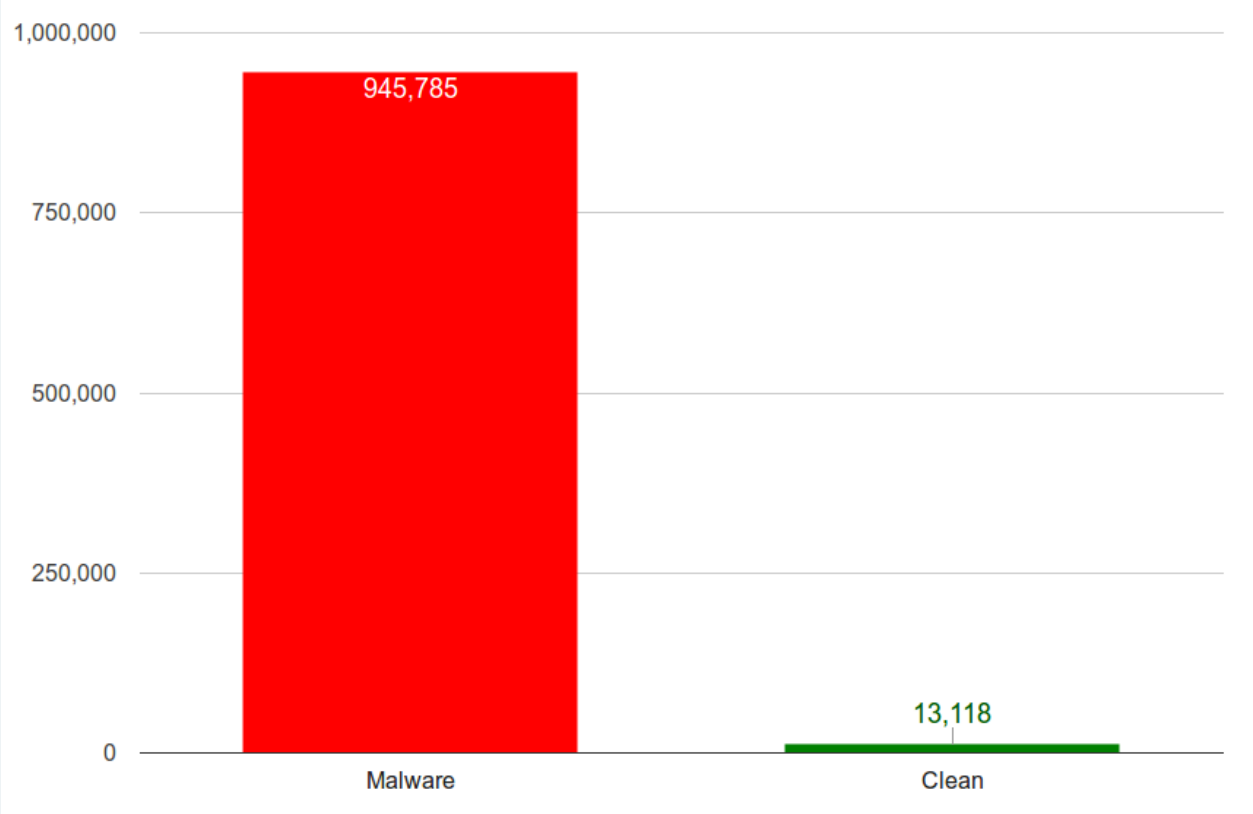
Hack.Lu 2014 or

IEEE TrustCom-15

(upcoming)



Datasets



- **Malware**: taken from Fortinet's DB - unique & non damaged samples only
- **Clean**: apps we analyzed manually, open source apps, top apps with known developer in Play Store

Why so few clean?

Hey, it's very difficult (and long) to be sure it's clean!

Number of samples

Unless specified otherwise, we considered:

Property type	**Nb of samples**
Package properties	945,785
DEX format properties	945,785
API call properties etc	**945,092**
Manifest properties	**617,942**

Properties in 3rd party kits (AdMob, JUnit...) are ruled out

Why not all?

- Some samples are incomplete (e.g. just classes.dex)
- Some samples are damaged
- Some properties are 'optional' (e.g targetSDK)

Comparisons

Many research papers use datasets of 100-1000 samples

We use close to 1 million

Comparisons

Many research papers use datasets of 100-1000 samples

We use close to 1 million

Android Malware Genome dates back to 2011

Our study is on samples collected before March 2015

Comparisons

Many research papers use datasets of 100-1000 samples

We use close to 1 million

Android Malware Genome dates back to 2011

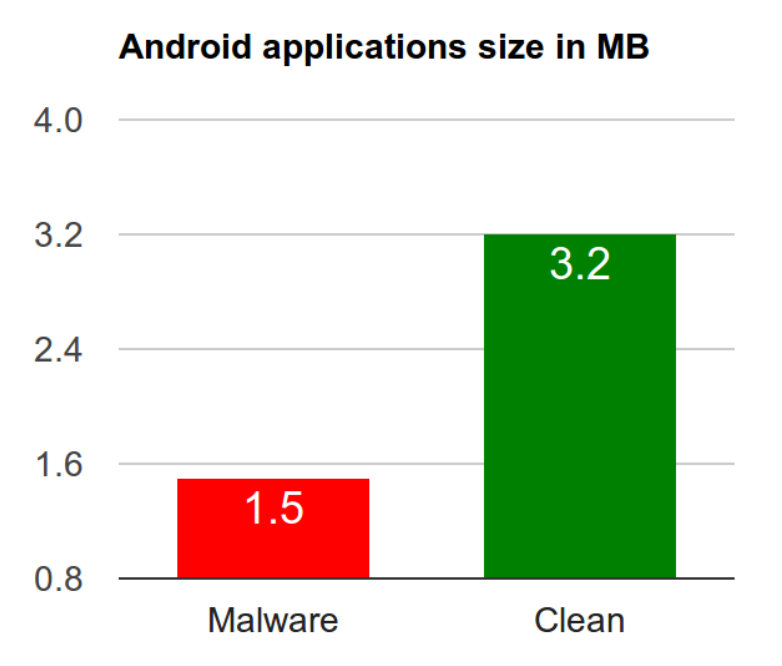
Our study is on samples collected before March 2015

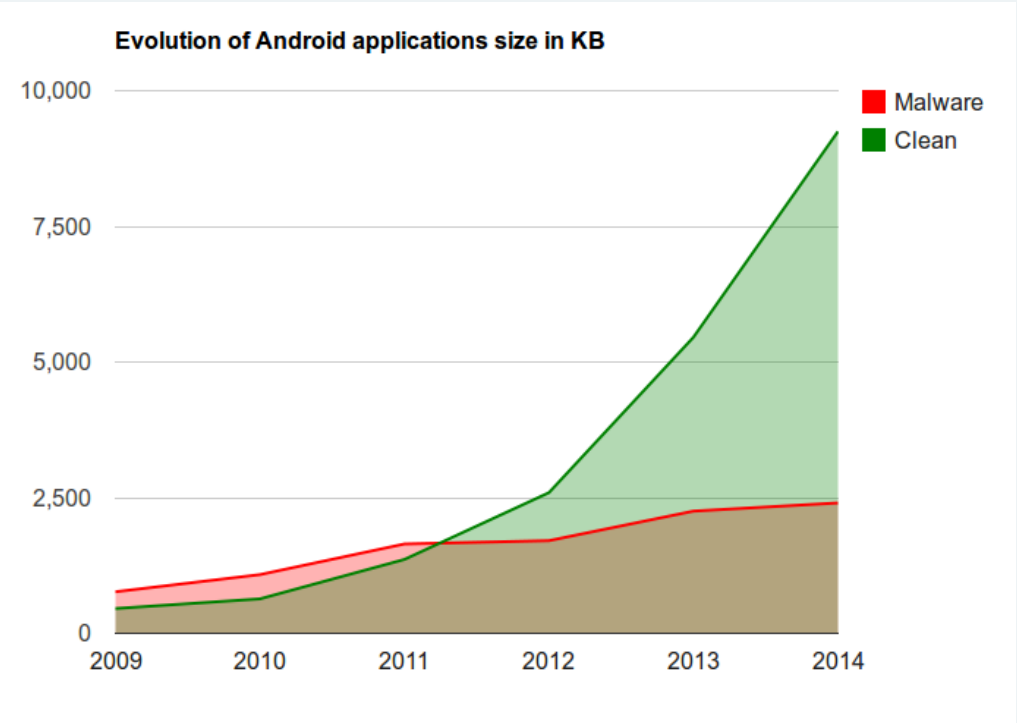
Extensive work: Andrubis (BADGERS'14), PlayDrone (SIGMETRICS'14)

Our study focuses on malware with stats on code-level properties

Criminal Profiling: What Do Malware Look Like?

