Reliable Third-Party Library Detection in Android and its

Security Applications

Third-party libraries on Android have been shown to be se-curity and privacy hazards by adding security vulnerabilitiesto their host apps or by misusing inherited access rights.Correctly attributing improper app behavior either to appor library developer code or isolating library code from theirhost apps would be highly desirable to mitigate these prob-lems, but is impeded by the absence of a third-party librarydetection that is effective and reliable in spite of obfuscatedcode. This paper proposes a library detection technique thatis resilient against common code obfuscations and that iscapable of pinpointing the exact library version used in apps.Libraries are detected with profiles from a comprehensivelibrary database that we generated from the original librarySDKs. We apply our technique to the top apps on GooglePlay and their complete histories to conduct a longitudinalstudy of library usage and evolution in apps. Our resultsparticularly show that app developers only slowly adapt newlibrary versions, exposing their end-users to large windowsof vulnerability. For instance, we discovered that two long-known security vulnerabilities in popular libs are still presentin the current top apps. Moreover, we find that misuse ofcryptographic APIs in advertising libs, which increases thehost apps’ attack surface, affects 296 top apps with a cu-mulative install base of 3.7bn devices according to Play. Tothe best of our knowledge, our work is first to quantify thesecurity impact of third-party libs on the Android ecosystem.

Third-party libraries have become a fixed part of mobileapps. Developers use them to, e.g., monetize their appsthrough advertisements, integrate their apps with online so-cial media, include single-sign-on services, or simply leverageutility and convenience libraries for their apps’ functionality.However, third-party libraries are a double edged sword:While they can provide convenience for the app developerand can greatly enhance their host apps’ features, they alsohave been shown to be a hazard to the end-users’ privacyPermission to make digital or hard copies of all or part of this work for personal orclassroom use is granted without fee provided that copies are not made or distributedfor profit or commercial advantage and that copies bear this notice and the full citationon the first page. Copyrights for components of this work owned by others than theauthor(s) must be honored. Abstracting with credit is permitted. To copy otherwise, orrepublish, to post on servers or to redistribute to lists, requires prior specific permissionand/or a fee. Request permissions from permissions@acm.org.CCS ’16, October 24–28, 2016, Vienna, Austria© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.ISBN 978-1-4503-4139-4/16/10...$15.00DOI: http://dx.doi.org/10.1145/2976749.2978333and security. A number of prior studies [10, 14, 4, 30, 34] hasdemonstrated that such libraries exhibit questionable privacypractices. For instance, they leak user-private information,exploit their host app’s privileges, or track users. Two re-cent incidents of such questionable practices were revealedin the popular SDKs of Taomike—China’s biggest mobile adprovider—and Baidu, that were found to be secretly spyingon users and uploading their SMS to remote servers [37] andopening backdoors to the users’ devices [35], respectively. Inaddition to such privacy violations, third-party libs increasethe attack surface of their host apps when they do not ad-here to security best practices and, hence, become a liabilityfor the users’ security. In the recent past, even popular li-braries by reputable software companies, such as Facebookand Dropbox, were affected by highly severe vulnerabilities.The found vulnerabilities could lead to the leakage of sensi-tive data to publicly readable data-sinks [26], code injectionattacks [28, 39], account hijacking [36], or linking a victim’sdevice to an attacker-controlled Dropbox account [8].Given the high prevalence of third-party libraries in appsand consequently their high impact on the health of the en-tire smartphone ecosystem, it is of no surprise that dedicatedresearch has investigated new mechanisms to sandbox orremove libs, with a strong focus on advertisement libs [31, 27,44, 30]. Yet, these proposals make one crucial assumptionthat currently limits them in their effectiveness: they assumethat libraries can be clearly identified, either through devel-oper input [27] or inspection of the app’s code [31, 44, 30].Reliably identifying libraries, however, forms a formidableand yet unsolved technical challenge. First, third-party li-braries are tightly integrated into their host app by staticallylinking them during the app’s build process into the app’sbytecode, thus blurring the boundaries between app andlibrary code. Second, app developers commonly make useof bytecode obfuscation tools, such as ProGuard [16]. Oneside-effect-free bytecode obfuscation technique is identifierrenaming. It turns identifiers into short, non-meaningfulstrings, i.e. a package name com.google is transformed intoa.c. Naïve library detection approaches based on identifiermatching, like in [34, 14, 31, 44] or applied