Dissecting Android Malware: Characterization and Evolution

By Authors

Abstract:

The popularity and adoption of smart phones has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android. In light of their ra...Show More

Metadata

Abstract:

The popularity and adoption of smart phones has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android. In light of their rapid growth, there is a pressing need to develop effective solutions. However, our defense capability is largely constrained by the limited understanding of these emerging mobile malware and the lack of timely access to related samples. In this paper, we focus on the Android platform and aim to systematize or characterize existing Android malware. Particularly, with more than one year effort, we have managed to collect more than 1,200 malware samples that cover the majority of existing Android malware families, ranging from their debut in August 2010 to recent ones in October 2011. In addition, we systematically characterize them from various aspects, including their installation methods, activation mechanisms as well as the nature of carried malicious payloads. The characterization and a subsequent evolution-based study of representative families reveal that they are evolving rapidly to circumvent the detection from existing mobile anti-virus software. Based on the evaluation with four representative mobile security software, our experiments show that the best case detects 79.6% of them while the worst case detects only 20.2% in our dataset. These results clearly call for the need to better develop next-generation anti-mobile-malware solutions.

Date of Conference: 20-23 May 2012

Date Added to IEEE *Xplore*: 09 July 2012

ISSN Information:

Conference Location: San Francisco, CA, USA

In recent years, there is an explosive growth in smartphone sales and adoption. According to CNN [1], smartphone shipments have tripled in the past three years (from 40 million to about 120 million). Unfortunately, the increasing adoption of smartphones comes with the growing prevalence of mobile malware. As the most popular mobile platform, Google's Android overtook others (e.g., Symbian) to become the top mobile malware platform. It has been highlighted [2] that “among all mobile malware, the share of Android-based malware is higher than 46% and still growing rapidly.” Another recent report also alerts that there is “400 percent increase in Android-based malware since summer 2010” [3].

Given the rampant growth of Android malware, there is a pressing need to effectively mitigate or defend against them. However, without an insightful understanding of them, it is hard to imagine that an effective mitigation solution can be practically developed. To make matters worse, the research community at large is still constrained by the lack of a comprehensive mobile malware dataset to start with.

The goals and contributions of this paper are three-fold. First, we fulfil the need by presenting the first large collection of 1260 Android malware samples in 49 different malware families, which covers the majority of existing Android malware, ranging from their debut in August 2010 to recent ones in October 2011. The dataset is accumulated from more than one year effort in collecting related malware samples, including manual or automated crawling from a variety of Android Markets. To better mitigate mobile malware threats, we will release the entire dataset to the research community at http://malgenomeproject.org/.2

Second, based on the collected malware samples, we perform a timeline analysis of their discovery and thoroughly characterize them based on their detailed behavior breakdown, including the installation, activation, and payloads. The timeline analysis is instrumental to revealing major outbreaks of certain Android malware in the wild while the detailed breakdown and characterization of existing Android malware is helpful to better understand them and shed light on possible defenses.

Specifically, in our 1260 malware samples, we find that 1083 of them (or 86.0%) are repackaged versions of legitimate applications with malicious payloads, which indicates the policing need of detecting repackaged applications in the current Android Markets. Also, we observe that more recent Android malware families are adopting update attacks and drive-by downloads to infect users, which are more stealthy and difficult to detect. Further, when analyzing the carried payloads, we notice a number of alarming statistics: (1) Around one third (36.7%) of the collected malware samples leverage root-level exploits to fully compromise the Android security, posing the highest level of threats to users' security and privacy; (2) More than 90% turn the compromised phones into a botnet controlled through network or short messages. (3) Among the 49 malware families, 28 of them (with 571 or 45.3% samples) have the built-in support of sending out background short messages (to premium-rate numbers) or making phone calls without user awareness. (4) Last but not least, 27 malware families (with 644 or 51.1% samples) are harvesting user's information, including user accounts and short messages stored on the phones.

Third, we perform an evolution-based study of representative Android malware, which shows that they are rapidly evolving and existing anti-malware solutions are seriously lagging behind. For example, it is not uncommon for Android malware to have encrypted root exploits or obfuscated command and control (C&C) servers. The adoption of various sophisticated techniques greatly raises the bar for their detection. In fact, to evaluate the effectiveness of existing mobile antivirus software, we tested our dataset with four representative ones, i.e., AVG

Antivirus Free, Lookout Security & Antivirus, Norton Mobile Security Lite, and Trend Micro Mobile Security Personal Edition, all downloaded from the official Android Market (in the first week of November, 2011). Sadly, wile the best case was able to detect 1,003 (or 79.6%) samples in our dataset, the worst case can only detect 254 (20.2%) samples. Furthermore, our analysis shows that malware authors are quickly learning from each other to create hybrid threats. For example, one recent Android malware, i.e., AnserverBot [4] (reported in September 2011), is clearly inspired from Plankton [5] (reported in June 2011) to have the dynamic capability of fetching and executing payload at runtime, posing significant challenges for the development of next-generation anti-mobile-malware solutions.

The rest of this paper is organized as follows: Section II presents a timeline analysis of existing Android malware. Section III characterizes our samples and shows a detailed breakdown of their infection behavior. After that, Section IV presents an evolution study of representative Android malware and Section V shows the detection results with four representative mobile antivirus software. Section VI discusses possible ways for future improvement, followed by a survey of related work in Section VII. Lastly, we summarize our paper in Section VIII.

SECTION II.

