Detecting Android Malware on Network Level

Danny Iland, Alexander Pucher, Timm Schäuble

December 5, 2011

Abstract

As Android OS establishes itself as the primary platform on smartphones, a substantial increase in malware targeted at Android devices is being observed in the wild. While anti-virus software is available, and Android limits applications to user approved permissions, many users remain unaware of the threat posed by malware and of actual infections on their devices. In this paper we explore techniques to enable mobile network operators to detect Android malware and violations of user privacy through network traffic analysis.

1 Introduction

Android is becoming the prevalent platform for smartphones today, with over 190 million activated devices in use in 130 countries. [4] Users of Android devices can install software from the official Android market, third party app stores, or by direct download and installation of an APK file. The Android Marketplace has no review process and makes it easy to publish a malicious application with relative anonymity. This problem is enhanced by an increasing number of unofficial app stores leading users to circumvent any existing review and reporting mechanisms. Third-party app stores are a viable distribution method for Android Malware, often hosting infected copies of legitimate applications and malware targeted towards users in a particular country. [6]

As worldwide market penetration increases, malware authors are taking notice of this largely unprotected platform. The Juniper Global Threat Center reported a 472% increase in Android Malware samples between July and November 2011. [11] More than half of US smartphone users in a survey sponsored by AVG reported that they do not worry that a hacker will attack their smartphone, and they do not feel their privacy is at risk when they use their smartphone. [5] 62% of respondents in the same survey reported that they do not always check if an app is from a trustworthy source. [5] Since apathy, ignorance, or inaction can lead to infections going unnoticed by the user, a detection mechanism that does not require user interaction is ideal. Detection by cellular providers would allow for notification of users about critical infections or disclosure of private information.

In a series of controlled experiments, we populate our virtualized Android devices with user data and infect them with malware samples. We collect the resulting network traces and analyze them for leaks of personal information and communication with command and control servers. We explore a number of lightweight approaches for detection and show that the suggested techniques are able to identify malicious behavior.

2 Related Work

Detection of malware by identification of network signatures is an established practice, utilized by network intrusion detection systems such as Snort. [8] However, the application of this approach has not been fully studied as it relates to modern mobile devices. While anti-virus companies regularly report on new threats to Android devices, these are typically the results of a single-binary analysis. One

large-scale analysis of network traffic generated by Android malware comes from Dasient, who performed automated analysis on 10,000 android applications from the Google Android marketplace. [3] Of those 10,000 applications, 8.4% were found to leak the International Mobile Equipment Identifier from the device. [3] Efforts to detect Android malware through dynamic analysis are being made by DroidBox, an open source project utilizing Googles Android Virtual Device to log Android application behavior. [7]

A Symantec report on monetization in Android malware identifies only a few approaches to making money with Android malware. Simple applications that send premium rate text messages are among the most prevalent. Malware such as Fakeplayer and AdSMS will only generate profit in Russia and China, respectively, since those are the countries where the included numbers are registered. Android spyware focuses on extracting information from the device, from stealing user credentials to uploading GPS coordinates and text messages. Adware on Android often fetch and display advertisements inside trojan applications, or simulate mobile search engine users to profit from pay per click schemes. [2]

3 Implementation

The experimental setup uses VirtualBox 4.0 as x86 emulator, running Android x86 2.2 generic and eeepc images. The VM is populated with dummy user data and a set of clean user applications. This image is then duplicated, executed and network traffic at VM boundaries is captured using the pcap library. We initially link VMs to an unprotected Internet connection for observing malware communication. Optionally, we perform a second pass using a DNS and HTTP server installed on the host machine to mock C&C servers that had been taken offline in the past.

In order to simulate a real users device, we created a virtual machine image containing contact data, account passwords, browser history and credit card information. We also installed a third party AppStore of a well-known e-commerce company, which requires users to allow the installation of applications from unknown source. This virtual machine image formed the baseline for our experiments. Following this, an infected application is installed. The infected application is then run by the user and a default browsing routine is performed. The VM is then left running idle for a period of at least 30 minutes and terminated before the network traces are analyzed using wireshark.

Our analysis of packet traces focuses on finding information leakage in HTTP traces and identifying connection attempts to command-and-control servers. Conversions containing International Mobile Equipment Identity number (IMEI), phone number or credit card information were tracked. If no abnormalities are detected so far, the packet dump is compared manually to a dump generated by the uninfected VM template image to determine whether the sample was not detected or simply inactive.