by third-party addetector apps, fail even to this simple obfuscation technique.Another problem, caused by the inability to reliably detectthird-party libraries within applications, is the lack of ac-countability for privacy and security violations. For instance,a wide range of security-related analyses studied apps forprivacy and security issues and raised awareness for vari-ous problem areas, including privacy leaks [12, 2, 42, 13, 3],permission usage [43], dynamic code loading [28], SSL/TLS(in-)security [25, 11], or (mis-)use of cryptographic APIs [9].However, without being able to distinguish app developercode from third-party library code, the reported results areon a per-app basis and do not distinguish whether bad orimproper behavior originates from app or library developers.To increase efficiency of library sandboxing mechanismsand to be able to hold the correct principal (app or libdeveloper) accountable for security and privacy violations, areliable and precise third-party library detection is requiredthat is resilient against common obfuscation techniques.Our contributions.In this paper, we make two tangible contributions: First,we present an efficient and reliable approach for detectingthird-party libraries within Android apps (see Section 4). Byanalyzing the original library SDKs, we extract profiles thatare resilient to common obfuscation techniques, such as iden-tifier renaming and API hiding. To achieve these propertiesour approach is based on class hierarchy information onlyand is independent of the libraries’ code. Still, our profilesare fine-grained enough to not only detect distinct libraries,but also the exact version used in an app. For the actuallibrary detection, we devised a profile matching algorithmthat reports whether an exact copy of a given library versionwas matched. In the negative case, either because the correctversion is missing in our database or dead-code eliminationwas applied to app code, a similarity score indicates the bestmatching profile for the library code in the app.Second, we use our library detection technique in a lon-gitudinal study of third-party libraries included in the topapps on Google Play (see Section 6). In this study, we areinterested in finding answers to security- and privacy-relatedquestions about libraries, such as “How prevalent are third-party libraries in the top apps and how up-to-date are thelibrary versions?”, “Do app developers update the libs includedin their apps and how quickly do they update?”, or “Howprevalent are vulnerabilities identified in prior research [28,9] in libraries and how many apps are affected?” To answerthese questions, we first built a comprehensive repository ofthird-party libraries and applications (see Section 5). Ourlibrary set contains 164 libraries of different categories (Ad-vertising, Cloud,..) and a total of 2,065 versions. We thencollected and tracked the version histories for the top 50apps of each category on Play between Sep 2015 and July2016, accumulating to 96,995 packages from 3,590 apps. Wecomplemented this database with meta-information, suchas app and library release dates, which we collected frompublic sources or developer websites. Based on this data set,we show that app developers commonly neglect updates ofthird-party libraries. By analyzing the time-to-fix of tworecent security/privacy incidents of the Facebook and Drop-box SDKs [36, 8], we show that this developer negligencein updating libraries exposes end-users to large windowsof opportunities for attacks (e.g., on average 190 days for51 apps with a vulnerable Facebook SDK in our data set).Lastly, we scan our library set for presence of API misusevulnerabilities [28, 9] that would expose the libs’ host appsto cryptanalytic and code injection attacks and discover 18vulnerable libs (61 versions), which together affect 296 appswith a cumulative install base of 3.7bn. Overall, our workconstitutes the first longitudinal security study of third-partylibraries in the Android ecosystem. We provide the first valu-able insights into the security-impact of libraries and as suchmotivate future work on improving library updatability onAndroid. In summary, we make the following contributions:1) We are the first to devise a light-weight and effectiveapproach (LibScout 1 ) to detect third-party libraries in An-droid apps that is resilient to common obfuscation techniquesand capable of pinpointing exact library versions.2) We created a large third-party library database including164 distinct libraries with 2,065 versions, which we profile.We collected the version history of 3,590 top apps on Playfor a total of 96,995 distinct packages. We complement thosedatabases with meta-data such as app/library release dates.3) We conduct a longitudinal study of third-party libs in ourapp set to investigate their prevalence, the update frequencyof apps and of libs, as well as the impact of app popularityand library API stability on the lib update frequency.4) We study time-to-fix and vulnerability windows of appsthat include vulnerable library versions (at the example ofrecently reported incidents of the popular Facebook andDropbox SDK). Our results show large windows of opportu-nity for attackers against apps including those libraries.5) Lastly, we