Malware Timeline

In Table I, we show the list of 49 Android malware families in our dataset along with the time when each particular malware family is discovered. We obtain the list by carefully examining the related security announcements, threat reports, and blog contents from existing mobile antivirus companies and active researchers [6] [7] [8] [9] [10] [11] [12] as exhaustively as possible and diligently requesting malware samples from them or actively crawling from existing official and alternative Android Markets. As of this writing, our collection is believed to reflect the state of the art of Android malware. Specifically, if we take a look at the Android malware history [13] from the very first Android malware FakePlayer in August 2010 to recent ones in the end of October 2011, it spans slightly more than one year with around 52 Android malware families reported. Our dataset has 1260 samples in 49 different malware families, indicating a very decent coverage of existing Android malware.

Table I The Timeline of 49 Android Malware In Our Collection (O† Offical Android Market; A‡: Alternative Android Markets)


Table I- The Timeline of 49 Android Malware In Our Collection (O† Offical Android Market; A‡: Alternative Android Markets)

For each malware family, we also report in the table the number of samples in our collection and differentiate the sources where the malware was discovered, i.e., from either the official or alternative Android Markets. To eliminate possible false positive in our dataset, we run our collection through existing mobile antivirus software for confirmation (Section V). If there is any miss from existing mobile antivirus security software, we will manually verify the sample and confirm it is indeed a malware.

To better illustrate the malware growth, we show in Figures 1(a) and 1(b) the monthly breakdown of new Android malware families and the cumulative monthly growth of malware samples in our dataset. Consistent with others [2] [3], starting summer 2011, the Android malware has indeed increased dramatically, reflected in the rapid

emergence of new malware families as well as different variants of the same type. In fact, the number of new Android malware in July 2011 alone already exceeds the total number in the whole year of 2010. Figure 1(b) further reveals two major Android malware outbreaks, including DroidKungFu (starting June, 2011) and AnserverBot (starting September, 2011). Among these 1260 samples in our collection, 37.5% of them are related to DroidKungFu [14] and its variants; 14.8% are AnserverBot [4]. Both of them are still actively evolving to evade the detection from existing antivirus software - a subject we will dive into in Section IV.


Figure 1. - The Android Malware Growth in 2010–2011

Figure 1.

The Android Malware Growth in 2010–2011

In this section, we present a systematic characterization of existing Android malware, ranging from their installation, activation, to the carried malicious payloads.

A. Malware Installation

By manually analyzing malware samples in our collection, we categorize existing ways Android malware use to install onto user phones and generalize them into three main social engineering-based techniques, i.e., *repackaging*, *update attack*, and *drive-by download*. These techniques are not mutually exclusive as different variants of the same type may use different techniques to entice users for downloading.

1) Repackaging

Repackaging is one of the most common techniques malware authors use to piggyback malicious payloads into popular applications (or simply apps). In essence, malware authors may locate and download popular apps, disassemble them, enclose malicious payloads, and then re-assemble and submit the new apps to official and/or alternative Android Markets. Users could be vulnerable by being enticed to download and install these infected apps.

To quantify the use of repackaging technique among our collection, we take the following approach: if a sample shares the same package name with an app in the official Android Market, we then download the official app (if free) and manually compare the difference, which typically contains the malicious payload added by malware authors. If the original app is not available, we choose to disassemble the malware sample and manually determine whether the malicious payload is a natural part of the main functionality of the host app. If not, it is considered as repackaged app.

In total, among the 1260 malware samples, 1083 of them (or 86.0%) are repackaged. By further classifying them based on each individual family (Table II), we find that within the total 49 families in our collection, 25 of them infect users by these repackaged apps while 25 of them are standalone apps where most of them are designed to be spyware in the first place. One malware family, i.e., GoldDream, utilizes both for its infection.

Table II An Overview of Existing Android Malware (Part I: Installation and Activation)


Table II- An Overview of Existing Android Malware (Part I: Installation and Activation)

Among the 1083 repackaged apps, we find that malware authors have chosen a variety of apps for repackaging, including paid apps, popular game apps, powerful utility apps (including security updates), as well as porn-related apps. For instance, one AnserverBot malware sample (SHA1: ef140ablad04bdge52c8c5f2fb6440f3agebe8ea) repackaged a paid app com. camelgames.mxmotor available on the official Android Market. Another BgServ [15] malware sample (SHA1: bc2dedad0507a916604f86167a9fa30693ge2080) repackaged the security tool released by Google to remove DroidDream from infected phones.

Also, possibly due to the attempt to hide piggy-backed malicious payloads, malware authors tend to use the class-file names which look legitimate and benign. For example, AnserverBot malware uses a package name com.sec.android.provider.drm for its payload, which looks like a module that provides legitimate **DRM** functionality. The first version of DroidKungFu chooses to use com. google .ssearch to disguise as the Google search module and its follow-up versions use com.google.update to pretend to be an official Google update.

It is interesting to note that one malware family - jSMSHider- uses a publicly available private key (serial number: b3998086d056cffa) that is distributed in the Android Open Source Project (AOSP). The current Android security model allows the apps signed with the same platform key of the phone firmware to request the permissions which are otherwise not available to normal third-party apps. One such permission includes the installation of additional apps without user intervention. Unfortunately, a few (earlier) popular custom firmware images were signed by the default key distributed in AOSP. As a result, the jSMSHider-infected apps may obtain privileged permissions to perform dangerous operations without user's awareness.

2) Update Attack

The first technique typically piggy- backs the entire malicious payloads into host apps, which could potentially expose their presence. The second technique makes it difficult for detection. Specifically, it may still repackage popular apps. But instead of enclosing the payload as a whole, it only includes an update component that will fetch or download the malicious payloads at runtime. As a result, a static scanning of host apps may fail to capture the malicious payloads. In our dataset, there are four malware families, i.e., BaseBridge, DroidKungFuUpdate, AnserverBot, and Plankton, that adopt this attack (Table II).

