Abstract—Android apps have become a primary domain of software applications. However, various causes (e.g., fragmentation and SDK evolution) have led to growing compatibility issues in Android, as especially experienced by end users as these issues causing installation/execution failures of apps. To understand systematically the compatibility issues in the Android ecosystem, this paper looks into developers’ intentions for achieving compatibility of apps and/or preventing potential compatibility issues. We characterized such intentions as reflected via relevant app attributes, in 100,925 benign and malicious apps developed in years 2010–2017. We observed that, among many other findings, there were always developers whose intentions were clearly against Android’s compatibility recommendations. Generally, malware developers’ intentions were significantly and largely different from those of benign apps. The intentions were also constantly evolving, gradually moving to target newer platforms yet with years of delay, with a slower pace in malware. The compatibility intentions, however, were not always fulfilled as expected, despite the specific platform versions intended for. The compatibility intentions, however, were not always fulfilled as expected, despite the specific platform versions intended for.

RQ1: How have Android app developers intended for the compatibility of their apps? Our study showed persistent presence of developers from any year, of benign or malicious apps, that violated compatibility recommendations: 0.32–1.24% benign apps and 0.39–16.88% malware ignoring minSdkVersion which is recommended to specify, while 0.52–6.1% benign apps and 0.14–2.71% malware specifying maxSdkVersion which is not recommended. On the other hand, app developers increasingly intended for specific API levels as target platforms by explicitly specifying targetSdkVersion, with even stronger intentions seen in benign apps developers.

RQ2: How were developers’ compatibility intentions evolving (for migrating to newer platforms)? We observed that all app developers were gradually moving to newer platforms, with a clearly slower pace by malware producers. There were also constantly evolving, gradually moving to target newer platforms yet with years of delay, with a slower pace in malware. The compatibility intentions, however, were not always fulfilled as expected, despite the specific platform versions intended for.

Index Terms—Android, app developer, compatibility, installation, evolution, benign, malware, security
were also considerable lags (about 4 years, and longer in malware) in the intentions (intended platforms) from the latest platforms available. Malware was moving slower potentially for a broader scope of attacks. Developers also appeared to take one to two years to start targeting new platforms, and purposely kept supplying apps just for older platforms.

RQ3: How have developers’ compatibility intentions been fulfilled? In general, developers did not always have their intentions fulfilled as expected, as evidenced by the persistent occurrence of compatibility issues. Apps with dominating intentions in varying years generally tended to have the highest rate of compatibility issues at installation time. We also found that whether the intentions were fulfilled had little to do with the particular platform versions intended for.

Our study artifacts have been made publicly available [here](#).

II. DEVELOPERS’ COMPATIBILITY INTENTIONS

We focus on developers’ intentions relevant to app compatibility as reflected in their specification of three possible attributes in an app’s manifest file (AndroidManifest.xml): minSdkVersion, targetSdkVersion, and maxSdkVersion [17]. These attributes of an app are used by Android to decide, in reference to an attribute of the Android system itself that corresponds to the system version (referred to as the system’s API level), whether the app can be installed or not.

During the build process of an app, the developer is recommended [18] to specify the minimum API level required for the app to function in the attribute minSdkVersion. When not specified, the attribute will be taken as 1 by default. Optionally, the developer may also specify the app’s targetSdkVersion to indicate the target API level for the app to run on. Usually, this attribute informs the system that the app has been tested against the corresponding API level. When not specified, the value of this attribute will be defaulted to that of minSdkVersion. When an app is installed to a device of an API level that is higher than the app’s minSdkVersion and/or targetSdkVersion, Android will enable backward compatibility behaviors for the app to function as expected. Despite an option, Android does not recommend developers to specify the maximum API level on which the app is supposed to run, through the attribute maxSdkVersion.

Intuitively, when app is attempted to be installed to a device, Android will reject the installation if the device’s API level does not fall within the range of [minSdkVersion, maxSdkVersion]. Since API level 6, however, Android does not check against maxSdkVersion during app installation, because Android promises that new versions of the platform are fully backward-compatible [17]. Yet, maxSdkVersion still indirectly affects the chance of an app getting installed to a device: if an app’s maxSdkVersion is lower than the device’s API level, the app will not show up in the list of recommended apps on the device’s Google Play store; or, if the app has been installed before, it will be automatically removed from the device when its API level is updated to be higher than the app’s maxSdkVersion if specified.

III. METHODOLOGY

This section describes our experimental datasets (benchmarks), experimental setup and study procedure, and metrics and measurements of our study, for answering the three research questions described earlier.

