Poster: Experiences of Landing Machine Learning onto Market-Scale Mobile Malware Detection

Hao Lin[1,*,†], Liangyi Gong[1], Zhenhua Li[1], Feng Qian[2], Zifan Zhang[1,3,*]
Qi Alfred Chen[4], Zhiyun Qian[5], Yunhao Liu[1,6]
[1]Tsinghua University [2]University of Minnesota – Twin Cities [3]Tencent Co. Ltd.
[4]University of California, Irvine [5]UC Riverside [6]Michigan State University

Abstract

Today app markets play a crucial role in publishing, updating and distributing mobile apps. However, they in turn have become a convenient channel for mobile malware to spread. Worse still, these app markets in a sense "lend credibility" to the apps published on them. In the past decade, machine learning (ML) techniques have been widely explored for automated and robust malware detection [2]. Unfortunately, we have yet to see any report of an ML-based malware detection solution deployed at market scales.

To learn the real-world challenges, we collaborate with a major Android app market Tencent App Market, or T-Market, which offers us large-scale ground-truth data. We comprehensively analyze the data and explore existing ML-based malware detection solutions on multiple dimensions. As a result, we confirm that the key challenges of developing such systems lie in *feature selection*, *feature engineering*, *app analysis speed*, ML *model evolution*, and *developer engagement*.

More importantly, we notice that failure in any of the above aspects would lead to the "wooden barrel effect" (and thus vain efforts) of the whole solution. Given the complex interplay among these five aspects, we document our judicious considerations and experiences in building an ML-based detection system, APICHECKER, which successfully improves and balances all five aspects (and thus no wooden barrel effect). It has been operational at T-Market since March 2018, vetting over 10K apps every day with a precision of 98%, recall of 96%, and an average per-app scan time of 1.3 minutes.

Feature selection: a principled, data-driven approach for the concrete selection of API features. In our design, we adopt *API-centric dynamic analysis* for feature extraction, which tracks Android SDK APIs at an app's runtime. However, with over 50K APIs in Android SDK, we need to carefully select critical APIs that can effectively balance detection accuracy and analysis speed. To this end, we extensively analyze our large dataset and reveal several intriguing findings. We find that strategically tracking a subset of the entire SDK APIs not only improves analysis speed, but also yields a better accuracy. Also, different sources of APIs complement each other and further improve accuracy. Guided by the findings, finally a total of 426 key APIs are selected as features. Detailed feature selection can be found in our full paper [1].

Feature engineering: an adversary's perspective for hidden features. We realize that solely relying on Android APIs is problematic due to adversaries' bypassing API invocations with other mechanisms, *e.g.,* Java reflection and intents. To address this, we further capture the requested permissions and used intents of apps in analysis. This information helps unveil hidden API invocations (hidden features), and provides a more complete landscape of apps' runtime behaviors.

Analysis speed: efficient app emulation on x86 servers. To boost the analysis speed, we incorporate the native x86 port of Android with *dynamic binary translation*, which translates ARM native code to x86, to build a efficient emulation infrastructure. Compared to Google's QEMU-based emulator with full-system binary translation, we achieve 70% reduction in app emulation time. As a result, we can vet an app in ~1.3 minutes on a single x86 server. For the concrete design of our emulation infrastructure, please refer to our full paper [1].

Model evolution: automatically updating the ML model with novel apps and Android SDK APIs. We combine the original dataset with the detection results of the continuous new app submissions in T-Market to automatically evolve our deployed model on a monthly basis. Based on our observations of real-world deployment, the number of selected key APIs only slightly fluctuates between 426 and 432, indicating that APICHECKER is robust to API evolution.

Developer engagement: complete avoidance of false positives. False positives and negatives are inevitable for an ML solution. Among them, false positives impact developer engagement while false negatives affect end users' experiences. We choose to actively and fully avoid the former by manually checking all apps vetted as malicious. Fortunately, ~90% of submissions are updates that can be quickly inspected. Thus, the incurred labor work of addressing false positives is acceptable in practice. However, the latter can hardly be eliminated, and thus we only address them based on user reports.

References

[1] Liangyi Gong, Zhenhua Li, Feng Qian, Zhang Zifan, Qi Alfred Chen, Zhiyun Qian, Hao Lin, and Yunhao Liu. 2020. Experiences of Landing Machine Learning onto Market-Scale Mobile Malware Detection. In *Proc. of EuroSys*. ACM.

[2] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. 2012. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *Proc. of NDSS*. Internet Society.

*Student † Presenter.