VI. DISCUSSION

The data presented in this study makes two conclusions

clear: that target fragmentation exists across the entire Android ecosystem and that target fragmentation has practical

implications on Android security. Making new Android versions compatible with un-updated apps ensures that apps do

not suddenly break, but distributing security changes in this

manner has clear limitations. This approach makes security

changes optional and mixes security changes with non-security

changes. Developers cannot pick-and-choose just the security

changes but must integrate all of the platform changes from a

new API level. Developers could try to sidestep this problem

by setting a maximum API level but this feature is not

enforced and would exacerbate device fragmentation problem

by discouraging platform updates.

With this in mind, we consider the alternative to the current

system: enforcing all security changes to the Android platform

regardless of target API level. Nearly four months since the

release of API level 23, less than 1% of active devices have

updated to version 23 [27]. This slow update process means

that developers have ample time to update their apps to work

with new behavior. There have also been security changes in

the past that were both mandatory and breaking that did not

appear to cause great pain. In Android level 21, a uniqueness

requirement was added for custom permissions to ensure

that malicious apps could not access protected content. This

change was mandatory and could prevent apps from being

reinstalled after a device update but searching on developer

forums reveals few complaints. We suspect that if developers

were simply forced to adjust to all security changes that it

would not be a problem for the majority of apps. If this is

not feasible then, at the very least, users should be informed

if their apps target out-of-date API levels so that they can be

empowered to make security conscious decisions.

The JavaScript Interface vulnerability described in Section V-B is a good case study to support our argument. After

failing to fix the problem in API level 17 with an optional

change, API level 19 banned all access to getClass from

the JavaScript Interface. This change is not made optional for

apps targeting lower API levels and would be unneeded if not

for the existence of apps that target API levels 16 or lower.

The vulnerability was only truly addressed with a change that

is applied to all apps, regardless of their target API version

A. Alternative Explanations

Ignorance is not the only reason why a developer might

not target the most current API level. One alternative is that

developers do not retarget their apps to the most current API

level because few devices run the most current API level. If

this were the case we would expect most apps to be no more

than one API level out of date. Although 16% of devices were

running API levels 22 or 23 when Dataset A was collected,

only 23% of apps in Dataset A targeted these levels.

Another possibility is that developers choose not to retarget

apps to higher API levels unless there is a security concern.

Because API levels 22 and 23 do not include security changes,

developers may have chosen not to retarget their apps. However, we show in Section V that numerous apps target outdated

API levels even though they miss relevant security changes

(54% of apps target API levels below 21). If we look at the

Fragment Injection vulnerability we find that apps that use a

PreferenceActivity are not more likely to target API

levels 19 or higher (66%) than the rest of the app population

(69%). This suggests that developers, as a whole, do not

consider security when deciding what API level to target.

A final option is that developers choose not to target

the most current API level to avoid breaking critical app

functionality. This is a real possibility but, if true, shows that

Google’s “all or nothing” design is flawed because it forces

developers to make an impossible choice and sacrifice security.

B. Threats to Validity

We only downloaded a subset of available apps for Dataset

A so there is some possibility of selection bias in our results.

However, a dataset size of 60,086 apps is within the normal

range for studies like ours. Because these apps were selected

randomly from the available apps we believe that the dataset

is large enough to smooth out any selection bias.

Our dataset is not a uniform snapshot of the apps available

on the Google Play store because we do not download updated

versions of previously downloaded apps. We do not have

longitudinal statistics on individual apps and we tend to have

young versions of apps. If apps that have been present on

the app store for a very long time are considerably more

likely to target current API levels then the statistics presented

in this paper may overestimate the problem from the user’s

perspective. However, there is an 18 month gap between the

collection of Dataset A and Dataset B. If long-lived apps target

outdated API levels less frequently then we would expect to

see some indication in the results for Dataset A.

Because we only study free apps, it is possible that our

results do not apply to non-free apps. We have no reason to

suspect that development practices are significantly different

for non-free apps. However, even if there is a considerable

difference between free and non-free apps, free apps comprise

89% of apps on the Google Play store [28] so any problem

present in free apps is critical and should be addressed.

Much of the analysis done in Section V assumes that apps

use certain dangerous features at uniform rates no matter

which API level they target. If this assumption is not true

then our conclusion that many apps unnecessarily use these

dangerous features because they target outdated API levels

might not be valid. In particular, it is possible that behavioral

changes discourage developers from retargeting their apps,

and we cannot assume that usage of dangerous features is

uniform across the app population. However, the differences

we obshat it would be extremely surprising for these differences to

be caused by different intended behavior in apps.erve in the usage of dangerous features are so great.

