Name	Component	Source	Venue	Privacy Policy	Installs (M for millions)
Texas Poker v4.0.1	App	Location	HTTP GET	×	10M-50M
Word Search v1.14	Mobfox	Location, IMEI	HTTP GET	~~app~~,~~lib~~	0.5M-1M
Speedtest v2.09	Mobfox	Location, IMEI	HTTP GET	app,lib	10M-50M
Brightest Flashlight v2.3.3	MDotm, Mobclix	IMEI, IMSI	HTTP GET	app,~~lib~~	50M-100M
Weather Underground v2.1.2	App, Mobclix	Location, IMEI	HTTP GET	app,lib	1M-5M
Fruit Ninja *(2 samples)*	Mobclix	IMEI	HTTP GET	app,lib	100M-500M
Angry Birds *(14 samples)*	Jumptap	IMSI	HTTP GET	app,lib	300M-900M
Bad Piggies *(3 samples)*	Jumptap	IMSI	HTTP GET	app,lib	10M-50M
Tap Tap Revenge v4.3.3 v4.4.5	Tapjoy	IMEI	HTTPS GET	×	0.1M
Logo Quiz v8.8	Tapjoy	IMEI	HTTPS GET	×	10M-50M
Trial Extreme v1.28 & v2.83	Tapjoy	IMEI	HTTPS GET	×	5M-10M
Big Win Basketball v2.0.4	Tapjoy	IMEI	HTTPS GET	app,~~lib~~	5M-10M
Solitaire v2.1.5	Tapjoy	IMEI	HTTPS GET	~~app~~,lib	50M-100M
Talking Tom 2 v2.0.3	Tapjoy	IMEI	HTTPS GET	app,lib	100M-500M

Table VIII: Free apps that spread certain personal information identified by AppAudit. For the "Privacy Policy" column, a "~~lib~~" means that the privacy policy does not cover the kind of data spread by advertising libraries.

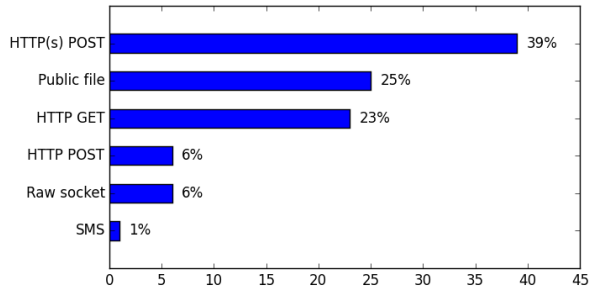

Figure 7: The venues of data leaking.

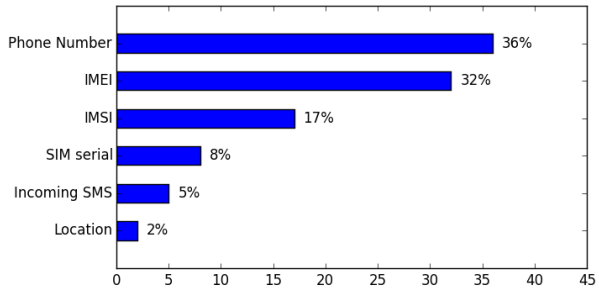

Figure 8: The types of leaked data.

which could effectively detect data leaks beforehand and avoid accidentally using data-leaking 3rd-party modules.

Finding 2: HTTP requests are the most prominent leaking venues. Figure 7 presents the leaking venues for all data-leaking cases in malware and free app datasets. HTTP(s) transmission turns out to be the most popular venue to leak data, since HTTP servers can be easily configured. This suggests that mobile application confinement tools [36], [34], [37], [38], [39] can focus on HTTP traffic to effectively confine data leaks.

Nevertheless, eight reported free apps transmit personal information in plain-text forms (HTTP GET requests). Consequently, some important personal information (locations and identity) can be easily obtained by traffic sniffing in the public. To make things worse, some of these report apps do not have a clear privacy policy statement, which makes users unaware of the potential risks.

