the extra effort needed to ensure that their obfuscations result in installable and runnable apps.

7 THREATS TO VALIDITY

This section presents our study's threats to external and construct validity, and the actions we have taken to mitigate them.

External validity measures the extent to which the results of our study can be generalized. One threat to external validity for our study is whether our study's findings can be generalized to other apps outside of our study. To mitigate this threat, we obtained benign and malicious apps from diverse sources that vary across application domains, in terms of app size, and originate from various time periods.

To ensure our findings are likely to generalize to other obfuscation strategies and tools, we employed 29 obfuscation strategies from 7 obfuscation tools—the largest number of strategies and tools utilized to date for a study about app obfuscation. We further obtained obfuscation tools that are academic, open-source, and commercial—aiding in generalizability to these three different sources.

Another threat to external validity is our selection of anti-malware products. To mitigate this threat, we have selected over 60 anti-malware products from VirusTotal, and focused on the most popular and well-rated 21 products for our study, due to space limitations. However, we make the results of our study, along with the complete list of anti-malware products and apps, available online [21]. The findings for the anti-malware products not discussed in this paper are consistent with the findings in this paper.

Construct validity is concerned with whether our study's measurements or measurement procedures validly quantify the constructs or concepts we intend to quantify. A threat to construct validity is the metrics and measurement procedures we used to quantify the ability of obfuscation tools to produce runnable apps whose behavior before obfuscation is similar to behavior after obfuscation. To measure these constructs, we compared the set or sequence of running components and thrown exceptions of apps before and after obfuscation. These measurement procedures and associated metrics are sensible given that components are functional units of behavior—making them a sensible means of identifying high-level behavior—and exceptions are the main means of identifying errors in apps whose test cases and oracles are unavailable, which is the case for many apps on Google Play.

Another threat to construct validity is the labeling of our apps as benign or malicious. To mitigate this threat, our dataset of benign apps are marked as benign by over 50 anti-malware products. Similarly, our malicious apps are obtained from repositories containing apps manually labeled as malicious by security experts.

8 RELATED WORK

We divide previous work related to our study into four categories: (1) studies about similarity of a repackaged app with its original version, (2) obfuscation strategies for PC and desktop software, (3) obfuscation tools specifically designed for Android, and (4) studies about the effects of obfuscation on anti-malware products. In the remainder of this section, we discuss each of these areas of related work and conclude the section with the key differences between our study and the most similar related work.

Researchers have studied the similarity between original apps and repackaged apps. Huang et al. [34] used Androguard [2], an Android reverse-engineering framework for malware analysis, to study the obfuscation resilience of repackaging detection algorithms. Faruki et al. [32] compared the performance of anti-malware products and Androguard's code similarity with AndroSimilar [33], their tool for detecting obfuscated apps. Wang and Rountev describe an approach for determining which obfuscation tool was applied to an obfuscated app [46].

A few studies have considered the application of obfuscation strategies in the context of PC and desktop software. Collberg et al. produced a taxonomy of transformations for obfuscation with a focus on Java [31]. Collberg et al. [30] also implemented a tool called SandMark for evaluating the effectiveness of code obfuscation to protect Java-based software systems from piracy, tampering, and reverse engineering. Christodorescu and Jha [28] evaluated the resilience of anti-malware products against code obfuscation applied to Visual Basic programs and proposed a semantics-aware malware-detection algorithm in [29].

Previous work has produced a few obfuscation tools specifically designed for Android apps. Zheng et al. [47] proposed ADAM, a framework for obfuscating Android apps and testing them on anti-malware products. They evaluated ADAM's effectiveness for evading 10 anti-malware products on 222 transformed, malicious apps. Rastogi et al. [44] presented DroidChameleon, a tool for obfuscating Android apps, and assessed obfuscations on six apps from Android Malware Genome [49].

Another set of studies focused on the effects of obfuscations on anti-malware products, without proposing new obfuscation tools. Maiorca et al. [40] studied the effects of code obfuscated by a single tool on 13 anti-malware products. Pomilia [43] studied the performance of 9 anti-malware products on a dataset obfuscated using Allatori. Morales et al. [41] studied the resilience of 4 anti-malware products after transforming 2 viruses on Windows Mobile OS.