The BaseBridge malware has a number of variants. While some embed root exploits that allow for silent installation of additional apps without user intervention, we here focus on other variants that use the update attacks without root exploits. Specifically, when a BaseBridge-infected app runs, it will check whether an update dialogue needs to be displayed. If yes, by essentially saying that a new version is available, the user will be offered to install the updated version (Figure 2(a)). (The new version is actually stored in the host app as a resource or asset file.) If the user accepts, an “updated” version with the malicious payload will then be installed (Figure 2(b)). Because the malicious payload is in the “updated” app, *not* the original app itself, it is more stealthy than the first technique that directly includes the entire malicious payload in the first place.


Figure 2. - An Update Attack from BaseBridge

Figure 2.

An Update Attack from BaseBridge

The DroidKungFuUpdate malware is similar to BaseBridge. But instead of carrying or enclosing the “updated” version inside the original app, it chooses to remotely download a new version from network. Moreover, it takes a stealthy route by notifying the users through a third-party library [16] that provides the (legitimate) notification functionality. (Note the functionality is similar to the automatic notification from the Google's Cloud to Device Messaging framework.) In Figure 3, we show the captured network traffic initiated from the original host app to update itself. Once downloaded, the “updated” version turns out to be the DroidKungFu3 malware. As pointed out in Table I, the DroidKungFuUpdate malware was available on both official and alternative Android Markets.

 Figure 3. - An Update Attack from DroidKungFuUpdate

Figure 3.

An Update Attack from DroidKungFuUpdate

The previous two update attacks require user approval to download and install new versions. The next two malware, i.e., AnserverBot and Plankton, advance the update attack by stealthily upgrading certain components in the host apps *not* the entire app. As a result, it does not require user approval. In particular, Plankton directly fetches and runs a jar file maintained in a remote server while AnserverBot retrieves a public (encrypted) blog entry, which contains the actual payloads for update! In Figure 4, we show the actual network traffic to download AnserverBot payload from the remote command and control (C&C) server. Apparently, the stealthy nature of these update attacks poses significant challenges for their detection (Table VII-Section V).

 Figure 4. - An Update Attack from AnserverBot

Figure 4.

An Update Attack from AnserverBot

3) Drive-by Download

The third technique applies the traditional drive-by download attacks to mobile space. Though they are not directly exploiting mobile browser vulnerabilities, they are essentially enticing users to download “interesting” or “feature-rich” apps. In our collection, we have four such malware families, i.e., GGTracker [17], Jifake [18], Spitmo [19] and ZitMo [20]. The last two are designed to steal user's sensitive banking information.

The GGTracker malware starts from its in-app advertisements. In particular, when a user clicks a special advertisement link, it will redirect the user to a malicious website, which claims to be analyzing the battery usage of user's phone and will redirect the user to one fake Android Market to download an app claimed to improve battery efficiency. Unfortunately, the downloaded app is not one that focuses on improving the efficiency of battery, but a malware that will subscribe to a premium-rate service without user's knowledge.

Similarly, the Jifake malware is downloaded when users are redirected to the malicious website. However, it is not using in-app advertisements to attract and redirect users. Instead, it uses a malicious QR code [21], which when scanned will redirect the user to another URL containing the Jifake malware. This malware itself is the repackaged mobile ICQ client, which sends several SMS messages to a premium-rate number. While QR code-based malware propagation has been warned earlier [22], this is the first time that this attack actually occurred in the wild.

The last two Spitmo and ZitMo are ported versions of nefarious PC malware, i.e., SpyEye and Zeus. They work in a similar manner: when a user is doing online banking with a comprised PC, the user will be redirected to download a particular smartphone app, which is claimed to better protect online banking activities. However, the downloaded app is actually a malware, which can collect and send mTANs or SMS messages to a remote server. These two malware families rely on the comprised desktop browsers to launch the attack. Though it may seem hard to infect real users, the fact that they can steal sensitive bank information raises serious alerts to users.

4) Others

We have so far presented three main social engineering-based techniques that have been used in existing Android malware. Next, we examine the rest samples that do not fall in the above three categories. In particular, our dataset has 1083 repackaged apps, which leaves 177 standalone apps. We therefore look into those standalone apps and organize them into the following four groups.

The first group is considered spyware as claimed by themselves - they intend to be installed to victim's phones on purpose. That probably explains why attackers have no motivations or the need to lure victim for installation. GPSSMSSpy is an example that listens to SMS-based commands to record and upload the victim's current location.

The second group includes those fake apps that masquerade as the legitimate apps but stealthily perform malicious actions, such as stealing users' credentials or sending background SMS messages. FakeNetflix is an example that steals a user's Netflix account and password. Note that it is not a repackaged version of Netflix app but instead disguises to be *the* Netflix app with the same user interface. FakePlayer is another example that masquerades as a movie player but does not provide the advertised functionality at all. All it does is to send SMS messages to premium-rate numbers without user awareness.

The third group contains apps that also intentionally include malicious functionality (e.g., sending unauthorized SMS messages or subscribing to some value-added service automatically). But the difference from the second group is that they are not fake ones. Instead, they can provide the functionality they claimed. But unknown to users, they also include certain malicious functionality. For example, one RogueSPPush sample is an astrology app. But it will automatically subscribe to premium-rate services by intentionally hiding confirmation SMS messages.