Sample file size



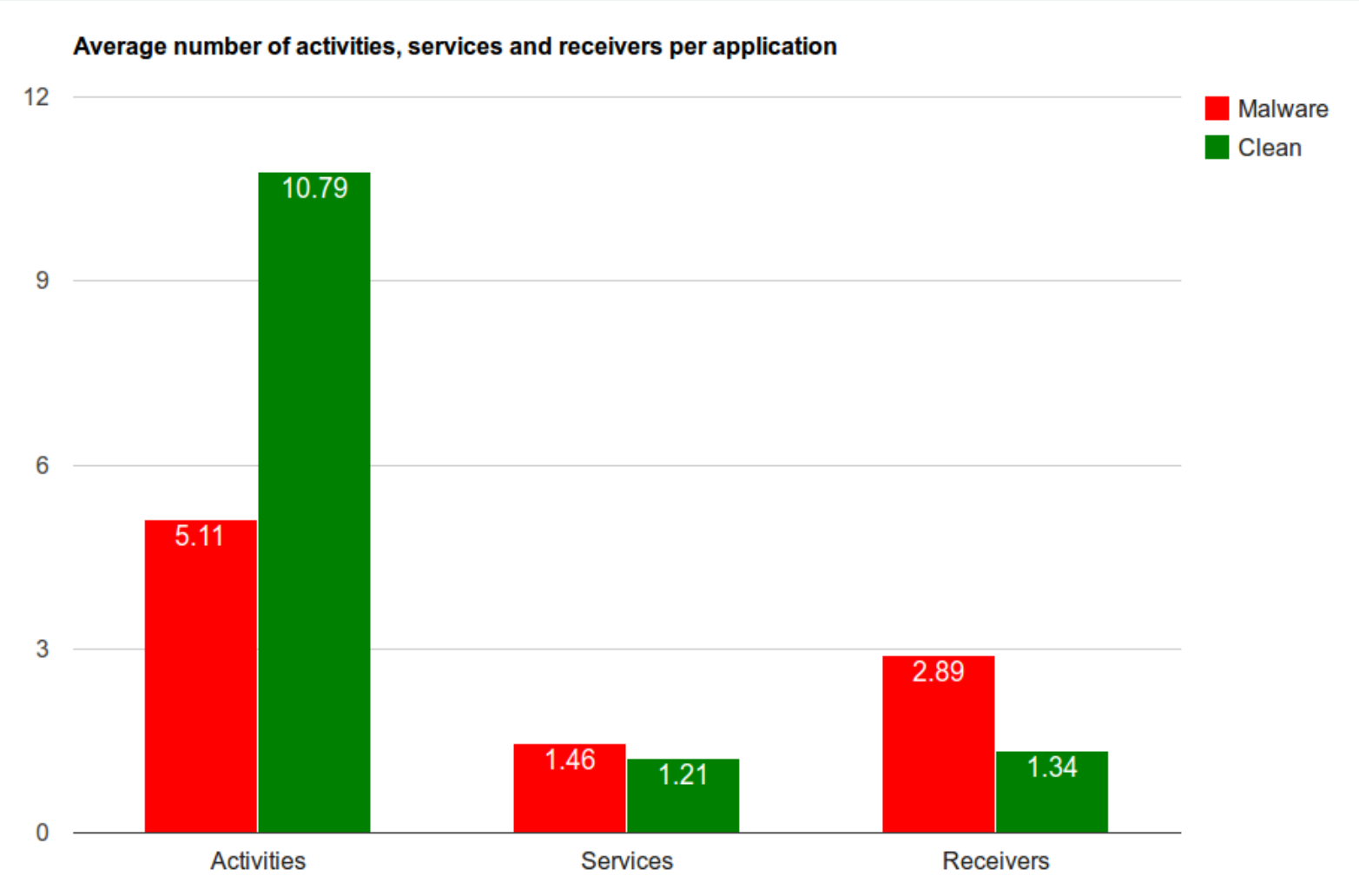


End of 2014

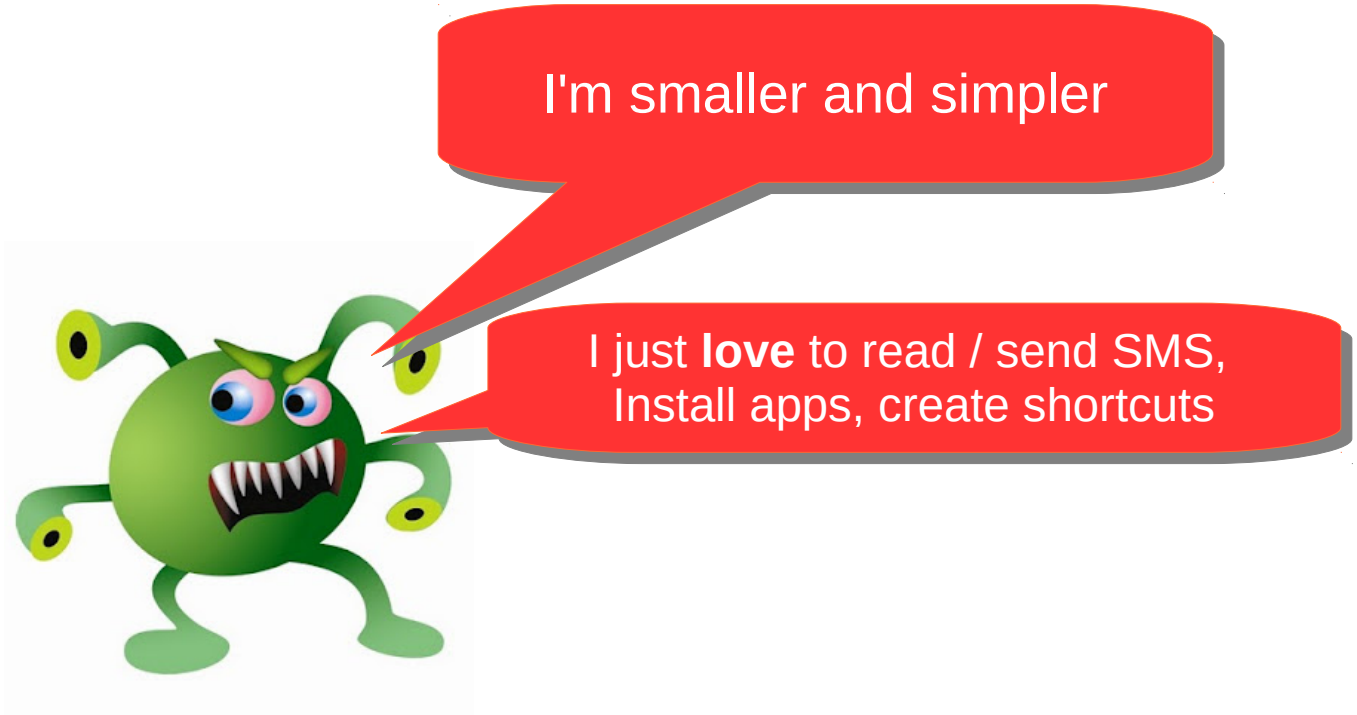
Clean: 9.2M average **4x bigger** than Malware: 2.4M average

Malware don't need to implement all features

Activities, services, receivers

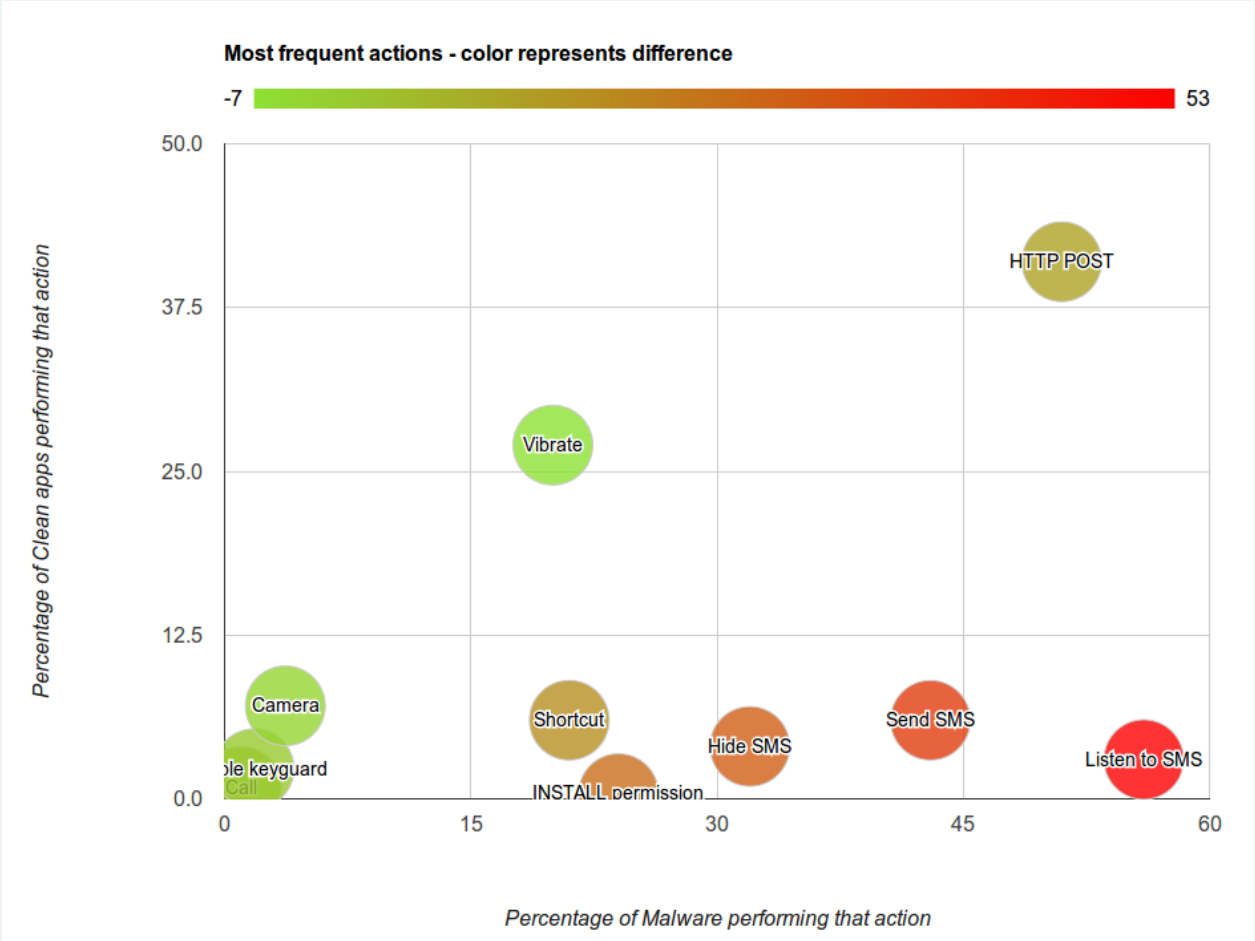


Criminal Profiling: What Do Malware Like?

