The Company You Keep: Mobile Malware Infection Rates and Inexpensive Risk Indicators

Hien Thi Thu Truong [ø], Eemil Lagerspetz [ø§], Petteri Nurmi [ø§], Adam J. Oliner [†], Sasu Tarkoma [ø§], N. Asokan [‡ø], Sourav Bhattacharya [ø§]

[ø] University of Helsinki, Gustaf Hällströmin katu 2b, 00560 Helsinki, Finland
[§] Helsinki Institute for Information Technology HIIT, Gustaf Hällströmin katu 2b, 00560 Helsinki, Finland
[‡] Aalto University, Otakaari 4, 02150 Espoo, Finland
[†] University of California at Berkeley, CA 94720, USA

ø first.last@cs.helsinki.fi, ‡ asokan@acm.org, † oliner@eecs.berkeley.edu

ABSTRACT

There is little information from independent sources in the public domain about mobile malware infection rates. The only previous independent estimate (0.0009%) [11], was based on indirect measurements obtained from domain-name resolution traces. In this paper, we present the first independent study of malware infection rates and associated risk factors using data collected directly from over 55,000 Android devices. We find that the malware infection rates in Android devices estimated using two malware datasets (0.28% and 0.26%), though small, are significantly higher than the previous independent estimate.

Based on the hypothesis that some application stores have a greater density of malicious applications and that advertising within applications and cross-promotional deals may act as infection vectors, we investigate whether the set of applications used on a device can serve as an indicator for infection of that device. Our analysis indicates that, while not an accurate indicator of infection by itself, the application set does serve as an inexpensive method for identifying the pool of devices on which more expensive monitoring and analysis mechanisms should be deployed. Using our two malware datasets we show that this indicator performs up to about five times better at identifying infected devices than the baseline of random checks. Such indicators can be used, for example, in the search for new or previously undetected malware. It is therefore a technique that can complement standard malware scanning. Our analysis also demonstrates a marginally significant difference in battery use between infected and clean devices.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Invasive software

WWW'14, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2744-2/14/04.
http://dx.doi.org/10.1145/2566486.2568046.

Keywords

mobile malware; infection rate; Android; malware detection

1. INTRODUCTION

How prevalent is mobile malware? There has been a steady stream of popular news articles and commercial press releases asserting that the mobile malware problem, especially on the Android platform, is dire [10, 5, 15]. For example, a recent press release [15] claims that 32.8 million Android devices were infected in 2012, which, given the estimated 750 million Android devices [17], works out to an infection rate of 4.3%. Lookout mobile reported that the global malware infection rate (which they call "likelihood of encountering application-based mobile threats") for Lookout users is 2.61%, consisting of various types of threats ranging from adware (most prevalent at 1.6%) to spyware (least prevalent at 0.1%) in their latest published estimate (June 2013) [13].

Some reports, however, speculate the opposite—that actual infections in the wild are rare, especially in the West [14]. There are other indications that the malware problem in mobile devices may indeed not be severe. Google Play, and other legitimate application markets, actively use malware scanners to detect and remove malware. Kostiainen *et al.* [9] describe the widespread availability of hardware and software platform security mechanisms on mobile devices, which could make them more robust against malware compared to traditional personal computers.

The research community has focused primarily on analyzing malware to detect if a given software package is malware or not and studying how malware may spread. The only independent research so far to address the question of malware infection rate was by Lever *et al.* [11] which used an indirect method of inferring infection by analyzing domain name resolution queries. They concluded that the infection rate in the United States is less than 0.0009% (c.f. 2.61% and 4.3% above). What accounts for this disparity?

There has been little direct measurement of malware infection rates in the wild by *independent sources*. In order to address this dearth, we carry out a large-scale study by gathering data from tens of thousands of Android devices in the wild. We instrumented Carat [16], a popular Android application, to collect information about the identities of applications run on devices. We used the resulting dataset to look for the presence of malware based on three different Android malware datasets.

Furthermore, we used our dataset to study the likely influence of several potential risk factors. First, hypothesizing that malware may be a battery hog, we compared the distribution of battery lifetime estimates between the sets of infected and clean devices. Second, following the adage "you are the company you keep," we hypothesized that the set of applications used on a device may predict the likelihood of the device being classified as infected in the future. Our intuition behind this hypothesis is the possibility that the set of applications used on a device is indicative of user behavior, such as which application stores they use.

Our intent is to identify a small set of "vulnerable" devices (from among a large population) based on indicators that can be measured inexpensively on the device. More extensive monitoring and analysis techniques can then be deployed on that subset of devices. Such an approach can complement and support anti-malware tools in several ways. First, since our technique focuses on estimating the infection susceptibility of a device (rather than on classifying whether a given package is malware), it may help in the search for *previously undetected malware* by allowing anti-malware vendors to focus on the applications present on the small set of vulnerable devices. Second, it can provide an early warning to enterprise administrators, especially in those enterprises with a Bring Your Own Device policy to identify vulnerable users for whom they can provide additional help or training.

The contributions of this paper are:

- **The first independent, directly measured estimate of mobile malware infection rate**. Taking a conservative approach to identifying malware, we used two malware datasets to find infection rates of 0.28% and 0.26%, which is significantly more than the previous independent estimate [11] (Section 4).
- **A lightweight technique to detect susceptibility of a device to infection**. We propose a new approach to quantify susceptibility of a device to malware infection based only on a set of identifiers representing the applications used on a device. Compared with random selection, our method—which requires only lightweight instrumentation—shows a five-fold improvement (Section 5.2).

We begin by discussing the background of our data collection methodology (Section 2), and the datasets we use (Section 3). We then present infection rates inferred from the datasets (Section 4) before analyzing potential infection indicators and explaining the method for building detection models (Section 5). We then discuss the big picture (Section 6) and provide an account of related work (Section 7), before concluding (Section 8).

2. BACKGROUND

2.1 Identifying Android Packages

An Android package is identified by a Java language-style package name that is intended to be globally unique. To ensure uniqueness, the recommended naming scheme is for developers to base the package name on an Internet domain name that they own. An example of such a package name is `com.facebook.katana`, which is the official Facebook application. An Android device enforces uniqueness of package names within it. Google Play, the official Android application market, also enforces unique package names across the application market. However, using unique package names is mere convention, and nothing prevents any developer from generating a package with any name. For example, the name `com.facebook.katana` is used in many malware packages. Zhou et al. [29] demonstrated that repackaging popular Android applications with a malicious payload is a favorite technique of Android malware authors. Therefore the package name alone is not a reliable identifier for uniquely identifying an Android package.