The last group includes those apps that rely on the root privilege to function well. However, without asking the user to grant the root privilege to these apps, they leverage known root exploits to escape from the built-in security sandbox. Though these apps may not clearly demonstrate malicious intents, the fact of using root exploits without user permission seems cross the line. Examples in this group include Asroot and DroidDeluxe.

B. Activation

Next, we examine the system-wide Android events of interest to existing Android malware. By registering for the related system-wide events, an Android malware can rely on the built-in support of automated event notification and callbacks on Android to flexibly trigger or launch its payloads. For simplicity, we abbreviate some frequently-used Android events in Table III. For each malware family in our dataset, we also report related events in Table II.

Table III The (Abbreviated) Android Events/Actions of Interest to Existing Malware

Among all available system events, BOOT_COMPLETED is the most interested one to existing Android malware. This is not surprising as this particular event will be triggered when the system finishes its booting process - a perfect timing for malware to kick off its background services. In our dataset, 29 (with 83.3% of the samples) malware families listen to this event. For instance, Geinimi (SHA1: 179e1c69ceaf2a98fdca1817a3f3f1fa28236b13) listens to this event to bootstrap the background service - com.geinimi. AdService.

The SMS_RECEIVED comes second with 21 malware families interested in it. This is also reasonable as many malware will be keen in intercepting or responding incoming SMS messages. As an example, zSone listens to this SMS_RECEIVED event and intercepts or removes all SMS message from particular originating numbers such as “10086” and “10010.”

During our analysis, we also find that certain malware registers for a variety of events. For example, AnserverBot registers for callbacks from 10 different events while BaseBridge is interested in 9 different events. The registration of a large number of events is expected to allow the malware to reliably or quickly launch the carried payloads.

In addition, we also observe some malware samples directly hijack the entry activity of the host apps, which will be triggered when the user clicks the app icon on the home screen or an intent with action ACTION_MAIN is received by the app. The hijacking of the entry activity allows the malware to immediately bootstrap its service before starting the host app's primary activity. For example, DroidDream (SHA1 : fdf6509b4911485b3f4783a72fde5c27aa9548c7) replaces the original entry activity with its own com.android.root.main so that it can gain control even before the original activity com.codingcaveman.SoloTrial.SplashActivity is launched. Some malware may also hijack certain UI interaction events (e.g., button clicking). An example is the zSone malware (SHA1: 00d6e661f90663eeffc10f64441b17079ea6f819) that invokes its own SMS sending code inside the onClick() function of the host app.

C. Malicious Payloads

As existing Android malware can be largely characterized by their carried payloads, we also survey our dataset and partition the payload functionalities into four different categories: *privilege escalation*, *remote control*, *financial charges*, and *personal information stealing*.

1) Privilege Escalation

The Android platform is a complicated system that consists of not only the Linux kernel, but also the entire Android framework with more than 90 open-source libraries included, such as WebKit, SQLite, and OpenSSL. The complexity naturally introduces software vulnerabilities that can be potentially exploited for privilege escalation. In Table IV, we show the list of known Android platform-level vulnerabilities that can be exploited for

privilege exploitations. Inside the table, we also show the list of Android malware that actively exploit these vulnerabilities to facilitate the execution of their payloads.

Table IV The List of Platform-Level Root Exploits and Their Uses in Existing android Malware


Table IV- The List of Platform-Level Root Exploits and Their Uses in Existing android Malware

Overall, there are a small number of platform-level vulnerabilities that are being actively exploited in the wild. The top three exploits are **exploid, RATC** (or **RageAgainstTheCage)**, and **Zimperlich.** We point out that if the **RATC** exploit is launched within a running app, it is effectively exploiting the bug i*n* the zygote daemon, *not* the intended **adbd** daemon, thus behavoring as the **Zimperlich** exploit. Considering the similar nature of these two vulnerabilities, we use **RATC** to represent both of them.

From our analysis, one alarming result is that among 1260 samples in our dataset, 463 of them (36.7%) embed at least one root exploit (Table V). In terms of the popularity of each individual exploit, there are 389, 440, 4, and 8 samples that contain **exploid, RATC**, GingerBreak, and **asroot**, respectively. Also, it is not uncommon for a malware to have two or more root exploits to maximize its chances for successful exploitations on multiple platform versions. (In our dataset, there are 378 samples with more than one root exploit.)

Table V An Overview of Existing Android Malware (Part II: Malicious Payloads)


Table V- An Overview of Existing Android Malware (Part II: Malicious Payloads)

A further investigation on how these exploits are actually used shows that many earlier malware simply copy verbatim the publicly available root exploits without any modification, even without removing the original debug output strings or changing the file names of associated root exploits. For example, **DroidDream** contains the **exploid** file name exactly the same as the publicly available one. However, things have been changed recently. For example, **DroidKungFu** does not directly embed these root exploits. Instead it first encrypts these root exploits and then stores them as a resource or asset file. At runtime, it dynamically uncovers these encrypted root exploits and then executes them properly, which makes their detection very challenging. In fact, when the first version of **DroidKungFu** was discovered, it has been reported that no single existing mobile antivirus software at that time was able to detect it, which demonstrated the “effectiveness” of this approach. Moreover, other recent malware such as **DroidCoupon** and GingerMaster apparently obfuscate the file names of the associated root exploits (e.g., by pretending as picture files with **png** suffix). We believe these changes reflect the evolving nature of malware development and the ongoing arms race for malware defense (Section IV).

2) Remote Control

During our analysis to examine the remote control functionality among the malware payloads, we are surprised to note that 1, 172 samples (93.0%) turn the infected phones into bots for remote control. Specifically, there are 1, 171 samples that use the HTTP-based web traffic to receive bot commands from their C&C servers.

