Understanding the Market-level and Network-level Behaviors of the Android Malware Ecosystem

Chao Yang
Niara, Inc.

Jialong Zhang
IBM Research

Guofei Gu
Texas A&M University

***Abstract*—The prevalence of malware in Android marketplaces is a growing and significant problem. Most existing studies focus on detecting Android malware or designing new security extensions to defend against specific types of attacks. In this paper, we perform an empirical study on analyzing the market-level and network-level behaviors of the Android malware ecosystem. We focus on studying whether there are interesting characteristics of those market accounts that distribute malware and specific networks that are mainly utilized by Android malware authors. We further investigate community patterns among Android malware from the perspective of their market account infrastructure and remote server infrastructure. Spurred by these analysis, we design a novel community inference algorithm to find more malicious apps by exploiting their community relationships. By using a small seed set (50) of known malicious apps, we can effectively find another extra 20 times of malicious apps, while maintaining considerable accuracy higher than 94%.**

I. Introduction

With the boom of Android applications (apps), a torrent of diverse Android malware has been released recently. Different from spreading desktop malware, Android malware authors can utilize Android markets to spread malware more efficiently. According to a recent report from [4], app markets become significant sources for spreading Android malware.

While most existing research efforts are spent on detecting Android malware [8], [11], [13], [23], [29]–[31] or designing new security extensions to defend against specific types of attacks [9], [12], we still lack some basic insights on the whole ecosystem of spreading Android malware. It is known that malware authors typically need to submit Android malware to the markets to attract victims' downloads, and build remote servers to communicate with the malware to achieve malicious goals (e.g., C&C control and compromising victims' privacy). However, the characteristics of the **market-level behaviors** and **network-level behaviors** of the Android malware ecosystem are still not well understood.

In this paper, we empirically perform a systematic measurement study on analyzing the market-level and network-level behaviors of the Android malware ecosystem. We crawled and analyzed over 82,000 Android apps and 28,000 Android market accounts from multiple representative markets (including both official and third-party markets). We further obtain a dataset of over 9,700 malicious Android apps, and another dataset of over 3,500 malicious market accounts that distribute at least one malicious app to the market. To facilitate the analysis of network-level behaviors, we also extract networking attempts of Android apps by running them in a customized Android runtime environment with randomly generated UI events. In total, we obtain over 239,000 unique URLs leading to over 25,000 unique remote servers.

To understand the market-level behaviors, we investigate whether there are any special characteristics of those market accounts that distribute malware. We investigate whether specific metrics, such as the popularity of the app are effective indications to the quality of Android apps or not. In particular, we investigate whether malicious accounts have specific temporal behavioral patterns in submitting malware samples.

To understand the network-level behaviors, we investigate whether there are any special network regions frequently utilized by Android malware authors to host their remote servers and whether there are any large communities among Android malware. Through the analysis, we aim at understanding more deeply on how Android malware is spread, and generating new defense insights against Android malware.

Motivated by the market and network behaviors of the Android malware ecosystem, we further designed a novel algorithm (`AMIA`) to infer more malicious apps. With the help of a small seed set of known malicious apps, AMIA can effectively find another extra 20 times of malicious apps with a considerable accuracy 94%.

II. Android Malware Ecosystem

A typical Android malware ecosystem includes malware developing, malware uploading, malware downloading and malware running. Figure 1 shows the working flow of the Android malware ecosystem.

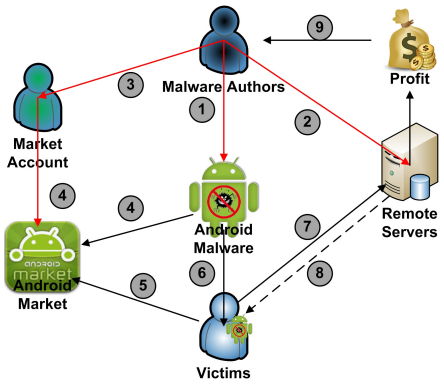


Fig. 1. Working flow of Android malware ecosystem.