All Android packages must be signed. Google recommends that each developer have a long-term key pair for signing all their applications[1]. Developer signing keys are often used with self-signed certificates. The certificate(s) (*devcert*) can be extracted from the certificate files in the META-INF directory within an Android package or by an application from within an Android device from the (misleadingly named) `Signature` field of the `PackageInfo` object via the `PackageManager` class. A package may be signed with multiple keys. We use the terms *package* and *application* interchangeably.

Each Android package also has version information in the form of a numeric "versionCode" (integer) and "versionName" (string, intended to be displayed to users). A developer is required to update the versionCode whenever they release a new version of a package because versionCode is used to decide if a package on a device needs to be updated. Therefore instead of just the package name we can use the combination of the package name, devcert, and versionCode as a reliable unique identifier for Android packages. We use the tuple <dc,p,v> to identify a package, where:

- **dc**: a statistically unique ID for the developer (devcert), generated as a SHA1 hash of devcert;
- **p**: the Android package name, extracted from `AndroidManifest.xml`, or from the system process list on the device;
- **v**: the versionCode the package, also obtained from `AndroidManifest.xml`.

2.2 Carat Application

We collected data using a modified version of Carat [16], a mobile application that runs on stock Android and iOS devices. Carat uses energy-efficient, non-invasive instrumentation to intermittently record the state of the device, including the battery level and process list, and uses this information to recommend actions that help users improve the device's battery life. The application is deployed to over 650,000 devices (41% Android) and is publicly available from Google's Play Store and Apple's App Store. In this paper, we discuss data from the Android version.

Carat records several pieces of information when the device battery level changes (sampling is triggered by the `BATTERY_CHANGED` Intent), including:

- Process list with package names, human-readable names, application type (system application or not), its priority and process id;

[1]`http://developer.android.com/tools/publishing/app-signing.html`

- Network type (Wi-Fi, cellular, or disconnected)
- Battery information (level, health, voltage, temperature and status)

Carat sends the collected data to a collaborative analysis backend running on a cloud-based cluster of Amazon EC2 instances. The analysis identifies applications that use anomalously large amounts of energy, either relative to other applications or to other instances of the same application on other devices, and reports any discovered anomalies back to the affected devices.

We chose Carat because of its high visibility, large user base, and the willingness of the Carat development team to let us instrument Carat as needed. Carat has an international user base, representative of current mobile device users. Roughly 36% of Carat users are based in North America, 24% in Asia, 23% in Europe, and 10% in the Middle East. Section 3.1 explains how we changed the application to help identify potential infection.

3. DATASETS

3.1 Carat Dataset

We modified the open-source[2] Carat application to record the unique developer IDs **dc**, as described in Section 2.1. The Carat development team published our modified version of Carat on March 11, 2013 and provided us with data collected from that date until October 15, 2013. There are 55,278 Android devices that were updated to the new version during the data collection period and reported package identification information.

Each device running Carat is identified by a Carat ID which is a statistically unique, Carat-specific device identifier, computed by applying SHA-1 to a concatenation of available device identifiers (e.g., IMEI and WiFi MAC address) and the Carat installation time. When Carat takes a sample, it walks the process list and generates a record for each running application. Package information is extracted on-device directly from `PackageInfo`. In addition to Carat ID and package identifiers, Carat also records the *translated name* of the package (the human-readable string identifying the application to users), the permissions of the package, and the *timestamp* when the package was recorded by Carat. The additional information is used for a different analysis, described in Section 5. An entry in the Carat dataset, summarized in Table 1, is of the form <**ID, dc, p, v**>. The authors of Carat provide more details about the privacy protection mechanisms used by Carat [16]. Data collection by Carat is subject to the IRB process of UC Berkeley[3]. For privacy reasons, Carat does not collect any personally identifying information about the user of the device. Carat users are informed about and provide consent to the data collected from their devices.

Type	Count
distinct devices	55,278
unique package names	64,916
unique devcerts (dc)	41,809
unique <dc,p> tuples	83,226
unique <dc,p,v> tuples	192,080
total unique records	5,358,819

Table 1: Summary of the Carat dataset.

The changes we made to Carat are to collect **dc** values of packages in addition to the package names (**p** values) already being collected. Since **dc** values carry no additional information about the user, our data collection technique does not have any impact on the privacy of Carat users.

We have made our Carat dataset available for research use[4]. In order to protect the privacy of Carat users, we have made the following changes to the shared dataset:

- Compute a device pseudonym for the published data set by computing a randomized cryptographic hash (SHA-1) of Carat ID using a fixed, secret salt. This will prevent an adversary from correlating a device pseudonym with its Carat ID or any of the other device identifiers that contributed to the computation of the Carat ID.
- Transform the package name (**p**) by computing its SHA-1 hash. This is intended to hide the names of any unpublished package names that may have been present on a Carat device of a developer or tester, while ensuring that the presence of a package in the dataset with a known package name can be verified.

3.2 Malware Datasets

We used malware data from three different sources: the Malware Genome dataset[5] provided by Zhou et al. [29], the Mobile Sandbox dataset[6] provided by Spreitzenbarth *et al.* [23], and the McAfee dataset provided by McAfee[7].

The source of each malware dataset used their own unique set of criteria to decide whether to include an Android package in the dataset. McAfee uses a proprietary classification technique. Using the Mobile Sandbox web interface, anyone can submit a package to Mobile Sandbox for consideration. Mobile Sandbox includes a package in their malware dataset if any one of over 40 anti-virus tools they use flag the package as malware. Malware Genome is a fixed set of known malware samples collected during the period from August 2010 to October 2011 [29].

From each Android package (.apk file) in a dataset, we extract the package identifier in the form of a <dc,p,v> tuple. A malware dataset is therefore a table of records, where each record is a <dc,p,v> tuple. Table 2 summarizes the malware datasets.

Type	Mobile Sandbox	McAfee	Mobile Genome	Union
unique dc	3,879	1,456	136	4,809
unique <dc,p>	13,080	2,979	756	15,084
unique <dc,p,v>	16,743	3,182	1,039	19,094
unique .apk files	96,500	5,935	1,260	103,695

Table 2: Summary of malware datasets.

[2] `http://carat.cs.berkeley.edu/`

[3] `http://cphs.berkeley.edu/`

[4] `http://se-sy.org/projects/malware/`

[5] `http://www.malgenomeproject.org/`

[6] `http://mobilesandbox.org/`

[7] `http://mcafee.com`

4. ANALYSIS OF INFECTION RATES

The instrumentation of Carat—intended to be a battery saving app—is necessarily lightweight. In particular, we cannot perform detailed analysis of the applications on the device to decide whether it is infected or not. Given the limited information in the Carat dataset, we assessed infection rates by examining if any of the malware packages (indicated by a malware dataset) match any of the applications in the Carat dataset.