A. Benchmarks

<table>
<thead>
<tr>
<th>App group</th>
<th>number of samples from each year within 2010-2017</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>benign</td>
<td>16,724, 9,871, 10,901, 9,635, 5,257, 8,368, 2,421</td>
<td>62,438</td>
</tr>
<tr>
<td>malware</td>
<td>2,139, 18,873, 5,384, 5,138, 14,514, 2,682, 258</td>
<td>38,487</td>
</tr>
</tbody>
</table>

As summarized in Table I, we used 62,438 benign and 38,487 malicious apps in our study, which were developed in eight different years (2010 through 2017). The 2,261 benign apps of 2017 were obtained from Google Play [19], and all other benign apps fromAndroZoo [20]. The malware of 2013 through 2016 was downloaded from VirusShare [21], and the malware of 2010 through 2012 from AndroZoo. Collecting malware of 2017 from publicly accessible sources was difficult, and we managed to manually gather 258 malware samples of that year from the wild. We determined the year of each app according to the dex date retrieved from the app’s APK. We discarded corrupted APKs which either cannot be unzipped or were missing resource files (e.g., the manifest file), and eventually had 100,925 apps for use in our study.

B. Experimental Setup and Procedure

In our study, we validated the compatibility of an app at installation time by actually trying to install (the original APK of) the app to an Android device. We collected the installation logs, and then analyzed these logs to recognize the installation as a success or failure, using our toolkit [22]. To understand how developers’ compatibility intentions are related to device API levels, we used three Android virtual devices (AVDs), Nexus One with 2G RAM and 1G SD storage but with varying API levels, 19, 23, and 25. We considered these three API levels because of their top proportions in the latest market share distribution of different Android platform versions [23]. We ran these AVDs through the Android emulator [24].

To retrieve the three app attributes that reflect developers’ compatibility intentions (Section II), we used apktool [25] to parse the manifest data of an app. We utilized the Android debug bridge (adb) [26] tool to install an app to a given device (AVD). In order to examine the effects of developers’ intentions on app compatibility at installation time, we needed to identify incompatibility-induced installation failures. However, installation failures, when occurred, could be due to reasons other than compatibility issues. Thus, for each app that failed at installation to one device, we kept trying to install the app to different devices (with different API levels and/or device configurations) until we found one device where the app can be installed successfully. In general, this process of excluding non-compatibility-induced installation failures is time-consuming. Fortunately, for this work, we were able to fulfill the process by only trying at most 5 devices including the three considered in our study.

C. Measurements

To answer our research questions, we characterized developers’ compatibility intentions by measuring the presence of the three manifest attributes that reflect the intentions, as well as their distribution over all possible API levels (i.e., attribute values). We also characterized the association between device API levels and developer intentions by looking at the most dominating API levels targeted by apps as specified in them.
versus the API levels targeted by app groups that are most prone to incompatibility-induced installation failures.

Importantly, with respect to all the above measures, we characterized the changing patterns of developers’ compatibility intentions over the eight-year span we considered to understand the respective evolutionary trends.

IV. MAIN FINDINGS

This section presents the results of our study, with a focus on major findings with respect to each of our research questions.

A. RQ1: Overall Developer Intentions for App Compatibility

Tables II through IV summarize the general compatibility intentions in terms of the three relevant attributes. While apps are recommended by the Android team to explicitly specify minSdkVersion, 0.84% and 4.24% of all the benign apps and malware, respectively, did not follow the recommendation. Although generally such violating apps were in decline, the movement was slow, especially in benign apps. On the other hand, for most of the years, malware tended to violate the recommendation more often than benign apps. One plausible reason is that malware generally has lower quality control than benign apps during app production (e.g., many malware apps are simply or even automatically built through repackaging).

Over the years, targetSdkVersion were specified increasingly in apps, benign or malicious—the growth was largely continuous. By 2017, about 90% of apps explicitly specified this attribute. Since it informs the platform version against which an app has been tested, the steady growth suggests potentially growing intentions of developers for targeting specific platforms (hence more focused scope of compatibility). In absolute terms, such intentions were always stronger with developers of benign apps versus malware producers. This contrast implies that malware tends to attempt to be compatible with more Android versions, possibly for affecting more users and apps.

While specifying maxSdkVersion is not recommended, there were apps (albeit accounting for relatively small portions) that violated this recommendation in any year. Notably, over time, there were increasing percentages of apps doing so, with more frequent violations seen in benign apps, suggesting that developers tended to ignore the recommendation. This is likely attributed to the fact that Android does not check maxSdkVersion during app installation since API 6.

For the two app groups in each of Tables II through IV we computed the significance (via a paired Wilcoxon signed-rank test [27]) and effect size (via Cliff’s Delta [28]) of the differences in benign and malware developers’ compatibility intentions across the eight years. Both analyses are non-parametric and were both conducted with $\alpha=.05$. We found that the differences were significant and large: $p$ values of 0.008, 0.016, and effect sizes of -1, -0.75, respectively, for targetSdkVersion and maxSdkVersion. The difference in minSdkVersion was also non-trivial (0.051 $p$ value and 0.63 effect size).