The flow begins with Android malware authors developing Android malware (①). To achieve malicious goals such as

compromising victims' privacy, malware authors typically need to build remote servers (②) to communicate with (or control) their malware samples. Next, malware authors require to register Android market accounts (③) to upload malware on specific Android markets (④). After successfully attracting victims' attention and obtaining their trust, the malware will be downloaded and further installed on victims' smartphones (⑤, ⑥). Once the victims' phones are infected, the malware typically communicates with the remote servers, to send out private/system information (⑦), or even to further receive instructions from remote servers (⑧). Finally, malware authors will obtain profits by selling victims' sensitive data or stealthily charging victims' mobile bills (⑨).

Throughout this process, we can see that Android malware authors typically require two types of behaviors: utilizing Android markets to spread malware and building remote servers to communicate with malware. Our research goal is to provide the first empirical analysis of the characteristics of the market-level and network-level behaviors of the Android malware ecosystem, and provide new defense insights.

III. Data Collection

A. Crawling Android Apps

To achieve our research goals, we crawled Android apps from four representative Android markets: the official Android market (*GooglePlay*) and three representative third-party Android markets (*SlideMe* from USA, *Anzhi* from China, and *Tapp* from Russia). SlideME is a leading independent Android Application Marketplace, "powering over 140 OEM's preloaded with the SlideME Market, positioning SlideMe second to Google Play in terms of global reach for Android Apps and Games distribution" [2]. Anzhi is one of the most popular Chinese third-party Android markets, which has over 11 million registered users and whose apps are averagely downloaded over 600 million times per month until the second quarter of 2014 [1]. Tapp is one of the largest and most popular Russian third-party Android markets.

Our crawler downloaded all free apps that were available in these third-party markets. Due to the crawling rate limit and the large amount of apps in GooglePlay, our dataset of official apps were randomly sampled from all 33 app categories in GooglePlay. Moreover, during the crawling process, besides downloading Android apps, our crawler also recorded those apps' market information (e.g., author, submission time, downloading number, and app category). Table I shows the number of collected Android apps for each market.

TABLE I
SUMMARY OF CRAWLING ANDROID APPS

	GooglePlay	SlideMe	Anzhi	Tapp
Location	U.S.A	U.S.A	China	Russia
Creation Time	2008	2008	2010	2012
Number of Unique Apps	18,751	15,109	38,458	11,822
Total (Unique)	18,751 (22%)	65,232 (78%)		
	82,966			

B. Identifying Android Malware

Next, we identified malicious apps from our Android app corpus by searching their values of MD5 in VirusTotal [6], which is a free anti-virus blacklist service providing the scanning reports from over 40 different anti-virus products. For each app, if it has been seen by VirusTotal, we obtained its full scanning report, which includes the first and the last time the app was seen, as well as the results from each individual virus scan. We consider an app to be malicious, if it is labeled as malware by at least one anti-virus product.

Table II shows the Android malware distribution for each market. We totally obtained 9,712 unique malicious apps, We term this dataset of malicious apps as `MalApps`. Apart from the dataset of 9,956 adware (`AdwareApps`), we term the dataset of the rest 63,298 apps as `RestApps`. Note that since a few anti-virus tools consider those non-malicious apps that use certain advertisement libraries as adware, to better guarantee the accuracy of our measurement results, we distinguish the adware from those truly malicious apps.