We also observe that some malware families attempt to be stealthy by encrypting the URLs of remote C&C servers as well as their communication with C&C servers. For example, Pjapps uses its own encoding scheme to encrypt the C&C server addresses. One of its samples (SH1: 663e8eb52c7b4a14e2873b1551748587018661b3) encodes its C&C server mobilemeego91.com into 2maodb3ialke8mdeme3gkos9g1icaofm. **DroidKungFu3**

employs the standard AES encryption scheme and uses the key Fuck_sExy-aL1!Pw to hide its C&C servers. Geinimi similarly applies DES encryption scheme (with the key 0x0102030405060708) to encrypt its communication to the remote C&C server.

During our study, we also find that most C&C servers are registered in domains controlled by attackers themselves. However, we also identify cases where the C&C servers are hosted in public clouds. For instance, the **Plankton** spyware dynamically fetches and runs its payload from a server hosted on the Amazon cloud. Most recently, attackers are even turning to public blog servers as their C&C servers. AnserverBot is one example that uses two popular public blog services, i.e., Sina and Baidu, as its C&C servers to retrieve the latest payloads and new C&C URLs (Section IV).

3) Financial Charge

Beside privilege escalation and remote control, we also look into the motivations behind malware infection. In particular, we study whether malware will intentionally cause financial charges to infected users.

One profitable way for attackers is to surreptitiously subscribe to (attacker-controlled) premium-rate services, such as by sending SMS messages. On Android, there is a permission-guarded function sendTextMessage that allows for sending an SMS message in the background without user's awareness. We are able to confirm this type of attacks targeting users in Russia, United States, and China. The very first Android malware FakePlayer sends SMS message “798657” to multiple premium-rate numbers in Russia. GGTracker automatically signs up the infected user to premium services in US without user's knowledge. zSone sends SMS messages to premium-rate numbers in China without user's consent. In total, there are 55 samples (4.4%) falling in 7 different families (tagged with ‡ in Table V) that send SMS messages to the premium-rate numbers hardcoded in the infected apps.

Moreover, some malware choose *not* to hard-code premium-rate numbers. Instead, they leverage the flexible remote control to push down the numbers at runtime. In our dataset, there are 13 such malware families (tagged with t in Table V). Apparently, these malware families are more stealthy than earlier ones because the destination number will not be known by simply analyzing the infected apps.

In our analysis, we also observe that by automatically subscribing to premium-rate services, these malware families need to reply to certain SMS messages. This may due to the second-confirmation policy required in some countries such as China. Specifically, to sign up a premium-rate service, the user must reply to a confirming SMS message sent from the service provider to finalize or activate the service subscription. To avoid users from being notified, they will take care of replying to these confirming messages by themselves. As an example, RogueSPPush will automatically reply “Y” to such incoming messages in the background; GGTracker will reply “YES” to one premium number, 99735, to active the subscribed service. Similarly, to prevent users from knowing subsequent billing-related messages, they choose to filter these SMS messages as well. This behavior is present in a number of malware, including zSone, RogueSPPush, and GGTracker.

Besides these premium-rate numbers, some malware also leverage the same functionality by sending SMS messages to other phone numbers. Though less serious than previous ones, they still result in certain financial charges especially when the user does not have an unlimited messaging plan. For example, DogWars sends SMS messages to all the contacts in the phone without user's awareness. Other malware may also make background

phone calls. With the same remote control capability, the destination number can be provided from a remote C&C server, as shown in Geinimi.

4) Information Collection

In addition to the above payloads, we also find that malware are actively harvesting various information on the infected phones, including SMS messages, phone numbers as well as user accounts. In particular, there are 13 malware families (138 samples) in our dataset that collect SMS messages, 15 families (563 samples) gather phone numbers, and 3 families (43 samples) obtain and upload the information about user accounts. For example, SndApps collects users' email addresses and sends them to a remote server. FakeNetflix gathers users' Netflix accounts and passwords by providing a fake but seeming identical Netflix UI.

We consider the collection of users' SMS messages is a highly suspicious behavior. The user credential may be included in SMS messages. For example, both Zitmo (the Zeus version on Android) and Spitmo (the SpyEpy version on Android) attempt to intercept SMS verification messages and then upload them to a remote server. If successful, the attacker may use them to generate fraudulent transactions on behalf of infected users.

D. Permission Uses

For Android apps without root exploits, their capabilities are strictly constrained by the permissions users grant to them. Therefore, it will be interesting to compare top permissions requested by these malicious apps in the dataset with top permissions requested by benign ones. To this end, we have randomly chosen 1260 top free apps downloaded from the official Android Market in the first week of October, 2011. The results are shown in Figure 5.


Figure 5. - The Comparison of Top 20 Requested Permissions by Malicious and Benign Apps

Figure 5.

The Comparison of Top 20 Requested Permissions by Malicious and Benign Apps

Based on the comparison, INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE, and WRITE_EXTERNAL_STORAGE permissions are widely requested in both malicious and benign apps. The first two are typically needed to allow for the embedded ad libraries to function properly. But malicious apps clearly tend to request more frequently on the SMS-related permissions, such as READ_SMS, WRITE_SMS, RECEIVE_SMS, and SEND_SMS. Specifically, there are 790 samples (62.7%) in our dataset that request the READ_SMS permission, while less than 33 benign apps (or 2.6%) request this permission. These results are consistent with the fact that 28 malware families in our dataset (or 45.3% of the samples) that have the SMS-related malicious functionality.

