2014 IEEE 38th Annual International Computers, Software and Applications Conference

A Forensic Analysis of Android Malware
How is Malware Written and How it Could be Detected?

Kevin Allix, Quentin Jerome, Tegawendé F. Bissyandé, Jacques Klein, Radu State and Yves Le Traon
Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
Luxembourg
{*firstname.lastname*}*@uni.lu*

***Abstract*—We consider in this paper the analysis of a large set of malware and benign applications from the Android ecosystem. Although a large body of research work has dealt with Android malware over the last years, none has addressed it from a forensic point of view.**

After collecting over 500 000 applications from user markets and research repositories, we perform an analysis that yields precious insights on the writing process of Android malware. This study also explores some strange artifacts in the datasets, and the divergent capabilities of state-of-the-art antivirus to recognize/define malware. We further highlight some major weak usage and misunderstanding of Android security by the criminal community and show some patterns in their operational flow. Finally, using insights from this analysis, we build a naive malware detection scheme that could complement existing anti virus software.

***Keywords*-Android Security, Digital Forensics, Malware Analysis, Malware development**

I. Introduction

Android has progressively grown to become in a few years the most widely used smartphone operating system [6]. With more and more users relying on Android-enabled handheld device, and able to install third party applications from official and alternative markets, the security of both devices and the underlying network becomes an essential concern for both the end user and his service provider. In recent years, practitioners and researchers have witnessed the emergence of a variety of Android malware. The associated threats range from simple user tracking and disclosure of personal information to advanced fraud and premium-rate SMS services subscription, or even unwarranted involvement in botnets. Although most users are nowadays aware that personal computers can and will be attacked by malware, very few realize that their smartphone is prone to an equally dangerous threat.

To assess the threat of software downloaded from the internet, discerning users rely on scan results yielded by antivirus products. Unfortunately, each antivirus vendor has its secret recipe on how/why it decides to assign a malware label to a given application. Thus, an application can be differently appreciated by distinct antivirus products, leading to damaging confusions. Indeed, both practitioners and researchers heavily rely on antivirus, whether to trust apps or to build the ground truths for assessment tasks.

For the purpose of our study, we have collected a large and up-to-date dataset of hundreds of thousands of Android applications from markets and repositories. We have then scanned each of these applications using about 45 antivirus products generously hosted by VirusTotal to assess whether they are labelled as malware or not. This effort was made to obtain a clear view of the business of malware writing and some insights in the evolution of malware and its detection by antivirus products. We also take this opportunity to investigate, indirectly, how skilled malware creators are.

Several research studies have investigated Android malware [1], [8], [23], [24]. Most of these academic works are however about using advanced code analysis and data mining techniques to study applications. Thus, there are scarce reports on the actual artefacts that a typical incident responder would rely on in practice. Our study aims at filling this gap by performing such an analysis and reporting our findings based on a large dataset. The main contributions of this paper are:

- We extensively provide evidence that malware labelling is not a precise science. Applications are flagged or not depending on the antivirus product;
- We show that most malware writers basically copy/paste code from fellow developer code and from public tutorials/samples from the Web;
- We find that malware writing is almost a regular business, with work cycles following a similar 5 working days per week;
- We also highlight that almost all malware writers are incapable to properly use digital certificates;
- Finally, we propose roadmaps for basic detection of malware that have not yet been detected by antivirus products.

The paper is structured as follows: Section II provides detailed information on the construction of our dataset of Android applications, and on the labelling process to categorize benign and malware applications. Section III depicts the main findings of our experimental analysis. We also provide a discussion to guide this analysis. We discuss related work in Section V. Section VI concludes the paper and outlines future work.

0730-3157/14 $31.00 © 2014 IEEE
DOI 10.1109/COMPSAC.2014.61

II. PRELIMINARIES

An investigation into the business of malware requires a significant dataset representing real-world applications. We have built our dataset by collecting applications from *markets*, i.e., the online stores where developers distribute their applications to end-users. Indeed, although Google –the main developer of the Android software stack– operates an official market named *Google Play*, the policy of Google makes it possible for Android users to download and install Apps from any other alternative market.

Alternative markets are often created to distribute specific selection of applications. For example, some of these markets may focus on a specific geographical area, e.g., Russia or China, providing users with Apps in their local languages. Other markets focus exclusively on free software, and at least one market is known to be dedicated to adult content. Users may also directly share Apps, either in close circles, or with application bundles released through BitTorrent. Such apps are often distributed by other users who have paid for them in non-free markets. Finally, we have included in our datasets, apps that have been collected by others to construct research repositories.

In the remainder of this section, we provide details on the different sources of our dataset, on the scanning process that were used to label each application as malware or benign, and on the artifacts that we have extracted from application packages to perform our study.

A. Dataset sources

We have developed specialized crawlers for several market places to automatically browse their content, find Android applications that could be retrieved for free, and download them into our repository. In this step we have found that several market owners took various steps in order to prevent their market to be automatically mined. Thus, for two of such markets, we cannot assure that we have retrieved their whole content. However, to the best of our knowledge, the total number of apps that we have collected constitutes the largest dataset of Android apps ever used in research studies.

Google Play[1]*:* The official market of Android is a web-site that allows users to browse its content through a web browser. Applications cannot however be downloaded through the web browser as any other file would be. Instead, Google provides an Android application[2] that uses a proprietary protocol to communicate with Google Play servers. Furthermore, no application can be downloaded from Google Play without a valid Google account – not even free Apps. Both issues thus outlined were overcome using open-source implementations of the proprietary protocol and by creating free Google accounts. The remaining constraint was *time*, as Google also enforces a strict account-level rate-limiting. Indeed, one given account is not allowed to download more than a given quantity of application in a given time frame.