TABLE II
SUMMARY OF COLLECTING ANDROID MALWARE

	GooglePlay	SlideMe	Anzhi	Tapp
MalApps	1,593	1,946	4,840	1,450
Total(Unique)	1,593 (16%)	8,229 (84%)		
	9,712			
AdwareApps	1,037	1,247	6,764	994
Total(Unique)	1,037 (12%)	8,977 (88%)		
	9,956			
RestApps	16,121	11,916	26,854	9,378
Total(Unique)	16,121 (29%)	38,726(71%)		
	63,298			

IV. Market-level Behaviors

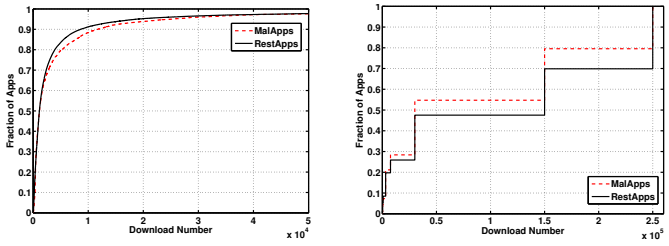
Different from desktop malware authors, who typically have to build their own platforms/websites to spread malware, Android malware authors can spread malware more effectively by utilizing popular Android markets. In this section, we first collected market accounts (uniquely identified by the author name) from those four representative markets and extracted malicious accounts that at least submitted one malicious app. Table III shows the overall accounts and malicious accounts for each market. Then we study the market behaviors of Android apps from the following perspectives.

TABLE III
SUMMARY OF COLLECTING MARKET ACCOUNTS

	GooglePlay	SlideMe	Anzhi	Tapp
# of Accounts	10,064	3,896	9,665	4,871
Total (Unique)	10,064 (35%)	18,432 (65%)		
	28,496			
# of Malicoius Accounts	883	432	1,493	709
Total(Unique)	883 (25%)	2,634 (75%)		
	3,517			

App Quality vs App Popularity. Many users prefer to trust those popular apps that have a large number of downloads. *Question 1: Are apps with higher downloading numbers safer?*

Empirical Answer: No. An app's downloading number does not have a strong correlation with its quality. Many malicious apps have been downloaded for a great number of times.

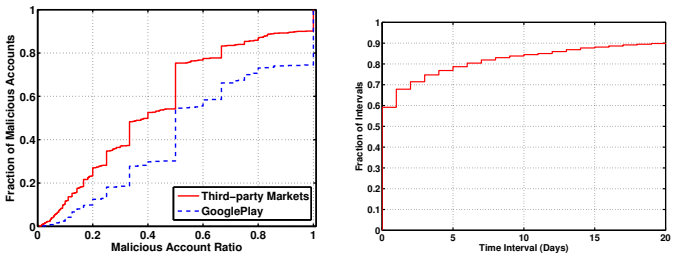


(a) Third-party Markets (b) GooglePlay

Fig. 2. App downloading numbers distribution.

Figure 2 shows the downloading numbers distribution in both third-party markets and GooglePlay, we can find that the distributions of the downloading number between `MalApps` and `RestApps` are similar. Specifically, in the third-party markets, the percentage of the apps in `MalApps` that have been downloaded more than 10,000 times is around 12%, which is even slightly larger than that (10%) in `RestApps`. Also, we can find that around 20% of apps in `MalApps` have been downloaded more than 5,000 times, whereas around 80% of apps in `RestApps` have been downloaded less than 5,000 times. Similar results can be seen for GooglePlay. Thus, such observation indicates that many popular apps with high downloading numbers are still malicious and an app's popularity is not an effective indicator to its quality.

Common Behaviors Among Malicious Accounts. *Question 2: Are there any common behavioral characteristics among malicious market accounts?* Our Empirical Answer: Yes. There is a spatial and temporal locality property in terms of malware authors' submissions of their malware samples. Malicious authors tend to repeatedly use the same accounts to post multiple malicious apps, and within a short time period.



(a) Account Malicious Ratio (b) Time Interval

Fig. 3. The distribution of account malicious ratios, and time intevals between two consequent malware submissions from the same malicious account.

