Update Behavior in App Markets and Security Implications: A Case Study in Google Play

Andreas Möller,
Stefan Diewald, Luis Roalter
Technische Universität München
Munich, Germany
andreas.moeller@tum.de,
stefan.diewald@tum.de,
roalter@tum.de

Florian Michahelles
ETH Zurich
Auto-ID Labs
Zurich, Switzerland
fmichahelles@ehtz.ch

Matthias Kranz
Luleå University of Technology
Department of Computer Science,
Electrical and Space Engineering
Luleå, Sweden
matthias.kranz@ltu.se

ABSTRACT
Digital market places (e.g. Apple App Store, Google Play) have become the dominant platforms for the distribution of software for mobile phones. Thereby, developers can reach millions of users. However, neither of these market places today has mechanisms in place to enforce security critical updates of distributed apps. This paper investigates this problem by gaining insights on the correlation between published updates and actual installations of those. Our findings show that almost half of all users would use a vulnerable app version even 7 days after the fix has been published. We discuss our results and give initial recommendations to app developers.

Author Keywords
Mobile applications; digital market places; update behavior; security

ACM Classification Keywords
D.4.6. Operating Systems: Security and Protection

INTRODUCTION AND MOTIVATION
Platform-specific marketplaces, such as the Apple App Store or Google Play (formerly Android Market), are nowadays an important source for mobile app distribution [13]. In March 2012, Apple reached in total 25 billion iOS app downloads[1]. Until 2011, 10 billion Android apps have been downloaded in total over Google Play[2]. Smartphone users find their applications bundled at one place and are informed about available updates (via a badge symbol on the App Store icon on iOS, or a message in the notification bar on Android). However, neither on iOS or Android, application updates are installed automatically. Android has a setting for installing updates without confirmation, but it is disabled by default.

[1]http://www.apple.com/pr/library/2012/03/05Apples-App-Store-Downloads-Top-25-Billion.html
[2]http://www.wired.com/gadgetlab/2011/12/10-billion-apps-detailed/

MobileHCI'12, September 21–24, 2012, San Francisco, CA, USA.
ACM 978-1-4503-1443-5/12/09.

This update mechanism implementation can be seen as a potential risk for security. Unfixed security holes increase the vulnerability of a device. As users need to take charge of keeping their system up to date themselves, important updates might not be installed timely or at all. Especially for research apps (e.g. [11, 10]) or at the beginning of an app's market lifetime, regular installation of updates is important. Being in state of development, such apps often are less stable and require more frequent fixes. Until the end of 2011, more than 20,000 new apps per month were published in Google Play[3], so that potentially a large number of apps is affected by this phenomenon. Security flaws become even more severe for the novel and upcoming category of apps that integrate with the home or automobile (so-called in-car apps, see e.g. [5]), since in that case not only the app itself, but also the connected property becomes insecure.

In a case study, we observed users' update behavior of an Android app we have placed in Google Play. We gained insights on the correlation between published updates and their actual installation and discuss the consequences and recommended actions on the part of the developers.

RELATED WORK
While inclusion in the Apple App Store requires a review process [1], Google Play is free of constraints for uploading apps. However, apps are scanned for viruses and malware [8] and in case of malicious content deleted. This is, however, just a method to uncover software that obviously tries to do 'evil' things, but not to detect programming bugs or security holes.

Automatic analysis of security problems during the submission process to digital market places has been proposed using several approaches [6, 14]. Di Cerbo et al. [4] present a methodology for mobile forensics analysis to detect 'malicious' (or 'malware') applications. The methodology relies on the comparison of the Android security permission of each application with a set of reference models, for applications that manage sensitive data. Thus, this research is focusing more on protecting the user from malicious apps whereas our

[3]http://www.androlib.com/appstats.aspx

paper focuses on capturing the (non-)compliances of users to install fixes of a trusted developer.

It has also been found that Android apps often require permissions that are actually unneeded. Extensions to Android's permission model have consequently been proposed which focus particularly on improving the (initially quite coarse) granularity of permissions [12, 15] or remove them in hindsight by inline reference monitoring[4]. Fewer rights inherently also decrease the probability for security-relevant bugs.

Miluzzo et al. [9] looked at implications and challenges of large-scale distribution of research apps through the Apple App Store. They pointed out that insufficient software robustness and poor usability may lead to a loss of confidence on the part of the users, but did not quantitatively examine this phenomenon (such as the number of uninstalls due to dissatisfaction). AppTicker [7] is a project that allows monitoring mobile app usage, (un)installation and more to gain information about usage patterns on smartphones. To our knowledge, the particular phenomenon of update behavior in app stores has not been examined yet. Despite the security approaches and measures we presented in this section, keeping the software up to date remains the central requirement for a stable and secure system.