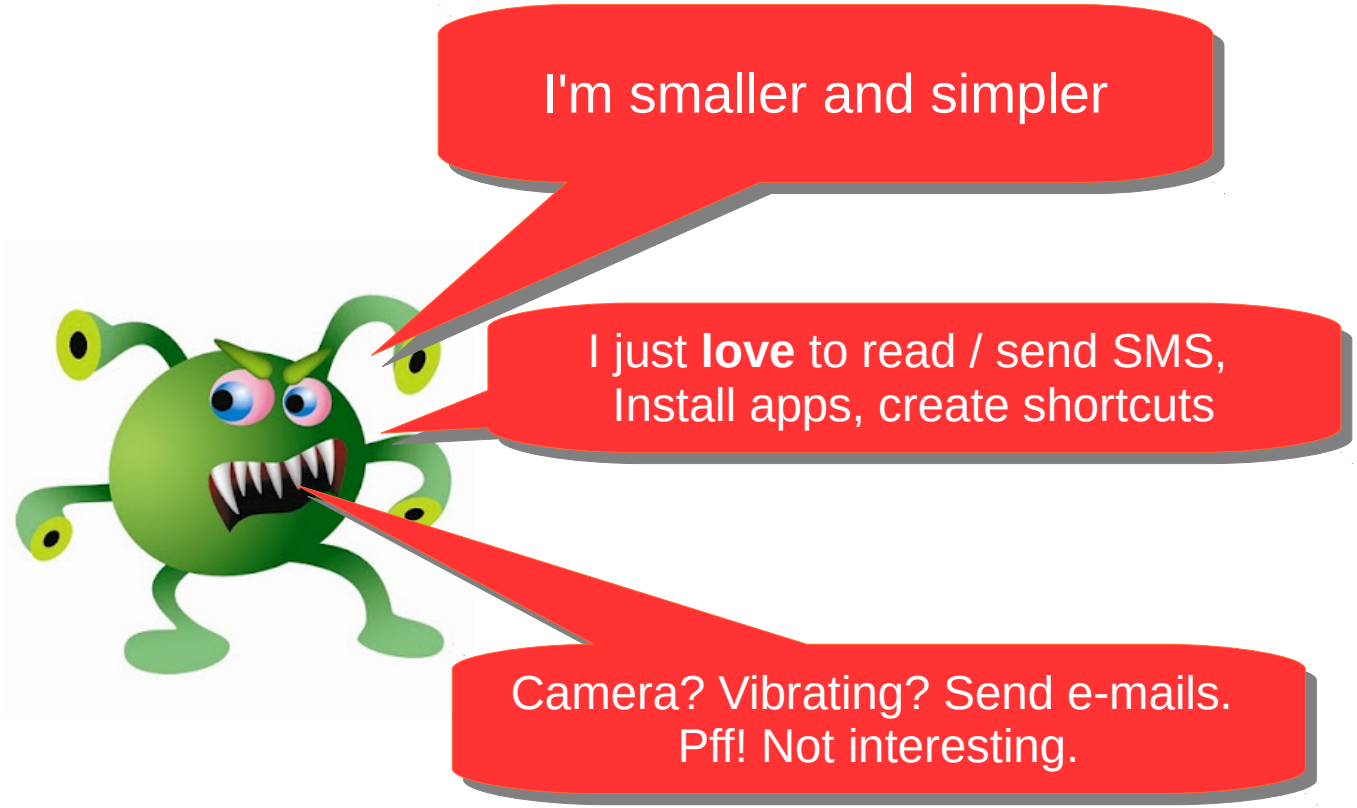


SMS: a strong indicator!

- 56% of malware implement a **SMS receiver**! (only 3% of clean)
- 43% of malware **send SMS**!
- 32% of malware use `abortBroadcast()` to conceal incoming SMS!



Criminal Profiling: Other Interests

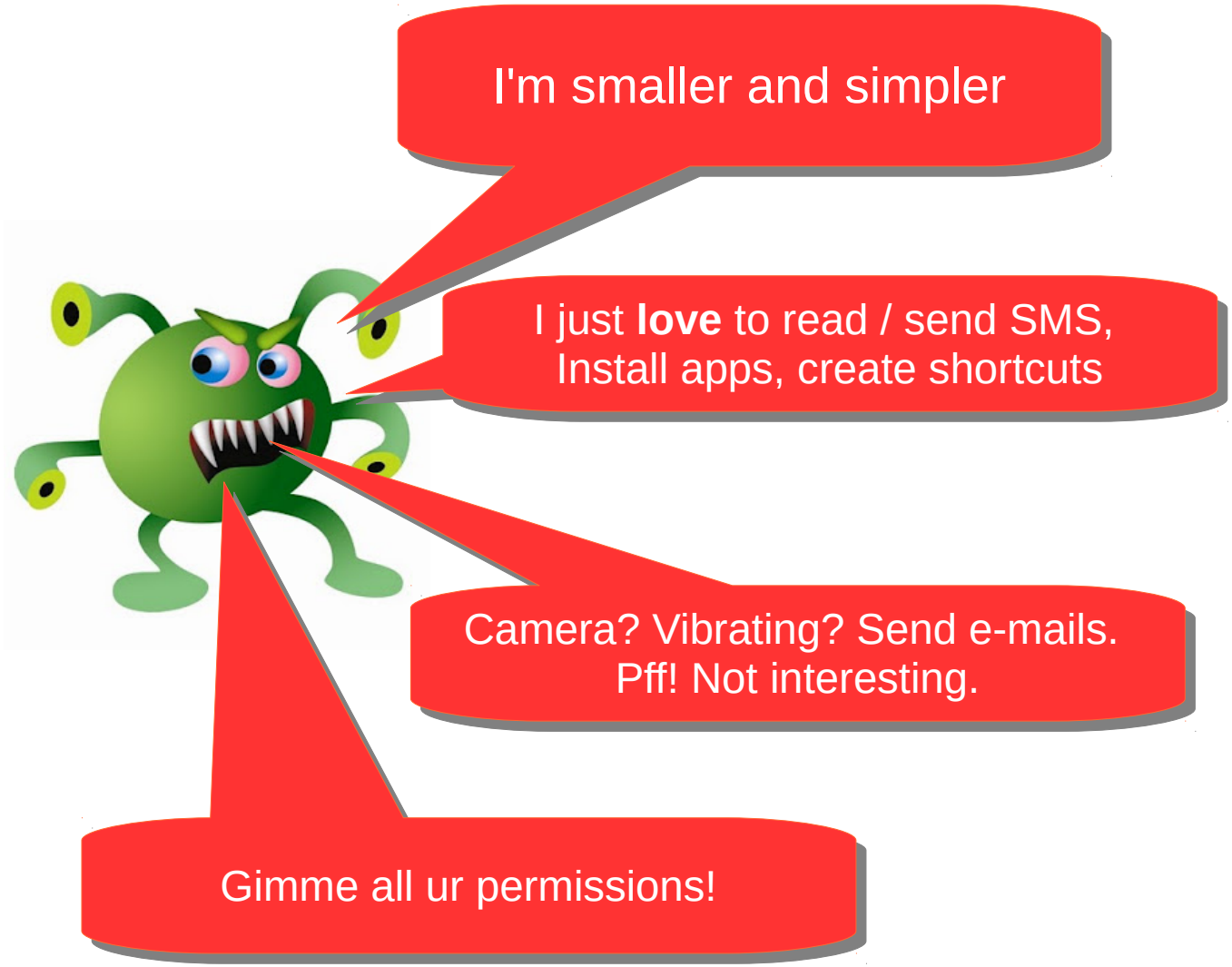


What Malware Like / Don't Like

- **INSTALL_PACKAGES**: 24% malware ask for it. Only 0.4% clean apps do. NB. Works for system applications only.
- Install **shortcuts**: 21% malware, 6% clean apps.

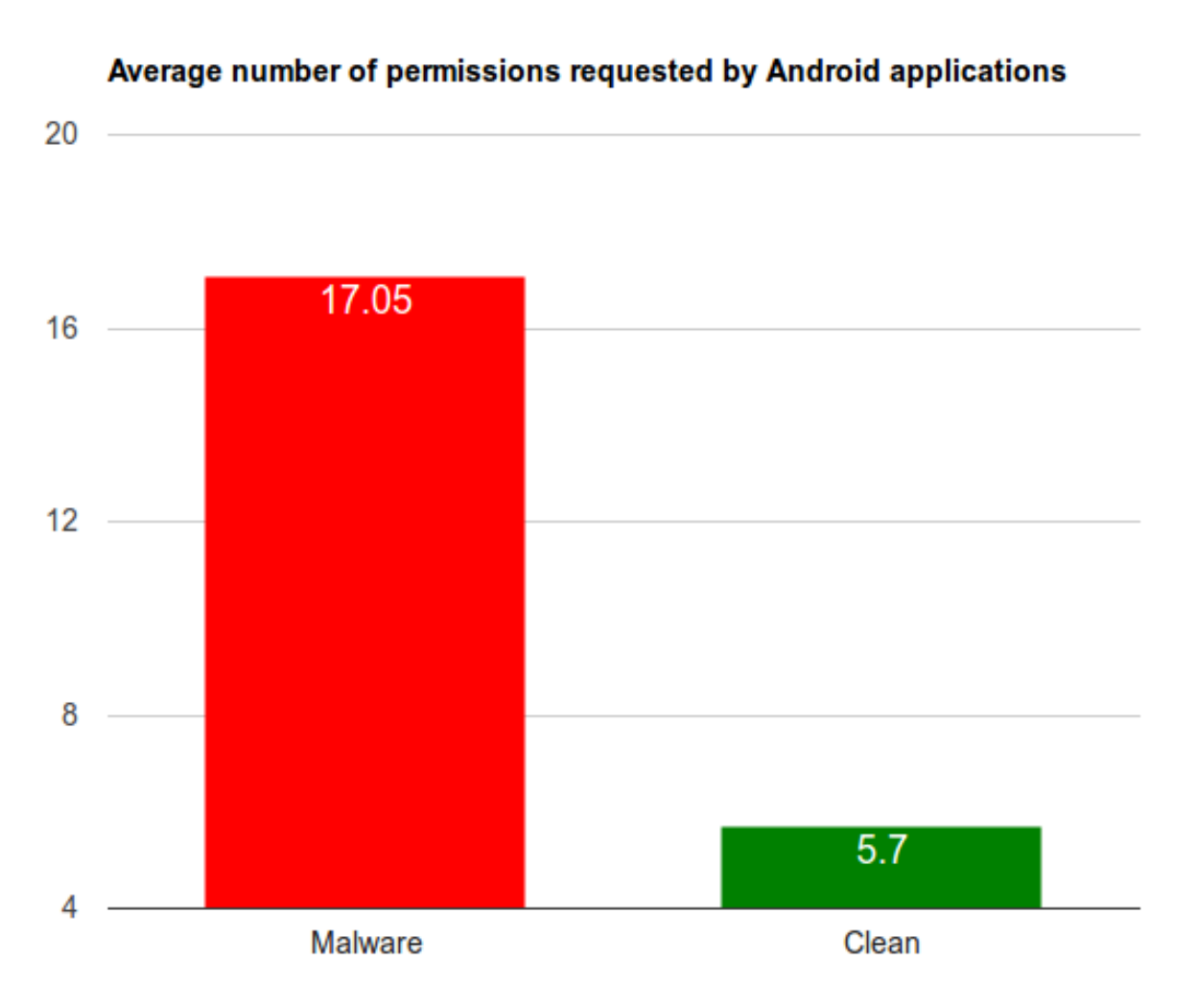
- **Emails**. 14% malware $<$ 29% clean (support/contact)
- **Vibrate**. 20% malware (ransomware?), 27% clean
- Is the era of **premium phone number** dialers over? 1%
- **Camera**. 3.7% malware, 7.1% clean. Only if you're a VIP ? ;)
- **Disable the keyguard**. Malware can run background tasks as services...