To answer this question, for each of 3,517 malicious market accounts, we first calculate its account malicious ratio, which is the percentage of malware samples in all of its submitted apps. As seen in Figure 3(a), around 50% of malicious accounts from third-party markets, and around 70% of malicious accounts from GooglePlay have a malicious ratio higher than 0.5. This observation indicates that malware authors tend to repeatedly use the same malicious accounts to submit malicious apps. Especially, we find that malicious accounts in GooglePlay typically have a higher ratio than those malicious accounts in third-party markets. That might be because GooglePlay requires a higher cost for registering accounts (e.g., valid Google accounts and registration fee) than third-party markets (e.g., SlideMe requires valid email addresses and physical contact addresses).

We also examine the time interval between two consequent submissions of malicious apps for the same malicious accounts. As seen in Figure 3(b), over 60% of time intervals are zero (i.e., those two consequent submissions happened in the same day), and around 80% of time intervals are less than 5 days. This observation indicates that malware authors tend to submit multiple malware samples within a short time period.

V. Network-level Behaviors

To facilitate the analysis of the network-level behaviors, we extract remote servers by running apps in a customized Android runtime environment (Android phone emulator). Before analyzing each app, the emulator will start from a clean snapshot to avoid possible effects generated by other apps. To trigger an app to execute more networking connections, we use Monkey to simulate real users' usage of the app by adding random UI events (e.g., click buttons, stretch the views, and type characters). As seen in Table[1] IV, we finally collect 239,582 unique URLs leading to 25,099 unique servers (including 19,342 domain names and 5,755 IP addresses).

TABLE IV
The summary of extracting remote servers.

Type	URLs	Domains	IPs	Servers[1]
MalServers	34,176	3,980	1,162	5,142
AdwareServer	35,267	3,112	1,057	4,169
RestServers	176,949	16,580	5,125	21,707
Total	239,582	19,342	5,755	25,099

[1] Each remote server is counted by one of its valid domain names, if available. Otherwise, the server is counted by its IP address.

Since our goal is to analyze the remote servers uniquely used by malware, we need to filter benign servers that are also visited by Android malware. We first filter top 10,000 Alexa [3] domain names. Then, we use two conservative strategies to further filter benign servers: (1) filtering all servers visited by apps in `RestApps`, thus generating a filtered dataset named as `FAMalServers` with 2,288 unique servers, and (2) filtering top frequently used servers by apps in `RestApps`, resulting in a filtered dataset named `FTMalServers` with 4,379 servers. We clearly acknowledge that these filtered datasets may still contain some benign servers, and miss some malware servers. However, these two strategies are essentially complementary, If our conclusions could hold under the usage of both two datasets, we believe that the same conclusions will also likely hold under the usage of another real dataset.

[1]The datasets of `MalServers`, `AdwareServers` and `RestServers` in the table represent the servers extracted from the apps in the dataset of `MalApps`, `AdwareApps` and `RestApps`, respectively.

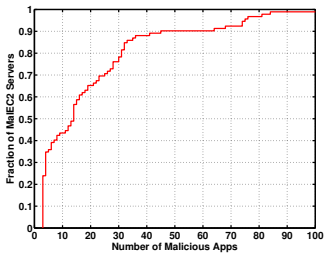
We now perform our detailed analysis of network behaviors of Android apps from following perspectives.

Usage of Network Space. *Question 1: Are there any special network regions frequently used by Android malware authors?* Our Empirical Answer: Yes. We find that Android malware authors tend to use cloud services to host their remote servers.