AppChina[3]*:* This market is by far the largest alternative market of our dataset. At the time of collection, AppChina was enforcing drastic scraping protections such as a 1Mb/s bandwidth limitation and a several-hour ban if using simultaneously more than one connection to the service.

Anzhi[4]*:* The anzhi market is operated from and for Chinese Android user base. It stores and distributes apps that are written in the Chinese languages, and provides a less-strict screening policy than e.g., Google Play.

Slideme[5]*:* Operated from the United States of America. this alternative market is a direct competitor of Google Play: it provides both free and paid Apps for the Android platform.

FreewareLovers[6]*:* A market run by a German company, FreewareLovers provides freeware for every major mobile platform, including Android. A big advantage of FreewareLovers is that it does not require any specific application and can be used with any web browser.

ProAndroid[7]*:* Operated from Russia, ProAndroid market is the smallest market that we crawled. It distributes free Apps only.

F-Droid[8]*:* This repository of Free and open-source software on the Android platform also provides a number of apps that users can download and install on their devices.

1mobile[9]*:* This market proposes free Android apps for direct downloads. It is a very large market that offers users with opportunities of browsing and retrieving thousands of apps.

In addition to market places, we also looked into other distribution channels to collect applications that are shared by bundles.

Torrents: We have collected a small set of apps which were made available through BitTorrent. We note that such applications are usually distributed without their authors' consent, and often include paid Apps. Nevertheless, when considering the number of leeches, we were able to notice that such collections of Android applications appear to attract a significant number of user downloads, increasing the interest for investigating malware distributed in such channels.

Genome[10]*:* Zhou et al. [25] have collected Android malware samples and gave the research community access to their built dataset. This dataset is divided in families, each containing malware that are closely related to each other.

Table I summarizes the number of applications collected from each market used to build our dataset. The largest share of applications are from the official Android market, Google Play. Using the SHA256 hash function on applications, we noticed that several thousands applications are found in more

[1] http://play.google.com (previously known as Google Market) [2] Also named Google Play

[3] http://www.appchina.com [4] http://www.anzhi.com
[5] http://slideme.org [6] http://www.freewarelovers.com
[7] http://proandroid.net [8] http://f-droid.org/ [9] market.1mobile.com
[10] http://www.malgenomeproject.org/

than one market. Hence, the total number of unique apps in Table I is less than the sum of unique applications in each market.

Table I
ORIGIN OF THE ANDROID APPS IN OUR DATASET

Marketplace	# of Android apps	Percentage
Google Play	325 214	54.73%
appchina	125 248	21.13%
anzhi	76 414	12.86%
1mobile	57 506	9.68%
slideme	27 274	4.68%
torrents	5 294	0.89%
freewarelovers	4 145	0.70%
proandroid	3 683	0.62%
fdroid	2 023	0.34%
genome	1 247	0.21%
apk_bang	363	0.06%
Total	**594 000 Unique apps**	

B. Artifacts of study

To perform our study we have mined information from the application packages focusing on two artifact metadata in Android package files.

Packaging dates: An Android application is distributed as an `.apk` file which is actually a ZIP archive containing all the resources an application needs to run, such as the application binary code and images. An interesting side-effect of this package format is that all the files that makes an application go from the developer's computer to end-users' devices without any modification. In particular, all metadata of the files contained in the `.apk` package, such as the last modification date, are preserved.

All bytecode, representing the application binary code, is assembled into a `classes.dex` file that is produced at packaging-time. Thus the last modification date of this file represents the packaging time. In the remainder of this paper, *packaging date* will refer to this date.

Certificate Metadata: In the Android platform, a first security measure was made mandatory to guarantee that the authenticity of each application can be traced back to its creator. Thus, all Android applications must be signed with a cryptographic certificate. Certificates are included in the app package to allow end-users to verify the package's signature. For each application from our dataset, we have collected the certificates and analyzed their attributes, including *owner* and *issuer*, as described by the X.509 standard [18].

C. Malware Labelling

Over the course of several months, while we collect the dataset, we have undertaken to analyze them with anti virus products actually used in the software market. For our study, we have relied on VirusTotal[11], a web portal that hosts about 40 products from renown anti virus vendors, including McAfee®, Symantec® or Avast®. We have sent all applications from our dataset to VirusTotal and collected the scan results for analysis and correlation studies.

D. Test of Statistical Significance

Our forensics analysis is based on a sample of Android applications. Although, to the best of our knowledge, no related study involving Android malware has ever exploited that many applications, there is a need to ensure, for some of our findings, that they are significant. To this end, we resort to the common metric of the Mann-Whitney-Wilcoxon (MWW) test.

The MWW test is a non-parametric statistical hypothesis test that assesses the statistical significance of the difference between the distributions in two datasets [19]. We adopt this test as it does not assume any specific distribution, a suitable property for our experimental setting. Once the Mann-Whitney U value is computed it is used to determine the p-value. Given a significance level $\alpha = 0.001$, if $p - value < \alpha$, then the test rejects the null hypothesis, implying that the two datasets have different distributions at the significance level of $\alpha = 0.001$: there is one chance in a thousand that this is due to a coincidence.

III. ANALYSIS

In this section, we describe and interpret the results of our findings on how malware are written, in comparison with benign applications, and how anti virus products perform in their detection.