Criminal Profiling: Your Permissions, or Your Life!



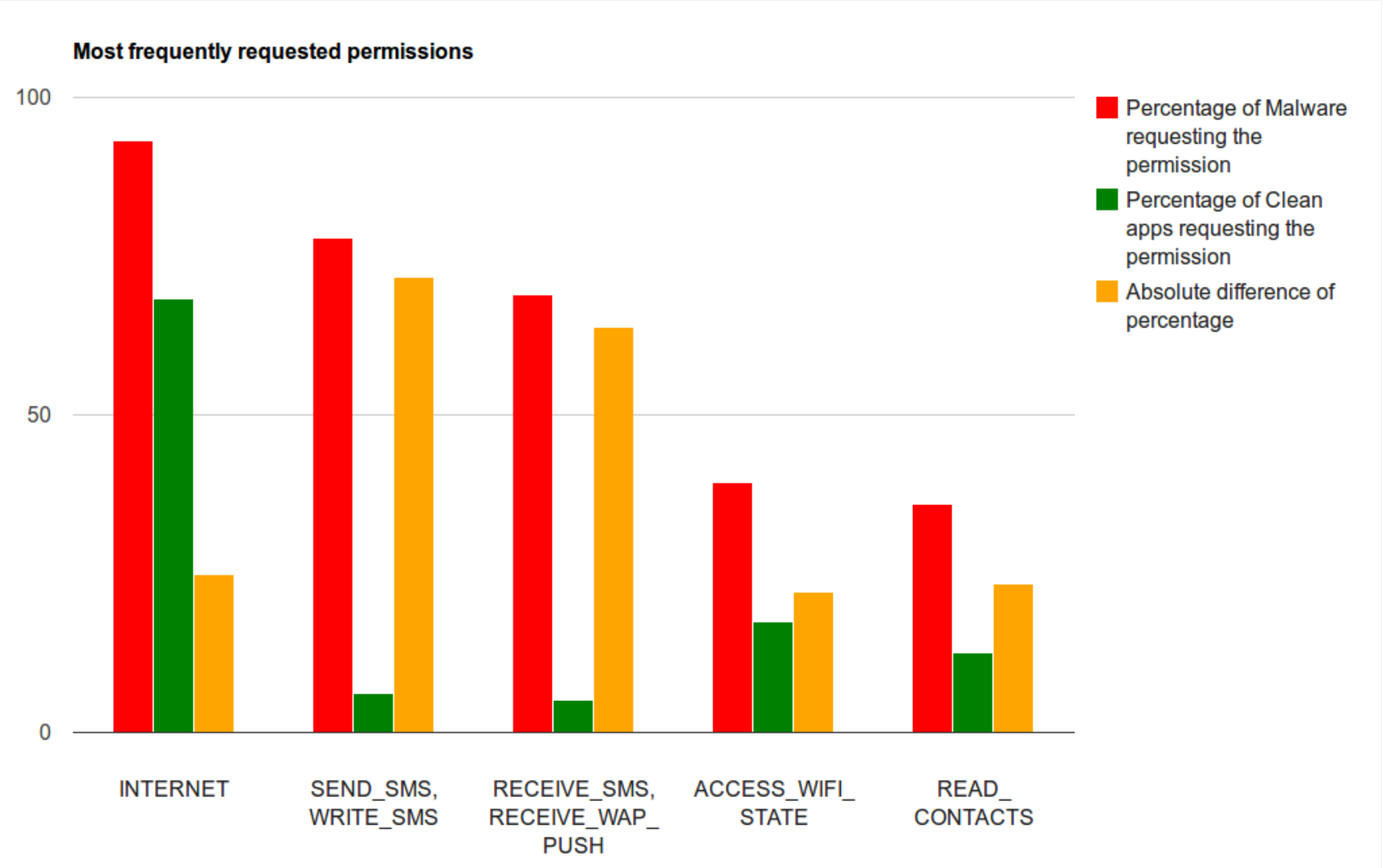
Web search for Angecryption

Permissions indicate evil will...

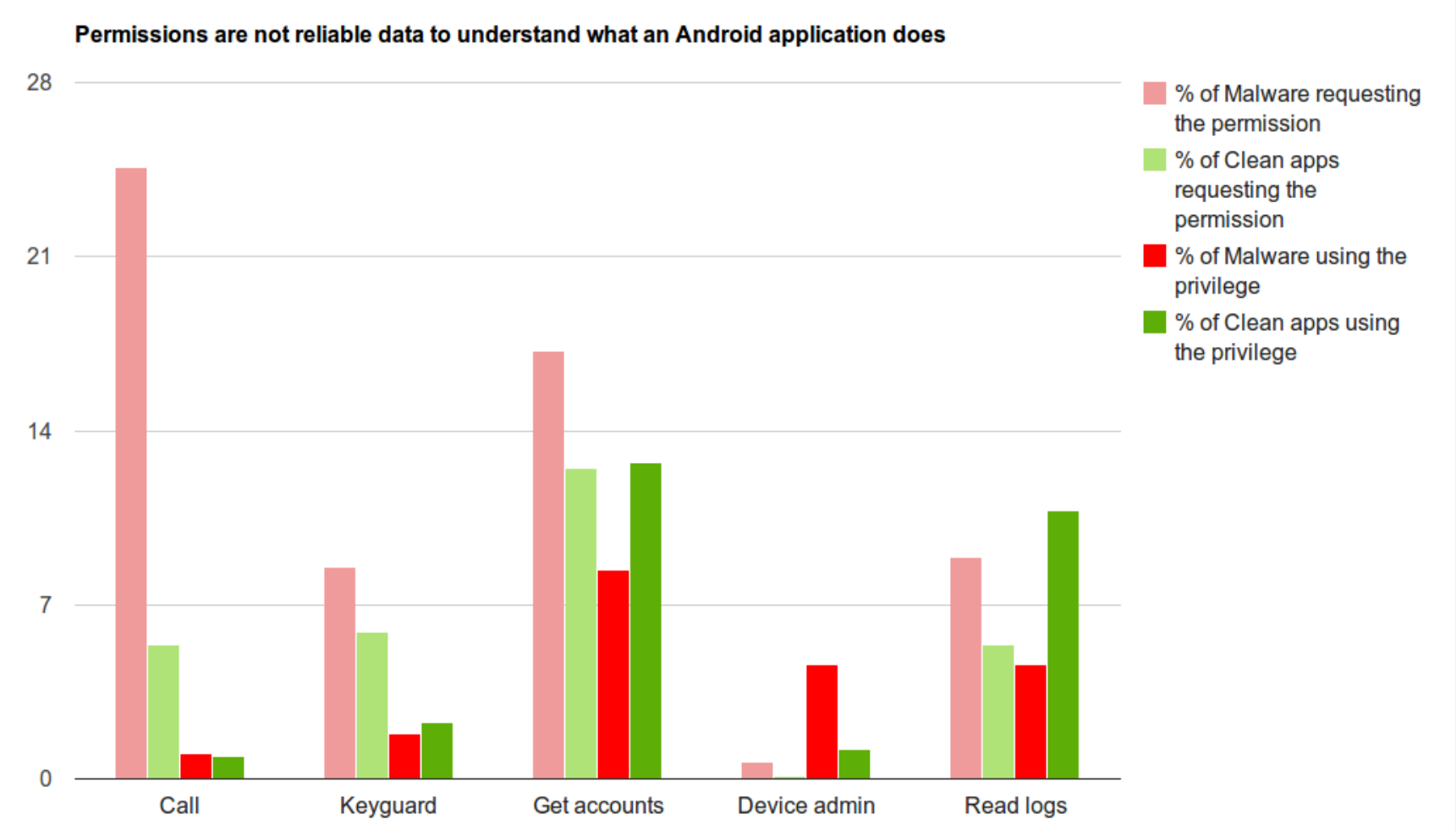


Clear over-use of permissions!!!

Top 5 permissions



Permissions are not so reliable



Why can't we rely on permission stats?

A permission may be requested but never used

Or the permission can be used within (legitimate?) third party code

Example: call permission vs ACTION_CALL/_DIAL

We don't have the manifest for all malware

Explains rare cases where use > request

Example: BIND_DEVICE_ADMIN permission vs DeviceAdminReceiver

Bypassing permissions

- Call another app that has the permission
- Escalate privileges via updating
- Hijacking the Android installer
- Use an exploit...

Wide Target


I target even “old” Android system
Don't want to lose an opportunity, do I?

Declared Target SDK

On average

- Malware target **Gingerbread**
- Clean apps target **Jelly Bean**

Stats

Considered 'only' 6,976 malware and 707 clean
Why not 900K?

- All samples don't come with a manifest
- All manifests don't come with target SDK

Malware profiling: targets

Geographic attribution statistics

Amount of data

- Country of application's certificate (575,396)
- Rule out unknown countries, buggy and fake entries
  - e.g. GF is not a correct country code
  - e.g. VU is Vanuatu but this entry is probably fake: CN=VU OU=VU O=VU L=VU ST=VU C=VU
  - **63% ruled out!**
- Rule out dev / debug certificates (12%)
- Remaining: **146,764 certificates**. 14,919 in 2014, and only 6,308 in 2015 (incomplete).

Geographic attribution is complicated

Attribution script turned out to be tricky

Plenty of cases!

- Certificates using call codes (e.g. +86 for China) or zipcodes
- Match towns or 'states' to countries (e.g Gweru is in Zimbabwe)
- Deal with errors e.g C=CH for China, C=CA for California...
- Fixed several bugs, but probably others :((
  - C=gg-2 (fake country) was counting for ... Guernsay
  - C=asd3f21asdf was counting for American Samoa

Malware certificates: target or origin?