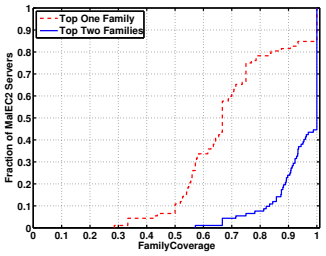
We extract the IP addresses of the remote servers and its corresponding Autonomous system (AS) names. Then, in each dataset, we rank the ASes according to the number of unique apps that visit them. Table V shows the top ten most frequently visited ASes in each dataset. We can find that in `RestServers`, only one of the top ten ASes belongs to the cloud vendors and it ranks the sixth. However, in `MalServers`, four of the top ten ASes belong to cloud vendors; in `FAMalServers`, six of the top ten ASes are allocated to five different cloud vendors, and the top AS (AS14618) belongs to the popular cloud vendor AmazonEC2. This observation implies that malware authors tend to host their remote servers in cloud vendors, which further motivate our analysis of those apps that share the same cloud vendors.

To further study the characteristics of Android malware samples whose remote servers are hosted in the cloud vendors, we use the popular cloud vendor AmazonEC2 as a case study. We extract AmazonEC2 servers, termed as `MalEC2Servers`, visited by malicious apps. Then, for each `MalEC2Server`, we extract its `MalEC2Families`, which are the malware families of the malware samples that visit that server.

We extract 92 different MalEC2Servers in total. Figure 4(a) shows the distribution of the number of unique apps in those 92 MalEC2Servers. We can find that these MalEC2Servers tend to be visited by multiple malicious apps (i.e., around 58% of those MalEC2Servers are visited by more than 10 different malicious apps).



(a) Number of Malicious Apps



(b) FamilyCoverage

Fig. 4. The distributions of the number of malicious apps, and the famliy coverages among MalEC2Servers.

For each MalEC2Familiy in each MalEC2Server, we calculate its value of FamilyCoverage, which is the percentage (coverage) of the malware samples belonged to the MalEC2Family in all malware samples that visit MalEC2Server. Next, in each MalEC2Server, we rank its MalEC2Families according to their values of FamilyCoverage. Figure 4(b) shows the distribution of the FamilyCoverage of the top MalEC2Family, and the sum of the top two values of FamilyCoverage, among those 92 servers. We can find that in over 90% of MalEC2Servers, the top MalEC2Family's FamilyCoverage is higher than 0.5 (i.e., more than half malware samples belong to the same family). While considering the sum of the top two families, the coverage is increased to 0.85 (i.e., the top two families coverage over 85% of malware samples). Also, we can see that in over 50% of MalEC2Servers, the sum of the top two families is 1.0 (i.e., over half of MalEC2Servers have only two families). These observations imply that the malware samples that visit the same AmazonEC2 server tend to belong to the same family. In other words, with the popularity of cloud hosting services, malware authors begin to use cloud machines as remote servers. That is mainly because comparing with deploying personal servers, it will cost less time/money to use a cloud vendor, and the anonymity is also better preserved.

Malware Community. Due to the observation that malware samples frequently share the same authors and remote servers, we next analyze whether the submissions of the malicious apps are more likely to be organized activities or isolated actions. *Question 2: Are there any Android malware communities?* Our Empirical Answer: Yes. A few large malware communities contribute to a great amount of malicious apps.

In this experiment, we cluster malicious apps into communities according to their community relationships. More specifically, we consider there is a *community relationship* between two malicious apps if they share the same author name or at least one malicious server.

To model such community relationships among malicious apps, we build a community relationship graph $G = (V, E)$. In this graph, each node (v_i) is represented as a two-tuple (app, author name). There is an edge e_{ij} between node v_i and v_j, if there is a community relationship between them. Accordingly, our relationship graph contains 9,850 nodes and 621,166 edges. The majority (80.67%) of the nodes in the relationship graph are connected with other nodes. Also, there are a few large subgraphs that are well connected, which implies that there are some large malware communities.

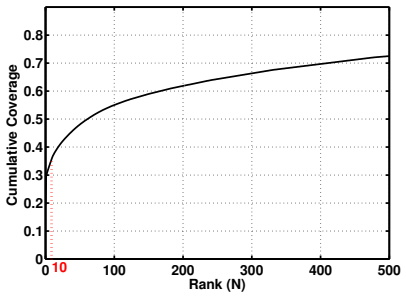


Fig. 5. The distribution of the cumulative community coverage.