I. 讨论

本研究提供的数据有两个结论: 目标碎片存在于整个 android 系统中, 目标碎片对 android 安全有实际意义。使新的 Android 版本与联合国更新的应用程序兼容, 确保应用程序不会突然中断, 但以这种方式分发安全更改具有明显的局限性。此方法使安全更改可选, 并将安全更改与非安全更改混合在一起。开发人员不能只选择安全更改, 但必须将所有平台更改从新的 API 级别集成。开发人员可以尝试通过设置最大 API 级别来避开此问题, 但此功能不会强制执行, 并通过阻止平台更新来加剧设备碎片问题。

考虑到这一点, 我们认为是当前系统的替代方案: 对 Android 平台实施所有安全更改, 而不管目标 API 级别如何。近四月以来, API 23的发布, 不到1% 的活动设备已更新到版本 23 [27]。这个缓慢的更新过程意味着开发者有足够的时间来更新他们的应用程序来处理新的行为。过去也有安全方面的变化, 既有强制性的, 也有破坏的, 似乎没有造成巨大的痛苦。在 Android API 21中, 为自定义权限添加了唯一性要求, 以确保恶意应用程序无法访问受保护的内容。此更改是强制性的, 可以防止在设备更新后重新安装应用程序, 但在开发人员论坛上搜索却很少有抱怨。我们怀疑, 如果开发人员只是被迫调整到所有的安全更改, 它不会是一个问题的大多数应用程序。如果这是不可行的, 那么至少, 应该通知用户, 如果他们的应用程序的目标是过时的 API 级别, 以便他们可以有权作出安全意识的决定。

第五-b 节中描述的 JavaScript 接口漏洞是支持我们的论点的一个很好的案例研究。在 API 17中没有用可选的更改解决问题后, api 级别19禁止从 JavaScript 接口访问 getClass。对于针对较低 api 级别的应用程序, 此更改不是可选的, 如果不是针对 api 级别16或更低的应用程序存在, 则不需要这样做。该漏洞只是真正解决的变化,

适用于所有应用程序, 无论其目标 API 版本如何。

A. 备选解释

不了解不是开发人员可能针对最新 API 级别的唯一原因。一种选择是, 开发人员不更新他们的应用程序到最新的 api 级别, 因为很少有设备运行最新的 api 级别。如果是这样的话, 我们希望大多数应用程序不超过一个 API 级别的日期。尽管在收集数据集 a 时, 16% 的设备正在运行 API 级别22或 23, 但数据集中的应用程序中只有23% 是针对这些级别的。另一种可能性是, 除非存在安全问题, 否则开发人员选择不将应用程序更新到更高的 API 级别。

因为 API 22和23不包括安全更改, 所以开发人员可能选择不重新他们的应用程序。然而, 我们在第五节显示, 许多应用程序的目标是过时的 api 级别, 即使他们错过了相关的安全更改 (54% 的应用程序目标 api 级别低于 21)。如果我们看一下片段注入漏洞我们发现使用 PreferenceActivity 的应用程序比应用程序的其余部分 (69%) 更容易目标 API 级别19或更高 (66%)。这表明,作为一个开发商不确定目标的 API 级别时, 请考虑安全性。

最后一个选择是开发人员选择不针对最新的 API 级别, 以避免破坏关键的应用程序功能。这是一个真正的可能性, 但如果真的, 表明谷歌的 "全部或没有" 的设计是有缺陷的, 因为它迫使开发商做出一个不可能的选择和牺牲安全。

B. 对有效性的威胁

我们只为数据集下载了一个可用的应用程序子集, 因此在我们的结果中存在选择偏差的可能性。然而, 60086个应用程序的数据集大小在正常范围内, 就像我们的研究一样。因为这些应用程序是从可用的应用程序中随机选择的, 所以我们认为数据集足够大, 可以避免任何选择偏差。

我们的数据集不单单来自Google play商店 ，因为我们不下载更新版本的以前的应用程序。我们没有关于单个应用程序的纵向统计数据, 我们倾向于有新版本的应用程序。如果应用程序商店已经存在了很长的时间, 更有可能的目标是当前的 API 水平, 那么本文中的统计数据可能会高估问题从用户的角度。然而,数据集 A和数据集 B 的集合，有18月之间的差距。如果长时间的应用程序的目标是过时的 API 级别, 那么我们希望在数据集 A 的结果中看到一些指示。