Examples

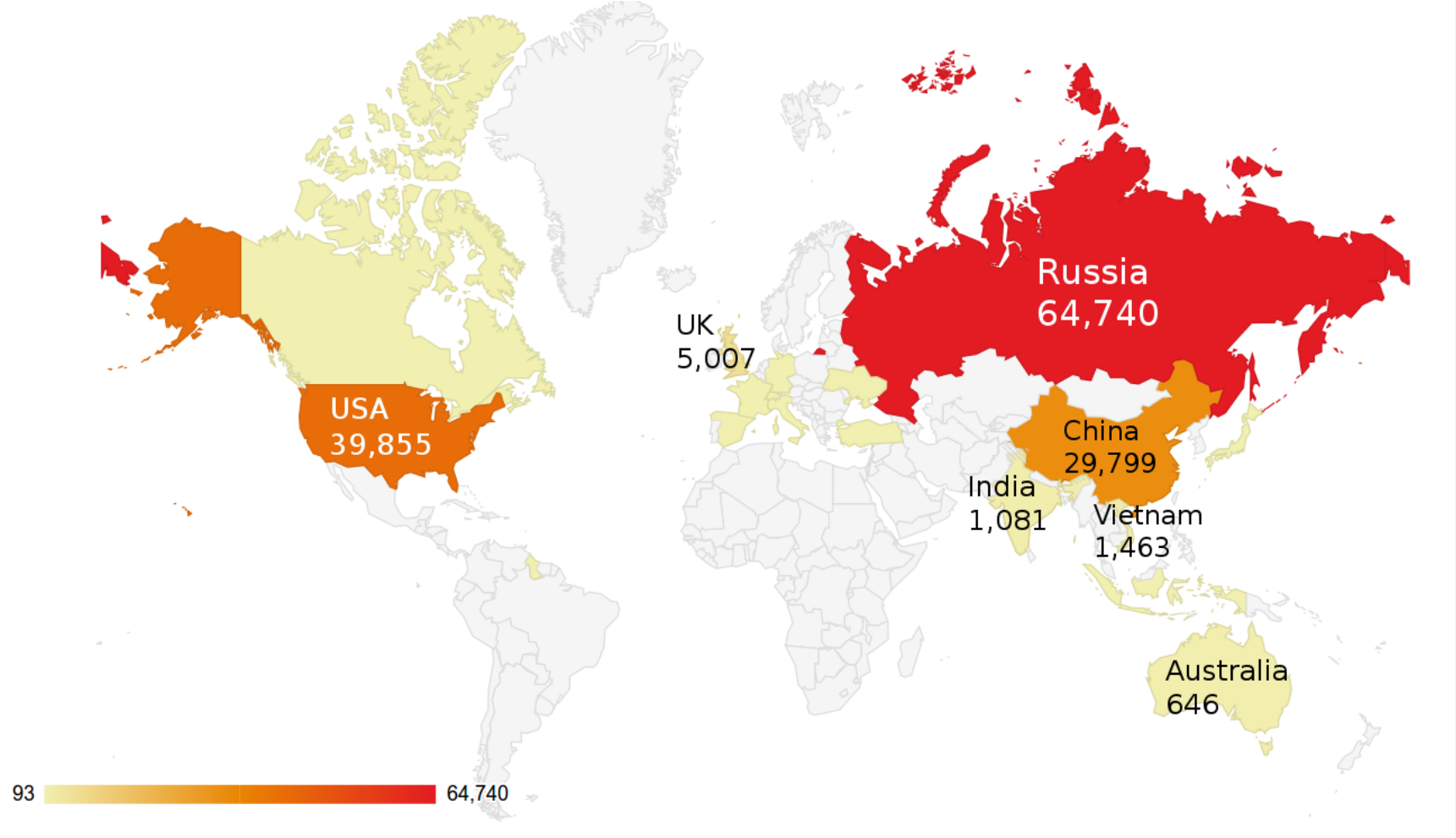
CN=Praveen Kumar Pendyala OU=Student O=IIT Bombay L=Mumbai ST=Maharastra C=400076

CN=Dau Dinh Manh O=Song Vang L=Ha noi ST=Ha Noi C=84

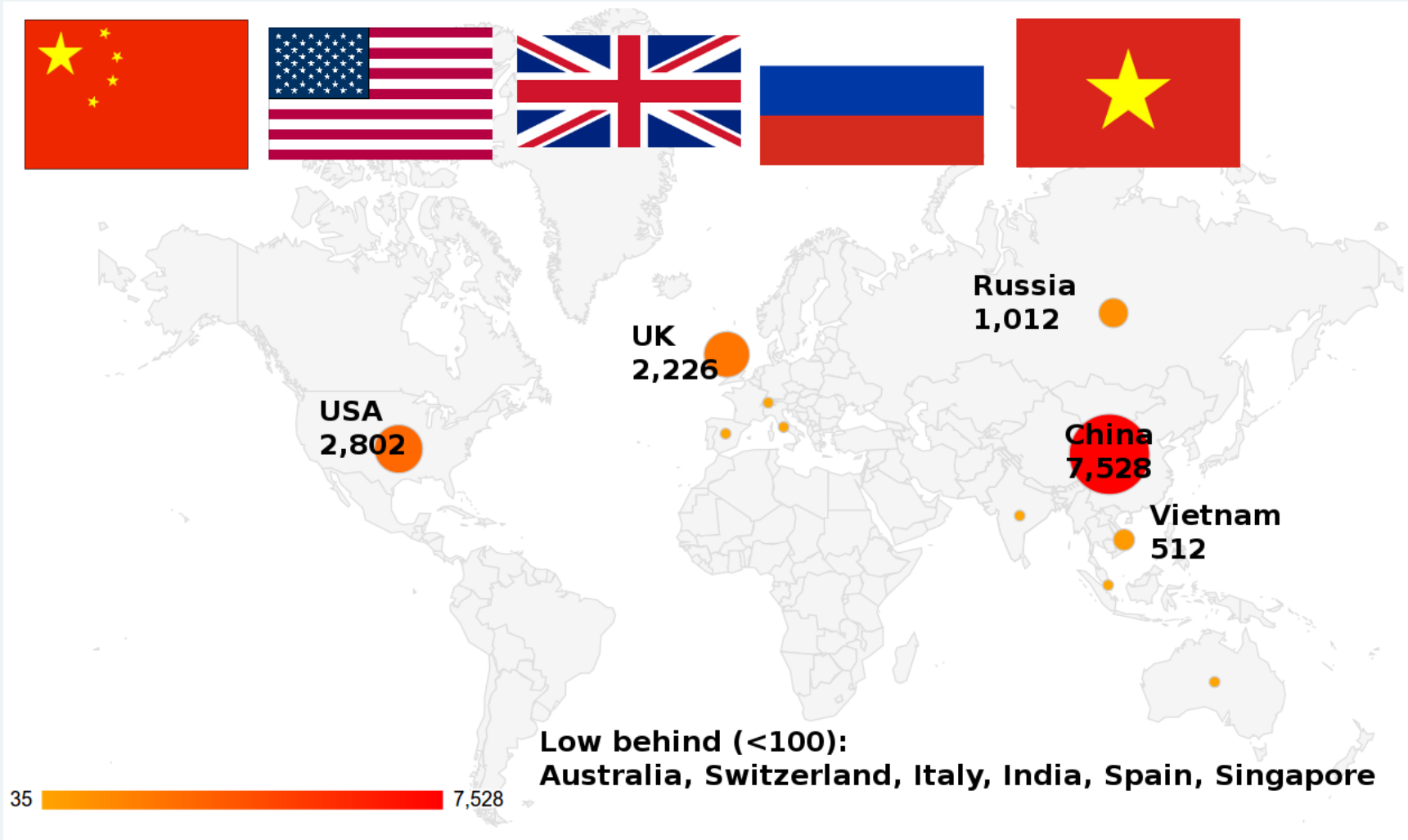
CN=Zhong Zhang OU=Zhainanzhi Inc O=Zhainanzhi Inc L=FuZhou ST=FuJian C=CN

- Many certificates with a seemingly valid identity
- Why mention a particular name?
  - For fame?
  - Because they don't believe their app is malicious?
  - Because they think we won't notice?
  - To complexify attribution?
  - Trojanized app where original certificate name was retained?