Finding 3: Tracking is universal. Figure 8 presents the breakdown of leaked data found in the malware and free app datasets. We discover that the IMEI number and phone number is the most commonly leaked information. The phone number is commonly sent by malware for follow-up SMS phishing. IMEI serves as the phone identity and is widely used to track user for targeted advertising. Nowadays, each free app is bundled with a couple of advertising libraries [1] and the user interacts with a number of apps. IMEIs to mobile devices is what cookies are to web browsers. Cookies are bound to individual websites, i.e. one website cannot access the cookies of another. However, IMEI tracking is not bound to individual apps but to individual advertising libraries. Thus, if two apps use the same advertising library, the advertiser can accurately track user's transition from one app to another. If the data transmission between the library and server is unencrypted, the trace can be acquired and used to predict user habits and launch social engineering attacks.

Finding 4: Apps and advertising libraries are gaining awareness of user privacy. We find that, apps (Word Search and Speedtest) are gaining awareness of privacy by removing problematic advertising libraries. We believe that AppAudit, when integrated with IDEs, could well assist developers for this purpose. On the other hand, we discover advertising libraries are gaining privacy awareness as well. For example, a newer version of the Tapjoy advertising library hashes

IMEI before sending it to the advertising server. Given that hashing is cryptographically hard to invert, hashing effectively avoids leaking plain-text IMEIs. The newer versions of Trail Extreme and Big Win Basketball benefit from this simple hashing and no long leak data because of Tapjoy. These advancements witness the improved awareness of user privacy for both apps and advertising libraries.

VI. Related Work

In this section, we introduce the related work about analyzing mobile applications. The related approaches can be divided into two categories. Static analysis produces analysis results by statically analyzing various files associated with the application. Dynamic analysis runs with the application in real devices and reports problems when they happen. The synergy of static and dynamic analysis is exploited by AppAudit as well as by existing work for various purposes [40].

A. Static Analysis

We discuss existing techniques in terms of their analysis granularities.

Permission-based analysis. Android defines permissions for an application to access various resources and system services. Every application is required to declare the permissions it uses in its manifest file. Permissions can serve as an approximation of application behavior, which has been leveraged to identify malicious apps [41]. Kirin [42] checks application permission usage with a set of security policies. However, permission analysis cannot distinguish if the application is abusing permissions. For example, having the access to personal information and network capability may not lead to the conclusion that the application will leak personal information via the network. As a result, permission-based analysis normally faces high false positive when used to analyze personal information leakage [43].

API analysis. To complement permission analysis, existing approaches have made an attempt to analyze the API usage of applications. Stowaway [31] extracts the APIs used by an application and checks if the app is overdemanding permissions. RiskRanker [43] uses API analysis to quickly identify applications that have higher security and privacy risks. API analysis refines the analysis granularity of permission-based analysis. However, API analysis does not consider the information flows within the application, which fails to justify privacy leakage with detailed code path.

Dataflow analysis. Dataflow analysis is a classic program analysis, used for information flow validation and data reachability test. PiOS [5] performs reachability dataflow analysis on iOS apps to identify potential privacy leaks. ContentScope [6] applies dataflow analysis to detect unwanted information leakage from personal information databases to third-party Android applications. In general, dataflow analysis can provide more accurate and informative results than previous analysis. However, dataflow analysis can encounter difficulties due to the wide use of GUI and event-driven programming paradigms in mobile apps [3].

Symbolic Execution. Symbolic execution is an alternative code analysis to finding information leakage. Symbolic execution faces the fundamental challenge of path explosion, especially with event-driven GUI programs. AppIntent [3] aims to reduce the number of paths to be executed based on Android intent propagation rules to improve performance of symbolic execution. Nevertheless, AppIntent still requires minutes to hours to examine an app.