4.1 Finding malware matches in Carat dataset

We consider two types of "match": matching devcert only (<dc>) and matching devcert, package name and version (<dc,p,v>).

<**dc**>: One approach to identifying malware matches is to deem a device in the Carat dataset as infected if it has *any* package signed with respect to a devcert found in a malware dataset. This way, when a **dc** associated with a record in a malware dataset is seen in a Carat dataset record, we flag that Carat record as infected. We proceed to compute the number of unique bad devcerts (N_C), the number of unique packages that correspond to bad devcerts (N_P), and the number of infected devices as a whole ($N_{I:C}$). If the same devcert is used to sign malware as well as benign packages, the <dc> approach may lead to an over-estimate of infection rate. However, it could serve as a way to detect previously undetected malware, as Barrera *et al.* [1] point out.

<**dc,p,v**> **matching**: We can be more conservative by marking an entry in the Carat dataset as infected if and only if every component of the reliable package identifier (Section 2.1) <dc,p,v> of that entry matches a record in the malware dataset. This type of matching will underestimate the infection rate, giving a lower bound. Table 3 summarizes the results of applying these two approaches on the Carat dataset with the three malware datasets, both separately and combined. We can make a number of observations. First, there is a significant disparity in infection rates computed using the two different approaches. Second, the number of infected devices using <dc,p,v> matching is largely disjoint for each malware dataset, leading to the question of whether there is common agreement as to what constitutes malware. In subsections 4.2 and 4.3, we examine these issues in more detail. No <dc,p,v> tuples from the Carat data matched the two-year-old Malware Genome dataset. This suggests that in the long run, malware on Android devices is detected and removed.

4.2 Disparity in <dc> vs. <dc,p,v> Matching

Figure 1 shows the distribution of infected devices using <dc> and <dc,p,v> matching. We now discuss two reasons for the discrepancy between <dc> matching and <dc,p,v> matching.

Reuse of devcerts: The first reason is that some devcerts are used to sign both malware and clean packages. Consider the case of the popular Brightest Flashlight Free application: v17 was flagged as malware by a number of anti–malware vendors, presumably because an ad library it was using was malicious. Subsequent versions are not flagged as malware. All versions, however, are signed with respect to the same devcert[8], which appeared in 2210 devices in our Carat dataset, but never with v17. The Android "same-origin" policy that allows applications signed using the same developer key to share data among themselves discourages (even legitimate) developers from changing their devcerts even if an earlier version of their package was flagged as malware.

A malware developer may also have non-malware packages, and it is known that malware developers are actively seeking to purchase verified developer accounts [10].

Widely available signing keys: The second reason is that some signing keys are widely available (either because they were leaked or because they were test keys). Surprisingly, some legitimate developers use these widely available signing keys to sign their packages. One particular devcert[9] is a widely available "test key". 544 packages in our malware datasets were signed with it. However, 1948 innocuous packages in our Carat dataset (several of them used to be in Google Play) were also signed with the same key. In the Carat dataset, 8188 devices had at least one package signed with it but only 11 devices had a package that matched the full <dc,p,v> of a known malware package.

In all of these cases, the same key is used to sign both malware and non-malware. Consequently the <dc> method of identifying applications results in an overestimate of malware infection rate. While it provides an upper bound of infection rate, the estimate is too high to be useful, marking more than 16% of devices as infected for all datasets, and over 30% for Mobile Sandbox and McAfee. Therefore, in the rest of this paper, we use only <dc,p,v> matching.

Figure 2 shows more details of how devcerts associated with malware are reused. Unsurprisingly, most malware devcerts are used to sign only one package each. However out of a total of 155 devcerts associated with malware (in both Mobile Sandbox and McAfee malware datasets taken together), there are still 13 devcerts signing more than one malware package, and 70 devcerts signing more than one package in general.

Interestingly, we found that `com.android.settings.mt` signed by a devcert[10] controlled by Samsung was flagged as malware by multiple anti-malware vendors[11]. (The same key is widely used by Samsung to sign their packages including some available on Google Play like `com.sec.yosemite.phone`, and `com.airwatch.admin.samsung`). Samsung has since updated the package, which is no longer flagged as malware, but continues to use the *same* the version code (1) for all versions. Consequently, <dc,p,v> matching resulted in devices containing all variants of this package (7845 devices in the Carat dataset) being labeled as infected. We were therefore forced to discount this package from our analysis because we have no way to distinguish between its malware and non-malware variants. We maintain that <dc,p,v> matching is still a reasonable approach for identifying malware because ordinary developers are required to update the version code when they release new versions[12].

[8] `https://androidobservatory.org/cert/27DDACF8860D2857AFC62638C2E5944EA15172D2`

[9] `https://androidobservatory.org/cert/61ED377E85D386A8DFEE6B864BD85B0BFAA5AF81`

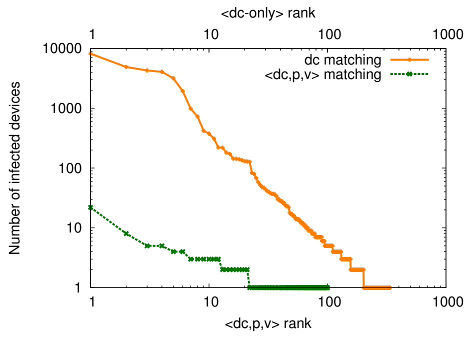
[10] `https://androidobservatory.org/cert/9CA5170F381919DFE0446FCDAB18B19A143B3163`

[11] `http://bit.ly/IFDMyl`

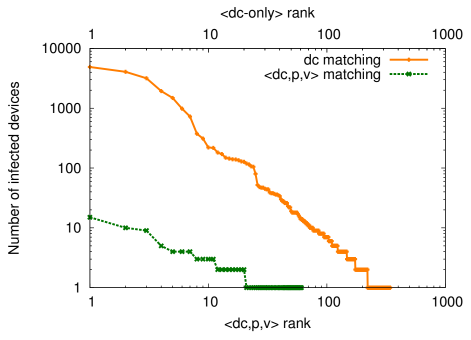
[12] `http://developer.android.com/tools/publishing/versioning.html`

Type / Malware dataset	Mobile Sandbox	%	McAfee	%	Malware Genome	%	Union of all sets	%
N_C: # of <dc> matches (bad devcerts)	337	0.81	343	0.82	8	0.019	534	1.3
N_P: # of packages (<dc,p,v>) matching malware <dc>	10,030	5.2	10,062	5.2	6137	3.2	11,510	6
$N_{C,P,V}$: # of packages matching malware <dc,p,v>	100	0.15	62	0.096	0	0	155	0.24
$N_{I:C}$: # of infected devices (<dc> match)	18,719	34	16,416	30	9240	17	20,182	37
$N_{I:C,P,V}$: # of infected devices (<dc,p,v> match)	154	0.28	144	0.26	0	0	285	0.52

Table 3: Incidence of infection in the Carat dataset.

