Android应用中的目标碎片

2016 IEEE安全和隐私研讨会

2016 IEEE安全和隐私研讨会

Patrick Mutchler斯坦福大学[pcm2d@stanford.edu](mailto:pcm2d@stanford.edu)

Yeganeh Safaei亚利桑那州立大学

[ysaf](mailto:ysafaeis@asu.edu)[aeis@asu.edu](mailto:aeis@asu.edu)

亚当杜佩亚利桑那州立大学

[doupe@asu.edu](mailto:doupe@asu.edu)

约翰米切尔斯坦福大学

[mitchell@cs.stanford.edu](mailto:mitchell@cs.stanford.edu)

***摘*要 - Android应用程序声明Android运行时平台的目标版本。 当在具有更新版本的Android设备上运行时，应用程序会以兼容模式执行，尝试模仿旧版目标版本的行为。 这种设计具有严重的安全后果。 针对过时的Android版本的应用会禁用对Android平台的重要安全更改。 我们将针对过时Android版本的应用程序称为目标碎片问题。**

**我们分析了2012年5月至2015年12月期间收集的1,232,696个免费Android应用程序的数据集，并显示目标碎片问题是整个应用程序生态系统中严重关注的问题，并且在几年内没有发生大的变化。 总共有93％的当前应用程序针对过时的平台版本，并且平均过时了686天; 79％的应用在上传到应用商店的那一天已经过时。 最后，我们检查了针对Android平台的七个与安全相关的更改，这些更改在针对过时平台版本的应用中被禁用，并显示目标碎片hamstrings尝试提高Android应用的安全性。**

1. 介绍

Android已经成为全球最受欢迎的智能手机平台，拥有超过10亿个有源设备[1]。 Android向用户提供安全代码，设备内碎片和应用程序内碎片两大安全挑战。 设备碎片是众所周知的问题[2]。 Google不会控制Android设备或Android软件更新的分发。 谷歌依靠其他企业网络以新设备或软件更新的形式向用户提供最新的Android软件。 由于此过程的分布式特性，大量Android设备正在运行Android平台的过时版本。 关键安全补丁不会覆盖数百万Android用户，从而延长安全漏洞的寿命。 与设备碎片相比，应用内的碎片尽管带来类似的安全后果，但几乎没有受到关注。

Android通过Android API向应用程序展示了许多重要的库功能。 Android平台的新版本可以改变现有库功能的行为。 例如，Android 4.4将AlarmManager.set的行为更改为为相似时间设置的批次警报。 这种改变有助于提高电池寿命，但会导致不精确的报警。 诸如此类的行为改变对应用程序提出了问题，当设备更新为新的Android版本时，这可能突然中断。

Google为每个Android平台版本分配一个称为API级别的整数。 保持一定程度的向前兼容性，并防止应用程序发生显着变化

行为没有警告，每个应用程序都有一个目标API级别。 在API级别高于其目标API级别的设备上运行的应用会以兼容模式执行，以尽可能接近地匹配目标API级别的设备行为。 例如，即使在最新的Android设备上运行，将目标API级别设置为18（Android 4.3）的应用也会使用无法访问的警报行为。 Android平台的变化可能包括重要的新安全功能，可以解决Android API的已知问题或提供额外的保护以抵御攻击。 针对过时的Android级别的应用将无法使用兼容模式禁用的这些功能。 这种设计意味着，即使运行最新的设备，针对过时的API级别的应用也无法访问最新的安全功能。 Google不会在Google Play商店中发布应用的目标API级别，因此用户无法知道应用是否面向过时的API级别。 相反，用户完全处于应用程序开发人员的摆布之中。

我们称这个应用程序针对过时的API级别问题

*目标碎片问题。*

目标碎片问题最为人熟知的后果是Android WebView [3]中的远程代码执行漏洞。 API级别17添加了新行为来解决此漏洞，但是此更改仅适用于定位API级别17或更高级别的应用程序。 漏洞在Android平台上被披露和固定多年后，即使在新平台上运行，针对过时的API级别，应用程序仍然可能存在漏洞。 一些研究这个漏洞的研究已经报告了16个或更低的应用程序的比例[4,5]。 但是，这个漏洞只是目标碎片问题如何导致Android应用程序安全性下降的一个例子，并且没有研究自行研究目标碎片。 对目标碎片问题的更完整分析对于理解Android生态系统的安全性至关重要。

据我们所知，本文报告了第一项研究，以大范围地确定和衡量目标碎片问题及其安全影响。 在本文中，我们研究了近四年来从Google Play商店收集的免费Android应用程序的数据集中的目标碎片。 这些数据集总共包含1,232,696个应用程序。 我们使用从Google Play商店获取的元数据来衡量目标分段问题的趋势。 最后，我们研究目标碎片问题对几个具体漏洞的安全影响。 我们研究的主要研究问题和结果如下：

204



2016 IEEE安全和隐私研讨会



©2016，Patrick Mutchler。 根据IEEE许可。 DOI 10.1109 / SPW.2016.31

# 

**A**ndroid应用程序生态系统中目标碎片问题的状态是什么？ 我们检查了2015年12月从Google Play商店收集的60,086个免费应用的数据集，发现93％的应用会针对过时的API级别。 我们定义了一种“过时”的措施，并且平均而言，这些应用程序的目标API级别是过时686天。

**开**发人员是否选择定位过时的Android版本，还是因开发者放弃其应用程序而导致碎片化？ 为了解决未维护的应用程序，我们定义了一种“疏忽过时”措施，即从应用程序上传到应用程序商店的日期开始衡量过期问题，并显示目标碎片不仅仅是停滞应用程序造成的。 我们发现，2015年12月收集的应用程序的平均过失时间为536天。

**目**标碎片是最受欢迎的应用程序中的问题吗？ 我们研究了应用程序受欢迎程度的目标碎片问题，并得出结论认为即使在最受欢迎的应用程序中，目标碎片化也是一个严重问题。 我们发现2015年12月收集的应用程序中有88％的应用程序安装了超过100万次，目标是过时的API级别。 这些应用的平均过期时间为607天，平均疏忽过期时间为493天，仅略低于一般人群。