B. Dynamic Analysis

Dynamic analysis is implanted into the mobile operating systems and monitors applications at runtime. TaintDroid [8] applies dynamic taint analysis to various components of Android OS, which tracks sensitive information flow and reports to the user when sensitive information leaves the device. AppFence [44] retrofits the Android OS to provide fake sensitive information to the applications upon user's requests. VetDroid [7] dynamically records the permission usage of untrusted applications, which is then analyzed offline to reveal malicious behavior.

Compared with static analysis, dynamic analysis only reports suspicious behavior that occurs at runtime. This feature avoid false alarms and is appealing to the end user. However, for other use cases (market-level vetting, detailed code analysis), dynamic analysis can be limited by low code coverage.

C. Compiler Techniques

The approximations in AppAudit are largely inspired by analysis techniques used in just-in-time compilers. Many of the design decisions in our execution engine is inspired by improvements to symbolic execution, e.g. prefix symbolic execution engine [45], directed symbolic execution [46]. Also our object representation is inspired by [47].

VII. Conclusion

Mobile devices carry abundant personal information. Program analysis can effectively reveal data leaks in apps and protect user privacy. App auditing has three major use cases. First, app market operators require automatic tools to detect and remove data-leaking apps. Second, app developers need to perform self-check before publishing apps. Third, mobile users expect to know if an app is leaking data before installation.

In this paper, we design AppAudit, an efficient analysis framework that can deliver high detection accuracy with significantly less time and memory. AppAudit comprises a static API analysis that can effectively narrow down analysis scope and an innovative dynamic analysis which could efficiently execute application bytecode to prune false positive and confirm data leaks.

According to our experiments on read-world malware and apps, AppAudit achieves a 99.3% true positive rate (comparable to static analysis) and no false positives. Most importantly AppAudit performs 8.3x faster than the state-of-the-art static analysis tools, which makes it the only solution viable for important use cases of app auditing.

VIII. Acknowledgment

This work was supported in part by the NSERC Discovery Grant 341823, Canada Foundation for Innovation (CFI) Leaders Opportunity Fund 23090. The testbed was supported in part by NSF China Grant 61272101 and the Shanghai Key Laboratory of Scalable Computing and Systems. The authors would like to thank Xinye Lin and our reviewers for the discussions and feedbacks.

References

[1] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WISEC '12. New York, NY, USA: ACM, 2012, pp. 101–112.

[2] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 21–21.

[3] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM Conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1043–1054.

[4] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '14. New York, NY, USA: ACM, 2013, pp. 259–269.

[5] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications," in *Proceedings of the 18th Network and Distributed System Security*, ser. NDSS '11, 2011.

[6] Y. Zhou and X. Jiang, "Detecting passive content leaks and pollution in android applications," in *Proceedings of the 20th Network and Distributed System Security*, ser. NDSS '13, 2013.

[7] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the 2013 ACM Conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 611–622.

[8] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[9] K. Ali and O. Lhoták, "Application-only call graph construction," in *Proceedings of the 26th European Conference on Object-Oriented Programming*, ser. ECOOP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 688–712.

[10] "Android callbacks (flowdroid project)," https://github.com/secure-software-engineering/soot-infoflow-android/blob/develop/AndroidCallbacks.txt.

[11] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "Bitblaze: A new approach to computer security via binary analysis," in *Proceedings of the 4th International Conference on Information Systems Security*, ser. ICISS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–25.

[12] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 317–331.

[13] G. Sarwar, O. Mehani, R. Boreli, and D. Kaafar, "On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices," in *Proceedings of the 10th International Conference on Security and Cryptography*, July 2013, pp. 461–467.

[14] "Scrubdroid/antitaintdroid project," http://gsbabil.github.io/AntiTaintDroid/.

[15] "dex2jar: Tools to work with android .dex and java .class files," https://code.google.com/p/dex2jar/.

[16] "Xml apk parser," https://code.google.com/p/xml-apk-parser/.

[17] "Android dynamic java virtual machine: Bytecode instruction set," https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html.