Our analysis of packet traces focuses on finding information leakage in HTTP traces and identifying connection attempts to command-and-control server DNS and IP-addresses. Conversions containing IMEI, phone number or credit card information were tracked, as well as unexpected binary downloads. Finally, if no abnormalities are detected so far, the packet dump is compared manually to a dump generated by the uninfected VM template image.

4 Results

4.1 Blacklisting

The vast majority of the malware we investigated uses DNS in order to obtain the ip address of their command and control servers. Since most Android malware is relatively simple in design, malware authors may not have considered this single point of failure, or put in the required effort to increase resilience. The Android.Crusewind Trojan for example is doing a DNS lookup for "crusewind.net" and then sends information to the the obtained IP address.

```
Protocol    Info
DNS         Query A crusewind.net
DNS         Query response A 209.85.51.152
HTTP        GET /flash/test.xml?[...]
```

Figure 1: Android/Crusewind DNS lookup

```
Protocol    Info
DNS         Query A amaz0n-cloud.com
DNS         Query response A 1.2.3.4 (fake DNS)
HTTP        POST /droid/droid.php HTTP/1.1 [...]
```

Figure 2: Android/Ggtracker DNS lookup with fake response

We also discovered that some samples, e.g. Android.Ggtracker, try to make their traffic less obvious for trustful users by mimicing domain names of well-known services.

During our research we did not discover any malware resilient to DNS blacklisting. If Android malware writers begin utilizing the more advanced techniques common in PC malware, blacklisting becomes a struggle between malware authors and researchers. Even faced with an adversary, it is possible to maintain blacklists through honeypot infections or, more effectively, reverse-engineering domain generation algorithms. [10] Hence a straight forward approach to detect a majority of Android malware infections is to create a blacklist of known domains. Our investigation also shows that required domain names can be extracted effectively from most binaries using string search.

4.2 String matching

The general approach of text-matching is the reliance on regular expressions or partial hashes of plain-text data transfers. We specifically focus on HTTP-based conversations as these make up the method of choice for information leaks in our set of samples.

4.2.1 HTTP header flags in server responses

Another approach could analyze HTTP server responses that apply uncommon header flags or build on uncommon software stacks. Header flags could include combinations of no-store, no-cache, etc. in combination with download activity. Specific combinations of these headers can be used as an indication for suspicious traffic. For example, an unclassified application love_more_and_more was found that hints intermediary proxies not to cache any transmitted data. It later continues to download and install software packages without user notification or interaction.

4.2.2 Literals and identifiers in GET requests

Another approach could match the client-side portion of HTTP-GET requests for specific variable names or values. Variable names can include hints about content such as imei. As we discuss in 4.2.4, values can be analyzed for syntactic matches to well-structured information such as IMEI, IMSI, and credit card numbers. In the example below, a conversation of Android/Crusewind is shown. Please note that our experimental setup only allows the usage of a null IMEI.

4.2.3 Contents and patterns of POST requests

Some malware authors seek predictable information about a device. In cases where we can identify what information the authors seek, we can use it to detect a message to the command and control server. For example, Android/Plankton.A! sends POST replies containing information about the infected device. One such request transmits a list of all permissions requested by the installed malware. Another delivers all available device information, from operating system version and display size to browser version information. Many malware authors are likely to seek the same data from devices, and similarities in both structure and data between responses can be exploited.

```
GET /client/xml/check_update/*/p:love_more_and_more/v:5/c:024 HTTP/1.1
User-Agent: Dalvik/1.2.0 (Linux; U; Android 2.2.2; eeepc Build/FRG83G)
Host: api.go108.cn
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 21 Nov 2011 09:00:13 GMT
Server: Apache
Set-Cookie: PHPSESSID=f6b7d05b2c8c1a1cee146b5956aaf895; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
```

Figure 3: Unclassified sample using multiple HTTP-header flags

```
GET /flash/test.xml?imei=null&time=01:57 HTTP/1.0
Accept: */*
User-Agent: Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; eeepc/eeepc-eng 2.2.2 FRG83G [...]
Host: crusewind.net
Connection: Keep-Alive
```