**随**着时间的推移，目标碎片问题会变得不那么严重吗？ 我们将2015年12月数据集的目标碎片结果与2012年5月至2014年7月期间从Google Play商店收集的其他四个数据集进行比较。这些数据集合并包含1,232,696个应用。 我们发现，除了极端超时应用程序的增长尾部之外，最近收集的四个数据集之间的过期分布非常相似，这表明目标碎片问题在几年内的严重程度没有发生显着变化。

**目**标碎片问题的具体安全含义是什么？ 我们通过检查Android平台的七个与安全相关的变化来扩展对目标碎片问题的讨论，并首次定量分析目标碎片对Android生态系统安全性的广泛影响。

1. 背景

与每个Android应用程序打包是一个清单文件。 清单文件是一个XML文档，其中包含有关应用程序的信息，例如应用程序组件列表，请求的权限以及应用程序响应的系统事件[6]。 清单包含与本研究相关的两个属性：最小API级别（minSdkVersion）和目标API级别（targetSdkVersion）\。 最低API级别的含义非常简单。 应用程序无法安装在Android级别低于最低API级别的设备上。 这种设计可确保应用程序不会安装在缺少必要功能的设备上。

\清单还可以包含最高的API级别，但自Android 2.0.1以来，除了在Google Play商店上过滤搜索之外，最大API级别不再用于其他任何用途。

|  |  |  |  |
| --- | --- | --- | --- |
| **API级别** | **版本代码** | **代码名称** | **发布日期** |
| 14 | 4.0–4.0.2 | 冰淇淋三明治  果冻豆  奇巧棒棒糖棉花糖 | 2011年10月 |
| 15 | 4.0.3–4.0.4 | 2011年12月 |
| 16 | 4.1–4.1.2 | 2012年7月 |
| 17 | 4.2–4.2.2 | 2012年11月 |
| 18 | 4.3–4.3.1 | 2013年7月 |
| 19 | 4.4–4.4.4 | 2013年11月 |
| 20 | 4.4W-4.4W.2 | 2014年6月 |
| 21 | 5.0–5.0.2 | 2014年11月 |
| 22 | 5.1–5.1.1 | 2015年3月 |
| 23 | 6.0–6.0.1 | 2015年10月 |

表I：缩写的Android版本历史记录。

应用程序的目标API级别用于维护与新Android平台的转发兼容性。 如果设备的API级别高于应用的目标API级别，则设备将启用兼容性功能以尽可能匹配目标API级别的行为。 在每个API级别启用的一组兼容性功能可以在Android文档中找到[7]。 请注意，可以将应用安全地安装在API级别低于目标API级别的设备上。 开发人员可以定位最新的API级别，而不会使其应用与旧版Android设备不兼容。

如果应用没有声明目标API级别，或者目标API级别低于最低API级别，则目标API级别将设置为最低API级别。 在本文的其余部分，我们区分原始目标API级别和目标API级别，后者是目录级别，后者是目标级别，后者是使用此规则计算并实际由Android操作系统使用的目标级别。 目前大约8％的应用程序未声明有效的原始目标API级别。 targetSdkVersion和minSdkVersion属性取对应于不同Android版本代码的称为“API级别”的整数值。 在讨论不同的API版本时，对于本文的其余部分，我们使用API​​级别值（例如17,18,19）而不是版本代码（例如4.2,4.3,4.4）。 最近的API级别和Android版本代码之间的对应关系如表1所示。

*A.安全问题*

从Android文档中可以看出，针对过时的API级别可能会产生安全隐患。 在targetSdkVersion属性[8]的文档页面上，Google建议开发者应该“增加[targetSdkVersion]属性的值以匹配最新的API级别”，但是没有提及针对过时API级别的安全后果。 在“安全提示”页面[9]中，没有提及目标API级别。 有人可能会认为针对过时的API级别的安全后果是最小的或不存在的。

但是，最近的Android版本中存在重要的安全更改，这些更改不适用于以过时的API级别为目标的应用程序。 例如，API级别17和19都包含防止广泛使用的功能中的代码注入漏洞的更改。 其他几个API级别改变了常用功能以具有更安全的默认行为，提供了额外的保护层。 表II列出了可通过定位禁用的Android平台的主要安全更改

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集** | **App Count** | **抓取日期** | **最新的API** |
| A  B C D E | 60,086  219,115  165,489  645,862  142,144 | 2015年12月  2014年6月  2014年1月  2013年7月  2012年5月 | 23  19  19  17  15 |

表II：与Android平台选择的与安全性相关的更改。 针对低于列出级别的API级别的应用程序不具备由这些更改提供的任何安全保护。

|  |  |
| --- | --- |
| **API** | **平台变化** |
| 16 | 从JavaScript访问文件URL默认是禁用的 |
| 17 | 内容提供商在默认情况下不再暴露于外国应用程序 |
| 17 | 无法通过JavaScript代码调用未注释的应用程序方法 |
| 19 | isValidFragment被添加以防止碎片劫持 |
| 19 | JavaScript URL在单独的WebView上下文中执行 |
| 21 | Context.bindService不再接受隐式意图 |
| 21 | WebView默认阻止混合内容 |

过时的Android级别。 这些变化的细节以及它们关闭的漏洞在第五节中讨论。如果大量应用程序针对的是过时的API级别，那么无论这些更改的意图如何，这些变化都会变得无效，并且应用程序被置于不必要的风险之中。

1. 方法

我们的研究分析了2012年5月至2016年1月期间从Google Play应用商店收集的1,232,696个免费应用的数据集。为了收集这些应用，我们开发了一个系统来抓取Google Play商店，以识别新应用，从Google Play中刮取元数据存储和下载实际的应用程序。 该系统在简短的时间窗口内运行，自然将我们的数据集分成与这些时间窗口相对应的较小的数据集（按反向时间顺序的数据集A，B，C，D和E）。 表III描述了这些数据集的详细信息，并列出了收集每个数据集时最新的API级别。

1. *收集应用程序*

我们的系统首先抓取Google Play商店的应用下载。 我们认为一个应用程序是唯一的，如果它有一个独特的应用程序ID\。 要抓取Google Play商店，我们使用以下四种技术：（1）为热门应用和收藏抓取Google Play指定类别，（2）抓取随机已知开发者页面以查找新应用，（3）在Google上搜索使用已知应用程序描述中的单词播放商店，以及（4）从已爬网页上的URL中提取所有应用程序ID。