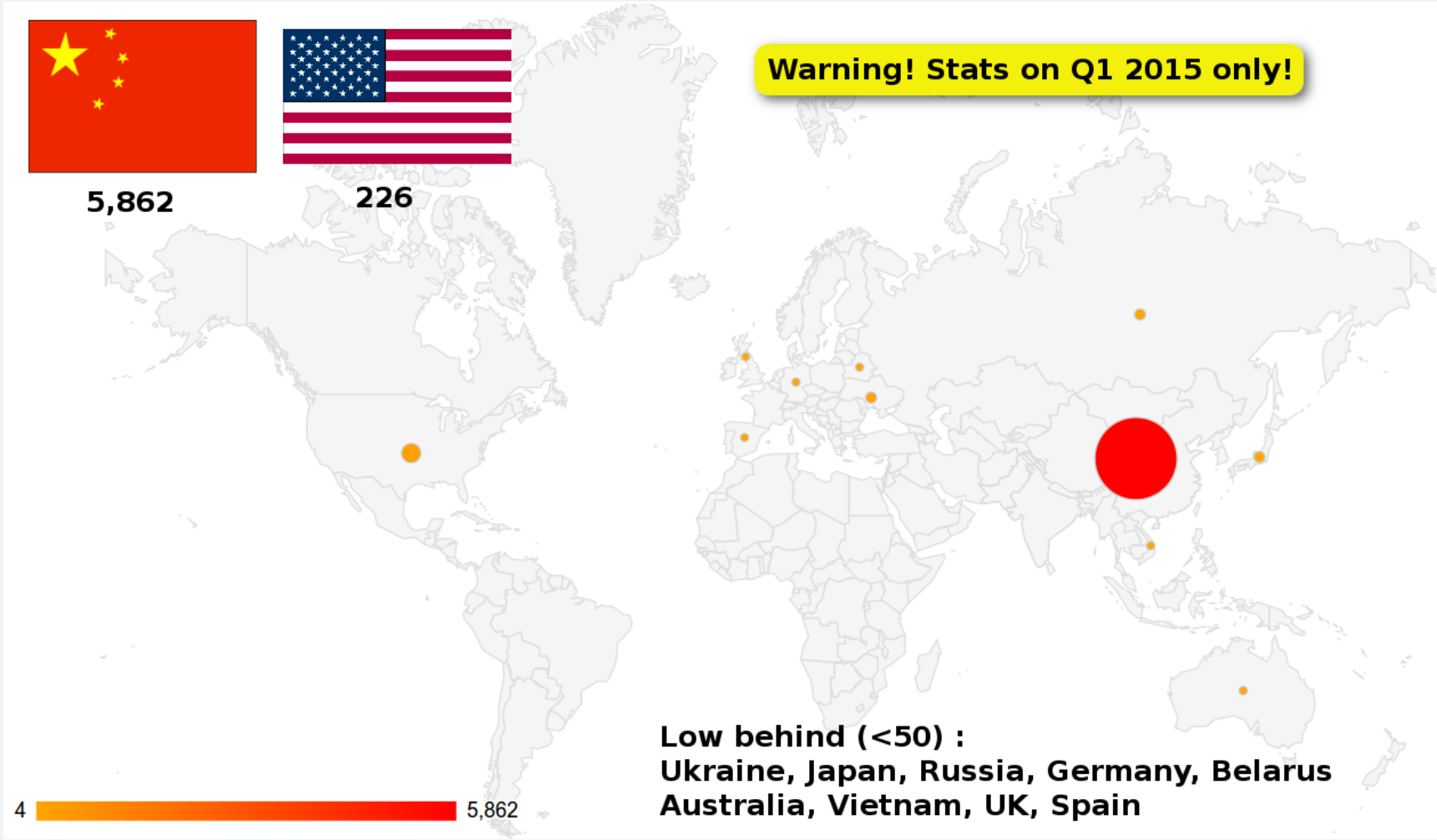
Presumed Targets of 146,764 malware



Top target countries in 2014



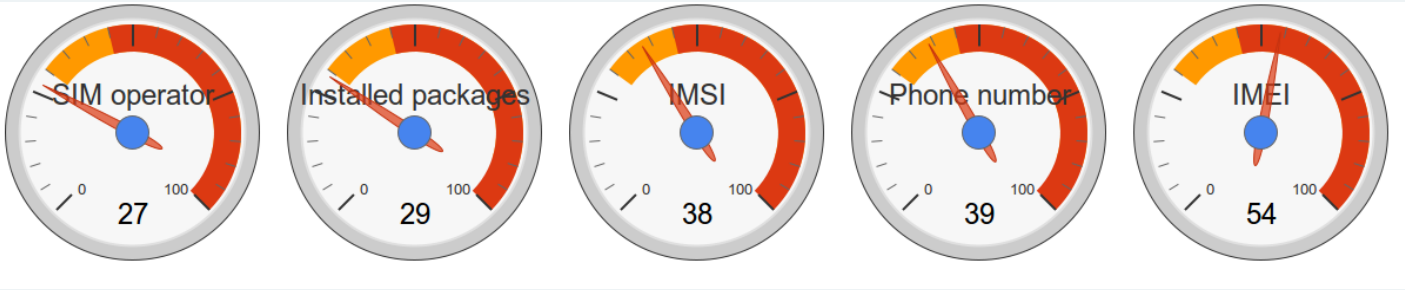
Top target countries in 2015



Information, I want information!

Most representative collected data

Collected Data



Captain Obvious:

- IMEI, IMSI, Phone number...

Not so obvious

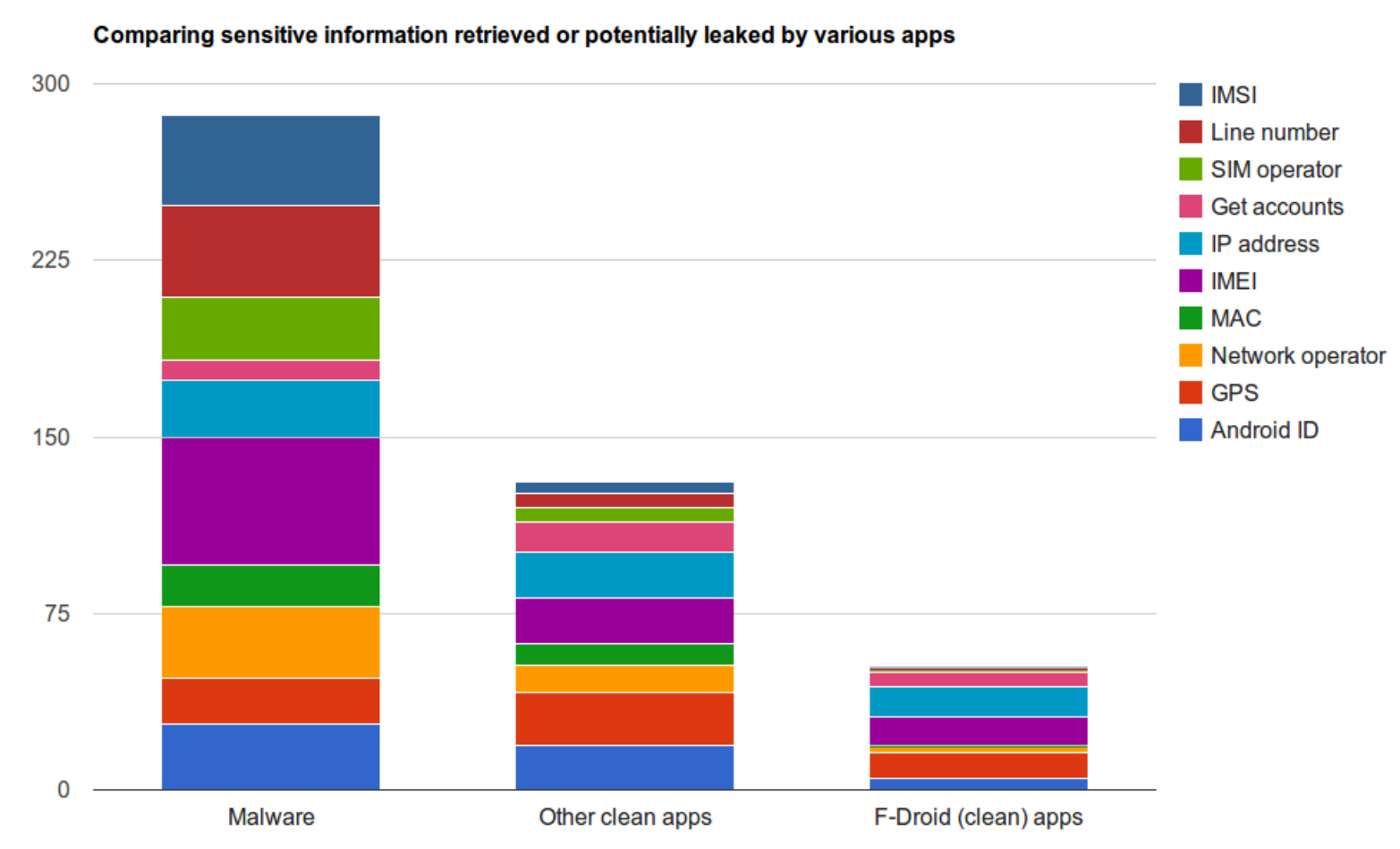
We hadn't expected the diff with clean apps would be so strong:

- IMEI collected $\approx$ **3 times more** for malware
- Phone number, IMSI, S/N: **6 times more**
- List apps, SIM operator: **4 times more**
- Android ID, MAC address: **twice**

What reason for those???

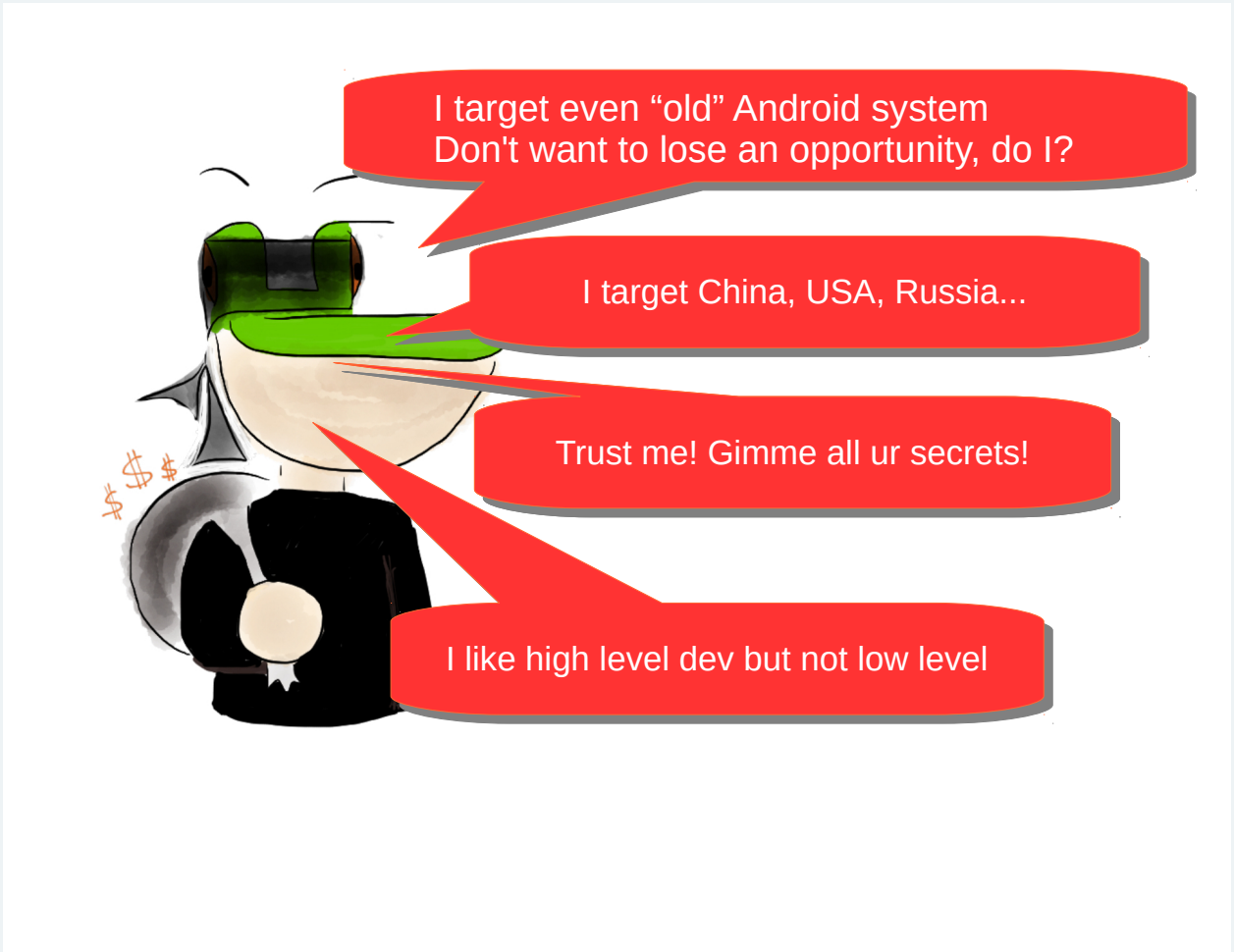
- GPS ($\approx$ 22% for both)
- Get accounts (9% malware, 13% clean)