All the aforementioned previous work that have studied either obfuscation tools or the effects of obfuscation strategies on anti-malware products focus on a single obfuscation tool and a small number of anti-malware products and apps. None of these studies have performed a large-scale empirical study considering the effects that occur due to the concurrent utilization of anti-malware products, various obfuscation tools, and their supported obfuscation strategies. None of these studies assessed these effects on benign apps; the effects of combining obfuscation strategies; and the ability of obfuscation tools to produce valid, installable, and runnable apps.

9 CONCLUSION

In this paper, we have evaluated the effectiveness of the top-rated Android anti-malware products against code obfuscation. We used a large dataset consisting of 3,000 benign apps, 3,000 malicious apps, and 73,362 obfuscated apps. To obfuscate Android apps, we applied 29 different obfuscation strategies using 7 commercial, open-source, and academic obfuscation tools. Our study includes the following key findings: (1) all anti-malware products succumb to code obfuscation; (2) most products are susceptible to trivial transformations, such as simple changes to the Android manifest file; (3) the detection rates of anti-malware products depend not only on the obfuscation strategy applied but also on the leveraged obfuscation tool; (4) anti-malware products are slow to adopt signatures of malicious apps; and (5) code obfuscation to a significant extent results in apps that do not exhibit the same behavior as their original versions. Based on our overall findings, we have provided guidance and recommendations for improving anti-malware products and obfuscation tools. To that end, Our study results, including the framework that we developed to conduct this study, are available publicly online [21].

10 ACKNOWLEDGMENTS

This work was supported in part by awards CCF-1252644, CNS-1629771, and CCF-1618132 from the National Science Foundation, HSHQDC-14-C-B0040 from the Department of Homeland Security, and FA95501610030 from the Air Force Office of Scientific Research.