Google会在商店中发布一些关于应用的元数据。 我们搜集并收集有关每个已爬网应用程序的元数据，包括该应用程序最新版本上传到应用程序商店的日期以及安装应用程序的设备数量\。

要下载应用程序，我们使用类似于Viennot，Garcia和Nieh [10]描述的方法。 为了尽可能高效地收集各种各样的应用程序，我们只下载了前所未有的应用程序ID的应用程序。 对于除数据集A以外的每个数据集，我们都尝试使用之前从未见过的应用程序标识下载每个应用程序

\该应用的Google Play网址的id查询参数的值，例如com.instagram.android中的play.google.com/store/apps/ details？id = com.instagram.android。 请注意，这不一定与应用程序的软件包名称相同

\精确的安装计数不可用。 Google Play会报告一系列可能的安装计数，每个数量级有两个存储桶。 例如，某个应用的报告安装次数可能在10,000到50,000之间。

表III：本研究中使用的ve数据集的概述以及收集每个数据集时的最新API等级。

爬行。 由于时间限制（和技术挑战），数据集A只是可用应用程序的一个子集。 我们在第VI-B节讨论这种收集方法的效果。

1. *分析*

Google Play商店发布每个应用的最低API级别，但不发布目标API级别。 我们使用apktool [11]，这是一个静态分析工具，可将打包的应用程序转换为可读取的文件，提取清单并记录其目标API级别。 由于我们的应用程序数据库非常大，因此执行复杂的静态分析是不切实际的。 本研究中使用的所有静态分析纯粹是句法，我们通过处理由apktool提取的app字节码的小数表示来执行。

1. 评估

在本节中，我们量化目标碎片问题的程度。 我们首先证明大多数采样应用都以过时的API级别为目标。 我们定义了一个过时度量标准，用于度量单个应用程序以及跨应用程序群体的目标碎片问题的严重程度。 我们发现目标碎片问题主要是由开发人员的疏忽导致的，而不是应用程序在应用程序商店中的闲置状态。 我们比较了流行和不受欢迎的应用程序之间的过时度量标准，并证明即使在最流行的应用程序中，目标碎片也是一个问题。 最后，我们显示在最近收集的四个数据集中，过期曲线类似。 这一结果表明，除非目标碎片化问题得到重新审查，否则将来可能会继续保持同样的规模。 由于我们的数据集规模很大，我们相信这些结果广泛适用于整个Google Play生态系统。

1. *目标碎片化*

图1显示了数据集A中包含最新应用程序的数据集中应用程序的目标API级别分布。 很明显，绝大多数应用程序并不针对API数据集A收集时的最新API级别API 23。 更准确地说，我们发现数据集A中93％的应用目标API等级为22或更低。

面向更多过时的API级别的应用程序更可能会错过关键的安全更改。 我们不仅对应用程序过时而且对应用程序过时有兴趣。 我们定义了一个称为“过期”的定量度量，因为应用程序的目标API级别的发布日期与应用程序收集时的最新API级别的发布日期之间的差异（天数）。 我们的中位数过期时间为704天，平均过期时间为686天

# 

图1：API级别23发布后两个月收集的数据集A中目标API级别的分布.API级别20特别针对可穿戴设备，解释了为什么少数应用会针对它。

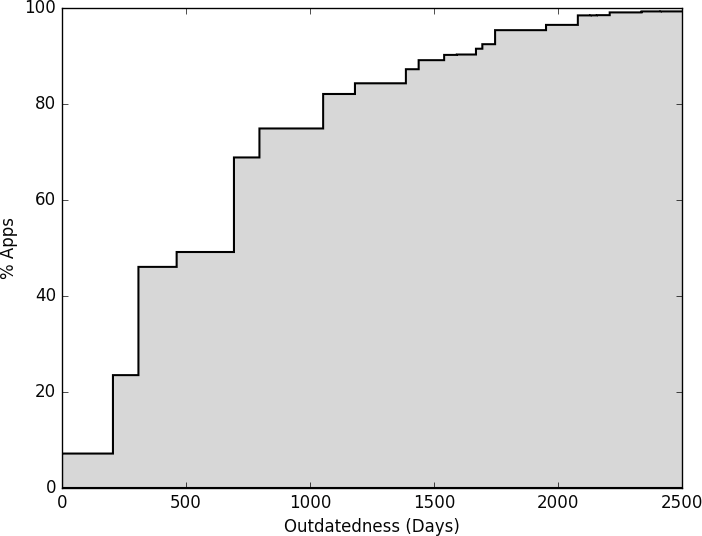


图2：数据集A中过期的累积分布。（X，Y）处的点表示Y％的应用程序具有X天或更少的过期。 过时值的低分辨率会导致此曲线的锯齿状特性。

对于数据集A中的应用程序。我们可以使用过期累积分布（图2）来衡量应用程序总体上目标碎片问题的严重程度。 通过检查曲线的低端，我们发现绝大多数应用程序都针对过时的API级别。 曲线顶端的长尾显示我们有相当多的应用程序针对已过时多年的API级别。

我们可能预计目标API级别的分布类似于偏斜的正态分布，随着API级别越来越过时，针对每个API级别的应用的百分比降低。 但是，如图1所示，我们发现针对API级别7到16的应用程序的百分比相对较低。 几乎同样多的应用将API级别设置为API级别16级别.API级别9特别突出，即使它没有提供任何关键兼容性功能，它仍将被数据集A中所有应用程序的近3％作为目标。 为什么应用程序会针对这种过时的API级别？

图3：应用程序上传到Google Play商店和收集应用程序之间的天数分布。

如果我们分析数据集A中的原始目标API级别，则发现8.2％的应用程序不包括原始目标API级别，请设置低于其最小API级别的值，或者具有原本无效的原始目标API级别。 这些应用程序使用其最低API级别作为其目标API级别。 这些应用程序占目标API级别分布的大部分长尾。 例如，目标API级别为15或更低的应用程序中有63％是这样做的，因为它们将其最低API级别用作其目标API级别。 这代表了通过说服开发人员正确使用targetSdkVersion属性来消除目标API级别分布中长尾的重要机会。