A. Malware identification by anti virus products

Malware identification by anti virus products is critical to practitioners and researchers alike. Indeed, anti virus products remain the most trusted means to flag an application as malware. Traditionally, the common detection scheme of anti virus is signature-based. Thus, to identify malware statically, antivirus software compares the contents of application files to their secret dictionary of virus signatures. This approach can be very effective, but can only help identify malware for which samples have already been obtained and associated signatures created. Some antivirus products add heuristics to their process in order to identify new malware or variants of known malware.

In Figure 1, we see that most of our data sources contain Android applications that are flagged as malware by at least 1 anti virus product hosted by VirusTotal. Even Google Play, where each application goes through the Bouncer[12], shows a malware-rate of 22%. These malware are often in the form of adware, i.e., applications that continuously display undesired advertisement during use. Anzhi and AppChina include the largest share of flagged applications. Each of all the malware samples from the Genome dataset are indeed flagged by at least one anti virus software.

[11] http://www.virustotal.com

[12] Google's in-house environment for screening malware

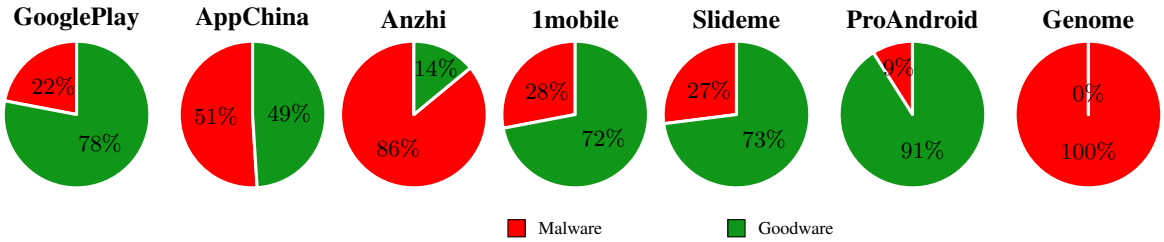

Figure 1. Share of Malware in Datasets: Applications are flagged by at least 1 antivirus product

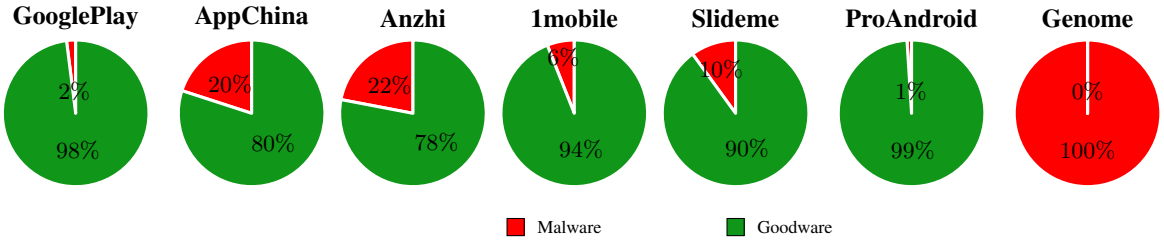

Figure 2. Share of Malware in Datasets: Applications are flagged by at least 10 antivirus products

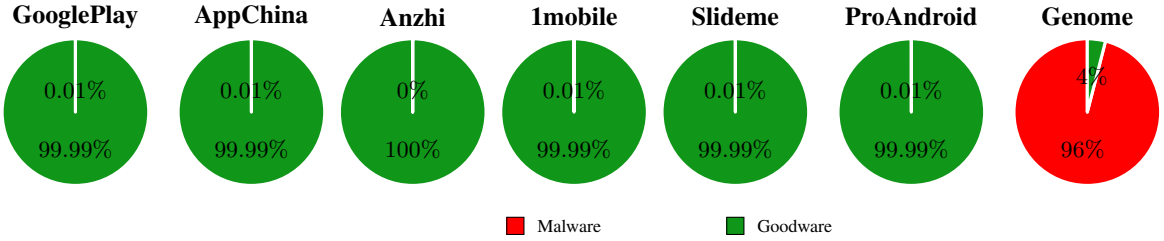

Figure 3. Share of Malware in Datasets: Applications are flagged by at least 26 antivirus products

Malware shares depicted in Figure 2 indicate that anti virus software have divergent scanning results. Indeed, if we require that an application should be tagged as a malware only if at least 10 anti virus products have found it suspicious, then the malware rate drops significantly for all our data sources. Google Play now only contains 2% of malware, while all Genome samples are still identified as true malware.

The Genome dataset being a reliable source of known malware, we change the threshold of anti virus until some of the applications in the dataset are missed in the scanning process. Figure 3 provides the different malware share when at least 26 anti virus, out of more than 40, are required to flag an application before it is considered a malware.

Anti virus software cannot each identify all existing malware. Only a small subset of widely known malware are recognized by a large number of anti virus software.

B. Android Malware Production

We proceed to investigate the production of Android malware to draw insights. The analysis of packaging dates of Android applications yields some distinct patterns. In Figure 4, we plot the packaging date, subdivided by hour, for benign applications and for all applications flagged by at least one anti virus. Despite the potential noise due to the threshold set by each anti virus to tag malware, we note a pattern in the compilation dates: it stands out that there are many more peaks of malware packaging. This suggests that malware often are compiled in batches, while compilation of benign applications are more spread over time.