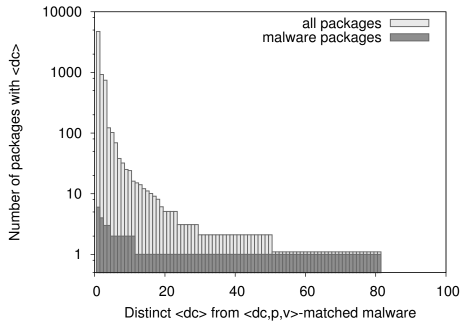


(a) Mobile Sandbox dataset

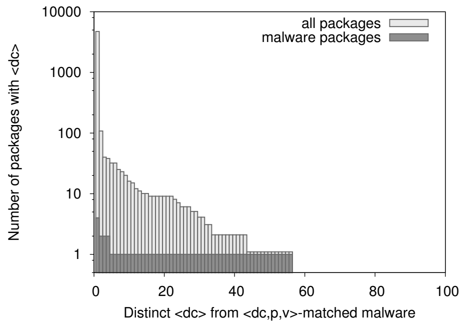


(b) McAfee dataset

Figure 1: The number of infected devices based on <dc> matching is orders of magnitude larger than with <dc,p,v> matching.



(a) Mobile Sandbox dataset



(b) McAfee dataset

Figure 2: Packages signed with respect to a potentially malicious <dc>.

4.3 Disagreement about What is Malware

Figure 3 shows the distribution of malware packages in the three datasets we used. As can be seen, a significant fraction of each dataset contains malware samples included only in that set, leading to the question whether there is any common agreement about what constitutes malware. This issue is illustrated even more dramatically in Table 3, which shows the number of devices labeled as infected according to each individual malware dataset. There were 154 and 144 devices labeled as infected according to the Mobile Sandbox and McAfee datasets, respectively, but only 13 devices were common to these sets. Table 4 lists the most frequent malware packages—those that were detected in five or more devices—in our Carat dataset. Each package was scanned by over 40 different anti-malware tools; *none* of the malware packages that match our Carat dataset is flagged as malware by a majority of anti-malware tools.

All of these observations confirm that there is no wide agreement among anti-malware tools about what constitutes malware. Given the extent of disagreement, we conclude

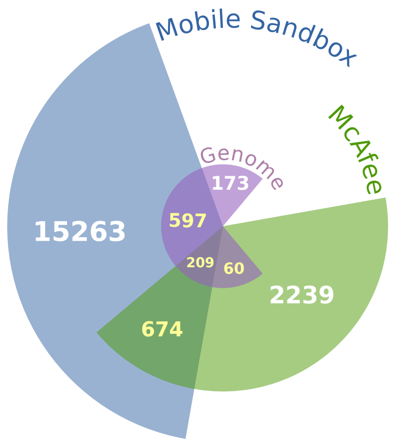


Figure 3: Sizes of the three malware datasets and the extent of overlaps among them.

Package name	Package hash MD5 & SHA1	#inf[a]	#det[b]	Description	Source
it.evilsocket.dsploit	7dedeca3c23462202f86e52dcc004231 07d522ab602352ab27523f92f94bd2ee9b10e40e	23	11	Monitoring	McAfee
com.noshufou.android.su	1e74b778765d2e49caaa06772c68edbc 600621d4c04fb3a699c292cd7aec21676510351d	23	17	Rooting	McAfee
ty.com.android.SmsService	af15196deb350db09eafe3e1cb751f2e 7658d936fd559c3dbe0d30ab669269c5d0e7e512	22	4	Trojan	Mobile Sandbox
com.mixzing.basic	30cf001554ef0c10bc0012e3a75950ba 6dc4477f570d77fdb4adf5db3ee3347a2e9a4b11	15	15	Adware	McAfee
pl.thalion.mobile.battery	2e0a763029636ee3555c59f92bd8bd92 4cf39278dba8453f1f58c28d08a8478bb2f18a30	10	11	Adware	McAfee
com.bslapps1.gbc	e911ba9d519c36eb02b19268ee67de35 f00ab5a4d720fc1489530354cb9fd6d11242a77b	9	7	Adware	McAfee
com.android.antidroidtheft	2d6130aaa0caa1a170fb0ed1c0d687c7 fcfa52c70c29f0b6712cb02ba31c053fbcf102e4	8	3	Monitoring	Mobile Sandbox
com.androidlab.gpsfix	9f6e1d4fad04d866f1635b20b9170368 e1c1661a4278d572adbf7439f30f1dcc1f0d5ea5	7	9	Adware	McAfee
com.adhapps.QesasElanbiaa	3a818a3a8c8da5bebbefdc447f1f782f 7b8d16c362e8ac19ceed6ec0e43f837ee811ac7a	7	15	Adware	McAfee
download.youtube.downloader.pro7	6bad5fa9b87d0a5d7e81994d8e6e4d38 074385ac4938cadc1f93f4a92b811786d2f01ac6	5	6	Adware	Mobile Sandbox
com.android.settings.mt	fa037c0c6dcfe0963d9e243ee6989dc1 c1c72cd10f3714c2fb4b1ca281b5f33192d2d872	5	4	Monitoring	McAfee

[a]Number of devices infected by this package
[b]Number of anti-malware tools flagging this package as malware according to `http://mobilesandbox.org`

Table 4: Most frequent malware.

that it is more appropriate to report infection rates separately for each malware dataset (0.26% for Mobile Sandbox and 0.28% for McAfee) rather than combined (0.51%).

One reason for the disagreement is that some packages are considered "potentially unwanted programs" (PUPs) by some anti-malware tools, but not outright malware. For example, Superuser (`com.noshufou.android.su`), a popular rooting application, cannot be considered malicious if it is installed intentionally by the user but is certainly dangerous if installed without the user's knowledge. Similarly, a WiFi-cracking tool is dangerous but not malicious towards the user. Some anti-malware tools consider applications containing intrusive adware libraries as malware while others do not. Some vendors eschew the term "malware" altogether and resort to identifying threats in different categories separately [13].

On the other hand, the disagreement among anti-malware tools is puzzling because there is evidence that anti-malware vendors use systems that monitor how other vendors rate a package and incorporate this information into their own rating. Sometimes, this can lead to mistakes being propagated widely. We encountered one such example in our dataset in the form of the package `com.android.vending.sectool.v1` signed with respect to a legitimate devcert, controlled by Google[13]. It corresponds to a malware removal tool published by Google almost three years ago[14]. Shortly thereafter, a malware package with the same name emerged [25][15]. Presumably some anti-malware vendor mistakenly labeled both packages as malware which consequently led the some of the rest to follow suit. The legitimate Google-signed package is flagged as malware by no less than 13 anti-malware tools to this day[16].