1. *陈旧的应用程序*

并非所有Google Play商店中的应用都由其开发者定期维护。 图3显示了应用程序收集和开发人员将其上传到Google Play商店之间的天数分布情况。 请注意，此日期可能是该应用第一次发布或最近推送的应用更新日期。 数据集A中只有38％的应用程序在2015年10月发布API级别23后上传。显然，在Google Play商店停滞不前的应用程序不会针对发布的新API级别。 未维护的应用程序的存在会导致目标API分布偏差，并且错误地暗示主动维护其应用程序的开发人员无法更新其应用程序以定位新的API级别。 在这里，我们试图区分由未维护的应用程序引起的过期和通过应用程序维护持续存在的过时。

我们将应用程序的“疏忽过期”定义为其目标API级别的发布日期与上载到Google Play商店时的最新API级别的发布日期之间的差异（以天计）。 疏忽过时的措施错失了开发者将其应用程序定位到当前API级别的机会。 图4描述了一个具体的例子来澄清过时性和疏忽过时性之间的区别。 我们有一个中位疏忽

过时的过时



Outdatedness



API 21

发布

API 22

发布

App Uploaded

API 23

发布

应用程序收集

图4：展示过时和疏忽过时的示例应用程序。 此应用程序的目标是API级别21，在API级别22发布后上载，并在API级别23发布后收集。过期是在应用程序收集之前从最高API级别开始测量的。 过期过期是指在应用上传之前从最高API级别开始衡量。

数据集A的应用程序过期377天，平均疏忽过期536天。

我们可能会更加慷慨，因为我们对疏忽过时的定义包括一些滞后时间，以允许开发人员在新的API级别发布后重新定位他们的应用程序。 在应用上传到应用商店时，我们选择最新的API级别，而不是在应用上传到应用商店时至少提供N天。 我们称这个滞后时间为“采纳窗口”。添加30天的采纳窗口仅略微影响疏忽过时，将中值疏忽过期减少到327天，平均过失过期减少到496天。

图5显示了使用30天采纳窗口的疏忽过期的累积分布。 即使采用慷慨的采用窗口，应用程序仍然无法定位适当的API级别。 79％的应用程序过于注重过时的API级别，这意味着这些应用程序一旦上传到应用程序商店就会过时。 开发人员在发布之前可以访问新的Android平台，因此应用程序可以在新的API级别的第零天更新。 从这些结果可以清楚地看出，目标碎片问题不能用陈旧的应用程序来解释，而是由于开发人员疏忽导致的结果，或者是由于无视目标过时API级别的后果或由于故意选择目标过时的API级别。

1. *流行的应用*

Google Play商店中的绝大多数应用都不是由很多用户下载的。 数据集A中只有2％的应用程序安装了至少1,000,000次，但这些应用程序占数据集A中应用程序总安装量的74％。了解应用程序受欢迎程度与目标碎片问题之间的关系很重要。

图6比较了不同流行的应用程序之间的累积过期分布。 我们看到过时曲线非常相似，最受欢迎的应用程序（至少安装了一百万次）的目标只是略微过时的API级别。 比较不同流行的应用程序之间的疏忽大意分布给出了类似的结果（并且由于空间原因，我们不包括它）。 已安装至少一百万的应用程序

图5：使用30天的收养窗口，数据集A中疏忽过期的累积分布。

次数平均过期607天，平均疏忽过期493天。

1. *目标碎片随着时间的推移*

从数据集A的分析中可以清楚地看到，目标碎片化是当今严重的问题。 即使是最受欢迎的应用程序也是针对过时的，并且通常非常过时的API级别。 通过分析在不同日期收集的数据集，我们可以看到问题随着时间的推移如何变化。 我们重复之前对其余四个数据集的分析，并将这些结果与数据集A的结果进行比较。

图7比较了我们的数据集的过时分布。 随着时间的推移，只有一个明显的趋势：针对极度过时的API级别的应用程序越来越多，数据集E和A之间的过时率的第90百分位增加了一倍以上。这是目标碎片问题的自然属性，因为最大过期随时间增长。 只要有针对最低API级别的应用程序，我们希望看到这个尾部继续增长。 比较过时曲线的低端并不显示明显的模式。 我们看到，在每个数据集中，绝大多数应用程序的目标都是过时的API级别，并且除了数据集E之外，在过时性曲线的低端似乎没有显着差异。 我们注意到数据集A和C分别在平台发布两个月后收集，数据集B和D分别在平台发布后七个月和九个月收集。 这种差异似乎对过时曲线的低端影响比任何模式都要大，其中超过20％的数据集B和D中的应用程序以当前API级别为目标，而低于10％的应用程序

数据集A和C以当前API级别为目标。

我们的数据集之间有一个非常有希望的趋势。 未指定原始目标API级别的应用百分比明显下降（图8），因此将其目标API级别设置为其最低API级别。 由于开发人员通常希望支持尽可能多的设备，因此应用程序通常具有非常低的最低API级别

# 

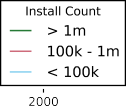


图6：不同应用程序流行度的累积过期分布比较，显示过时统计数据对应用程序流行度变化的强大。 拥有1,000,000或更多安装量的应用程序具有过时曲线，仅比其他人群稍好一些。 数据点是线性内插的，以提高可读性。



图7：所有数据集的过期累积分布。 最近收集的四个数据集（A到D）遵循相似的曲线，但尾部增长。 数据点是线性插值的，以提高可读性。

未能指定目标API级别极其危险。 这种趋势表明，开发人员已经越来越意识到目标API功能，并且针对其最低API级别的开发人员数量将随着时间消失。

1. 安全含义

目标碎片问题意味着只要在兼容模式下禁用更改，对Android平台的任何安全更改效果就会降低。 在本节中，我们将探讨目标碎片问题对Android平台的七个安全更改的实际后果。 针对过时的API级别的应用程序可能不一定容易受到攻击，但是存在安全漏洞的风险增加。 除了一种情况，我们只能表明应用程序不必要地处于高风险状态。 因为Google认为过时的行为不够安全

图8：每个数据集中应用程序的百分比，其清单中不包含有效的targetSdkVersion属性，因此将其目标API版本设置为等于其最小API版本。

值得改变Android平台，过时行为的广泛使用令人不安。 我们还引用研究表明，这些应用程序可以被利用的必要条件是频繁的。 本节中的统计数据是在数据集A（最近收集的数据集）上计算的。