To further investigate and strengthen the validity of our finding, we consider the samples of confirmed malware from known families exposed in the Genome dataset, and consider all other applications from our datasets as benign. This process is valid when considering a very strict threshold where an application is labelled as malware if at least half, i.e., 22, of the anti virus software from VirusTotal flag it. Figure 5 thus confirms more strongly that Android malware are compiled in batches. The 1258 malware of the genome dataset have been packaged on only 244 different days. 51 malware were packaged on 2011-09-21 alone, representing 16% of all Android apps packaged on this day. Only 72 malware were packaged each alone in a distinct day when

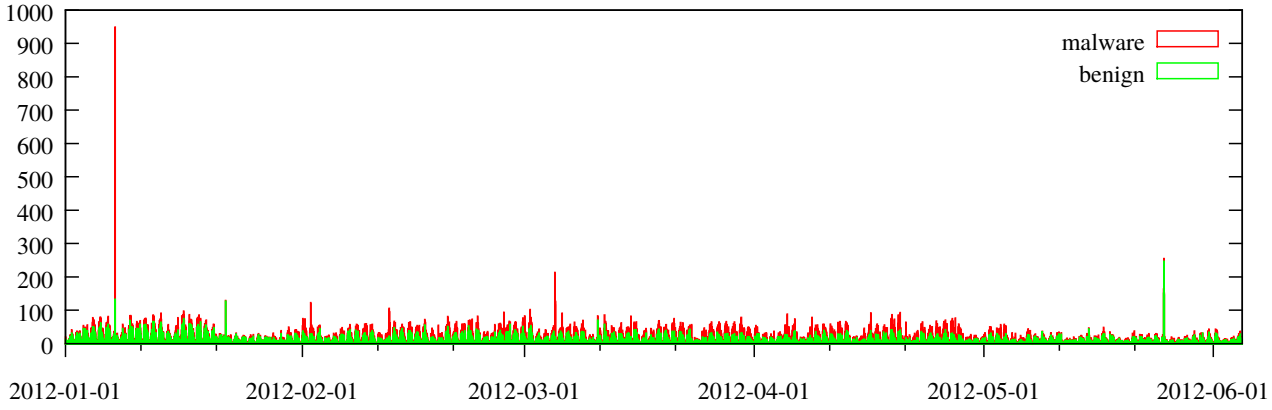

Figure 4. Number of benign and malware packaged between 01 January 2012 and 01 June 2012

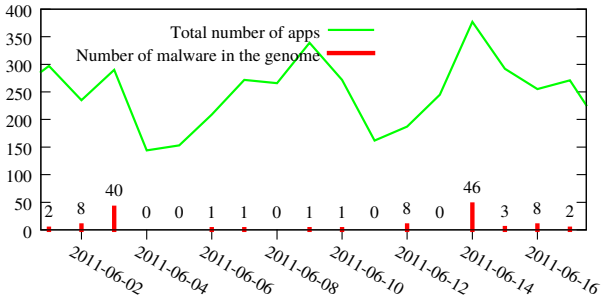

Figure 5. Number of packaged application and of packaged malware over time: Focus on period 2011-06-01 to 2011-06-17

no other malware was packaged. We counted 78 cases where at least two malware were packaged in the same second. At 15 instances, four or more malware were packaged in the same second; Two of those instances saw ten or more new malware being packaged. Such a strong time locality suggests that malware writers have set up tools to automate the malware packaging process. One single certificate (md5: *264BF7D71E0EDC4FCB8A9A16AB7C3357*) even managed to sign 781 apps detected as malware by at least one anti virus in the same second (2012-01-07 14:25:06).

Malware development is often a standardized process that aims at producing a large number of malware at once. Aside from rare cases of target-specialized malware, malware are built in bulk in the like of slightly different applications.

C. Business of Malware Writing

We now look into the process of malware writing. We focus our analysis on their apparition cycles by clustering Android applications based on the week day during which they were packaged. For this experiment, we only consider malware that were detected by at least half of the anti virus software operated by VirusTotal.

Table II highlights the distribution of app packaging dates for both benign applications and malware across week days. The percentage of apps packaged during business days are actually similar for malware and benign applications. A test of significance with the MWW test further confirms that the statistical difference is near null.

On average, 19% of benign applications are packaged during weekends, while this is the case for only 13% of malware. We further use the MWW test to confirm that the difference between weekdays and week-end days is statistically more significant for malware than for benign. There is thus a clear pattern of five-day work per week. A possible explanation to this pattern could be that malware writing is performed by some developers during their regular office hours while working for their employer. A second reason might be that malware writers follow a standard work schedule and do not work during weekends, thus suggesting an industrial process in the building of malware rather than a spare-time hobby.

There appear to be evidence that the business of malware writing, or at least their proliferation, is at an industrial scale.

D. Digital Certificates

Android applications rely on digital certificates to build a trust model between developers and end users. Applications signed using the same certificate can share information and data at runtime (if allowed by explicit permissions). Certificates also allow to link a set of applications with their developer, although this linking does not ensure that the identity of a developer is certified. Indeed, certificates can be self-signed, rather than signed by a competent trustworthy authority, and therefore do not necessary lead to the real developer. However, finding the same certificate (serial number, fingerprint , issuer and owner) in several

Table II
DISTRIBUTION OF ANDROID PACKAGING DATES ACROSS WEEK DAYS

	Monday		Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday	
Benign Apps	56,476	15.75%	57,728	16.10%	58,078	16.20%	58,995	16.45%	58,926	16.43%	34,223	9.54%	34,182	9.53%
Malware *(Threshold=24)*	236	14.68%	376	23.38%	284	17.66%	276	17.16%	225	13.99%	90	5.60%	121	7.52%
Malware *(Threshold=25)*	211	14.46%	342	23.44%	265	18.16%	254	17.41%	205	14.05%	81	5.55%	101	6.92%
Malware *(Threshold=26)*	200	14.60%	328	23.94%	249	18.18%	234	17.08%	190	13.87%	74	5.40%	95	6.93%

applications is a strong indicator of either a unique origin, or of advanced certificate stealing and reuse.