因为我们只研究免费的应用程序, 所以我们的结果可能不适用于非免费的应用程序。我们没有理由怀疑, 对于非免费应用程序, 开发实践有明显的不同。然而, 即使自由和非免费的应用程序之间也有相当大的区别, 免费的应用程序在 Google 游戏商店中占了89% 的总应用程序数量 [28]， 所以在免费应用中存在的任何问题都是至关重要的, 应该加以解决。

第五节中的许多分析都假定应用程序使用特定的危险特性, 无论其目标是什么 API 级别。如果这个假设是不正确的, 那么我们的结论是, 许多应用程序不必要地使用这些危险特性, 因为它们的目标是过时的 API 级别可能无效。特别是, 行为变化可能阻止开发者重他们的应用程序, 而且我们不能假设在应用程序群体中使用危险特性是统一的。然而, 我们在使用危险特性时所观察到的差异是如此之大, 以至于这些差异会因应用程序中不同的预期行为而引起。

VII. RELATED WORK

The most similar work to this study is by McDonnell, Ray,

and Kim [29], who study the use of deprecated API methods

and the adoption of updated API methods in ten open source

Android apps. Unlike our study, which focuses on target API

levels, they focus on the usage of methods changed in new

API levels. Targeting an API level is not dependent on using

methods added in that platform version and apps should target

the most current API level even if they do not use any newly

added methods. Their results say nothing about the security

consequences of outdated apps but do show that developers

are slow to adapt to the changing Android platform.

The target fragmentation problem has been discussed in

relation to specific vulnerabilities in several studies. Thomas

et al. [5] study the changes in Android 17 that closed the

JavaScript Interface vulnerability in depth. Their study focuses

on the slow adoption of new devices and its effect on the

lifetime of the vulnerability but they also find that 22% of

studied apps use the JavaScript Interface and targeted API

levels below 17. Mutchler et al. [4] identify apps that load

untrusted content in WebView and note that the JavaScript

Interface vulnerability puts these apps at risk.

The vulnerabilities mentioned in this paper have been studied without mention of target fragmentation. Lu et al. [30]

build a static analyzer to identify vulnerabilities including

Service Hijacking. Georgiev, Jana, and Shmatikov [15] provide

a tool to prevent attacks through the JavaScript Interface. Chin

and Wagner [13] statically analyze apps and find unsafe use

of file: URLs. Jin et al. [31] build a tool to detect a variety

of XSS-like vulnerabilities in WebView apps.

Because both apps and devices must be updated in order to

take advantage of new security features, target fragmentation

and device fragmentation are linked. Thomas, Beresford, and

Rice [2] study device fragmentation using volunteers who

install their device monitor app. They find that 88% of devices

are vulnerable to at least one of selected vulnerabilities and

that devices are updated infrequently (1.26 times per year on

average). Zhou et al. [32] find that more than 1,000 of 2,423

factory images can be exploited through misconfigurations of

device drivers. Xing et al. [33] identify how apps can exploit

the OS update process to obtain sensitive system permissions.

Mulliner et al. [34] provide a scalable method for applying

third party patches to vulnerable Android devices.

The app update process has also been studied. Moller et ¨

al. [35] investigate the update patterns of Android users and

find that only half of users install an app update within

one week of the update being published. McIlroy, Ali, and

Hassan [36] mine update data from 10,713 apps and find that

only 1% of apps receive at least one update per week.

Several other studies analyze large datasets of Android apps.

Viennot, Garcia, and Nieh [10] crawl, download, and analyze

1,100,000 apps to obtain statistics about permission and library

distributions as well as identify apps that unsafely embed

credentials. Other studies [37, 38] also study permission and

library usage. Kavaler et al. [39] compare usage of Android

classes and questions asked on StackOverflow.