1. *WebView默认值*

WebView [12]是一个UI元素，充当Android应用程序中的嵌入式Web浏览器。 在数据集A中，91％的应用程序至少包含一个WebView实例\。 对于WebView的默认行为，通过针对过时的API级别禁用了三项主要安全更改。 以下各节将介绍这些更改并显示定位当前API级别的应用不太可能使用不安全行为。

* 1. *文*件方案同源策略：根据同源策略，JavaScript代码只能访问从同一来源加载的内容。 将所有文件：URL作为属于同一来源的URL在互相不信任的系统中存在安全风险。 如果WebView使用file：URL加载任何不受信任的内容，则该内容可以完全访问应用程序可以访问的其他所有内容。 在API级别16中，WebView的默认行为已更改为将所有文件：URL都视为属于不同的来源，但此更改仅适用于定位API级别16或更高的应用程序。 希望覆盖此行为并将所有文件：URL视为属于同一来源的应用程序可以使用setAllowFileAccessFromFileURLs和setAllowUniversalAccessFromFileURLs方法。

我们发现，目标API级别15或更低的应用程序中有82％，以及目标API级别16或更高的应用程序的18％使用file：URL的不安全策略。 因为定位API级别为16或更高且想要使用不安全策略的应用程序必须明确这样做，所以我们可以使用这些应用程序的百分比，这些应用程序包括对setAllowFileAccessFromFileURLs

\由于我们无法确保广告库不包含漏洞，因此这些广告库包含WebViews以及主要的应用程序功能。

或setAllowUniversalAccessFromFileURLs作为想要使用不安全策略的应用程序的上限。 如果我们假设这两个应用的百分比是统一的，那么我们得出结论：目标API级别为15或更低的应用中有64％不必要地使用了不安全的策略，如果将它们重定向到API级别16，它们将变得更加安全。这些应用占9.6％的数据集A中的所有应用程序。我们在第VI-A节中讨论这个假设的有效性。

允许不安全访问的应用程序风险较高。 要被利用，应用程序必须使用文件：URL加载不受信任的内容。 应用程序可能被利用的两种方式是加载文件：从其他应用程序接收的URL或通过导航到不可信的网页，这可能会丢失磁盘并将WebView重定向到该文件。 参见Chin和Wagner [13]详细描述这个漏洞。 研究表明，23％的Web浏览器应用程序不安全地加载文件：来自外部应用程序的URL [14]，WebView应用程序经常加载不受信任的网页[13,15,4]。

* 1. *J*avaScript网址：应用可以通过调用loadUrl来加载WebView中的网页内容。 在API等级为18或更低的应用程序中，在JavaScript伪URL\上调用loadUrl将在当前呈现的网页中执行脚本。 这种行为在加载从外国应用程序检索到的未经过滤的URL的应用程序中可被利用。 恶意应用程序可以发送请求以在foo.com加载URL，然后发送要在foo.com上下文中执行的脚本以读取私人内容。 这种攻击被称为交叉应用脚本[16]。

目标API级别为19或更高的应用程序会在空的上下文中加载JavaScript Psuedo-URL，而必须使用方法evaluateJavascript在当前WebView上下文中执行JavaScript代码。 我们发现，API等级为19或更高的60％应用程序至少包含一次对此方法的调用。 再次，我们可以将此用作以这种方式故意执行JavaScript代码的应用数量的上限。 因为90％的API API级别为18或更低的应用程序包含WebView并启用JavaScript，所以我们得出结论：30％的API API级别为18或更低的应用程序无需以此方式评估JavaScript代码，并且如果它们更安全被重定向到API级别19.这些应用程序占数据集A中所有应用程序的9.4％。

* 1. *混*合内容：包含通过HTTP检索的网页元素的网页在通过HTTPS加载时称为包含混合内容。 加载混合内容是一种安全风险，几个主流浏览器阻止混合内容[17,18]。 默认情况下，WebView在API级别为21或更高的应用中阻止混合内容。 希望覆盖此行为并允许混合内容的应用程序可以使用setMixedContentMode方法指定自定义策略。

我们发现，76％的目标API等级为20或更低的应用和42％的目标API等级为21或更高的应用允许混合内容。 如果我们假设希望允许混合内容的应用百分比在两个应用群体中是统一的，那么我们可以得出结论，目标应用的34％

\以协议JavaScript开头并包含要执行的JavaScript代码的URL。

API级别20或更低不必要地允许混合内容，并且如果将其重定向到API级别，则将变得更安全

21.这些应用程序占数据集A中所有应用程序的18％。

1. *JavaScript接口远程执行代码*

Android允许应用程序通过使用称为JavaScript Interface [19]的功能将应用程序级对象暴露给在WebView中运行的JavaScript代码。 2012年，发布了JavaScript界面​​的远程代码执行攻击[20]。 由于JavaScript代码可以访问添加到JavaScript Interface的所有对象的公共方法，所以恶意脚本可以通过调用暴露对象上的getClass（一种从java.lang.Object继承的方法）来访问Java Reection API。 从那里，恶意脚本可以构建任意的Java对象并执行任意代码。

Android通过强制应用程序注释应该从JavaScript代码调用的方法来解决API级别17中的此漏洞。 从JavaScript代码调用未注释的方法什么也不做。 因为开发人员不太可能需要将getClass公开为JavaScript代码，所以这限制了恶意脚本可能造成的损害。 但是，此更改不适用于定位API级别为16或更低的应用程序。 可以利用使用JavaScript界面​​，目标API级别为16或更低的应用程序以及加载不受信任的Web内容的应用程序。 我们通过寻找调用来以smali代码添加JavaScript接口来识别使用JavaScript接口的应用程序。 数据集A中50％的应用程序使用JavaScript界面​​。 在这些应用程序中，15％的目标API级别为16或更低。 这些在WebView中加载不可信JavaScript代码的应用程序都可以被利用。 识别可加载不受信任的JavaScript代码的应用程序超出了本研究的范围，因此我们无法说明这些应用程序的哪些部分是可利用的，但我们注意到现有的工作表明，应用程序加载并不罕见