CASE STUDY

For our case study, we are looking at *VMI Mensa*[5], an Android application developed by the research group of the authors of this paper. *VMI Mensa* shows meals and prices of cafeterias and canteens of university campuses in our city. The application, targeted at students and university employees, has been available in Google Play since July 21, 2011 and meanwhile (as of July 2012) reached 2,294 downloads. It has received 123 ratings (averagely rated with 4.8 out of 5 stars) and 40 user comments. Since its launch, the app has continuously been extended in its functionality, e.g. by a location-aware canteen finder, details on ingredients, accessibility information (e.g. on elevators), and much more.

Update Installation Analysis

Since *VMI Mensa* was first available in Google Play, we have shipped 21 updates. For our analysis, we used the built-in statistics tools of the Android Developer Console in Google Play. They allow keeping track of the number of installations over time, monitor installed app versions and a lot more. All data is anonymous and cannot be related with individual users. As stated before, updates may install automatically or manually by user confirmation. We cannot track whether automatic update installation was enabled on users' devices.

For our analysis, we looked at the latest five updates, published at December 22, 2011, January 17, January 26, February 24 and April 02 (all 2012). The average time between updates was 26 days, which we consider not as an unreasonable effort for users to regularly install them. All updates added new functionality to the app and/or fixed small problems, but none were critical for security. For each update, we observed how many users downloaded the update on the initial day of publishing and in the 6 consecutive days. We calculated the update installation ratio by relating the download count to the total count of active device installations on the respective days.

User Communication Analysis

In addition to the anonymous update installation statistics, we considered available user communication in form of feedback emails, comments and ratings in Google Play for our analysis. We will bring in these findings in the discussion section.

Results

In the following, we describe and visualize the quantitative results of our case study.

Update Behavior

Table 1 shows the installation percentages on the update publishing day (day 0) and the six consecutive days (day 1 to day 6), averaged over all five updates that were considered in this study. The exact ratios are very similar for all updates, which is implied by the low standard deviations (see last column of the table). In average, 17.0% installed the update on day 0. On the following days, the numbers continuously and exponentially decrease: 14.6% installed the update on day 1, only 7.8% on day 2, and 5.1% on day 3. On day 6, only another 2.3% downloaded the update.

Day after Update	Update Installed	Standard Deviation
Publishing Day	17.0%	2.7%
Day 1	14.6%	2.0%
Day 2	7.8%	1.3%
Day 3	5.1%	0.9%
Day 4	3.5%	0.7%
Day 5	2.8%	0.5%
Day 6	2.3%	0.4%
Total in 7 days	53.2%	2.7%

Table 1. Percentage of all users who installed an update within 7 days after it was published. Only slightly more than half of all users installed a recent update within one week. Data was averaged based on five subsequent updates published within 102 days. Standard deviation is related to the five individual updates we observed in our use case.

This trend is visualized in Fig. 1 and can be summarized as follows: Most of those users who actually *do* install updates install them quickly. We hypothesize that the relatively high ratios of the first two days might partly be due to the automatic update option. Users that did *not* install the update early are also not likely to do so in the subsequent days. In total, just 53.2%, slightly more than a half, had the most recent update installed one week after publication.

Version Distribution

We also looked at the distribution of the latest five versions of the app on users' devices, illustrated by different colors in Fig. 2. The seven-day periods after an update has been published are slightly shaded for illustration. The visualization shows the spread of new versions due to cumulative installs (visualized with a steep graph that flattens out more and more), and the decrease of older versions. It also becomes