Figure 4: Android/Crusewind leaking IMEI in HTTP-GET request

```
POST /ProtocolGW/protocol/commands HTTP/1.1
Content-Type: application/json
[...]

{"initiationType":"schedule","needSpecificParameters":true,"applicationDetails":
{"applicationId":"325842966#752469853","build":{"brand":"generic_x86","device":
"eeepc","manufacturer":"asus","model":"eeepc","versionRelease":"2.2.2",
"versionSDKInt":8},"deviceId":null,"displayMetrics":{"density":1.5,"densityDpi"
:240,"heightPixels":600,"scaledDensity":1.5,"widthPixels":800,"xdpi":160.0,
"ydpi":160.42105},"locale":"hd_US","protocolVersion":"0.0.2","userAgent":
"Mozilla/5.0 (Linux; U; Android 2.2.2; hd-us; eeepc Build/FRG83G)
AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1","userId"
:"NOT IN USE!!!"}}
```

Figure 5: Android/Plankton.A leaking IMEI and IMSI

A specific and stateful matching technique could listen into the details of POST and PUT requests and seek out well-known communication patterns and questionable transfers taking the form of user or device IDs.

4.2.4 Well-structured identifiers in POST requests

A different approach is matching of cetain types of well-structured data that should not typically be transferred across the network in plain-text. From our investigation it became clear that the International Mobile Equipment Identity number is a prime candidate for detecting information leaks. Specifically, sequences of 14 digits can be identified and compared to the sending devices IMEI. Even if the IMEI of the origin device is not available Luhns algorithm can be used to identify a potential IMEI. [1] The same technique could be applied furthermore to detect unencrypted transmission of IMSI or credit card numbers which use comparable check digits.

Because of the simple and well defined structure of IMEIs, it is feasible to build lookup tables for IMEIs and IMEI hashes. In particular the first 8 digits are a Type Allocation Code, each corresponding to a model of phone. [1] This leaves only 7 digits of unique input, making building complete lookup tables for IMEIs from popular phones and hashing algorithms space-feasible. For example, all HTC Photon devices will have an IMEI of the form 35721503XXXXXXX.

5 Evaluation

The follow table gives an overview of malware we investigated and the identifying properties that were used for detection on network-level with the approaches explored above. Although some C&C servers of known malware are down as of the time of this study, we were able to observe client-side communication using mock DNS and HTTP server responses. In total, 18 samples were investigated, generating traces compared against the patterns of identifying information. Of those, 8 samples were detected, 2 evaded detection and 8 samples failed to execute in the virtualized environment.

Looking at the active samples only, detection rate is acceptable at 80%. With the small sample size and large number of inactive samples it is clear however further tests are necessary in a real-world environment to reliably determine the detection rate. Some malware failed to execute, or was otherwise inactive. These samples are flexispy, lovetrap, kungfu, droiddeluxe, basebridge, ggtracker, netisend and droiddream. Spygold and zsone executed, but were not detected.

Detected malware:

Name	Identifying information
crusewind	crusewind.net, http-text, IMEI
walkandtext	incorporateapps.com, http-text IMEI, phone number
tonclank	searchwebmobile.com, http-json IMEI
bgserv	www.youlubg.com, IMEI, phone number
smspacem	biofaction.no-ip.biz, http-soap phone number
lovetrap	cooshare.com, http-text positionrecorder.asmx, IMSI
? (DL/installer)	api.go108.cn, http-xml no-store no-cache, love_more_and_more
HippoSms.A	info.ku6.cn, http-text, IMEI

Table 1: list of identifying information

6 Limitations

Our approach to detection on the network level is inherently limited by a number of factors. Endpoint-related detection such as DNS and IP-address blacklisting heavily rely on static malware behavior, false-negatives are inevitable. Addresses cannot be blacklisted until a malicious application is identified and the connections it makes are analyzed. The process of generating a blacklist from this analysis ranges from simple to impossible. Most of the malware identified on Android devices currently use command and control addresses statically assigned in

the binary, ideal for automated blacklisting. More complex approaches, such as fast-flux or peer to peer communication patterns, are more difficult to effectively detect.

Also the blacklist approach is likely to generate some rate of false positives. IP addresses and domain names frequently change hands, and what was a command and control server a month ago many not be today. Stale blacklists could cause compromised hosts to be flagged for a long period of time even though an infection was temporary and resolved. False positives are likely in the application layer content as well. Malware writers record IMEIs to uniquely identify infected devices, and although the practice is discouraged, some legitimate applications do the same. Malware authors request information about the operating system and web browser of a device, but so do legitimate applications. These transmissions would be identified as malicious traffic. These false positives could serve as useful reports, however, from a user privacy perspective. Since we are operating at the provider level we can not actively remove an infection. Alerts can be used to notify users or, in the worst case, to terminate data or premium SMS service only. Hence, we do not strive to achieve zero false-positive rate.