不受信任的Web内容[13,15,4]。

1. *导出的内容提供者*

管理对结构化数据的访问的应用程序组件称为内容提供者[21]。 内容提供者在应用程序的展示文件中声明，并且可以将它们设置为本地应用程序，也可以将其暴露给具有导出属性的其他应用程序。 无意中导出的持有敏感数据的内容提供商是安全威胁，因为设备上的每个应用都可以访问它们。 但是，如果未指定导出的属性，则会回退到默认值。 API级别17将默认值更改为false，但指向API级别16或更低级别的应用使用默认值true。 研究表明，65％的导出内容提供商的应用程序泄露私人数据[22]。 9.7％的应用程序目标API级别为16或更低，8.0％的应用程序目标API级别为17或更高级别包括至少一个导出的内容提供程序。 4.9％的目标API级别为16或更低的应用包含由于默认行为而导出的内容提供商。 假设想要导出内容提供商的应用程序的百分比在两个应用程序人口中是统一的，我们得出结论：1.7％的API应用程序的目标级别为16或更低，不必要地导出内容提供程序。

这些应用程序占数据集A中应用程序的0.3％。

1. *片段注射*

2013年，安全研究人员在PreferenceActivity类中发现了一个漏洞[23]。 恶意应用可以将制作的消息发送到继承自PreferenceActivity的导出类。 这些消息被解释为Fragment实例，并使用Reflection API动态加载，从恶意应用程序执行任意代码。 API级别19将方法isValidFragment添加到PreferenceActivity以关闭此漏洞。 预计开发人员将使用此方法检查注入片段的包名称并拒绝未经授权的片段。 定位API级别19或更高的应用会继承isValidFragment的实现，该实现始终引发异常，因此默认情况下是安全的，但是定位API级别的应用

18或更低继承始终返回的实现

真实的，它不能防范这种攻击。

我们发现，1.7％的API API级别为18或更低的应用程序至少包含一个继承自PreferenceActivity的导出类，并且不会覆盖isValidFragment\的不安全实现。 与以前的例子不同，这是足够的信息来证明这些应用程序是可利用的，而不仅仅是高风险。 可利用的应用程序占数据集A中所有应用程序的0.5％。

1. *服务劫持*

服务是一个无需用户交互即可执行操作的应用程序组件。 UI组件可以使用bindService方法与服务交互。 应用程序通过使用Intent指定要与哪些服务进行交互[24]。 意图可以是显式的也可以是隐式的。 显式意图使用其类名列出唯一的服务。 隐式意图仅指定要执行的一般操作，系统选择适当的服务来处理请求。 使用隐式意图与服务进行通信并不安全。 由于服务不是面向用户的组件，因此用户无法控制哪个服务响应隐式意图。 如果多个服务匹配bindService中使用的隐式意图，则选择这些服务中的一个随机服务。 恶意应用程序可以创建一个匹配Implicit Intent并模拟可信代码的服务。 在API级别为21或更高的应用程序中，将隐式意图传递给bindService会引发安全异常。

目标API等级20或更低的应用中有83％包含at

至少一次调用bindService。 这些应用程序占数据集A中所有应用程序的43％。通过静态识别这些应用程序中哪些应用程序使用隐式意图绑定服务是非常平凡的，超出了本研究的范围。 然而，之前的研究[25]发现，19％的应用程序可能容易受到服务劫持的攻击，因为它们使用隐式意图。

1. *检测过期的应用程序*

Google Play不会发布应用的目标API级别。 具有安全意识的用户无权根据目标API级别做出决策，而必须信任开发人员

\我们假设isValidFragment的所有实现都正确地验证了注入片段的包名称。 未来的工作可能会通过查找isValidFragment的错误实现来扩展我们的结果。

重新定位他们的应用程序。 让用户访问这些信息是非常重要的。 为此，我们创建了一个开源工具[26]，用于报告用户手机上每个应用程序的目标API级别，并向用户提供任何安全隐患。

1. 讨论

本研究中提供的数据表明两个结论很明显：目标分散存在于整个Android生态系统中，而目标碎片对Android安全性有实际影响。 使新的Android版本与未更新的应用程序兼容可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限制。 此方法使安全更改成为可选项，并将安全更改与非安全更改混合在一起。 开发人员不能挑选和选择安全更改，而必须从新的API级别集成所有平台更改。 开发人员可以通过设置最大的API级别来避开这个问题，但是这个特性并没有被强制执行，并且会通过阻止平台更新来加剧设备碎片问题。

考虑到这一点，我们考虑替代当前系统：不管目标API级别如何，对Android平台实施所有安全更改。 API级别23发布后将近四个月，不到1％的活动设备已更新至版本23 [27]。 这种缓慢的更新过程意味着开发人员有足够的时间更新他们的应用程序以处理新的行为。 过去也出现过安全方面的变化，这两方面都是强制性的，违规行为似乎并未引起巨大的痛苦。 在Android级别2​​1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，可能会阻止应用在设备更新后重新安装，但在开发人员论坛上搜索几乎没有发现任何投诉。 我们怀疑，如果开发人员只是被迫适应所有安全性改变，那么对于大多数应用程序来说这不会造成问题。 如果这是不可行的，那么至少应该告知用户他们的应用程序是否瞄准了过时的API级别，以便他们有权做出有安全意识的决定。

第VB节中描述的JavaScript接口漏洞是一个很好的案例研究来支持我们的论点。 通过可选更改未能在API级别17中解决问题后，API级别19禁止从JavaScript界面​​访问所有getClass。 对于针对较低API级别的应用程序，此更改不可选，如果不是针对API级别为16或更低级别的应用程序，则不需要此更改。 无论其目标API版本如何，只有适用于所有应用程序的更改才能真正解决该漏洞。

1. *替代解释*

无知并不是开发人员可能无法定位最新API级别的唯一原因。 另一种选择是开发人员不会将他们的应用程序重定向到最新的API级别，因为很少有设备运行最新的API级别。 如果是这种情况，我们希望大多数应用不会超过一个API级别过时。 虽然16％的设备是

在收集数据集A时运行API级别22或23，但数据集A中只有23％的应用程序针对这些级别。

另一种可能性是开发人员选择不将应用重定向到更高的API级别，除非存在安全问题。 由于API级别22和23不包含安全更改，因此开发人员可能选择不重新部署应用程序。 然而，我们在第五部分展示了即使许多应用程序错过了相关的安全更改（54％的应用程序将API级别设置为21以下），仍然有许多应用程序针对过时的API级别。 如果我们看一下片段注入漏洞，我们会发现使用PreferenceActivity的应用程序不太可能将API等级设置为19或更高（66％），而不是其他应用程序人口（69％）。 这表明开发人员在决定确定目标API级别时并不考虑安全性。