VII. 相关工作

这项研究最相似的工作是由McDonnell, Ray,and Kim [29],——研究使用过时的 API 方法在十个开源Android 应用程序中采用了更新的 API 方法。与我们的研究不同, 它侧重于目标 API级别, 他们侧重于在新的API 级别方法中更改的用法。针对 API 级别的目标不依赖于使用在该平台版本和应用程序中添加的方法。针对最新的 API 级别, 即使它们不使用任何新的添加的方法。他们的结果说过时的应用程序与安全无关, 但确实表明, 开发人员很慢, 无法适应不断变化的 Android 平台。

对目标碎片问题进行了讨论几个研究中的特定漏洞的关系。Thomaset al. [5] 研究了 Android API 17 关闭JavaScript 接口漏洞的深度的变化。他们重点研究新设备的缓慢采用及其对生存期的漏洞, 但他们也发现, 22% 研究的应用程序使用 JavaScript 接口和目标 API等级低于17。Mutchler et al. [4] 识别加载的应用程序web 中不受信任的内容, 并注意 JavaScript接口漏洞使这些应用程序面临风险。

本文中提到的漏洞已被研究, 而没有提到目标碎片。Lu et al.[30]。

构建静态分析器来识别漏洞, 包括服务劫持。Georgiev, Jana, 和 Shmatikov [15] 提供

一种防止通过 JavaScript 接口进行攻击的工具。 Chin 和Wagner [13] 静态分析应用程序和发现不安全的使用的文件: 网址。Jin et al. [31] 建立一个工具来检测各种web 应用程序中的类似 XSS 的漏洞。

因为必须更新应用程序和设备才能利用新的安全功能, 目标碎片和设备碎片链接。Thomas, Beresford 和Rice [2]研究设备碎片使用志愿者安装他们的设备监视器应用程序。他们发现88% 的设备容易受到至少一个选定的漏洞和该设备不经常更新 (平均每年1.26 次)。 Zhou et al. [32] 在2423工厂中超过1000家图像可以通过错误设备驱动程序。Xing et al.. [33] 识别应用程序如何利用操作系统更新过程以获取敏感的系统权限。Mulliner et al. [34]提供了一种可伸缩的应用方法通过第三方补丁到易受攻击的 Android 设备。

我们还研究了 app 更新过程。Moller et al [35] 调查 Android 用户的更新模式发现只有一半的用户在应用程序更新发布更新的一周内安装了。McIlroy Ali和Hassan [36]从10713个应用的更新数据里发现只有1% 的应用程序每周至少更新一次。

其他一些研究分析了 Android 应用的大数据采集。 Garcia和Nieh [10] 抓取、下载和分析110万应用程序获取有关权限和库的统计信息分布以及识别安全嵌入的应用程序凭据.其他研究 [37, 38] 也研究允许和库使用。Kavaler et al. [39]比较了使用了 Android在

StackOverflow 的相关课程和问题。

VIII. CONCLUSION

Android apps specify a target API level and run in a

compatibility mode on devices with higher API levels. The

compatibility mode can disable important security changes in

the Android platform. We call the problem of apps targeting

outdated API levels the target fragmentation problem. In this

study we analyze a dataset of more than one million Android

apps collected over four years and show that the large majority

of collected apps target outdated API levels. We examine

the practical implications of target fragmentation on seven

security changes to the Android platform and show that target

fragmentation hamstrings new security features.

We believe that applying security changes in this optional

manner is a flawed approach that sacrifices security at the

altar of compatibility. Developers become a new obstacle

to securing apps and users have no means of ensuring that

their apps target the most current API levels. The target

fragmentation problem is further compounded by the coupling

of security changes and non-security changes. We hope that by

shedding light on this problem, developers can become more

aware of the consequences of targeting outdated API levels

and this flawed design can be reexamined and changed so that

there is less opportunity for Android apps to operate without

access to important security features.

VIII. 结论

Android 应用程序指定一个目标 API 级别, 并在具有较高 API 级别的设备上的兼容模式运行。兼容模式可以禁用Android 平台重要的安全更改。我们叫这个问题为针对过时API级别的应用程序目标碎片化问题。我们花了四年里研究收集分析了超过 100万 Android 应用，显示大部分收集的应用的目标是过时的 API 级别。我们研究目标碎片化对Android 平台七个安全变化的实际影响 , 并显示目标碎片化对新的安全功能的削弱。我们认为, 应用可选的安全更改方式是一个有缺陷的方法, 牺牲安全性来确保兼容性。开发商为了保证应用程序和用户的安全成为新的障碍，没有办法确保他们的应用程序的目标是最新的 API 级别。目标碎片化问题进一步复杂化加剧安全更改和非安全更改。我们希望通过在这个问题上, 开发者可以变得更意识到以过时的 API 级别为目标的后果，对这种有缺陷设计可以重新审视和改变, 以便Android应用运行时不会产生重要的安全功能。