Sidenote: comparing with F-Droid apps

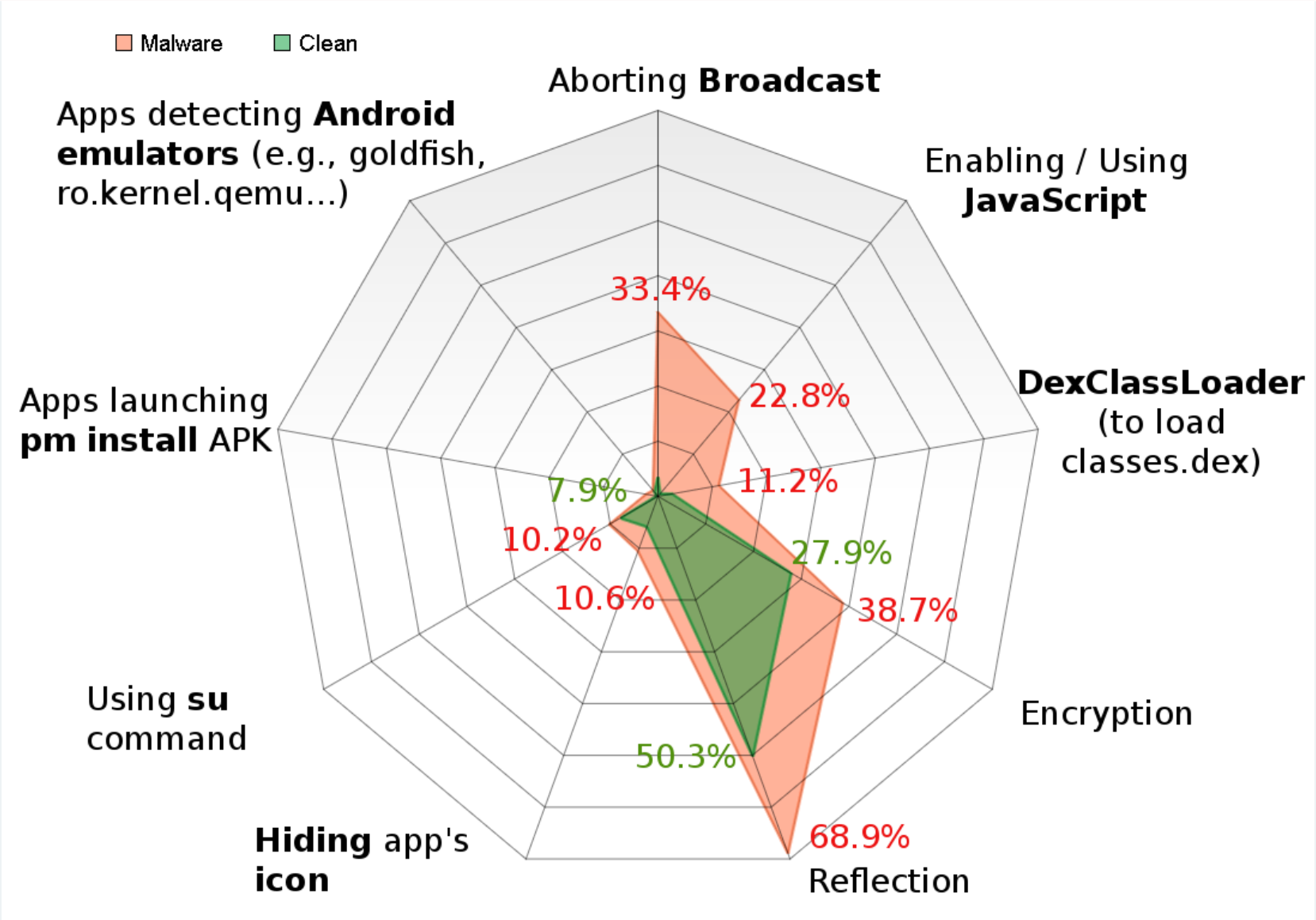


F-Droid (Free and Open Source Software Android apps) far cleaner than the average

Malware authors: how much skills?



Most frequent techniques



Reminder: code from third party kits are ruled out

Techniques: What Do We Make Out of It?

Malware authors are not Unix geeks:

- `su` (8-10%), chmod ($< 2\%$), mount ($< 1\%$), busybox ($\approx 1.5\%$)
- Command line installation `pm install`: only 2.2%
- Android emulator detection: only 1.4%

Techniques: What Do We Make Out of It?

Malware authors are not Unix geeks:

- `su` (8-10%), chmod ($< 2\%$), mount ($< 1\%$), busybox ($\approx 1.5\%$)
- Command line installation `pm install`: only 2.2%
- Android emulator detection: only 1.4%

Malware authors are not (particularly) keen on native dev:
No significant difference in using JNI (23-26%), executing native process (21-24%)

Techniques: What Do We Make Out of It?

Malware authors are not Unix geeks:

- `su` (8-10%), chmod ($< 2\%$), mount ($< 1\%$), busybox ($\approx 1.5\%$)
- Command line installation `pm install`: only 2.2%
- Android emulator detection: only 1.4%

Malware authors are not (particularly) keen on native dev:
No significant difference in using JNI (23-26%), executing native process (21-24%)
Malware authors have development skills:

- Android SDK: abortBroadcast(), DexClassLoader, setComponentEnabledSetting()
- JavaScript (22.8% malware - only 0.6% clean)

Techniques: surprises

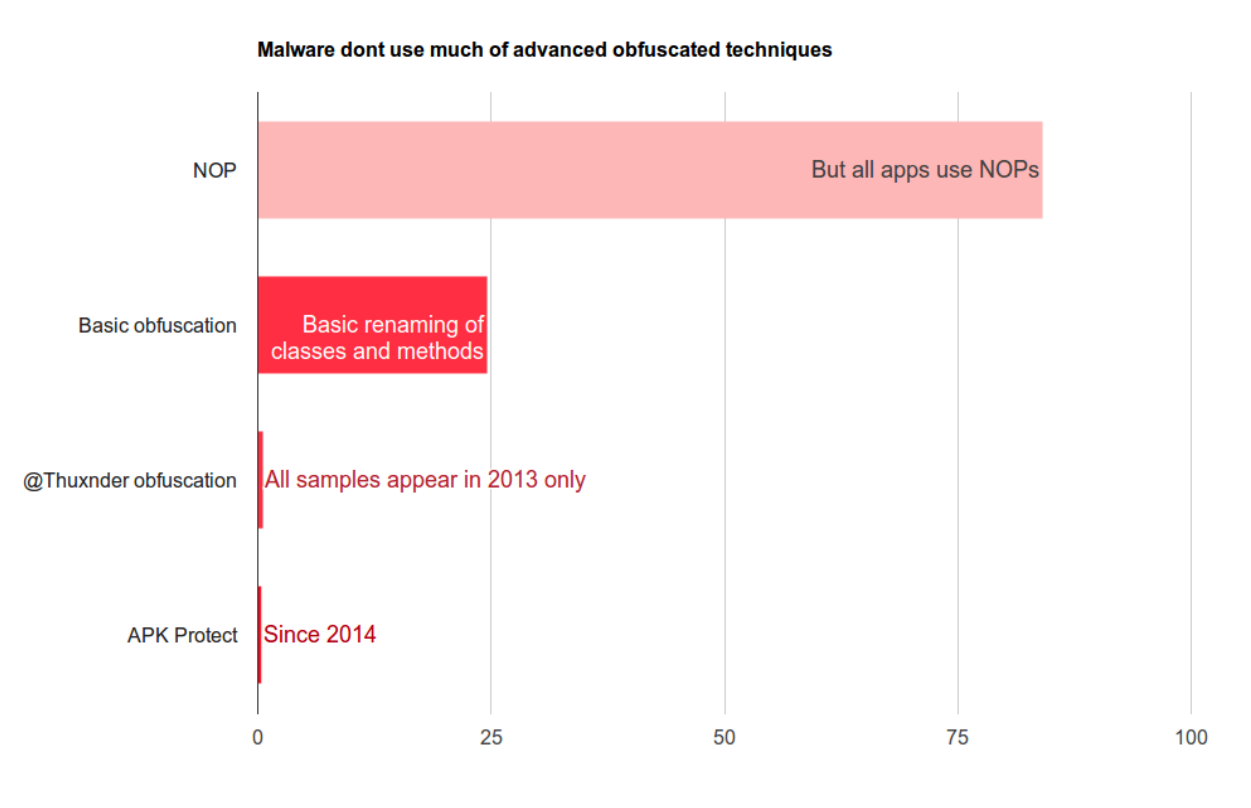
Why is everybody fond of reflection and encryption?
Reflection: 68.9% malware, 50.3% clean
Encryption: 39.7% - 27.9%
Because they're old/well-known techniques?

Techniques: surprises

Why is everybody fond of reflection and encryption?
Reflection: 68.9% malware, 50.3% clean
Encryption: 39.7% - 27.9%
Because they're old/well-known techniques?

What are clean apps doing with openDexFile and loadDex?!
0.3% malware - 0.4% clean
Dalvik.system.DexFile - openDexFile() is private

Obfuscation: smaller than expected?



- NOPs are meaningless
- Basic obfuscation = ProGuard a, b, c renaming
- @thuxnder obfuscation (2012) = abusing linear sweep with `fill-array-data` = 0.5%. All 4,800 samples in 2013.
- APKProtect: since 2014

Obfuscation (continued)

Reliable properties

nop opcode, APKProtect string, @thuxnder

```
if-eq v0, v0, +9
fill-array-data v0, +3
fill-array-data-payload
```