最后一种选择是开发人员选择不定位最新的API级别，以避免破坏关键应用功能。 这是一个真正的可能性，但如果这是真的，则表明Google的“全部或全部”设计是令人敬畏的，因为它迫使开发人员做出不可能的选择和圣餐安全。

1. *威胁有效性*

我们只下载了数据集A的可用应用程序的子集，因此在我们的结果中存在选择偏差的一些可能性。 但是，数据集大小为60,086个应用程序在我们这些研究的正常范围内。 由于这些应用程序是从可用应用程序中随机选择的，因此我们认为该数据集足够大以消除任何选择偏差。

我们的数据集不是Google Play商店中可用应用的统一快照，因为我们不下载以前下载的应用的更新版本。 我们没有个别应用程序的纵向统计数据，我们倾向于使用年轻版本的应用程序。 如果应用程序在应用程序商店中存在了很长一段时间，并且目标API水平的可能性更大，那么本文中提供的统计数据可能会从用户的角度来高估问题。 然而，数据集A和数据集B的集合之间存在18个月的差距。如果长期存在的应用程序针对过时的API水平较低，那么我们期望在数据集A的结果中看到一些指示。

因为我们只研究免费应用程序，所以我们的结果可能不适用于非免费应用程序。 我们没有理由怀疑非免费应用的开发实践显着不同。 然而，即使免费和非免费应用程序之间存在相当大的差异，免费应用程序占Google Play商店中应用程序的89％[28]，因此免费应用程序中存在的任何问题都至关重要，应予以解决。

第五部分所做的大部分分析都假设应用程序使用某些统一比率的危险功能，而不管它们的目标API级别如何。 如果这种假设不成立，那么我们的结论是许多应用程序不必要地使用这些危险的功能，因为它们针对过时的API级别可能无效。 特别是，行为变化可能会阻止开发人员重新定位他们的应用程序，我们不能认为危险功能的使用在应用程序群体中是统一的。 但是，我们在使用危险功能时观察到的差异非常大

这些差异是由应用程序中不同的预期行为引起的，这将是非常令人惊讶的。

1. 相关工作

这项研究最类似的工作是McDonnell，Ray和Kim [29]，他们研究了在十个开源Android应用中使用已弃用的API方法和采用更新的API方法。 与我们关注目标API级别的研究不同，他们关注在新API级别更改方法的使用。 定位API级别不依赖于使用在该平台版本中添加的方法，应用程序应该定位最新的API级别，即使它们不使用任何新添加的方法。 他们的结果没有提到过时应用程序的安全后果，但确实表明开发人员适应不断变化的Android平台的速度很慢。

在一些研究中已经针对特定脆弱性讨论了目标碎片问题。 Thomas等人 [5]研究Android 17中的变化，深入解决JavaScript Interface漏洞。 他们的研究主要集中在新设备的缓慢采用及其对漏洞生命周期的影响，但他们还发现22％的研究应用使用JavaScript接口和17以下的目标API级别.Mutchler et al。 [4]识别在WebView中加载不可信内容的应用程序，并注意JavaScript接口漏洞使这些应用程序处于危险之中。

已经研究了本文中提到的漏洞，但未提及目标碎片。 Lu等人 [30]构建一个静态分析器来识别包括服务劫持在内的漏洞。 Georgiev，Jana和Shmatikov [15]提供了一种防止通过JavaScript接口进行攻击的工具。 Chin和Wagner [13]静态地分析应用程序并发现文件：URL的不安全使用。 Jin等人 [31]构建一个工具来检测WebView应用程序中的各种类似XSS的漏洞。

由于必须更新应用程序和设备才能利用新的安全功能，因此目标碎片和设备碎片相关联。 Thomas，Beresford和Rice [2]使用安装设备监视器应用程序的志愿者研究设备碎片。 他们发现，88％的设备容易受到至少一个选定的漏洞的攻击，并且设备很少更新（平均每年1.26次）。 周等人 [32]通过设备驱动程序的错误纠正，可以利用2,423个以上的工厂映像中的1,000多个。 Xing等人 [33]确定应用程序如何利用操作系统更新过程来获取敏感的系统权限。 Mulliner等人 [34]为易受攻击的Android设备提供可扩展的第三方补丁程序。

应用程序更新过程也进行了研究。 莫勒尔等人。 [35]调查Android用户的更新模式，并发现只有一半的用户在更新发布后的一周内安装应用更新。 McIlroy，Ali和Hassan [36]从10,713个应用程序中更新数据，并发现只有1％的应用程序每周至少接收一次更新。

其他几项研究分析了Android应用程序的大数据集。 Viennot，Garcia和Nieh [10]对1,100,000个应用程序进行抓取，下载和分析，以获取有关权限和库分布的统计信息，并识别未安装嵌入的应用程序

证书。 其他研究[37,38]也研究许可和图书馆的使用。 Kavaler等人 [39]比较Android类和在StackOverow上提出的问题的使用情况。

1. 结论

Android应用程序指定目标API级别，并在具有更高API级别的设备上以兼容模式运行。 兼容模式可以禁用Android平台中的重要安全更改。 我们将针对过期API级别的应用程序问题称为目标碎片问题。 在这项研究中，我们分析了四年内收集的超过100万个Android应用程序的数据集，并显示绝大多数收集的应用程序都针对过时的API级别。 我们检查目标碎片对Android平台的七个安全更改的实际影响，并显示目标碎片会ha吓新的安全功能。

我们认为，以这种可选方式应用安全更改是一种令人敬畏的方法，即在兼容性的祭坛上的安全性。 开发人员成为保护应用程序的新障碍，用户无法确保他们的应用程序定位最新的API级别。 目标碎片问题由于安全更改和非安全更改的耦合而变得更加复杂。 我们希望通过阐明这个问题，开发人员可以更加了解针对过时API级别的后果，并且可以重新审视和更改此类设计，以便Android应用在没有重要安全功能的情况下运行的机会较少。