We next examine the percentage of the malware samples covered by those large communities in all malware samples. More specifically, we consider each connected subgraph as one community, and thus obtain 847 communities. We next rank those communities based on their size. For each community, we calculate its coverage (c_i), which is the percentage of its malware samples in all malware samples. Then, summing up the coverages from the top community to the nth community,

TABLE V
TOP TEN ASES FOR RESTAPPS AND FAMALAPPS

	RestServers		MalServers		FAMalServers	
Rank	**AS Number**	**AS Name**	**AS Number**	**AS Name**	**AS Number**	**AS Name**
1	AS15169	Google	AS9308	Abitcool	AS14618	Amazon (**Cloud**)
2	AS4134	Chinanet	AS4134	Chinanet	AS4134	Chinanet
3	AS9308	Abitcool	AS15169	Google	AS15169	Google
4	AS4808	China169 Beijing	AS17964	Beijing Dian-Xin-Tong (**Cloud**)	AS16509	Amazon (**Cloud**)
5	AS24400	Shanghai Mobile	AS23724	IDC, China (**Cloud**)	AS4837	China169 Backbone
6	AS14618	Amazon (**Cloud**)	AS14618	Amazon (**Cloud**)	AS36351	SoftLayer (**Cloud**)
7	AS22577	Google	AS24400	Shanghai Mobile	AS4808	China169 Beijing
8	AS4837	China169 Backbone	AS17431	Beijing TONEK	AS37963	Alibaba (**Cloud**)
9	AS17431	Beijing TONEK	AS3549	Global Crossing	AS26496	GoDaddy (**Cloud**)
10	AS20645	PurePeak Limited	AS33494	IHNetworks(**Cloud**)	AS24940	Hetzner (**Cloud**)

we calculate the cumulative coverage as $C_n = \sum_{i=1}^{n} c_i$. Figure 5 shows the distribution of the cumulative coverage with the value of n. We can find that the top 100 communities cover over 55% of all malware samples which implies that a few communities contribute to a large number of malware samples.

From the above observation, we can conclude that those malicious apps have strong community relationships, which can be applied to find more malicious apps.

VI. COMBATING MALICIOUS APPS

In this section, we propose a lightweight Android malware inference algorithm (AMIA) to infer malicious apps based on the market-level and network-level behaviors of Android malware ecosystem.

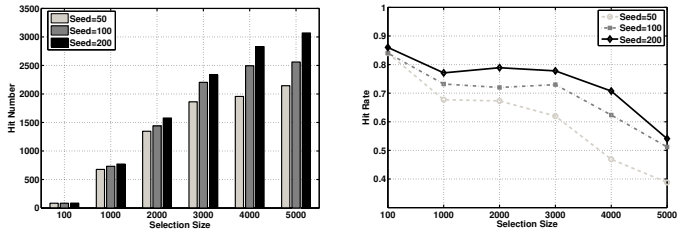
A. Design of Inference Algorithm

In brief, our inference algorithm (AMIA) propagates malicious scores from a seed set of known malicious apps to other apps according to the closeness of their community relationships. If an app accumulates a sufficient malicious score, it is more likely to be a malicious app.

Specifically, given a set of unknown apps U and a seed set of known malicious apps M. We build a Malicious Relevance Graph $G = (V, E)$ by using these $(M + U)$ apps. In this graph, each node (v_i) is represented as a two-tuple (app, author name). There is an edge e_{ij} between node v_i and v_j, if these two nodes has the same app (i.e., the same value of MD5) or the same author name, or their apps share at least one remote server. Then, for each node whose app is malicious, we assign a non-zero malicious score and propagate this score to other nodes according to the weights of the edges between them by using the PageRank algorithm [5]. When the score vector converges after several propagation steps, we infer the apps in those nodes with high malicious scores as malicious apps.