Our analysis on certificates aims at understanding the practice of certificate use by malware writers. We first note, based on our datasets, that self-signed certificates are the norm rather than the exception for Android developers. Of the 165 542 certificates in our dataset, only 51 are not self-signed. Self-signed certificates were used to sign 99.88% of the apps in our dataset. Consequently, most certificates carry no information that could be trusted about the identity of the application developer.

Our findings apply particularly to malware development. We focus our study on the subset of malware in the Genome dataset. These are well established malware that most anti virus products can identify. For instance, the certificate that holds the serial number `E6EFD52A17E0DCE7` was used in at least two different malware applications. Manual searching for the Issuer-related fields[13] does lead to the blog of a well known Android developer. One entry of this blog addressed the issue of signing of Android applications. After reading this entry, we found out that writers of the referred malware just copy/pasted the command in the posted example without any effort to change the basic information that indicates what a certificate is supposed to certify.

We have further investigated this copy/paste strategy and found that it occurs too often. Thus, although a certificate issued to Android Debug can be used to develop and test an application, the release version cannot be published with such a certificate. This basic rule is stated in almost every online tutorial and Android textbook. Yet, we identified more than 50 well-known malware which use such a certificate: this questions the competency or may highlight the laziness of malware writers as in a day-to-day job.

Finally, our manual investigation into the attributes of certificates in malware, reveal that, sometimes, malware writers brag or use obvious offensive names. For instance, a certificate whose *owner* is named *PhoneSniper* appears in at least 281 different malware. If users were able to carefully inspect certificates before installation, such malware would have been less propagated. Similarly, this information could be used with techniques of natural language processing to silently filter some malware in application markets.

The vast majority of Android apps in our datasets are signed with a certificate that was used to sign very few other applications. Indeed, we have found that 95% of the certificates signed less than 10 apps. However, for the remainder of certificates that were used in large numbers of applications, different patterns emerge.

Table III summarizes the top certificates used in malware packages. Once again, we consider as malware all applications that were flagged as suspicious by at least half of the anti virus products in VirusTotal. The numbers distinctly provide evidence that there a mass development and deployment of Android benign and malware apps was put in place. For instance, three certificates were used each for more than 160 malware. The top-used certificate by malware is also used by over 4 623 benign applications: a realistic hypothesis to support this fact would be that the private key was somehow leaked, leading to many otherwise unrelated writers to use and share the same certificate.

We further consider the overlap between benign and malicious applications that share the same certificate. In Table IV, we indicate the top certificates that are used by both malware and goodware. We note that there is a clear overlap showing the usage of certificates for both malicious and benign applications. A number of explanations can be provided for this phenomenon:

- *Dr Jekyll and Mr Hyde syndrome*. Developers use the same development tools and environment for both legitimate and malicious applications. This observation supports the 5 working day behavior shown in table II. This means that developers write malware during their regular working hours.
- *Reputation biasing*. In this hypothesis, a developer might increase her/his reputation by developing benign applications. As soon as enough positive reviews have been obtained, successive malware might be more easily downloaded and installed. For instance, the certificate with the serial 4DFF5300, has been observed signing both a malicious and a non malicious application on 2011-08-30, in the very same time: 21:52:38. On the overall 1 benign application and 176 malware are associated with this certificate. The most recent application in our dataset using this certificate was packaged on 2012-03-11 17:19:54, while its first usage can be traced back to 2011-07-14 21:45:12.
- *Anti virus false negatives*. Probably, some of the applications tagged as benign are in fact malicious. It is possible that existing tools have not detected them yet as malicious, due to a better obfuscation and stealthier behavior.

[13] `Issuer: C=ID, ST=Jawa Barat, L=Bandung, O=Londatiga, OU=AndroidDev, CN=Lorensius W. L. T/emailAddress=lorenz@londatiga.net`

Table III
TOP 20 CERTIFICATES WHICH WERE USED TO SIGN THE MOST MALWARE

Certificate MD5	Number of Benign	Number of Malware	Certificate Issuer & Owner
E8...87	4 623	192	C=US, ST=California, L=Mountain View, O=Android, OU=Android, CN=Android/emailAddress=android@android.com
E5...3F	0	167	C=keji0003
CF...26	1	166	C=cn, ST=shenzhen, L=china, O=Phone, OU=Phone, CN=PhoneSniper
50...BA	0	98	C=kejikeji, ST=kejikeji, L=kejikeji, O=kejikeji, OU=kejikeji, CN=kejikeji
E5...C2	0	95	C=US, OU=Google Inc.
8B...D2	0	52	CN=Fujian Kaimo Network Tech
3C...3E	0	29	C=a, ST=a, L=a, O=a, OU=a, CN=a
AC...A7	1	21	CN=Sexy
C4...2B	0	20	C=CA, ST=Ontario, L=Toronto, O=Typ3 Studios, OU=Typ3 Studios, CN=Typ3 Studios
CF...6C	0	19	C=0
1D...07	6	17	C=CN, ST=Sichuan, L=Chengdu, O=jiemai-tech, OU=jiemai-tech, CN=Jiemai Technology
77...F3	8	17	CN=alan
B1...A4	0	17	OU=Safe System Inc., CN=Safe System Inc.
74...50	0	16	C=cn, ST=guangdong, L=shenzhen, O=hynoo, OU=hynoo, CN=wang
21...37	2	15	C=cn, ST=fujian, L=xiamen, O=guopai, OU=guopai, CN=jtwang
76...A8	1	14	C=CN, CN=picshow1
AC...94	0	13	C=86, ST=BeiJing, L=BeiJing, O=Gold Dream Studio, OU=Gold Dream Studio, CN=Hong Fu
73...A3	0	12	C=001, ST=US, L=LSA, O=www.android.com, OU=www.android.com, CN=Android
C6...1B	0	12	C=86, ST=SH, L=CN, O=MJ, OU=MJ, CN=MJ
E7...AE	34	12	C=0086, ST=Beijing, L=Beijing, O=Gall me, OU=Android, CN=Gall me