[4]AppGuard. http://www.backes-srt.de/produkte/srt-appguard

[5]https://play.google.com/store/apps/details?id=de.tum.ei.lmt.vmi.mensa

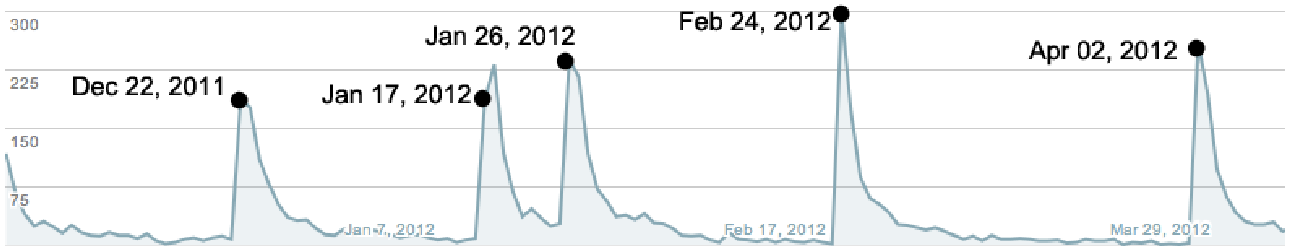

Figure 1. Visualization representing the number of five subsequent update downloads (vertical axis) over time. The graph shows maxima on the update publishing day (possibly also due to activated auto-updates) and exponentially decreases thereafter. Modified diagram based on Android Developer Console statistics.

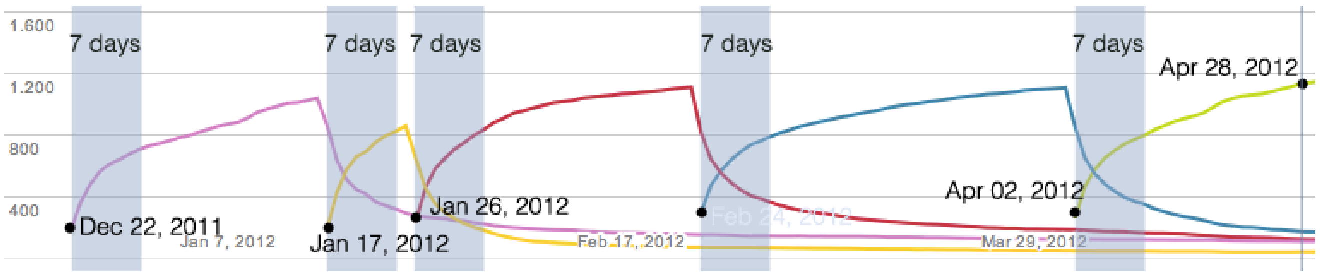

Figure 2. Visualization representing the number of installations by version (vertical axis); the colored lines indicate the five latest versions. The diagram reveals how long old versions are active on user's devices. The 7-day periods after an update has been published are highlighted. Modified diagram based on Android Developer Console statistics.

evident how long outdated versions (up to four versions older than the latest one) are still circulating. As an example, we look at April 28, 2012, which is two weeks after the latest update has been published: Only 56.4% of all users have installed the latest version (v.27) at this time. The previous four versions were still in use by 8.5% (v.26), 6.0% (v.25), 5.5% (v.24) and 2.1% (v.23). Most severely, 21.5% had even older versions installed on their devices at that time.

DISCUSSION

Results from our case study reveal a problematic update behavior: Even one week after their publication, updates were installed only by about 50% of users. The rest used different outdated versions; one fifth even did not install even one of the last five updates. This implies two potential groups of users: those who update in an exemplary manner, and those who barely update at all. Hence, developers must not make the mistake to rely on the belief that at least the penultimate version of their app would run on most devices.

If we project this result to general update behavior, our findings imply a critical security situation. The harmless feature updates in our case study could be important security-related fixes in another app. On average, almost half of all users would use a vulnerable app version even 7 days after the fix has been published. The time from detection of a security hole to the final update shipment is not even considered here. Further reasons indicate that the 'real' update situation could even be worse than in our exemplary case analysis. A high number of installed apps could further decrease the amount of up-to-date apps, since more time would be required for individual updates. Furthermore, the fact that users are presumably highly engaged with our examined canteen app could have an impact on update frequency as well. We see an even more critical situation with apps that are not regularly used, but for which security is crucial just then (e.g. for online banking apps). In-depth usage monitoring [2] is required for better understanding the relation between usage frequency and update behavior.

We also looked at users' behavior in case of problems. Our app contained a 'Give feedback' item in the preferences menu that allowed sending an email to the developers. In the app description in Google Play, we asked users to give us feedback using this function. We also linked to a Q&A page from which users could contact the developers as well. Our experience revealed that few users actually used these opportunities. They rather made use of the rating functionality in Google Play. For example, the download of the daily menu was not working for one day due to a server migration. Several users immediately left a bad rating in Google Play, complaining about the app not working any more. Apparently, they had not read the requests to provide feedback per mail or not found the feedback link in the app. A similar case illustrates as well that not all users read the description texts in Google Play: One user commented that it would be good to have an English translation. In fact, the app is fully localized to 6 languages (amongst them English), and localizations automatically adapt to the device's system language. Similarly, this user rated the app worse because of this complaint.