B. Evaluation

To evaluate the effectiveness of AMIA, we consider both the number of correctly inferred malicious apps, termed as "Hit Number", and the ratio of Hit Number to the total number of inferred apps, termed as "Hit Rate". Thus, a higher Hit Number indicates AMIA can catch more malicious apps; and a higher Hit Rate indicates AMIA can infer malicious apps more accurately. Next, we provide our evaluation results by varying different selection sizes (i.e., the number of apps inferred in the top list), and different seed sizes.



(a) Hit Number (b) Hit Rate

Fig. 6. The hit number and hit rate based on the ground truth of VirusTotal.

Varying Selection Size: As shown in Figure 6(a), while increasing the selection size, more malicious apps could be identified by AMIA. This implies that our lightweight algorithm can be effectively used to infer more malicious apps. Also, as seen in Figure 6(b), the Hit Rate decreases with the increase of the selection size. That is mainly because the apps with higher malicious scores are more likely to be malicious.

Varying Seed Size: As in seen in Figure 6, the more seeds we use, the higher Hit Number and Hit Rate we can achieve by selecting the same size of apps. This is because when we use more malicious seeds, we have more knowledge about the community relationships among malicious apps. In addition, many of those inferred apps do not share the same family (type) with those seeds. This implies although as a complementary and lightweight strategy to quickly find those more suspicious apps, AMIA can still find new types of malware.

Further Analysis: We further manually analyze those inferred apps that are not labeled as malicious by VirusTotal. More specifically, while selecting 1,000 apps, we manually scan the APK files of those apps, with the usage of multiple most recent Android Anti-Virus tools[2]. Table VI shows the actual Hit Number by adding the numbers of malicious apps identified by VirusTotal (VT) and reported by other Android anti-virus tools. We can see that by using three different sets of seeds, AMIA can all correctly infer more than 940 malicious apps with the Hit Rate higher than 0.94, while selecting 1,000

[2]This is a time-consuming work. Thus, it is not practical to use this strategy to identify malware from a large-scale (e.g., over 70,000) corpus of apps.

apps. This also indicates that our inference algorithm is a good complement to existing Android malware blacklist service.

TABLE VI
ACTUAL HIT NUMBER WHILE SELECTING THE TOP 1,000 APPS.

Seed Size	50		100		200	
	VT	Actual	VT	Actual	VT	Actual
Hit Number	677	943	732	949	771	950

VII. RELATED WORK

Android Malware Detection: An extensive body of systems have been developed to analyze and detect Android malware by monitoring system calls [10], [20]–[22], [25], analyzing the usage of Android permissions [7], [14]–[16], [19], and analyzing the usage of Framework APIs [8], [11], [13], [23], [29]–[31] . These detection systems usually focus on mobile level detection and require deep domain knowledge about Android system, the development of Android malware and a lot of resources for the detection.

Community based detection: Community based techniques have also been widely researched in both spam and malware detection. [26], [27] explored community behaviors to detect comment spam and social network spam respectively. [17], [28] can infer more malicious servers based on different community patterns.

Analysis of market: Recently [18] studied the role of third party markets with a focus on malicious apps and designed a system to quickly locate the same malicious app across the markets. [24] studied the app promoting patterns on app markets and proposed a system to detect suspicious apps that are promoted by attackers.

VIII. CONCLUSION

In this paper, we present the first in-depth empirical analysis of the market-level and network-level behaviors of the Android malware ecosystem. We identify interesting spatial-temporal behavioral patterns of malware market accounts, spatial locality patterns of malware servers, and the community patterns of Android malware from the perspectives of their market and server infrastructure. We hope this study can inspire further interest and research to study the Android malware ecosystem.