re-apply existing app analysis techniques tolibrary code to investigate improper usage of dynamic codeloading and crypto APIs. We identify 61 library versions thataffect 296 top apps on Play with a cumulative install-base of3.7bn users and expose these apps to cryptanalytic attacks.2. RELATED WORKCode and app clone detection techniques for Android appshave been studied in many different respects. Prior workcomputed the similarity between apps using code-based simi-larity techniques [5, 17, 48] or by extracting semantic featuresfrom program dependency graphs [6, 7].Other approaches are tailored to third-party libraries onAndroid, e.g., by employing the concept of whitelisting pack-age names to detect libraries within app code [14, 4, 5].As such approaches fail to cope with even simple obfusca-tion techniques such as identifier renaming, more robustapproaches based on machine learning or code clusteringhave been investigated. AdDetect [24] and PEDAL [20] usemachine-learning to detect advertising libraries. AdDetectuses hierarchical package clustering to detect (non-)primarymodules of apps whereas PEDAL extracts code-features fromlibrary SDKs and uses package relationship information totrain a classifier to detect libraries even when identifiers areobfuscated. AnDarwin [7] and WuKong [40] detect app cloneswith high accuracy by filtering library code that is detectedby means of code clustering techniques. Such approachesrely on the assumptions that libraries are pervasively usedby many apps, and that app developers do not modify thelibrary. However, this second assumption is unrealistic, sinceautomatic/manual dead-code elimination during app build-ing will necessarily modify the library code 2 . Moreover, theseapproaches only provide binary classifications since they can-not name the concrete library versions used within the apps.The recent LibRadar [22] extends WuKong’s clustering ap-proach and generates unique profiles for each detected cluster.Profiles are generated from the frequency of API calls withindistinct packages in a cluster and can subsequently be usedfor fast library detection. With this approach, LibRadar1 https://projects.cispa.uni-saarland.de/derr/libscout2 https://developer.android.com/studio/build/shrink-code.htmlAllatori dashO DexGuard DexProtector DIVILAR ProGuard Stringer[32] [29] [15] [18] [47] [16] [19]API hiding (\*) – – ? ? – – –Class encryption – – ? ? – – –Control-flow randomization (\*) ? ? – – – – –Identifier renaming (\*) ? ? ? ? – ? ?String encryption (\*) ? ? ? ? – – ?Virtualization-based protection – – – – ? – –Table 1: Feature comparison of Android app obfuscators. Our approach is robust against features marked with (\*).was able to find 29K potential libraries on a large corpus ofGoogle Play apps. This number presumably constitutes anover-approximation, since the original code clustering andthe subsequent feature extraction are not performed on theoriginal libraries. This lack of ground truth produces falsepositives when multiple libraries have the same root package(e.g. com.google for the various Google Play Service librariesand Google libs like Gson/Guice). To avoid such heuristics,we extract our profiles from the original library binaries. Thiscomes at the cost of completeness but has several advantagessuch as less false positives and the possibility of inferringexact library versions. In addition, this allows computationof more reliable similarity values for partial library inclusions(as a result of code optimizations).Besides library detection, research has also proposed tech-niques for privilege separation between host apps and adver-tising libraries. To this end, AdSplit [31] puts library codeinto separate processes, while PEDAL [20] uses an inlinereference monitoring approach to allow users to selectivelyenable/disable functionality in libs that requires a permission.AdDroid [27] uses a system-centric approach and proposesa new ad-API in Android’s application framework for appdevelopers to allow privilege separation by construction. Asa more rigorous approach, APKLancet [44] removes malwareand ad library code from an app package. The code to beremoved is identifed through semantic fingerprints that havepreviously been extracted from malware/library samples.A separate line of work studied questions related to secu-rity and privacy in advertising libraries. Stevens et al. [34]investigate the permission (mis-)use of ad libs. Book et al. [4]conducted a longitudinal study of ad lib permissions anddiscovered that the absolute number of required permissionsincreases over time. However, in contrast to our longitudinalstudies, they estimated the library release dates as the termi-nus ad quem of the release date of an app that includes thelibrary. Library profiles are generated by hashing the librarycode detected via package name matching. In this case, thedetection only works if the original library code is includedwithout any modifications. Other approaches analyse pri-vate data exfiltration [14, 33] and security vulnerabilitiesin authentication/authorization SDKs [41]. In Section 7,we re-apply existing app security analyses to library codeand use LibScout to measure the real-world impact of thediscovered vulnerabilities in our app set.