Also, we observe 688 malware samples request the RECEIVE_BOOT_COMPLETED permission. This number is five times of that in benign apps (137 samples). This could be due to the fact that malware is more likely to run background services without user's intervention. Note that there are 398 malware samples requesting CHANGE_WIFI_STATE permission, which is an order of magnitude higher than that in benign apps (34

samples). That is mainly because the Exploid root exploit requires certain hot plug events such as changing the WIFI state, which is related to this permission.

Finally, we notice that malicious apps tend to request more permissions than benign ones. In our dataset, the average number of permissions requested by malicious apps is 11 while the average number requested by benign apps is 4. Among the top 20 permissions, 9 of them are requested by malicious apps on average while 3 of them on average are requested by benign apps.

SECTION IV.

Malware Evolution

As mentioned earlier, since summer of 2011, we have observed rapid growth of Android malware. In this section, we dive into representative samples and present a more indepth analysis of their evolution. Specifically, we choose DroidKungFu (including its variants) and AnserverBot for illustration as they reflect the current trend of Android malware growth.

A. DroidKungFu

The first version of DroidKungFu (or DroidKungFu1) malware was detected by our research team [30] in June 2011. It was considered one of the most sophisticated Android malware at that time. Later on, we further detected the second version DroidKungFu2 and the third version DroidKungFu3 in July and August, respectively. The fourth version DroidKungFu4 was detected by other researchers in October 2011 [31]. Shortly after that, we also came across the fifth version DroidKungFuSapp, which is still a new variant not being detected yet by existing mobile antivirus software (Section V). In the meantime, there is another variant called DroidKungFuUpdate [32] that utilizes the update attack (Section III). In Table VI, we summarize these six DroidKungFu variants. In total there are 473 DroidKungFu malware samples in our dataset.

Table VI The Overview of Six DroidKungFu Malware Families

Table VI- The Overview of Six DroidKungFu Malware Families

The emergence of these DroidKungFu variants clearly demonstrates the current rapid development of Android malware. In the following, we zoom in various aspects of DroidKungFu malware.

1) Root Exploits

Among these six variants, four of them contain encrypted root exploits. Some of these encrypted files are located under the directory “assets”, which look like normal data files. To the best of our knowledge, DroidKungFu is the first time we have observed in Android malware to include encrypted root exploits.

The use of encryption is helpful for DroidKungFu to evade detection. And different variants tend to use different encryption keys to better protect themselves. For example, the key used in DroidKungFu1 is Fuck_sExy-aL1!Pw, which has been changed to Stak_yExy-eLt!Pw in DroidKungFu4.

It is interesting to notice that in DroidKungFu1, the file name with the encrypted root exploit is “rate” - the acronym of RageAgainstTheCage. In DroidKungFu2 and DroidKungFu3, this file name with the same root exploit has been changed to “myicon”, pretending to be an icon file.

2) C&C Servers

All DroidKungFu variants have a payload that communicates with remote C&C servers and receives the commands from them. Our investigation shows that the malware keeps changing the ways to store the C&C server addresses. For example, in DroidKungFu1, the C&C server is saved in plain-text in a Java class file. In DroidKungFu2, this C&C server address is moved to a native program in plaintext. Also, remote C&C servers have been increased from 1 to 3. In DroidKungFu3, it encrypts the C&C server addresses in a Java class file. In DroidKungFu4, it moves the C&C address back to a native program as DroidKungFu2 but in cipertext. In DroidKungFuSapp, we observe using a new C&C server and a different home-made encryption scheme.

3) Shadow Payloads

DroidKungFu also carries with itself an embedded app, which will be stealthily installed once the root exploit is successfully launched. As a result, the embedded app will be installed without user's awareness. An examination of this embedded app code shows that it is almost identical to the malicious payload DroidKungFu adds to the repackaged app. The installation of this embedded app will ensure that even the repackaged app has been removed, it can continue to be functional. Moreover, in DroidKungFu1, the embedded app will show a fake Google Search icon while in DroidKungFu2, the embedded app is encrypted and will not display any icon on the phone.

4) Obfuscation, JNI, and Others

As briefly mentioned earlier, DroidKungFu heavily makes use of encryption to hide its existence. Geinimi is an earlier malware that encrypts the constant strings to make it hard to analyze. DroidKungFu instead encrypts not only those constant strings and C&C servers, but also those native payloads and the embedded app file. Moreover, it rapidly changes different keys for the encryption, aggressively obfuscates the class name in the malicious payload, and exploits JNI interfaces to increase the difficulty for analysis and detection. For example, both DroidKungFu2 and DroidKungFu4 uses a native program (through JNI) to communicate with and fetch bot commands from remote servers.

The latest version, i.e., DroidKungFuUpdate, employs the update attack. With its stealthiness, it managed into the official Android Market for users to download, reflecting the evolution trend of Android malware to be more stealthy in their design and infection.

B. AnserverBot

AnserverBot was discovered in September 2011. This malware piggybacks on legitimate apps and is being actively distributed among a few third-party Android Markets in China. The malware is considered one of the most sophisticated Android malware as it aggressively exploits several sophisticated techniques to evade detection and analysis, which has not been seen before. Our full investigation of this malware took more than one week to

complete. After the detailed analysis [33], we believe this malware evolves from earlier BaseBridge malware. In the following, we will highlight key techniques employed by AnserverBot. Our current dataset has 187 AnserverBot samples.

1) Anti-Analysis

Though AnserverBot repackages existing apps for infection, it aims to protect itself by actively detecting whether the repackaged app has been tampered with or not. More specifically, when it runs, it will check the signature or the integrity of the current (repackaged) app before unfolding its payloads. This mechanism is in place to thwart possible reverse engineering efforts.