4.4 Geographic Regions

Mobile malware infection rates in China and Russia are reported to be higher than in the West [12]. Since we do not have demographic information about Carat users, we do not know their location and thus cannot verify this claim directly. However, the translated name (transname) of a package can sometimes indicate the local language of the device. The length of the string is too short for robust automated language detection. However, we can, based on the presence of operator-specific (e.g., `com.vzw.hss.myverizon`, "My Verizon Mobile") and currency-specific (e.g., `com.fdj.euro`, "Euro Millions") package names, estimate the number of infected devices in certain regions like the US, Europe, and Japan. Consequently, we can conjecture that the infection rate in USA is likely to be more than 0.02% based on the fact that 13 infected devices (as per McAfee malware dataset) have USA-specific packages.

5. DETECTING POTENTIAL INFECTION

Mobile device owners, enterprise IT administrators, and vendors of anti-malware tools would all benefit from techniques that can provide early warnings about the susceptibility to malware infection. In this section, we report on the in-depth analyses of our data and look at factors that have a strong influence on the risk of malware infection.

5.1 Energy and Number of Applications

We first considered two factors: the extent of energy consumption on the device, and the number of applications installed on a device. Our dataset exhibits the patterns one might expect: the average battery life of infected devices

[13]`https://www.androidobservatory.org/cert/24BB24C05E47E0AEFA68A58A766179D9B613A600`

[14]`http://googlemobile.blogspot.com/2011/03/update-on-android-market-security.html`

[15]`http://bit.ly/IDjR3W`

[16]`http://bit.ly/IDjWos`

was less than that of clean devices and the average number of installed applications on infected devices was larger than that in clean devices. However, the differences were not always significant.

To examine the relationship between infection and energy consumption, we analyzed the mean battery life between infected devices and the subset of clean devices that match the models and OS versions of infected devices. The mean battery life after outlier removal for the infected devices was 6.56 hours (median 6.06), and 7.88 hours for the clean devices (median 7.5) for Mobile Sandbox. A Wilcoxon rank sum test indicated statistical significance in the lifetime distributions ($Z = -4.2215$, $p < 0.01$). For McAfee, the mean battery life was 7.88 hours for infected (median 7.74) and 8.27 hours for clean (median 7.9). The difference was marginally significant ($Z = -1.199$, $p = 0.23$).

The average number of applications observed on the infected devices (93, median 83 for Mobile Sandbox / 115, median 93 for McAfee) was higher than the average number of applications on the clean devices (88, median 73 for Mobile Sandbox / 90, median 72 for McAfee) during our observation period. This would match with the intuition that every newly installed application is an opportunity for infection and that users who install and run more applications would therefore be more likely to become infected. To assess this hypothesis more rigorously, we used the Wilcoxon rank sum test to compare the number of installed packages between infected and clean devices. The difference was statistically significant for McAfee ($Z = -3.086946$, $p < 0.01$) and marginally significant for Mobile Sandbox ($Z = -1.287166$, $p = 0.1980$). More details of these analyses are found in the full version of this paper [26].

5.2 Applications Used on a Device

The density of malware in different application stores tends to vary considerably, with some stores having a high malware incidence rate [30]. The set of applications used on a device can serve as a (weak) proxy for the application stores used by the user of the device, thus potentially providing information about the device's susceptibility to malware. Cross-application promotions and on-device advertising are other factors that can affect susceptibility to malware infections. As the next step of analysis, we examined how much information about malware infection the applications run on the device can provide.

We conduct our investigation by considering malware detection as a classification task where the goal is to classify the device as infected or clean using the set of applications used on the device as the input feature. As discussed earlier, each anti-malware vendor may have their own proprietary algorithm to classify applications as malware or not. Therefore, we report our detection experiments separately for each malware dataset.

Analogously to, e.g., some spam filters, we rely on (Multinomial) Naïve Bayes classifiers in our experiments. Despite making a strong independence assumption, Naïve Bayes classifier is a powerful and computationally efficient tool for analyzing sparse, large-scale data. As the input features to the classifier we consider bag-of-applications vectors, i.e., sparse binary vectors containing value one for each application run on the device and the value zero otherwise. If a device contains an application that is present within the list of known malware applications, we label the device as infected. Otherwise, the device is labeled as clean.

As we are interested in looking at associations between malware and other applications, and as anti-malware tools would easily detect known malware applications, all applications known as malware were removed from the data before carrying out the classification experiments (they are used only to label devices). Since our data has been collected over a period of multiple months, we also removed all applications that corresponded to earlier (or newer) versions of one of the malware applications (i.e., which have the same devcert and package key, but different version code) to avoid any potential temporal effects.

We start with a simple cross-validation experiment (detecting infection by known malware) and then describe two variants of detecting infection by new malware. Finally we report on a real-life scenario of detecting infection by previously undetected malware. In each experiment, the baseline for detection is the success rate of finding an infected device by randomly selecting a subset of supposedly clean devices.

5.2.1 *Cross-Validation*

We used stratified 5-fold cross-validation. Accordingly, the set of devices was partitioned into five subsets, each having a similar class distribution as the entire dataset. Four of the subsets were used to train the classifier and the remaining subset was used for testing. This process was repeated five times so that each subset served once as the test set. Classification results were aggregated over the five folds. The results are shown in Table 5 (lines 1-2).

The results clearly indicate the difficulty in accurately distinguishing malware infections solely on the basis of applications run on the device. Only a small number of the actually infected devices were correctly identified, however, the classification algorithm also made relatively few false detections (approximately 1000 devices out of over $55,000$). The precision is significantly better than baseline (1.37% and 1.07%), random chance (it is also comparable to what some anti-virus tools provide for these applications [29]). More importantly, considering the relatively low level of false positives, the results indicate that applications run on the device could potentially be used as one of the features for detecting likelihood of malware infection.

5.2.2 *Infection by New Malware*

Next we evaluate the potential of using the set of applications used on a device as an indicator for infection by *new, previously unknown* malware. To do this, we partitioned the set of clean devices into a training set consisting of 80% and a test set consisting of 20%, chosen randomly. We partitioned the malware five ways, according to the number of infected devices. These groups therefore correspond to roughly equal number of infected devices.

In each run (of a total of five runs), four groups were used as "known malware" and the devices they infected were combined with the training set (80% clean devices). The remaining group was used as "unknown malware" and its infected devices combined with the test set (20% clean devices).

