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 analyses, 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 |
| Number of Unique Apps | 18,751 | 18,109 | 38,458 | 11,822 |
| Total (Unique) | 18,751 (22%) | 65,232 (78%) |
| | | | | |

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 | 1,946 | 1,247 | 6,764 | 994 |
| Total (Unique) | 1,953 (16%) | 8,229 (84%) |
| AdwareApps | 1,037 | 1,450 | 3,896 | 709 |
| Total (Unique) | 1,037 (12%) | 7,977 (88%) |
| RestApps | 16,121 | 11,916 | 26,854 | 9,378 |
| Total (Unique) | 16,121 (29%) | 38,726 (71%) |
| | | | | |

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%) |
| # of Malicious Accounts | 883 | 432 | 1,493 | 709 |
| Total (Unique) | 883 (25%) | 2,634 (75%) |
| | | | | |

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.

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.

Figure 3(a) shows the distribution of account malicious ratios, and time intervals 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 on 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 I, we finally collect 239,582 unique URLs leading to 25,099 unique servers (including 19,342 domain names and 5,755 IP addresses).

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 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.

**Table IV**

<table>
<thead>
<tr>
<th>Type</th>
<th>URLs</th>
<th>Domains</th>
<th>IPs</th>
<th>Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>MalServers</td>
<td>34,176</td>
<td>3,980</td>
<td>1,162</td>
<td>5,142</td>
</tr>
<tr>
<td>AdwareServer</td>
<td>35,267</td>
<td>3,112</td>
<td>1,057</td>
<td>4,169</td>
</tr>
<tr>
<td>RestServers</td>
<td>176,949</td>
<td>16,580</td>
<td>5,125</td>
<td>21,707</td>
</tr>
<tr>
<td>Total</td>
<td>239,582</td>
<td>19,342</td>
<td>5,755</td>
<td>25,099</td>
</tr>
</tbody>
</table>

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

![Figure 4](image-url)  
(a) Number of Malicious Apps  
(b) FamilyCoverage  

**Top One Family**  
**Top Two Families**

**Fig. 4.** The distributions of the number of malicious apps, and the family 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.

![Figure 5](image-url)  
**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 $n$th community,
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 \((v_j)\) 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.

![Fig. 6. The hit number and hit rate based on the ground truth of VirusTotal.](image)

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 apps.

\(^2\)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.
This also indicates that our inference algorithm is a good complement to existing Android malware blacklist service.

<table>
<thead>
<tr>
<th>TABLE VI</th>
<th>ACTUAL HIT NUMBER WHILE SELECTING THE TOP 1,000 APPS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed Size</td>
<td>VT Actual</td>
</tr>
<tr>
<td>50</td>
<td>677 943</td>
</tr>
</tbody>
</table>

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