IX. ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation (NSF) under Grant no. CNS-1314823 and CNS-0954096. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] About anzhi. http://www.anzhi.com/aboutus.php.
[2] About slideme. http://slideme.org/about-slideme.
[3] Alexa top websites. http://www.alexa.com/topsites/category/Top/Computers/Internet/Domain_Names.
[4] Android app stores become significant sources for malware. http://www.cmcm.com/blog/en/security/2016-01-20/925.html.
[5] Pagerank algorithm. http://en.wikipedia.org/wiki/PageRank.
[6] Virus total. https://www.virustotal.com/.
[7] K. Au, Y. Zhou, Z. Huang, D. Lie, X. Gong, X. Han, and W. Zhou. Pscout: Analyzing the android permission specification. In *Proceedings of CCS*, 2012.
[8] A. Bose, X. Hu, K. G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Proceeding of MobiSys*, 2008.
[9] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry. Towards taming privilege-escalation attacks on android. In *Proceedings of NDSS*, 2012.
[10] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for android. In *Proceedings of the 1st Workshop on CCSSPSM*, 2011.
[11] K. Chen, N. Johnson, V. Silva, S. Dai, K. MacNamara, T. Magrino, E. Wu, M. Rinard, and D. Song. Contextual policy enforcement in android applications with permission event graphs. In *Proceedings of NDSS*, 2013.
[12] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. Quire: lightweight provenance for smart phone operating systems. In *Proceedings of USENIX Security*, 2011.
[13] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. Mc-Daniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacymonitoring on smartphones. In *Proceedings of OSDI*, 2010.
[14] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of android application security. In *Proceedings of the 20th USENIX*, 2011.
[15] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th CCS*, 2009.
[16] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystied. In *Proceedings of CCS*, 2011.
[17] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang. Finding the Linchpins of the Dark Web: a Study on Topologically Dedicated Hosts on Malicious Web Infrastructures. In *Proceedings of S&P*, 2013.
[18] M. Lindorfer, S. Volanis, A. Sisto, M. Neugschwandtner, E. Athanasopoulos, F. Maggi, C. Platzer, S. Zanero, and S. Ioannidis. AndRadar: Fast Discovery of Android Applications in Alternative Markets. In *Proceedings of DIMVA*, 2014.
[19] H. Peng, C. Gates, B. Sarm, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 19th CCS*, 2012.
[20] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of ACSAC*, 2010.
[21] A. Schmidt, R. Bye, H. Schmidt, J. Clausen, O. Kiraz, K. Yxksel, S. Camtepe, and A. Sahin. Static analysis of executables for collaborative malware detection on android. In *ICC Communication and Information Systems Security Symposium*, 2009.
[22] A. Schmidt, H. Schmidt, J. Clausen, K. Yuksel, O. Kiraz, A. Sahin, and S. Camtepe. Enhancing security of linux-based android devices. In *Proceedings of 15th International Linux Kongress*, 2008.
[23] D. Wu, C. Mao, T. Wei, H. Lee, and K. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Proceedings of ASIAJCIS*, 2012.
[24] Z. Xie and S. Zhu. AppWatcher: Unveiling the Underground Market of Trading Mobile App Reviews. In *Proceedings of WiSec*, 2015.
[25] L. Yan and H. Yin. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *Proceedings of USENIX Security*, 2012.
[26] C. Yang, R. Harkreader, J. Zhang, S. Shin, and G. Gu. Analyzing Spammers' Social Networks For Fun and Profit – A Case Study of Cyber Criminal Ecosystem on Twitter. In *Proceedings of WWW*, 2012.
[27] J. Zhang and G. Gu. NeighborWatcher: A Content-Agnostic Comment Spam Inference System. In *Proceedings of NDSS*, 2013.
[28] J. Zhang, S. Saha, G. Gu, S. Lee, and M. Mellia. Systematic mining of associated server herds for malware campaign discovery. In *Proceedings of ICDCS*, 2015.
[29] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zhou. Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, 2012.
[30] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 19th NDSS*, 2012.
[31] Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of MobiSys*, 2012.