No two devices infected by the same malware application appear in both (training and test) sets in the same run to ensure that malware applications for testing are truly unknown. In line with our other experiment, we also re-

Experiments	TP	FN	FP	TN	Precision	Baseline	Gain
1. (§5.2.1) Mobile Sandbox, Cross-validation	14	140	1008	54116	1.37%	0.28%	3.8 times
2. (§5.2.1) McAfee, Cross-validation	11	133	1020	54114	1.07%	0.26%	4.1 times
3. (§5.2.2) Mobile Sandbox, new malware	53	687	5100	267450	1.03%	0.05%	3.8 times
4. (§5.2.2) McAfee, new malware	40	680	5146	267654	0.77%	0.26%	2.9 times
5. (§5.2.3) Mobile Sandbox, undetected malware	5	132	5098	270572	0.10%	0.05%	2.0 times
6. (§5.2.3) McAfee, undetected malware	7	120	5044	270721	0.14%	0.05%	3.0 times
7. (§5.2.4) Mobile Sandbox, real-life set	4	71	609	54515	0.65%	0.14%	4.8 times
8. (§5.2.4) McAfee, real-life set	4	112	407	54727	0.97%	0.21%	4.6 times

Table 5: Detection of infection based on the set of applications used on a device. TP – True Positive, FN – False Negative, FP – False Positive, TN – True Negative.

moved all application features corresponding to newer or older versions of a malware application. Lastly, to mitigate the influence of random partitioning, we repeated the entire experiment five times and report the summed results (the aggregation was done over 25 runs). Results are reported in Table 5 (lines 3-4).

The classification results for detecting infection by new malware are 3.8 (Mobile Sandbox) and 2.9 (McAfee) times better than baseline.

In the experiment, all the devices that had an application from the set of "undetected malware" were always assigned to the test set. While this is reasonable for truly new, unknown malware, in reality, it is possible that undetected malware was present on some devices in the set of devices used to train the model (naturally those devices would have been classified as "clean" at the time of training). We next investigate the potential for detecting infection by previously undetected malware.

5.2.3 *Infection by Previously Undetected Malware*

Infection by previously undetected malware (including new as well as old but previously undetected malware) may also be detectable using the same classification approach. To evaluate this possibility, we ran a new experiment. We partitioned the set of *all* devices randomly into two sets: a training set containing 80% of the devices and a test set containing the remaining 20%. Next, we partitioned the set of malware applications five ways according to the number of infected devices. In each run, only the malware applications from four malware sets (i.e., "known malware" sets) were used to label "infected" devices in the training set. Any device in the training set that contains an application from the remaining malware set (i.e., "undetected malware") was labeled as "clean" to reflect the fact that at the time of training such devices were not known to be infected. We moved any device in the test set that is infected by "known malware" to the training set. To minimize the effect of random partitioning, as before, we repeated the entire experiment five times, with five runs in each round, and report the summed results of 25 runs in Table 5 (lines 5-6). The results are 2 (Mobile Sandbox) and 3 (McAfee) times better than the baseline.

5.2.4 *Detection of 'Real life" Infections*

So far, we simulated "new" or "previously undetected" malware by dividing our malware datasets randomly. For each of our malware datasets we received a first version (the "original" set) in March 2013 and an updated version (the "updated" set) in subsequently (in September 2013 for Mobile Sandbox and November 2013 for McAfee). This allowed us to validate our model for detection of infection by previously undetected malware under "real life" conditions. Let us denote the difference between the updated set and the original set as the "new" set. We labeled the full set of devices using the original malware set, trained our model using the resulting set and used the set of all "clean" devices (with respect to the original set) as the test set to detect infection. We compared the results with respect to infections labeled by the "new" set. The results are shown in Table 5 (lines 7-8). The detection performance is 4.8 (Mobile Sandbox) and 4.6 (McAfee) times better than the baseline of random selection.

6. DISCUSSION

Infection rates: The infection rates we find are nowhere as high as the more alarmist claims from some in the anti-malware industry [15]. But even our most conservative estimates (0.28% and 0.26%) are still (statistically) significantly[17] higher than what was reported by Lever et al. [11]. There are a number of possible explanations:

- Lever et al. looked at devices in the United States only. It is believed that the prevalence of mobile malware infection is much higher in China and Russia. We cannot be certain about how many of the infected devices we found are located in those regions. However, as we discussed in Section 4.4, we can estimate a lower bound of at least 13 infected devices (0.02%) as being in the United States.

- Lever et al. identified infected devices by examining device DNS requests for tainted hosts. Not all malware generates DNS requests (e.g., malware that aims to send premium SMS messages may not bother with DNS requests). Malware authors presumably change their data collection and command and control addresses frequently or may have used hardwired IP addresses. Both of these factors may have led to an underestimation of the infection rate.

Our conservative malware infection rate estimates are closer to Google's recent estimate, 0.12% [19]. However, Google's estimate refers to the percentage of application installations that they have marked as potentially harmful, while ours is the percentage of infected devices in a community of mostly clean devices. Also, Google affects the result by warning

[17]A two-sample proportions test (Chi-squared) verified that the difference is statistically highly significant (Mobile Sandbox: χ^2=44132.24, df=1, p<0.001, and McAfee: χ^2=38657.59, df=1, p<0.001.)

the user at installation time, while Carat collects the information on running applications after installation. Further, installation of an application does not guarantee that it will ever be run.

Detecting infection: The results of the classification experiments in Section 5.2 demonstrate that the set of applications used on the device is a potential source of information for detecting malware applications. The detection techniques discussed in Section 5.2, are not intended to replace the standard anti-malware scanning for infection by *known* malware. Rather they can complement standard anti-malware tools in a number of ways. We foresee two ways in which early warning could be used:

- *Search for previously undetected malware*: An anti-malware vendor can, after doing the standard scanning for malware using known malware signatures, apply our detection technique to the list of applications used on a device to determine if the device falls in the "vulnerable" class and inform the vendor of the tool if it does. The vendor can then apply expensive analysis techniques on all the applications in vulnerable devices. Without such a detection mechanism, the vendor would have had to take a random sample of devices.

- *Training enterprise users*: Consider an enterprise that has a small budget for training users on good application hygiene. Rather than selecting trainees at random, the enterprise administrators could target the training towards users of the "vulnerable" class of devices.

For example, in the case shown in Table 5 (line 8), the success rate for finding infection by previously unknown malware by taking a random sample of devices is 0.21%. Our detection mechanism results in an almost five-fold improvement (precision=0.97%). It is important to note that this improvement comes at virtually no cost because the instrumentation needed to collect the data is extremely lightweight. Our detections were based only on the set of applications used on a device, where each application is identified only by the <dc,p,v> tuple. Measuring this set of active applications intermittently using Carat incurs negligible performance or battery overhead (literally below the precision of our hardware instrumentation).