Content matching techniques and heuristics mainly infer information from plain-data transmissions. End-to-end encryption could nullify this detection mechanisms, especially when relying on protocols common for commercial applications such as SSL as this makes malicious conversations indistinguishable from benign traffic from a content perspective.

Our use of Android x86 virtual machines also introduces several limitations. Primarily, the x86 environment, while compatible with most Android software, caused some malware samples to fail to run. Also, an Android x86 VM is not a cellphone, it does not support text messages, and has a non-standard IMEI and IMSI. Requests of IMEI and IMSI return the null value.

7 Future Work

For our network-level malware detection system to be viable in an evolving environment, it must rely on automatically generated signatures. Dynamic and Static analysis tools such as DroidBox and Androguard generate real-time reports on the behavior of Android binaries. These open-source tools could be modified to extract addresses of command and control servers, message formats, and premium rate SMS numbers used. The Honeynet Project is developing and releasing an open source Scalable Tailored Application Analysis Framework, designed for scalable analysis of a very large number of Android binaries. Our improved DroidBox could be incorporated with STAAF to analyze large numbers of android applications and generate blacklists and signatures suitable for deployment to network-level deep packet inspection detection systems.

Integrating text message logs into our solution would be invaluable. Text messages sent by malware are likely to be detectable or distinguishable from those sent by users. Literature suggests that a large portion of Android Malware sends text messages to premium rate numbers, most of which are hardcoded into the malware binary. [2] Additionally, Android spyware such as BGServ and GoldDream leak information about text message traffic, sending the source, destination, and content of SMS traffic to a third party via SMS or HTTP. [6] With access to text messaging logs, Android spyware of this type can be detected.

Detecting infected APKs as they are transferred to a users device is another viable approach. When an APK is identified in network traces, the installation URL can be checked against a blacklist and whitelist of known sources. If a source is unknown, the APK file can be extracted and submitted for analysis. If the APK matches a known malware family, or dynamic analysis shows suspicious behaviors, the APK and the URL it came from can be flagged as malicious, and the user notified that some software they have recently installed may be malicious.

8 Conclusion

In this paper we discussed lightweight approaches to detecting Android malware on the network level. Most malware on Android devices we observed uses very basic communication techniques, specifically static C&C server addresses and plain-text transmission of data. Our preliminary results show that the presented detection techniques are viable, but large-scale testing is required to determine real-world performance. As Android malware evolves the effectiveness of these measures will decrease. However these techniques can still be valuable as they raise the bar of entry for repackaged and newly created malware and come at low overhead.

References

[1] GSM Association. Imei allocation and approval guidelines, 2010. Available online at http://www.gsmworld.com/documents/DG06_v5.pdf; visited on December 4th 2011.

[2] Eric Chien. Motivations of recent android malware. Available online at http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/motivations_of_recent_android_malware.pdf; visited on December 4th 2011.

[3] Gerry Eisenhaur, Michael N. Gagnon, Tufan Demir, and Neil Daswani. Mobile malware madness and how to cap the mad hatters, 2011. Available online at https://media.blackhat.com/bh-us-11/Daswani/BH_US_11_Daswani_Mobile_Malware_Slides.pdf; visited on December 4th 2011.

[4] Google management discusses q3 2011 results, 2011. Available online at http://seekingalpha.com/article/299518-google-management-discusses q3-2011-results-earnings-call-transcript; visited on December 4th 2011.

[5] New avg study reveals smartphone users not aware of significant mobile security risks, 2011. Available online at http://www.avg.com/gb-en/press-releases-news.ndi-973; visited on December 4th 2011.

[6] Xuxian Jiang. Security alert: New android malware – golddream – found in alternative app markets. Available online at http://www.cs.ncsu.edu/faculty/jiang/GoldDream/; visited on December 4th 2011.

[7] Droidbox. Available online at http://code.google.com/p/droidbox/; visited on December 4th 2011.

[8] Martin Roesch and Stanford Telecommunications. Snort - lightweight intrusion detection for networks, 1999.

[9] A primary profit mechanism in Android Malware is to send premium text messages to predefined numbers. Malware such as Fakeplayer and AdSMS will only generate profit in Russia and China, respectively, since those are the countries where the included numbers are registered.

[10] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Imei allocation and approval guidelines, 2009. Available online at http://www.cs.ucsb.edu/~seclab/projects/torpig/torpig.pdf; visited on December 4th 2011.

[11] Mobile malware development continues to rise, android leads the way, 2011. Available online at http://globalthreatcenter.com/?p=2492; visited on December 4th 2011.