Table IV
TOP 15 CERTIFICATES WHICH WERE USED TO SIGN MANY MALWARE AND WHICH SIGNED BENIGN APPS AS WELL

Certificate MD5	Number of Benign	Number of Malware	Certificate Issuer & Owner
E8...87	4 623	192	C=US, ST=California, L=Mountain View, O=Android, OU=Android, CN=Android/emailAddress=android@android.com
1D...07	6	17	C=CN, ST=Sichuan, L=Chengdu, O=jiemai-tech, OU=jiemai-tech, CN=Jiemai Technology
77...F3	8	17	CN=alan
21...37	2	15	C=cn, ST=fujian, L=xiamen, O=guopai, OU=guopai, CN=jtwang
E7...AE	34	12	C=0086, ST=Beijing, L=Beijing, O=Gall me, OU=Android, CN=Gall me
8D...F9	92	10	C=US, ST=California, L=Mountain View, O=Android, OU=Android, CN=Android/emailAddress=android@android.com
DE...92	3	9	C=CN, ST=Guangdong, L=Guangzhou, O=synkay, OU=sunkay, CN=sunkay
C7...80	56	8	C=US, ST=Fl, L=Miami, O=Gp Imports, OU=Gp Imports, CN=Gp Imports
69...A5	87	7	C=CN, ST=beijing, L=beijing, O=Wali, OU=Wali, CN=Lee
34...F5	2	6	C=KR, ST=South Korea, L=Suwon City, O=Samsung Corporation, OU=DMC, CN=Samsung Cert/emailAddress=android.os@samsung.com
3D...10	6	4	CN=Ngan Viet Dung
BA...26	48	3	C=CN, ST=Zhejiang, L=Hangzhou, O=Feelingtouch, OU=Feelingtouch, CN=Feelingtouch
82...C5	2	3	C=86, ST=china, L=ysler, O=ysler, OU=ysler, CN=ysler.com
59...EE	178	2	C=86, ST=Guangdong, L=Guangzhou, O=3g.cn, OU=GAU, CN=Jarod Yv
51...B3	7	2	C=CN, ST=ShenZhen, L=ShenZhen, O=nmting.com, OU=nmting.com, CN=Ale Zhao

- *Anti virus false positives*. Anti virus can also wrongly flag a benign application as malware. For instance the digital certificate whose md5 is *75BDB3531C04EB8246846532A7AE2050* has been observed to sign 2 844 total applications, only one (1) of which being tagged as malicious. In this case, we suspect that either the certificate was stolen, but using it for only one single malicious application does not really make sense. More probable is the hypothesis that the single malicious application is a false positive. We have correlated this information also with the time-line of the packaging dates for this certificate. The single malicious application was packaged on 2013-11-15 19:16:04; On this very same day, this certificate signed 55 other apps that are all undetected by anti virus products. The usage pattern for this certificate shows very frequent application signing, often with just a few minutes between two apps, and the application detected as a malware exhibits no deviation from this pattern. Furthermore, it would make sense for the developer to create a new certificate if he once wrote a malware, in order to avoid having his/her future benign applications signed with a certificate that is associated with a malware.

Malware writers do not use digital certificates properly, and often reuse compromised keys that were used to build certificates of benign applications.

IV. DISCUSSION

The forensic analysis that we have performed and whose results were outlined in the previous section has yielded a number of insights for the research and practice of malware detection. In this section, we summarize these insights and discuss how this empirical study could be instrumented in our work on malware detection.

A. Summary of findings

On Anti virus software: Our large-scale analysis of hundreds of thousands of Android applications with over

40 anti virus products have revealed that most malware are not simultaneously identified by several anti virus. Only a small subset of common malware is detected by most anti virus software. This finding actually supports the idea that there is a need to invest in alternative tools for malware detection such as machine-learning based approaches which are promising to flag more malware variants.

On malware business: We have presented empirical evidence that malware were mass produced. This raises a number of questions leading to hypothesis on how malware developers manage to remain productive. The first hypothesis would be that, malware is not written from scratch, thus providing an opportunity to detect malware by discovering the piece of code that was grafted to existing, potentially popular, apps.

B. Insights

Building a naive anti virus software: Exploring the rate of shared certificates within malware, we were able to devise a naive malware detection mechanism based on the appearance of a tagged certificate. In its simplest form, the scheme consists in tagging any application as malicious if the signing key has been already observed for a confirmed malicious application.