Moreover, AnserverBot aggressively obfuscates its inter-nal classes, methods, and fields to make them humanly unreadable. Also, it intentionally partitions the main payload into three related apps: one is the host app and the other twos are embedded apps. The two embedded apps share the same name *com.sec.android.touchScreen.server* but with different functionality. One such app will be installed through the update attack while the other will be dynamically loaded without being actually installed (similar to Plankton). The functionality partitioning and coordination, as well as aggressive obfuscation, make its analysis very challenging.

We have the reason to believe that AnserverBot is inspired by the dynamic loading mechanism from Plankton. In particular, the dynamic mechanisms to retrieve and load remote code is not available in earlier BaseBridge malware. In other words, it exploits the class loading feature in Dalvik virtual machine to load and execute the malicious payload at run time. By employing this dynamic loading behavior, AnserverBot can greatly protect itself from being detected by existing antivirus software (Section V). Moreover, with such dynamic capability in place, malware authors can instantly upgrade the payloads while still taking advantage of current infection base.

2) Security Software Detection

Another related self- protection feature used in AnserverBot is that it can de- tect the presence of certain mobile antivirus software. In particular, it contains the encrypted names of three mobile antivirus software, i.e., *com.qihoo360.mobilesafe, com.tencent.qqpimsecure* and *com.lbe.security*, and attempts to match them with those installed apps on the phone. If any of the three antivirus software is detected, AnserverBot will attempt to stop it by calling the restartPackage method and displaying a dialog window informing the user that the particular app is stopped unexpectedly.

3) C&C Servers

One interesting aspect of AnserverBot is its C&C servers. In particular, it supports two types of C&C servers. The first one is similar to traditional C&C servers from which to receive the command. The second one instead is used to upgrade its payload and/or the new address of the first type C&C server. Surprisingly, the second type is based on (encrypted) blog contents, which are maintained by popular blog service providers (i.e., Sina and Baidu). In other words, AnserverBot connects to the public blog site to fetch the (encrypted) current C&C server and the new (encrypted) payload. This functionality can ensure that even if the first type C&C server is offline, the new C&C server can still be pushed to the malware through this public blog, which is still active as of this writing.

SECTION V.

Malware Detection

The rapid growth and evolution of recent Android malware pose significant challenges for their detection. In this section, we attempt to measure the effectiveness of existing mobile antivirus software. To this end, we choose four representative mobile antivirus software, i.e., AVG Antivirus Free v2.9 (or AVG), Lookout Security & Antivirus v6.9 (or Lookout), Norton Mobile Security Lite v2.5.0.379 (Norton), and TrendMicro Mobile Security Personal Edition v2.0.0.1294 (TrendMicro) and download them from the official Android Market in the first week of November 2011.

We install each of them on a separate Nexus One phone running Android version 2.3.7. Before running the security app, we always update it with the latest virus database. In the test, we apply the default setting and enable the real-time protection. After that, we create a script that iterates each app in our dataset and then installs it on the phone. We will wait for 30 seconds for the detection result before trying the next app. If detected, these antivirus software will pop up an alert window, which will be recorded by our script. After the first iteration, we further enable the second-round scanning of those samples that are not detected in the first round. In the second round, we will wait for 60 seconds to make sure that there is enough time for these security software to scan the malware.

The scanning results are shown in Table VII. In the table, the first two columns list the malware family and the number of the samples in this malware family. The rest columns show the number of samples as well as the percentage being detected by the corresponding security software. At the end of the table, we show the number of detected samples for each antivirus software and its corresponding detection rate. The results are not encouraging: Lookout detected 1003 malware samples in 39 families; TrendMicro detected 966 samples in 42 families; AVG detected 689 samples in 32 families; and Norton detected the least samples (254) in 36 families.

Table VII Detection Results From Four Representative Mobile Anti-Virus Software


Table VII- Detection Results From Four Representative Mobile Anti-Virus Software

Apparently, these security software take different approaches in their design and implementation, which lead to different detection ratio even for the same malware family. For example, AVG detects all ADRD samples in our dataset, while Lookout detects 59.0% of them. Also, Lookout detects most of DroidKungFu3 samples and all DroidKungFu4 samples while AVG can detect none of them (0.0%) or few of them (4.1%).

There are some malware families that completely fail these four mobile security software. Examples are BeanBot, CoinPirate, DroidCoupon, DroidKungFuSapp, NickyBot and RogueLemon. One reason is that they are relatively new (discovered from August to October 2011). Therefore, existing mobile antivirus companies may not get a chance to obtain a copy of these samples or extract their signatures. From another perspective, this does imply that they are still taking traditional approaches to have a signature database that represents known malware samples. As a result, if the sample is not available, it is very likely that it will not be detected.

Our characterization of existing Android malware and an evolution-based study of representative ones clearly reveal a serious threat we are facing today. Unfortunately, existing popular mobile security software still lag behind and it becomes imperative to explore possible solutions to make a difference.

First, our characterization shows that most existing Android malware (86.0%) repackage other legitimate (popular) apps, which indicates that we might be able to effectively mitigate the threat by policing existing Android Markets for repackaging detection. However, the challenges lie in the large volume of new apps created on a daily basis as well as the accuracy needed for repackaging detection. In addition, the popularity of alternative Android Markets will also add significant challenges. Though there is no clear solution in sight, we do argue for a joint effort involving all parties in the ecosystem to spot and discourage repackaged apps.