The effectiveness of the detections could be improved with more data, and with additional features, such as battery consumption and the amount and extent of permissions required by the application. More effective detection techniques can lead to other early warning applications. For example, Carat or a standalone mobile security application on the device might visualize the detection as a traffic light indicator of "threat level".

Predicting expected time before infection: Our approach is well–suited to answering the question of the *expected time before infection*. This is an interesting and important question and a solution would provide a much needed temporal risk measure for end users. In this section, we outline our simple solution for this problem that will be verified in our future work. In order to fully develop and deploy a solution, we need the application specific time-to-infection distributions from devices. We expect to see more such cases as we continue the data collection over the next few months. Once we have enough such cases, how can we predict the expected time to infection?

Our proposed solution is based on the application–specific time-to-infection distributions. For a given application and device, we record the time the application has been installed on a clean device until the device became infected. This time is zero for applications installed on already infected devices and infinite for clean devices. Thus we obtain per application distributions of the time-to-infection.

Based on the application specific time-to-infection distributions we then determine the device specific time-to-infection. This is performed by considering each application on the given clean device and then determining the expected minimum time-to-infection based on the application specific distributions. We believe that the minimum time is a reasonable starting point for assessing the risk to the user.

The three important parts of the process are the summary statistic for the application specific distributions, application correlations, and the determination of the expected minimum time.

Our initial solution uses the median to summarize the application specific distributions. The median is a robust statistic and it copes well with outliers. We take correlations between applications into account by considering also subsets of applications. The motivation is that the summary statistic may hide interesting patterns pertaining to certain groups of applications that together significantly reduce the time-to-infection. A group is included in the analysis if its time to infection value is lower than the respective values of its elements. Given the application and group statistics, it is then easy to choose the minimum value.

Energy consumption: We found a marginally significant difference between infected and clean devices in terms of remaining battery life. Our results show that malware reduces the remaining operating time of the devices by an average of 1.3 hours (Mobile Sandbox) and 0.4 hours (McAfee). We are incentivized to detect and remove malware in order to conserve energy. In addition, we can use this observation to detect malware. The combination of installed and running applications with the energy consumption data make for a new way to assess the risk of having malware. Crowdsourcing of malware detection has been proposed before [2], however, our approach is unique because it only uses knowledge of the installed, running applications and the energy consumption, making it non-intrusive and lightweight. The combination of these two features—applications and energy—appears to be a promising avenue for future research.

Reliable malware detection: A typical anti-virus tool performs extensive analysis of a package on a device in order to determine if it is malware. We did not have this luxury because we wanted to minimize the overhead we add to Carat. Consequently, we resorted to identifying malware by comparing reliable identifiers for packages (e.g., <dc,p,v> tuples) with those in known malware datasets. This may underestimate the incidence of malware in the sense that it will not detect any previously unknown malware, just as any other signature-based malware detection mechanism. Despite this limitation, our results provide new, and more accurate, information about malware infection rates which suggests that mobile malware infection may be more widespread than previous rigorous independent estimates.

Our detection techniques are independent of the method used to decide if a device is infected or not. Consequently, their efficacy will be improved when they are used together with better techniques for identifying malware.

Limitations: In our Carat dataset, we did not make use of the time information associated with Carat samples. It is possible that users infected by malware go on to install anti-malware or other performance analysis tool in order to troubleshoot. Since our analysis technique is based on applications occurring together on infected and clean devices, it may incorrectly infer that the presence of such applications is an indicator of potential infection. Removing such applications from the analysis would be one way to address this problem. For privacy reasons, Carat does not collect any demographic data about its users. Consequently, we cannot be certain that the Carat dataset corresponds to a representative sample of Android users in general besides the geographical distribution information we presented in Section 2.2.

7. RELATED WORK

Work by Lever et al. [11] was the first (and until now, to the best of our knowledge, the only) public, independent study of mobile malware infection rates. They used large datasets consisting of DNS requests made by the customers of a US-based Internet service provider and a cellular carrier and tried to identify infected mobile devices based on DNS requests to known tainted hostnames. Our work, in contrast, uses data collected directly from the mobile devices.

Analyzing mobile malware has been an active research area. Felt et al. [6] presented one of the first surveys of malware on three different mobile platforms. Zhou and Jiang [29] provide a detailed, systematic analysis and classification of Android malware based on a large set of malware samples. Some researchers have explored collaborative techniques for malware detection [2, 28]. Our work differs from these in that rather than detecting malware as such, we use data collected from a large number of devices to quantify the susceptibility of infection for a given device.

This paper proposes using proxy signals, like energy use or the set of applications run on a device, to detect or predict infection. That kind of approach bears a resemblance to anomaly detection, especially in the intrusion detection field, which has a rich history [20, 22]. Kim et al. [8] proposed a power-aware malware detection framework for mobile devices targeted for detecting battery exhaustion malware. One paper suggests that energy is an insufficient metric for detecting malware [7], but energy issues have been successfully attributed to buggy and malicious applications [18, 16].

Motivated by the prohibitive cost of doing comprehensive malware detection on mobile phones, Portokalidis et al. proposed Paranoid Android [21] which maintains exact virtual replicas of mobile devices on a server where the expensive malware analysis is done. Maintaining exact replicas may be considered privacy-invasive [3]. Also, analyzing application permissions has been done [4], but also safe applications often request extensive permissions, making application permissions an insufficient indicator of malware.

Our approach of using lightweight instrumentation to identify potentially vulnerable devices can help reduce the impact of the privacy concern. Our work uses data collected from a large number of clients (sometimes called a *community*) to build statistical models. Much of the academic research in applying statistical analysis and machine learning to the problem of mobile malware takes a *software-centric* view, focusing on analyzing software packages to determine if they are malware. In contrast, we take a *device-centric* view, attempting to estimate the propensity of a device for infection. The closest prior work in this aspect was by Wagner et al. [27] and subsequently Sumber and Wald [24] who took a similar approach in the context of Twitter. They use publicly visible characteristics of Twitter users to predict their susceptibility to fall victim to social bots.

Finally, some recent activity focuses on collecting and collating mobile applications (both malware and otherwise) and making them available publicly in a systematic manner with well-designed interfaces [1, 23]. These have been extremely useful in our work.

8. CONCLUSION

In this paper, we addressed a gap in the research literature regarding malware infection rates on mobile devices by direct measurement on tens of thousands of mobile devices. Our estimates for infection rate of mobile malware, although small (0.26% for Mobile Sandbox and 0.28% for McAfee), is still higher than previous estimates.