[18] "Droidbench, an open test suite for evaluating the effectiveness of taint-analysis," https://github.com/secure-software-engineering/DroidBench.

[19] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 95–109. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.16

[20] "Lookout mobile security," https://www.lookout.com/.

[21] "Avg mobile security," http://www.avgmobilation.com/.

[22] "Sophos security," http://www.sophos.com/en-us.aspx.

[23] "Nviso apkscan," http://apkscan.nviso.be/.

[24] "Semantec mobile security," http://www.symantec.com/mobile-security.

[25] "Fortiguard virus encyclopedia," http://www.fortiguard.com/encyclopedia/.

[26] "Trendlabs security intelligence blog," http://blog.trendmicro.com/trendlabs-security-intelligence/.

[27] "An evaluation of the application verification service in android 4.2," http://www.cs.ncsu.edu/faculty/jiang/appverify/.

[28] "Virustotal: Free online virus, malware and url scanner," https://www.virustotal.com/.

[29] X. Jiang, "Security alert: New beanbot sms trojan discovered," http://www.csc.ncsu.edu/faculty/jiang/BeanBot/.

[30] M. X. Lu Gong, "Beanbot analysis report," https://github.com/mingyuan-xia/AppAudit/wiki/BeanBot-analysis-report.

[31] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638.

[32] "Free apps used to spy on millions of phones: Flashlight program can be used to secretly record location of phone and content of text messages," http://www.dailymail.co.uk/news/article-2808007/Free-apps-used-spy-millions-phones-Flashlight-program-used-secretly-record-location-phone-content-text-messages.html.

[33] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '12. New York, NY, USA: ACM, 2012, pp. 71–72.

[34] S. Lee, E. L. Wong, D. Goel, M. Dahlin, and V. Shmatikov, "πbox: A platform for privacy-preserving apps," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '13, 2013, pp. 501–514.

[35] X. Zhang, A. Ahlawat, and W. Du, "Aframe: Isolating advertisements from mobile applications in android," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13. New York, NY, USA: ACM, 2013, pp. 9–18.

[36] G. Russello, A. B. Jimenez, H. Naderi, and W. van der Mark, "Firedroid: Hardening security in almost-stock android," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13. New York, NY, USA: ACM, 2013, pp. 319–328.

[37] Y. Huang, P. Chapman, and D. Evans, "Privacy-preserving applications on smartphones," in *Proceedings of the 6th USENIX Conference on Hot Topics in Security*, ser. HotSec'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 4–4.

[38] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications," in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. SEC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 27–27.

[39] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Flexible and fine-grained mandatory access control on android for diverse security and privacy policies," in *Proceedings of the 22Nd USENIX Conference on Security*, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 131–146.

[40] M. D. Ernst, "Static and dynamic analysis: Synergy and duality," in *WODA 2003: ICSE Workshop on Dynamic Analysis*, Portland, OR, May 9, 2003, pp. 24–27.

[41] 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 Network and Distributed System Security*, ser. NDSS '12, 2012.

[42] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 235–245.

[43] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th International Conference on Mobile systems, Applications, and Services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 281–294.

[44] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 639–652.

[45] W. R. Bush, J. D. Pincus, and D. J. Sielaff, "A static analyzer for finding dynamic programming errors," *Softw. Pract. Exper.*, vol. 30, no. 7, pp. 775–802, Jun. 2000.

[46] K.-K. Ma, K. Y. Phang, J. S. Foster, and M. Hicks, "Directed symbolic execution," in *Proceedings of the 18th International Conference on Static Analysis*, ser. SAS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 95–111.

[47] M. Gorbovitski, Y. A. Liu, S. D. Stoller, T. Rothamel, and T. K. Tekle, "Alias analysis for optimization of dynamic languages," in *Proceedings of the 6th Symposium on Dynamic Languages*, ser. DLS '10. New York, NY, USA: ACM, 2010, pp. 27–42.