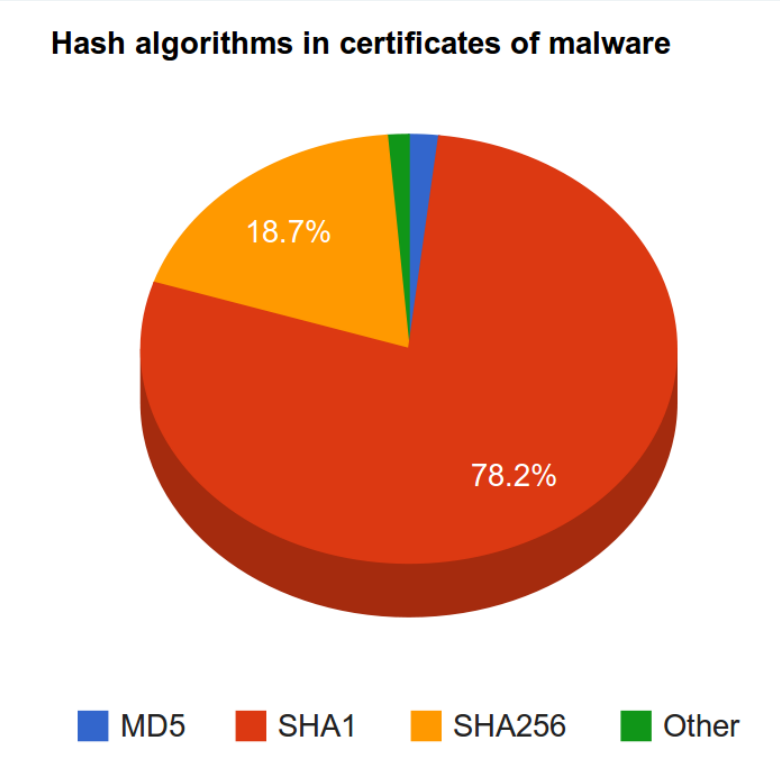
Unreliable property: basic obfuscation

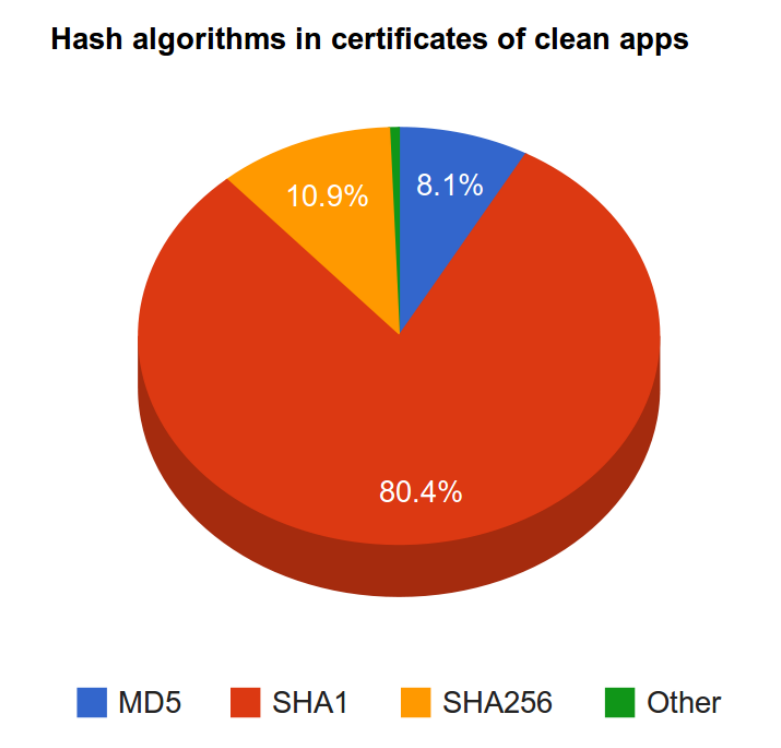
- AESObfuscator-1: used by Android LVL
- /a/a;->a: simplistic!!!

Issues

- NOPs mentioned by Mody (VB 2013)
- Lipovsky (CARO 2014) estimates all abusing linear sweep up to 30%
  - Seems too high
  - Unless I miss samples or case detections?

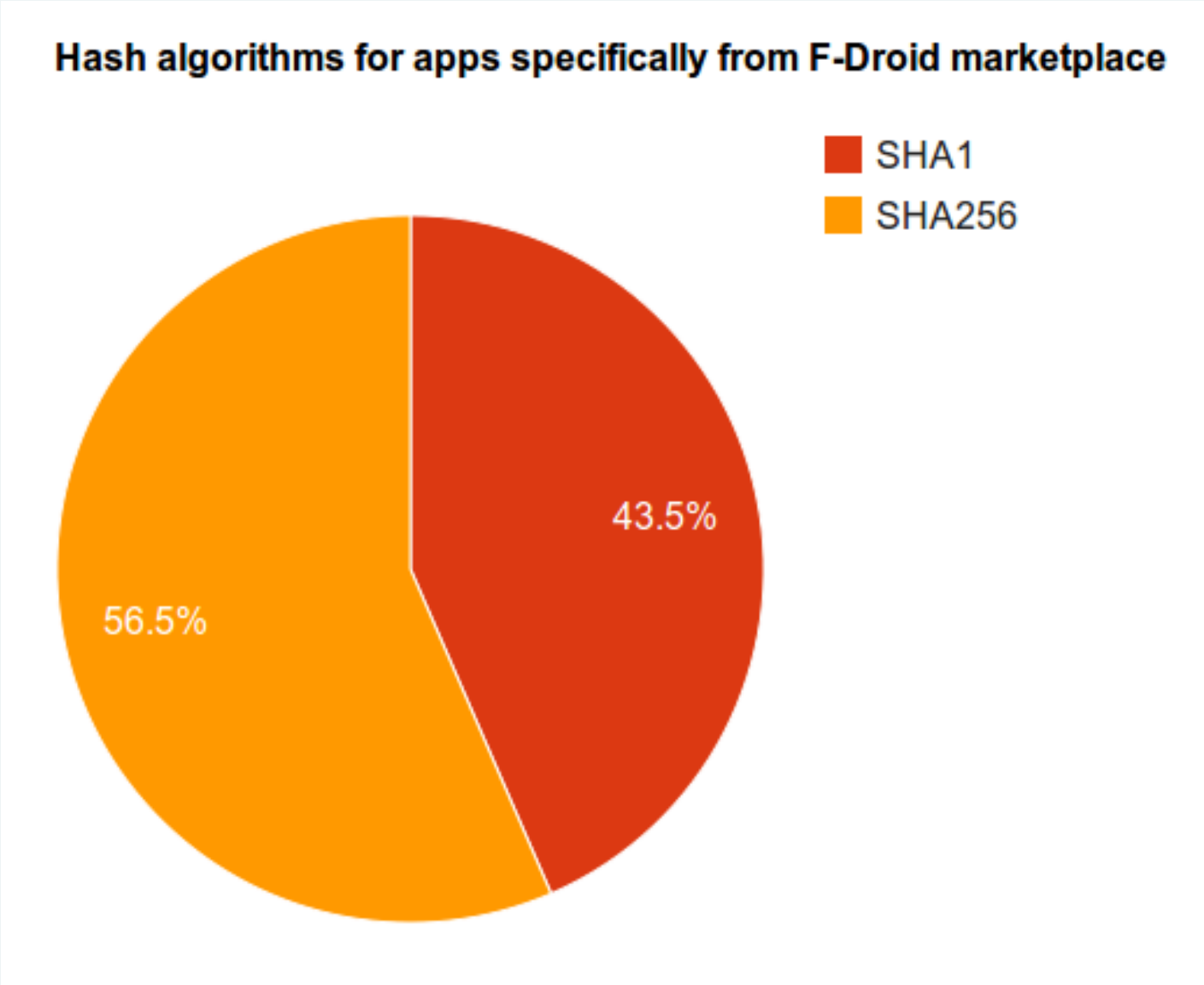
Hash algorithms of app certificates





- malware: 617,942
- clean: 13,110
- Are malware authors more tech-savvy than regular developers?

Sidenote: F-Droid developers even more tech-savvy?



Use of exploits is not widespread

Detectors

Specific root exploits (Rage in the Cage, Levitator, Zerg Rush...)
Generic (and very imperfect) exploit detector

Result 1: my specific root exploit detectors don't work

Rage in the Cage	3
Exploid	4
Levitator	0
Mempodroid	0
Towel Root	0
Zerg Rush	0

Result 2: generic exploit detector works

Detected in **1.6% malware** - I certainly miss cases though
Yet, exploits are not widespread

Rooting is not specific to malware

Property looks for evidence of tools used on rooted devices:

- com.cyanogenmod
- com.noshufou.android.su
- Superuser.apk
- eu.chainfire.supersu

Both clean and malicious apps look for those

$\approx 2\%$

Recap

- Stats computed on ≈ 1 million malware. However, some properties (obfuscation, country...) are difficult to spot accurately.

Recap

- Stats computed on ≈ 1 million malware. However, some properties (obfuscation, country...) are difficult to spot accurately.
- There's a general belief that malware are complicated (assembly, emulator detection, exploits etc). **Statistically, this is wrong.**
  - Rooting is not specific to malware
  - Unix commands, exploits, emulator detection $< 2\%$
  - Malware authors are skilled Android developers
  - They don't like low level dev + Unix

Recap

- Stats computed on ≈ 1 million malware. However, some properties (obfuscation, country...) are difficult to spot accurately.
- There's a general belief that malware are complicated (assembly, emulator detection, exploits etc). **Statistically, this is wrong.**
  - Rooting is not specific to malware
  - Unix commands, exploits, emulator detection $< 2\%$
  - Malware authors are skilled Android developers
  - They don't like low level dev + Unix
- Why implement complex schemes when simple code achieves the goal?
  - Malware focus on their goals: money!
  - They are smaller (why code useless stuff?)

Recap

- Stats computed on $\approx$ 1 million malware. However, some properties (obfuscation, country...) are difficult to spot accurately.
- There's a general belief that malware are complicated (assembly, emulator detection, exploits etc). **Statistically, this is wrong.**
  - Rooting is not specific to malware
  - Unix commands, exploits, emulator detection $< 2\%$
  - Malware authors are skilled Android developers
  - They don't like low level dev + Unix
- Why implement complex schemes when simple code achieves the goal?
  - Malware focus on their goals: money!
  - They are smaller (why code useless stuff?)
- $\approx$ **half malware read or send SMS, grab IMEI**. They retrieve **twice**+ more sensitive data than clean apps

Recap

- Stats computed on ≈ 1 million malware. However, some properties (obfuscation, country...) are difficult to spot accurately.
- There's a general belief that malware are complicated (assembly, emulator detection, exploits etc). **Statistically, this is wrong.**
  - Rooting is not specific to malware
  - Unix commands, exploits, emulator detection $< 2\%$
  - Malware authors are skilled Android developers
  - They don't like low level dev + Unix
- Why implement complex schemes when simple code achieves the goal?
  - Malware focus on their goals: money!
  - They are smaller (why code useless stuff?)
- $\approx$ **half malware read or send SMS, grab IMEI**. They retrieve **twice**+ more sensitive data than clean apps
- Geographic attribution is difficult. Countries like China, Russia, USA, UK, Vietnam, Ukraine are top targets.

Thanks for your attention!

Contact info

@cryptax or `aapvrille` (at) `fortinet` (dot) `com`

Thanks to

.. my husband  Alligator, Lobster...

More

A. Apvrille, L. Apvrille, SherlockDroid: an Inspector for Android Marketplaces, Hack.Lu 2014

M. Lindorfer, M. Neugschwandtner et al ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors, BADGERS 2014

N. Viennot, E. Garcia, J. Nieh, A Measurement Study of Google Play, SIGMETRICS 2014