To assess this naive approach we have considered that in a first phase we have manually discovered all malware packaged before 01/Jan/2013 in our dataset. We consider for this step only malware that are detected by at least half of the anti virus products. Then based on the certificates recorded for the found malware, we arbitrarily tag as malicious all applications packaged after 01/Jan/2013 and that are signed with any of the flagged certificates. Table V provides the results for this experiment. We were able to build a malware detector with a Precision of 84% ($2,166$ false positives out of $2,166 + 11,460$ tagged). While we succeed in flagging almost 1 actual malware out of 10, we only wrongly tag as malicious about 1 benign app in 100.

Table V
PERFORMANCE OF A NAIVE ANTI VIRUS SOFTWARE BASED ON CERTIFICATES

	Benign apps tagged	**Malware tagged**
Number	2 166	11 460
Percentage	1.19%	8.82%

At the minimum, the obtained results show that our naive approach could be used by anti virus vendors to improve their recall, by being suspicious of more apps, and improve precision by trusting apps signed with certificates that have been used in a large number of benign apps.

Localizing malware: Our findings on the potential mass production of malware could be leveraged in an approach of malware localization. Indeed, simultaneous development and packaging of malware suggests a redundant insertion of malware code in all applications. Thus, a similarity measure of the bytecode could allow to isolate this code and then locate it in other malware samples.

V. RELATED WORK

In this section, we enumerate a number of related work to emphasize on the importance of understanding the development of malware in order to devise efficient techniques for their detection. These related work span from empirical studies on on datasets of malware, to malware detection schemes.

A. Malicious datasets analysis

Researchers have already shown interest in malicious application datasets analysis. Felt *et al.* have analyzed several instances of malware deployed on various mobile platforms such as iOS, Android and Symbian [15]. They detail the wide range of incentives for malware writing, such as users' personal information and credentials exfiltration, ransom attack and the easiest way to profit from smartphone malware, premium-rate SMS services. Their study is however qualitative, while we have focused on a quantitative study to draw generalizable findings on common patterns.

The Genome dataset, our source of well-established malware, was built as part of a study by Zhou *et al.* [25]. They expose in details features and incentives of the current malware threat on Android. They also suggest that existing anti virus software still need improvements. Our analysis also comes to this conclusion when we demonstrate that most malware cannot be found by all anti virus products.

Opposite to our lightweight forensic analysis approach, Enck et al. [13] did an in-depth analysis of Android applications by using advanced static analysis methods. Doing so, they were able to discover some risky features of the android framework that could be used by malicious applications. However, our approach allowed to highlight interesting patterns that are could be leveraged more easily.

In recent work [1], Allix et al. have devised a sophisticated Feature set to use in a machine learning-based malware detection for Android. This approach has however proved to be resource-intensive, suggesting further investigations into more straightforward features. The study detailed in this paper is part of the roadmaps we have devised for our investigations.

B. Dynamic analysis

Various solutions have been proposed to detect malicious Android applications. Crowdroid, presented by Burguera et al. [8], performs dynamic analysis of Android applications by first collecting system calls patterns of applications, and then applying clustering algorithms to discriminate benign from suspicious behaviors. Crowdroid strongly rely on crowd sourcing for system calls patterns collection.

Vidas and Christin has investigated applications from alternative markets and compared them to applications from

the official market [22]. They have found that certain alternative markets almost exclusively distribute repackaged applications containing malware. They have proceeded to propose AppIntegrity to strengthen the authentication properties offered in application marketplaces. Our findings are in line with those, when we note that malware seem to be mass produced, and that the same certificates overlap between malware and benign applications.

C. Similarity and Heuristics based malware detection

In order to detect repackaged applications, which malware authors often do to embed their malicious payloads, Zhou et al. [23] presented *DroidMOSS*. Their approach consists in building a signature of the whole application by using a fuzzy hashing technique on the application's opcodes. Then a similarity score is computed for all Apps of a reference dataset, thus concluding to the detection of rep if a similarity score is higher than a given threshold

DroidRanger presented by Zhou et al. [24] tries to detect suspicious applications by first performing a fast filtering step based on permissions requested by an application. It then analyze the application code structure, as well as other properties of applications. Finally, an heuristics based detection engine is run with the data gathered about applications. With this approach, the authors were able to find malware on the official Android market but also two zero-day malware.

Regarding information leakage detection, Zhou et al. also proposed TISSA [26] allowing an end-user to have a fine grained control of the access to her personal data.

D. On device mitigation

The topic of embedded mitigation solution was covered by a wide range of previous works.

XManDroid [7] provides a mechanism capable of analyzing Inter Process Communications and decide if connections between applications are compliant with the system policy. This full dynamic solution addresses the problem of application level privilege escalation introduced in [10].

DroidChecker [9] attends to address the same issue by tracking permissions from the manifest files until their utilization within the application. To achieve this, Chan et al. proposed the use of control flow graphs and taint checking techniques.

Apex [20] proposes an extension to the Android permission manager allowing users to customize permissions owned by applications.

Kirin [14] extends the package installer and analyze permissions before installation. It embeds security rules based on permissions sets and can prevent a program from being installed according to the permissions it requests. A similar approach has been presented in [3].

Enck et al. introduced Taintdroid [12] which uses taint analysis techniques to detect sensitive data leaks and warn the end-user by showing him/her with relevant information.

In [17], authors have also studied data leaks. Hornyack et al. present a framework capable of shadowing sensitive user data and of blocking outgoing connexion implying data leaked.