We also investigated whether we can build models that can detect malware infection based on the set of applications currently installed on a device. Although the precision and recall of this initial detection attempt are not high, the approach can still constitute one line of defense in a suite of techniques, especially given that the data collection needed for the detection is extremely lightweight. In particular, our models can be used by enterprise IT administrators and anti-malware vendors to identify a small pool of vulnerable devices, e.g., to deploy more expensive analysis techniques on them or to provide training to their users. In our experiments, the precision of the model is up to five times better than the baseline of random selection of devices.

There are several interesting directions to continue the work, including the following:

- *Using a larger dataset*: Repeating and improving our analysis using the larger Carat dataset.
- *Improving detection accuracy*: Using additional features and/or experimenting with better detection techniques to improve the precision and recall.
- *Predicting expected time to infection*: Validating our proposed approach for predicting expected time to infection using a larger dataset from Carat.
- *Energy vs. infection*: Investigating whether malware infection has an impact on the expected battery life and whether another proxy for energy use could be predictive or indicative of infection.

An extended version of this paper is available [26].

Acknowledgements

We thank Igor Muttik, Michael Spreitzenbarth, Xuxian Jiang, and Yajin Zhou for giving us their respective malware datasets. In particular, Michael and Igor responded promptly to several e-mail requests for clarifications additional information. We thank the anonymous reviewers for their valuable feedback. The work of Truong and Bhattacharya was supported by the Intel Institute for Collaborative Research in Secure Computing (ICRI-SC).

9. REFERENCES

[1] D. Barrera, J. Clark, D. McCarney, and P. C. van Oorschot. Understanding and improving app installation security mechanisms through empirical analysis of android. In *Proc. the second ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM'12, pages 81–92. ACM, 2012.

[2] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proc. the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.

[3] M. Chandramohan and H. B. K. Tan. Detection of mobile malware in the wild. *Computer*, 45(9):65–71, 2012.

[4] P. H. Chia, Y. Yamamoto, and N. Asokan. Is this app safe?: a large scale study on application permissions and risk signals. In *Proc. the 21st international conference on World Wide Web*, WWW '12, pages 311–320. ACM, 2012.

[5] Damballa Labs. Damballa threat report – first half 2011. Technical report, 2011. `https://www.damballa.com/downloads/r_pubs/Damballa_Threat_Report-First_Half_2011.pdf`.

[6] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proc. the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM'11, pages 3–14. ACM, 2011.

[7] J. Hoffmann, S. Neumann, and T. Holz. Mobile malware detection based on energy fingerprints - a dead end? In *RAID*, 2013.

[8] H. Kim, J. Smith, and K. G. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proc. the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 239–252. ACM, 2008.

[9] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan. Old, new, borrowed, blue: a perspective on the evolution of mobile platform security architectures. In *First ACM Conference on Data and Application Security and Privacy*, pages 13–24. ACM, 2011.

[10] B. Krebs. Mobile Malcoders Pay to Google Play, Mar. 2013. `http://krebsonsecurity.com/2013/03/mobile-malcoders-pay-to-google-play/`.

[11] C. Lever, M. Antonakakis, B. Reeves, P. Traynor, and W. Lee. The core of the matter: Analyzing malicious traffic in cellular carriers. In *Proc. the 2013 Network and Distributed Systems Security Conference (NDSS 2013)*. Internet Society, 2013.

[12] Lookout Mobile. 2013 mobile threat predictions, Dec 2012. `https://blog.lookout.com/blog/2012/12/13/2013-mobile-threat-predictions/`.

[13] Lookout Mobile. Lookout tours the current world of mobile threats. Lookout blog, June 2013. `https://blog.lookout.com/blog/2013/06/05/world-current-of-mobile-threats/`.

[14] R. McGarvey. Threat of the week: Mobile malware, menace or myth? *CreditUnion Times*, Apr. 2013.

[15] NQMobile. Mobile malware up 163% in 2012, getting even smarter in 2013. *PRNEwsWire*, 2013. `http://ir.nq.com/phoenix.zhtml?c=243152&p=irol-newsArticle&id=1806588`.

[16] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. In *Proc. 11th ACM Conference on Embedded Networked Sensor Systems*, Nov 2013.

[17] L. Page. Update from the CEO, Mar. 2013. `http://googleblog.blogspot.fi/2013/03/update-from-ceo.html`.

[18] A. Pathak, A. Jindal, Y. C. Hu, and S. Midkiff. What is keeping my phone awake? Characterizing and detecting no-sleep energy bugs in smartphone apps. In *Mobisys*, 2012.

[19] S. M. Patterson. Contrary to what you've heard, Android is almost impenetrable to malware, Oct 2013. `http://qz.com/131436/contrary-to-what-youve-heard-android-is-almost-impenetrable-to-malware/`.

[20] V. Paxson. Bro: a system for detecting network intruders in real-time. In *Computer Networks*, volume 31, 1999.

[21] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid android: versatile protection for smartphones. In *Proc. the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 347–356, New York, NY, USA, 2010. ACM.

[22] M. M. Sebring and R. A. Whitehurst. Expert systems in intrusion detection: a case study. In *National Computer Security Conference*, 1988.

[23] M. Spreitzenbarth, F. Echtler, T. Schrek, F. C. Freiling, and J. Hoffman. MobileSandbox: looking deeper into android applications. In *Proc. the 28th ACM Symposium on Applied Computing (SAC)*, 2013.

[24] C. Sumner and R. Wald. Predicting susceptibility to social bots on twitter. BlackHat US presentation. `https://media.blackhat.com/us-13/US-13-Sumner-Predicting-Susceptibility-to-Social-Bots-on-Twitter-Slides.pdf`.

[25] Trend Labs. Trojanized security tool serves as backdoor app, Mar 2011. `http://blog.trendmicro.com/trendlabs-security-intelligence/trojanized-security-tool-serves-as-backdoor-app/`.

[26] H. T. T. Truong, E. Lagerspetz, P. Nurmi, A. J. Oliner, S. Tarkoma, N. Asokan, and S. Bhattacharya. The Company You Keep: Mobile Malware Infection Rates and Inexpensive Risk Indicators. *CoRR*, abs/1312.3245, 2013. `http://arxiv.org/abs/1312.3245`.

[27] C. Wagner, S. Mitter, C. Körner, and M. Strohmaier. When social bots attack: Modeling susceptibility of users in online social networks. In *2nd workshop on Making Sense of Microposts at WWW2012*, 2012.

[28] L. Yang, V. Ganapathy, and L. Iftode. Enhancing mobile malware detection with social collaboration. In *Privacy, Security, Risk and Trust (PASSAT), IEEE Third International Conference on Social Computing (SocialCom)*, pages 572–576, 2011.

[29] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy (SP)*, pages 95–109, 2012.

[30] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proc. the 2012 Network and Distributed Systems Security Conference (NDSS 2012)*, Feb. 2012.