参考

1. I.伦登。 Android在2014年打破了1b的标记，所有13b智能手机的81％出货。 [线上]。 可用性：<http://techcrunch.com/2015/01/29/android-breaks-1b-mark-for-> 2014-81-的最1-3B智能手机 - 运 - 中计
2. DR Thomas，AR Beresford和A. Rice在2015年第五届ACM智能手机和移动设备安全与隐私讲座会议论文集“安全生态系统的安全性指标”。
3. Webview addjavascriptinterface远程代码执行。 [线上]。 可用：<http://labs.mwrinfosecurity.com/blog/2013/09/24/webview-> addjavascriptinterface-远程代码执行
4. P. Mutchler，A. Doupe'，J. Mitchell，C. Kruegel和G. Vigna，2015年移动安全技术学术期刊“移动网络应用安全的大规模研究”。
5. DR Thomas，AR Beresford，T. Coudray，T. Sutcliffe和A. Taylor，“android api漏洞的生命周期：关于javascript-to-java界面的案例研究”，2015年安全协议XXII。
6. 应用清单。 [线上]。 可用：<http://developer.android.com/guide/> 专题/清单/清单，intro.html
7. Build.version代码。 [线上]。 可用：<http://developer.android.com/> 参考/ android / os / Build.VERSION CODES.html
8. *<*用途-SDK>。 [线上]。 可用：<http://developer.android.com/guide/> 专题/清单/使用-SDK-element.html
9. 安全提示。 [线上]。 可用：<http://developer.android.com/training/> 文章/安全tips.html请
10. N*. Viennot，E. Garcia和J. Nieh在2014年ACM计算机系统测量和建模会议论文集中的“谷歌播放的测量研究”。*
11. Apktool。 [线上]。 可用：<http://code.google.com/p/android-apktool>
12. 网页流量。 [线上]。 可用：<http://developer.android.com/reference/> 安卓/ WebKit的/ WebView.html
13. E. Chin和D. Wagner在2014年的“信息安全应用”中的“Bifocals：分析Android应用程序中的webview漏洞”。
14. D. Wu和RK Chang，2014年信息安全“分析Android浏览器应用程序for le：//漏洞”。
15. M. Georgiev，S. Jana和V. Shmatikov，2014年“2014年网络和分布式系统安全研讨会论文集”中的“Breaking and xing origin-based access control in hybrid web / mobile application frameworks”。
16. M. Backes，S. Gerling和P. von Styp-Rekowsky，“针对Android手机的本地跨站脚本攻击”，2011. [Online]。 可用：<http://infsec.cs.uni-saarland.de/projects/android-vuln/android> xss.pdf
17. 什么 is 杂 内容？ [线上]。 可用：<http://developers.google.com/web/fundamentals/security/prevent-> 混合内容/什么，是混合内容
18. refox中的混合内容阻塞。 [线上]。 可用：[HTTP：//支持。](http://support/) mozilla.org/en-US/kb/mixed-content-blocking-ﬁrefox
19. 将javascript代码绑定到android代码。 [线上]。 可用：http：// developer.android.com/guide/webapps/webview.html#BindingJavaScript
20. N.伯格曼。 滥用webview javascript桥梁。 [线上]。 可用：<http://50.56.33.56/blog/?p=314>
21. *<*提供商>。 [线上]。 可用：<http://developer.android.com/guide/> 专题/清单/提供商element.html
22. YZX Jiang，“在Android应用中检测被动内容泄漏和污染”，2013年第20届网络和分布式系统安全研讨会论文集。
23. R. Hay。 Android框架中的一个新漏洞：片段注入。 [线上]。 可用：<http://securityintelligence.com/new-> 漏洞的机器人的框架片段注入
24. 意图和意图过滤器。 [线上]。 可用：[http://developer.android。](http://developer.android/) COM /引导/组件/意图-lters.html
25. E. Chin，AP Felt，K. Greenwood和D. Wagner，“分析Android中的互动应用通信”，在2011年第9届移动系统，应用和服务国际会议论文集中。
26. Api解析器。 [线上]。 可用：<http://github.com/yeganehs/API-Parser>
27. 仪表板。 [线上]。 可用：<http://developer.android.com/about/> 仪表板/ index.html的
28. Google Play统计信息。 [线上]。 可用：<http://appbrain.com/stats>
29. T.McDonnell，B.Ray和M.Kim，“An android study of api stability and adoption in the android ecosystem，”Proceedings of the 29th IEEE Con​​ference on Software Maintenance，2013。
30. L. Lu，Z. Li，Z. Wu，W. Lee和G. Jiang，2012年ACM计算机和通信安全会议论文集“Chex：静态审查android应用程序中的组件劫持漏洞”。
31. X. Jin，X. Hu，K. Ying，W. Du，H. Yin和GN Peri，“基于html5的移动应用程序的代码注入攻击：特征化，检测和缓解”，2014年ACM SIGSAC会议论文集计算机和通信安全，2014年。
32. X. Zhou，Y. Lee，N. Zhang，M. Naveed和X. Wang，“碎片的危险：Android设备驱动程序定制中的安全危害”，2014年IEEE安全与隐私专题讨论会会议论文集。
33. L. Xing，X. Pan，R. Wang，K. Yuan和X. Wang，“2014年IEEE安全和隐私专题研讨会论文集”升级您的Android，升级我的恶意软件：通过手机操作系统更新升级特权“ 2014年
34. C. Mulliner，J. Oberheide，W. Robertson和E. Kirda，2013年第29届年度计算机安全应用会议论文集“Patchdroid：可伸缩的第三方安全补丁for Android设备”。
35. A.Müller，F. Michahelles，S. Diewald，L. Roalter和M. Kranz，“应用市场和安全影响的更新行为：谷歌播放的案例研究”，在Proceedings of the 3rd Workshop on Research in大。，2012年。
36. S. McIlroy，N. Ali和AE Hassan，“2015年度经验软件工程”新鲜应用：谷歌Play商店中经常更新的移动应用的实证研究。
37. ML Dering和P. McDaniel在“2014年IEEE军事通信会议论文集”中的“Android市场重建与分析”。
38. R. Johnson，Z. Wang，C. Gagnon和A. Stavrou，“安卓应用程序的权限分析”，2012年IEEE软件安全与可靠性协会会议论文集，2012。
39. D. Kavaler，D. Posnett，C. Gibler，H. Chen，P. Devanbu和V. Filkov在2013年社交信息学杂志上发表的题为“使用和询问：Apis用于android市场并询问关于堆栈栈的问题”。