REFERENCES

[1] Apktool. https://ibotpeaches.github.io/Apktool/. (2010).
[2] Androguard: Reverse engineering and malware analysis of Android apps by BlackHat. https://github.com/androguard. (2011).
[3] Allatori Obfuscator. http://www.allatori.com/. (January 2012).
[4] VirusTotal-Free virus, malware and URL scanner. https://www.virustotal.com/en. (2012).
[5] VirusShare. http://virusshare.com/. (August 2013).
[6] Contagio Malware Repository. http://contagiodump.blogspot.it. (2015).
[7] Brain Test Lookout Report. https://blog.lookout.com/blog/2016/01/06/brain-test-re-emerges/. (2016).
[8] DressCode Android malware. http://blog.checkpoint.com/2016/08/31/dresscode-android-malware-discovered-on-google-play/. (2016).
[9] Kaspersky Security Bulletin. https://kasperskycontenthub.com/securelist/files/2016/12/Kaspersky_Security_Bulletin_2016_Review_ENG.pdf. (2016).
[10] McAfee mobile threats report. https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf. (2016).
[11] Smartphone OS Market Share, 2017 Q1. International Data Corporation. http://www.idc.com/promo/smartphone-market-share/os. (2016).
[12] VikingHorde Android malware. http://blog.checkpoint.com/2016/05/09/viking-horde-a-new-type-of-android-malware-on-google-play/. (2016).
[13] Android Open Source Project. https://source.android.com/. (July 2017).
[14] Android Studio. https://developer.android.com/studio/build/shrink-code.html. (2017).
[15] DashO. https://www.preemptive.com/. (2017).
[16] Dex2jar: Tools to work with android. dex and java. class files. https://github.com/pxb1988/dex2jar. (2017).
[17] DexGuard. https://www.guardsquare.com/en. (2017).
[18] FalseGuide Android malware. http://blog.checkpoint.com/2017/04/24/falseguide-misleads-users-googleplay/. (2017).
[19] Google Play App Store. https://play.google.com/store?hl=en. (2017).
[20] jarsigner - JAR Signing and Verification Tool. https://docs.oracle.com/javase/6/docs/technotes/tools/windows/jarsigner.html. (2017).
[21] Obfuscation Study Framework. http://www.ics.uci.edu/~seal/projects/obfuscation/index.html. (August 2017).
[22] ProGuard. https://www.guardsquare.com/en/proguard. (2017).
[23] Smali/Backsmali. https://github.com/JesusFreke/smali. (2017).
[24] UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html. (August 2017).
[25] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Are your training datasets yet relevant. In *International Symposium on Engineering Secure Software and Systems*. Springer, Milan, Italy, 51–67.
[26] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, Austin, Texas, 468–471.
[27] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceeding of Network and Distributed System Security Symposium*. San Diego, California.
[28] Mihai Christodorescu and Somesh Jha. 2004. Testing malware detectors. *International Symposium on Software Testing and Analysis (ISSTA'04)* (July 2004).
[29] Mihai Christodorescu, Somesh Jha, Sanjit A Seshia, Dawn Song, and Randal E Bryant. 2005. Semantics-aware malware detection. In *IEEE Symposium on Security and Privacy*. IEEE, Oakland, CA, 32–46.
[30] Christian Collberg, GR Myles, and Andrew Huntwork. 2003. Sandmark-a tool for software protection research. *IEEE security & privacy* 99, 4 (2003), 40–49.
[31] Christian Collberg, Clark Thomborson, and Douglas Low. 1997. *A taxonomy of obfuscating transformations.* Technical Report TR148. Department of Computer Science, The University of Auckland, New Zealand.
[32] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. 2014. Evaluation of android anti-malware techniques against dalvik bytecode obfuscation. In *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, Beijing, China, 414–421.
[33] Parvez Faruki, Vijay Laxmi, Ammar Bharmal, Manoj Singh Gaur, and Vijay Ganmoor. 2015. AndroSimilar: Robust signature for detecting variants of Android malware. *Journal of Information Security and Applications* 22 (June 2015), 66–80.
[34] Heqing Huang, Sencun Zhu, Peng Liu, and Dinghao Wu. 2013. A framework for evaluating mobile app repackaging detection algorithms. In *International Conference on Trust and Trustworthy Computing*. Springer, London, UK, 169–186.
[35] Li Li, Tegawendé François D Assise Bissyande, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Yves Le Traon. 2016. *Static analysis of android apps: A systematic literature review.* Technical Report. SnT.
[36] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Haipeng Cai, David Lo, and Yves Le Traon. 2017. Automatically locating malicious packages in piggybacked android apps. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17)*. IEEE Press, Buenos Aires, Argentina, 170–174.
[37] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security* 12, 6 (June 2017), 1269–1284.
[38] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding Android app piggybacking. In *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, Buenos Aires, Argentina, 359–361.
[39] Federico Maggi, Andrea Valdi, and Stefano Zanero. 2013. AndroTotal: a flexible, scalable toolbox and service for testing mobile malware detectors. In *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*. ACM, Berlin, Germany, 49–54.
[40] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. 2015. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security* 51 (March 2015), 16–31.
[41] Jose Andre Morales, Peter J Clarke, Yi Deng, and BM Golam Kibria. 2006. Testing and evaluating virus detectors for handheld devices. *Journal in Computer Virology* 2, 2 (2006), 135–147.
[42] Damien Octeau, Somesh Jha, and Patrick McDaniel. 2012. Retargeting Android applications to Java bytecode. In *International Symposium on the Foundations of Software Engineering*. ACM, Cary, North Carolina, 6.
[43] M. Pomilia. A Study on Obfuscation Techniques for Android Malware. (2016). http://www.dis.uniroma1.it/~midlab
[44] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. 2014. Catch me if you can: Evaluating android anti-malware against transformation attacks. *IEEE Transactions on Information Forensics and Security* 9, 1 (2014), 99–108.
[45] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. 1999. Soot - a Java Bytecode Optimization Framework. In *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '99)*. IBM Press. http://dl.acm.org/citation.cfm?id=781995.782008
[46] Yan Wang and Atanas Rountev. 2017. Who Changed You? Obfuscator Identification for Android. (May 2017).
[47] Min Zheng, Patrick PC Lee, and John CS Lui. 2012. ADAM: an automatic and extensible platform to stress test android anti-virus systems. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, Heraklion, Crete, Greece, 82–101.
[48] Wu Zhou, Yajin Zhou, Michael Grace, Xuxian Jiang, and Shihong Zou. 2013. Fast, scalable detection of piggybacked mobile applications. In *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, San Antonio, TX, 185–196.
[49] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy*. IEEE, San Francisco, California, 95–109.