B. RQ2: Evolution of Developers’ Compatibility Intentions

Figure 1 depicts the percentage distribution of compatibility intentions in terms of the values of the three relevant app attributes, in benign apps (marked by prefix B) and malware (marked by prefix M). The basis of the distribution for each attribute was the apps that explicitly specified the attribute values, shown at the center of each colored bar segment. The percentage of apps having specified an attribute value is presented by the height of each bar segment. For example, as shown in the top chart, over 20% of benign-2010 apps specified minSdkVersion as 4, and over 50% specified it as 3.

Overall, apps were very conservative in specifying the minimal API level to work with. For example, in benign apps, those created in 2014 were still dominated by API level 8 (which was released in 2010 [29]) as the minSdkVersion. By 2017, the dominating minSdkVersion moved to 16 (released in 2012). In general, benign-app developers intended for a backward compatibility of 4 years or longer. Malware producers had intentions for backward compatibility of a noticeably longer period, up to 7 years as seen by more recent apps (e.g., even those created in 2017 mostly specified minimal API level of 9, released in 2010). Intuitively, this can be explained by the intention of malware producers to affect more apps in the past. The targetSdkVersion of an app potentially indicates the best platform version the developer wanted the app to run on.
Thus, this attribute reflects how developers might have wanted to migrate to newer platforms. As expected, newer apps, benign or malicious, were generally dominated by higher API levels. However, compared to benign apps, malware appeared to migrate to newer platforms noticeably slower. Also, in both app groups, the migration was often abrupt (e.g., the most dominating targetSdkVersion in malware jumped to 17 in year 2014 without seeing any lower levels between 10 and 16 dominating previously). Evolution of the attribute values revealed that apps targeted platforms of one or two year older than themselves, which is reasonable as the developers needed to take some time to develop apps for the new platforms.

Compared to the other two attributes, maxSdkVersion saw the greatest similarity in the value distribution between benign apps and malware. The values of this attribute, in contrast to the years of the apps, suggest that there were constantly noticeable gaps between the highest API level released when an app was created and the API level the app declared as the maximal to work with. These gaps suggest that a few (given that only a small portions of apps specified this attribute) apps were created every year just for older platforms (as they will not be shown on Google Play on newer platforms).

C. RQ3: Intention Fulfillment

Table V lists the top dominating API level specified (as targetSdkVersion or, if not specified, by default as minSdkVersion) in all of the benign/malicious apps of a specific year (the last row) versus the API level targeted by the subset of those apps that had the highest rate of incompatible apps at installation time with a device API level (the third to fifth rows). For instance, among benign-2011 apps, the group of apps that had the highest incompatibility rate with the device of API level 25 targeted API level 8, while the top dominating API level targeted by all benign-2011 apps as specified was 4.

V. RELATED WORK

Compatibility issues have been attended previously, through studies of the fragmentation problem [5], [13] and API changes [15], [16], both of which constitute the main causes of incompatibilities. In [13], developers’ strategies for preventing compatibility issues were studied. In our earlier work [10], we studied a particular kind of compatibility issues—those due to incompatible use of run-time permissions. These prior works focus on app code traits relevant to compatibility issues, concerning app incompatibilities that will only be exhibited at runtime. In contrast, our study addresses human (developer) intentions reflected in (non-code) app metadata, regarding not only compatibility issues that affect app execution, but also those that can fail apps earlier at installation time.

In [4], SDK evolution was characterized, during which the API changes can cause app incompatibilities. Our study incorporated an evolutionary perspective as well, but we examined how developers’ intentions for app compatibility evolved.

VI. CONCLUSION AND FUTURE WORK

We studied the compatibility intentions of Android app developers as reflected by relevant attributes in app manifest, by examining 100,925 benign and malicious apps created in 2010–2017. Our study addressed how developers intended for app compatibility, how the intentions evolved in terms of the attributes’ values changing over time, and how the intentions have been fulfilled in terms of the achievement of compatibility in relation to the values of those attributes. We found that (1) malware developers had less explicit expression of compatibility intentions than did developers of benign apps, while both app groups have seen noticeable violations against compatibility recommendations and growing intention for particular platform versions targeted, (2) malware developers intended for backward compatibility of longer periods than benign developers did (plausibly for affecting apps more broadly), (3) in terms of developers’ intentions, malware was migrating to newer platforms noticeably slower than benign apps, although both app groups saw a delay of one to two years in terms of the API levels they targeted, (4) the compatibility intentions were not always fulfilled, at least for app installation, benign apps or malware, and (5) how well the intentions were fulfilled was largely independent of the intention for specific API levels. As future work, we will explore the symptoms and root causes of app incompatibilities as observed, not only at installation time but also during app executions, with respect to developers’ relevant intentions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>specified in app</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>17</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>17</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td>23</td>
</tr>
</tbody>
</table>

TABLE V: Device API levels with most incompatible apps versus most dominating API levels in apps
REFERENCES