Second, our characterization also indicates that more than one third (36.7%) of Android malware enclose platform-level exploits to escalate their privilege. Unfortunately, the open Android platform has the well-known "fragmentation" problem, which leads to a long vulnerable time window of current mobile devices before a patch can be actually deployed. Worse, the current platform still lacks many desirable security features. ASLR was not added until very recently in Android 4.0. Other security features such as TrustZone and eXecute-Never need to be gradually rolled out to raise the bar for exploitation. Moreover, our analysis reveals that the dynamic loading ability of both native code and Dalvik code are being actively abused by existing malware (e.g., DroidKungFu and AnserverBot). There is a need to develop effective solutions to prevent them from being abused while still allowing legitimate uses to proceed.

Third, our characterization shows that existing malware (45.3%) tend to subscribe to premium-rate services with background SMS messages. Related to that, most existing malware intercept incoming SMS messages (e.g., to block billing information or sidestep the second-confirmation requirement). This problem might be rooted in the lack of fine-grain control of related APIs (e.g., sendTextMessage). Specifically, the coarse-grained Android permission model can be possibly expanded to include additional context information to better facilitate users to make sound and informed decisions.

Fourth, the detection results of existing mobile security software are rather disappointing, which does raise a challenging question on the best model for mobile malware detection. Specifically, the unique runtime environments with limited resources and battery could preclude the deployment of sophisticated detection techniques. Also, the traditional content-signature-based approaches have been demonstrated not promising at all. From another perspective, the presence of centralized marketplaces (including alternative ones) does provide unique advantages in blocking mobile malware from entering the marketplaces in the first place.

Last but not least, during the process of collecting malware samples into our current dataset, we felt confusions from disorganized or confusing naming schemes. For example, BaseBridge has another name AdSMS (by different antivirus companies); ADRD is the alias of Hongtoutou; and LeNa is actually a DroidKungFu variant. One possible solution may follow the common naming conventions used in desktop space, which calls for the cooperation from different mobile security software vendors.

Smartphone security and privacy has recently become a major concern. TaintDroid [34] and PiOS [35] are two systems that expose possible privacy leaks on Android and iOS, respectively. Comdroid [36] [37] and Woodpecker [38] expose the confused deputy problem [39] on Android. Accordingly, researches have proposed several possible solutions [37] [40] [41] to this issue. Stowaway [42] exposes the over-privilege problem (where an app requests more permissions than it uses) in existing apps. Schrittwieser *et al*. [43] reports that certain security flaws exist in recent network-facing messaging apps. Traynor *et al*. [44] characterizes the impact of mobile botnet on the mobile network. AdRisk [45] systematically identifies potential risks from in-app advertisement libraries. Our

work is different from them with a unique focus on systematically characterizing existing Android malware in the wild.

To improve the smartphone security and privacy, a number of platform-level extensions have been proposed. Specifically, Apex [46], MockDroid [47], TISSA [48] and AppFence [49] extend the current Android framework to provide find-grained controls of system resources accessed by untrusted third-party apps. Saint [50] protects the exposed interfaces of an app to others by allowing the app developers to define related security policies for runtime enforcement. Kirin [51] blocks the installation of suspicious apps by examining the existence of certain dangerous permission combination. L4Android [52] and Cells [53] run multiply OSes on a single smartphone for improved isolation and security. Note that none of them characterizes (or studies the evolution of) existing Android malware, which is the main focus of this work.

Among the most related, Felt *et al. [54]* surveys 46 malware samples on three different mobile platforms, i.e., iOS, Android and Symbian, analyzes their incentives, and discusses possible defenses. In contrast, we examine a much larger dataset (with 1,260 malware samples in 49 different families) on one single popular platform - Android. The size of our dataset is instrumental to systematically characterizing malware infection behavior and understanding their evolution. Moreover, the subsequent test of existing mobile security software further necessitates a change for effective anti-mobile-malware solutions.

From another perspective, Becher *et al*. [55] provides a survey of mobile network security, from the hardware layer to the user-centric attacks. DroidRanger [56] detects malicious apps in existing official and alternative Android Markets. DroidMOSS [57] uses the fuzzy hashing to detect the repackaged apps (potential malware) in third-party android markets. Enck *et al. [58]* studies 1,100 top free (benign) Android apps to better understand the security characteristics of these apps. Our work differs from them by focusing on 1, 260 malicious apps (accumulated from more than one year effort) and presenting a systematic study of their installation, activation, and payloads.

In this paper, we present a systematic characterization of existing Android malware. The characterization is made possible with our more than one-year effort in collecting 1260 Android malware samples in 49 different families, which covers the majority of existing Android malware, ranging from its debut in August 2010 to recent ones in October 2011. By characterizing these malware samples from various aspects, our results show that (1) 86.0% of them repackage legitimate apps to include malicious payloads; (2) 36.7% contain platform-level exploits to escalate privilege; (3) 93.0% exhibit the bot-like capability. A further indepth evolution analysis of representative Android malware shows the rapid development and increased sophistication, posing significant challenges for their detection. Sadly, the evaluation with four existing mobile antivirus software shows that the best case detects 79.6% of them while the worst case detects only 20.2%. These results call for the need to better develop next-generation anti-mobile-malware solutions.

ACKNOWLEDGMENT

We would like to thank our shepherd, Patrick Traynor, and the anonymous reviewers for their comments that greatly helped improve the presentation of this paper. We also want to thank Michael Grace, Zhi Wang, Wu Zhou, Deepa Srinivasan, Minh Q. Tran, and Lei Wu for the helpful discussion. This work was supported in part by the US National Science Foundation (NSF) under Grants 0855297, 0855036, 0910767, and 0952640. Any opinions,

findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.