Android及其安全应用程序中可靠的第三方库检测

Android上的第三方库已被证明是安全和隐私的危险，通过增加他们的主机应用程序的安全漏洞或滥用继承的访问权限。正确地归因不适当的应用程序行为，以appor图书馆开发者代码或孤立库代码从他们的主机应用程序会非常希望缓解这些问题，但是由于缺乏第三方库检测而受到阻碍，尽管存在混淆的代码，该检测仍然有效且可靠。本文提出了一种图书馆检测技术，它可以抵御常见的代码混淆，并且能够查明应用程序中使用的确切的库版本。通过从我们从原始库SDK生成的综合库数据库中检测配置文件来检测库。我们将我们的技术应用于GooglePlay上的顶级应用及其完整历史记录，以对应用中的库使用和演变进行纵向研究。我们的结果特别表明，应用程序开发人员只能慢慢适应新的图书馆版本，使他们的最终用户暴露于大型的Windows漏洞。例如，我们发现流行库中的两个已知安全漏洞仍存在于当前的顶级应用程序中。此外，根据Play的研究，我们发现滥用广告库中的加密API会增加主机应用程序的攻击面，从而影响296个顶级应用程序，其中包含3.7亿个设备的总体安装量。尽我们所知，我们的工作首先是量化第三方库对Android生态系统的安全影响。

第三方库已成为移动应用程序的固定部分。开发人员使用它们来例如通过广告实现货币化，将其应用程序与在线社交媒体集成，包括单一登录服务，或者简单地为其应用程序的功能提供功能和便利库。但是，第三方库是双刃剑：虽然它们可以为应用程序开发提供方便，并且可以大大提高其主机应用程序的功能，但它们也被证明是对最终用户隐私的危害。允许制作全部或部分此工作的数字或硬拷贝为个人或课堂使用的授予不收取费用，前提是复制品不是为了营利或商业利益而制作或分发的，并且复制品在第一页中包含本通知和完整的引文。必须尊重他人拥有的作品组成部分的版权。允许用信用抽象。要另外复制或重新发布，要在服务器上发布或重新发布到列表，需要事先具体许可和/或收费。请求权限：permissions@acm.org.CCS 2016年10月24日至28日，维也纳，奥地利©2016版权归作者/作者所有。授权给ACM.ISBN 978-1-4503-4139-4 / 16/10的发布权... $ 15.00DOI：http://dx.doi.org/10.1145/2976749.2978333和安全性。一些先前的研究[10,14,4,30,34]已经证明这些图书馆存在可疑的隐私实践。例如，他们泄漏用户私人信息，利用其主机应用程序的权限或跟踪用户。在Taomike--中国最大的移动广告供应商 - 百度的流行SDK中，发现了两起此类可疑做法的最新事件，这些事件被发现是秘密间谍用户并将他们的短信上传到远程服务器[37]，并打开用户设备的后门[35]，分别。除了这样的隐私侵犯之外，当第三方库不适用于安全最佳实践时，第三方库增加了其主机应用程序的攻击面，因此成为用户安全性的责任。最近，即便是信誉良好的软件公司，如Facebook和Dropbox，受到高度严重漏洞的影响，发现的漏洞也可能导致敏感数据泄漏到公众可读的数据库中[26]，代码注入攻击[28,39]，帐户劫持[36]，或将受害者的设备链接到攻击者控制的Dropbox帐户[8]。鉴于应用程序中第三方库的高流行性，因此它们对这是一个全面的智能手机生态系统，专门研究调查了沙盒或去除库的新机制并不奇怪，并且强调了广告资源库[31,27,44,30]。然而，这些建议提出了一个至关重要的假设：目前限制了它们的有效性：它们假设通过开发输入[27]或检查应用程序的代码[31,44,30]，可以清楚地识别图书馆。可靠地识别图书馆，然而，这形成了一个艰巨而尚未解决的技术挑战。首先，通过在应用程序的构建过程中将它们静态链接到应用程序的字节码中，第三方内容与他们的主机应用程序紧密集成，从而模糊了应用程序和库代码之间的界限。其次，应用程序开发人员通常使用字节码混淆工具，如ProGuard [16]。无效的字节码混淆技术是标识性的。它将标识符变成简短而无意义的字符串，即将com.google的包名称转换为a.c文件。基于标识匹配的简单的库检测方法，如[34,14,31,44]，或由第三方addetector应用程序应用，甚至无法实现这种简单的混淆技术。另一个问题，由于无法可靠地检测到第三方库应用程序是缺乏隐私和安全侵犯的可计数性。例如，广泛的安全相关分析研究了应用程序的隐私和安全问题，并提高了对各种问题领域的认识，包括隐私泄漏[12,2,42,13,3]，权限使用[43]，动态代码加载[28]，SSL / TLS（in）安全[25,11]或加密API的使用（错误）[9]。然而，如果不能区分应用程序开发者代码与第三方库代码，结果不是基于每个应用程序的基础，并且不区分坏的或不恰当的行为是源自应用程序还是库开发人员。为了提高库沙箱机制的效率并且能够保持正确的主体（应用程序或libdeveloper）对安全和隐私侵犯负责，可以并且需要精确的第三方库检测，这对于常见的混淆技术具有适应性。我们的贡献。本文中，我们做出了两项具体的贡献：首先，我们提出了一种检测Andr中第三方库的高效可靠方法oid应用程序（请参阅第4节）。通过分析原始库SDK，我们extrac