E. Miscellaneous approaches

An offensive framework was presented in [16] which embeds a broad range of available Android exploits such as *Rage Against The Cage*, known to overflow the number of processes allowed. In the wild, this exploit is used by various malware. The framework is able to run arbitrary root exploit and to maintain privileges among reboots.

Rootkits possibilities on smartphones are exposed in [5], showing that smartphones are as vulnerable as desktop computers. The most valuable incentive to deploy rootkits on smartphones would be the interesting personal data such as voice communications and location.

With *Androguard*[14], Desnos et al. provide a tool to decompile Android applications and perform code analyses [11]. Built on top of these features, *Androguard* also provides a way to detect a large selection of malware, and to measure the similarity of two applications, to detect repackaging for instance.

Finally, concerning the detection of private data leaks, static analysis tools [4], [21], including taint analysis [2], have been proposed to deal with the specificities of Android.

VI. Conclusion

The recent and steady rise of Android malware over the past four years has lead to a rapidly growing automation in the malware creation process. Due to the specific nature of development of Android applications, important artifacts leak out and can provide some insights about their creators. We have analyzed the available data through this perspective. For our large-scale study, we have considered over 500,000 Android applications, which included both benign applications and malware.

Packaging dates show substantial time localization behavior. Waves of packaging can be observed thus shedding a new light on the malware creation process. Digital certificates, albeit self-signed also provide valuable pieces of information. We have observed huge quantities of malware sharing the same private key and thus proving that either keys have been stolen, or those malware have the same origin. On the other hand massive copy/paste coding, relying on directly copying code from popular tutorials and blogs, shows that the malware programming is done at a fast pace by developers lacking elementary cryptography knowledge.

This, unfortunately shows that current Android malware as well as mitigation techniques are still in the infancy.

[14] http://code.google.com/p/androguard/

It's surprising to see that most malware writers do not use digital certificates properly and that many of the current mitigation techniques did not check them. However, more troubling is the extent to which private keys seem to have been compromised and that both benign applications and malware share the same certificates.

In the future, we plan to leverage the insights discussed in Section IV. Furthermore, we plan to extend this work by considering also the automated analysis of the bytecode. Some preliminary work have been done and the results are promising.

ACKNOWLEDGEMENTS

This work was supported by the Fonds National de la Recherche (FNR), Luxembourg, under the project AndroMap C13/IS/5921289.

We would like to thank VirusTotal for providing us access to their tool.

REFERENCES

[1] K. Allix, T. F. Bissyandé, Q. Jerome, J. Klein, R. State, and Y. Le Traon, "Large-scale machine learning-based malware detection: Confronting the "10-fold cross validation scheme" with reality," in *CODASPY '14*, 2014.

[2] S. Arzt, S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Conference on Programming Language Design and Implementation (PLDI)*, 2014.

[3] A. Bartel, J. Klein, M. Monperrus, K. Allix, and Y. Le Traon, "Improving privacy on android smartphones through in-vivo bytecode instrumentation," Technical Report, May 2012.

[4] A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon, "Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot," in *ACM Sigplan Workshop on the State Of The Art in Java Program Analysis (SOAP)*, 2012.

[5] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy, and L. Iftode, "Rootkits on smart phones: attacks, implications and opportunities," in *HotMobile '10*, Maryland, 2010.

[6] J. Brodkin, "On its 5th birthday, 5 things we love about android," Nov. 2012, http://arstechnica.com/gadgets/2012/11/on-androids-5th-birthday-5-things-we-love-about-android/.

[7] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks," Technische Universität Darmstadt, Technical Report TR-2011-04, Apr. 2011.

[8] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *SPSM '11*, Chicago, Illinois, USA, 2011, pp. 15–26.

[9] P. P. Chan, L. C. Hui, and S. M. Yiu, "Droidchecker: analyzing android applications for capability leak," in *WISEC '12*. Tucson, Arizona, USA: ACM, 2012, pp. 125–136.

[10] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *ISC'10*. Boca Raton, FL, USA: Springer-Verlag, 2011, pp. 346–360.

[11] A. Desnos, "Android: Static analysis using similarity distance," in *HICSS '12*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 5394–5403.

[12] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *OSDI'10*. Vancouver, BC, Canada: USENIX Association, 2010, pp. 1–6.

[13] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *SEC'11*. San Francisco, CA: USENIX Association, 2011, pp. 21–21.

[14] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *CCS '09*. Chicago, Illinois, USA: ACM, 2009, pp. 235–245.

[15] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *SPSM '11*. New York, NY, USA: ACM, 2011, pp. 3–14.

[16] S. Höbarth and R. Mayrhofer, "A framework for on-device privilege escalation exploit execution on android," in *Proc. IWSSI/SPMU*, June 2011.

[17] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 639–652.

[18] ITU, "Information technology Open Systems Interconnection The Directory: Public-key and attribute certificate frameworks Technical Corrigendum 2," ITU, Genebra, Series X: Data Networks, Open System Communications and Security Directory, nov 2008, ITU-T Recommendation X.509.

[19] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[20] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," in *ASIACCS '10*, 2010, pp. 328–332.

[21] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon, "Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis," in *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[22] T. Vidas and N. Christin, "Sweetening android lemon markets: Measuring and combating malware in application marketplaces," in *CODASPY '13*, 2013.

[23] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *CODASPY '12*. ACM, 2012, pp. 317–326.

[24] 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 *NDSS'12*, 2012.

[25] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *SP '12*, Washington, DC, USA, 2012, pp. 95–109.

[26] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *TRUST'11*. Pittsburgh, PA: Springer-Verlag, 2011, pp. 93–107.