For developers, our observations have three consequences. First, they show how quick users are with bad ratings, which may be problematic especially for commercial apps – other work already stated that user reviews can be brutal [9]. Hence, it is important to keep the application bug-free and provide timely updates in case of problems.

Second, developers cannot rely on users reading instructions and employing the built-in feedback functions. We gained the insight that ways to further improve such functions should be found, and we also learned that keeping track of ratings and comments in Google Play is important. Otherwise, in some cases, we would not have been aware of potential problems. In our case, they were related to usability and minor issues, but they could have been security bugs as well. This is especially important since security holes not necessarily go along with unresponsive or crashing apps and thus are not covered by the built-in error reporting function of Google Play.

Third, as a first step towards an improved security on mobile phone platforms and in light of sometimes difficult download mechanisms [3], we encourage developers to support users in updating, e.g. by built-in update checks within their application and/or forwarding users to the platform market place, as we use it in our research apps [11].

CONCLUSION

In this paper, we have analyzed update behavior and security implications in application markets at the example of an Android application we developed and offer for download in Google Play. We found that, in average, half of all users did not install an update even seven days after it has been published and thus would use a potentially vulnerable application. Although generalizations of our initial findings must be carried out carefully and further studies will be necessary, we raised the awareness for a potential slow update propagation on Android and other mobile platforms.

Further automatic quality assessments for uploaded apps in digital market places and more automated update mechanisms could be ways to increase the level of security on mobile devices.

Acknowledgments

We thank all of our students involved in the development of *VMI Mensa* and our other research apps.

REFERENCES

1. Apple Inc. App store review guidelines. `https://developer.apple.com/appstore/guidelines.html`, 2012.
2. Böhmer, M., Hecht, B., Schöning, J., Krüger, A., and Bauer, G. Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, ACM (2011), 47–56.
3. Cramer, H., Rost, M., Belloni, N., Bentley, F., and Chincholle, D. Research in the large. using app stores, markets, and other wide distribution channels in ubicomp research. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing - Adjunct*, Ubicomp '10 Adjunct, ACM (New York, NY, USA, 2010), 511–514.
4. Di Cerbo, F., Girardello, A., Michahelles, F., and Voronkova, S. Detection of malicious applications on android OS. In *Proceedings of the 4th international conference on Computational forensics (IWCF'10)*, H. Sako, K. Y. Franke, and S. Saitoh, Eds., Springer (Berlin, Heidelberg, 2010), 138–149.
5. Diewald, S., Möller, A., Roalter, L., and Kranz, M. Mobile Device Integration and Interaction in the Automotive Domain. In *AutoNUI: Automotive Natural User Interfaces Workshop at the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2011)* (Nov.–Dec. 2011).
6. Gilbert, P., Chun, B.-G., Cox, L. P., and Jung, J. Vision: automated security validation of mobile apps at app markets. In *Proceedings of the second international workshop on Mobile cloud computing and services*, MCS '11, ACM (New York, NY, USA, 2011), 21–26.
7. Henze, N., and Sahami, A. Appticker. `http://projects.hcilab.org/appticker/`, 2012.
8. Lockheimer, H. Google Mobile Blog. Android and Security. `http://googlemobile.blogspot.de/2012/02/android-and-security.html`, February 2012.
9. Miluzzo, E., Lane, N., Lu, H., and Campbell, A. Research in the app store era: Experiences from the CenceMe app deployment on the iPhone. In *Proc. Ubicomp* (2010).
10. Möller, A., Roalter, L., Diewald, S., Scherr, J., Kranz, M., Hammerla, N., Olivier, P., and Plötz, T. Gymskill: A personal trainer for physical exercises. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on* (march 2012), 213 –220.
11. Möller, A., Thielsch, A., Dallmeier, B., Roalter, L., Diewald, S., Hendrich, A., Meyer, B. E., and Kranz, M. Mobidics – improving university education with a mobile didactics toolbox. In *Ninth International Conference on Pervasive Computing (Pervasive 2011), Video Proceedings* (San Francisco, CA, USA, June 2011).
12. Nauman, M., Khan, S., and Zhang, X. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ACM (2010), 328–332.
13. Research, and Markets. *Application Distribution Channels 2011*. Evans Data Corp., Sep. 2011.
14. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., and Weiss, Y. "Andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems 38* (2012), 161–190. 10.1007/s10844-010-0148-x.
15. Vidas, T., Christin, N., and Cranor, L. Curbing android permission creep. In *Proceedings of the Web*, vol. 2 (2011).