小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems
DOI:10.20009/j.cnki.21-1106/TP.2025-0122
2026 年 4 月 第 4 期
Vol.47 No.4 2026

移动应用登录设备数量管控机制的安全性研究

刘智晨,张 歆,张晓寒

(复旦大学 计算机科学技术学院,上海 200433)
E-mail:liuzc22@m.fudan.edu.cn

摘 要: 当前,移动应用普遍采用登录设备数量管控机制,限制一个账号仅能在有限台设备上同时保持登录状态,从而防止服务方资源被滥用.然而,此类管控机制的安全性尚未得到充分研究.本文针对安卓应用的登录设备数量管控机制开展了系统性实证研究.本文提出了基于登录凭证复用和设备标识伪造的绕过攻击,并依此构建了分析系统,对流行应用的管控机制安全性进行了测试与分析.研究发现,本文测试的208款热门应用的管控机制均存在可被绕过的安全漏洞,导致黑灰产能以较低成本实现服务资源滥用.对此,本文从凭证存储和设备标识技术两方面提出了安全建议,以提升移动应用登录设备数量管控机制的安全性.

关 键 词: 移动安全;登录设备数量管控机制;设备标识;绕过攻击
中图分类号: TP309 **文献标识码:** A **文 章 编 号:** 1000-1220(2026)04-0919-08

Study on the Security of Login Device Count Control Mechanisms in Mobile Applications

LIU Zhichen,ZHANG Xin,ZHANG Xiaohan
(School of Computer Science,Fudan University,Shanghai 200433,China)

Abstract: Currently,it is common for mobile applications to use login device count control mechanisms that limit an account to a fixed number of simultaneously logged-in devices,in order to prevent misuse of the service provider's resources. However,the security of such control mechanisms has not been fully studied. In this paper,we conduct a systematic empirical study on login device count control mechanisms for Android applications. This paper proposes a bypass attack based on login credential multiplexing and device identity forgery,and builds an analysis system accordingly to test and analyze the security of the control mechanisms of popular applications. It is found that the control mechanisms of 208 popular applications tested in this paper have security vulnerabilities that can be bypassed,enabling illicit industries to exploit service resources at a lower cost. To address this issue,we propose security recommendations regarding credential storage and device identification techniques to enhance the security of the login device count control mechanisms in mobile applications.

Keywords: mobile security;login device count control mechanisms;device identification;bypass attack

0 引 言

随着移动互联网的迅速发展,智能手机已深度融入人们的日常生活,移动应用也在社交[1]、支付[2]、物联网设备控制[3]等多个领域提供了便捷的服务.然而,随着应用数量和用户规模的不断增长,服务方资源也面临着严重威胁.相关报告表明,2024年上半年,黑灰产从业人员规模已超过427万,国内作恶手机号数量高达323万,日活跃风险IP数量更是达到1136万[4].恶意刷单、薅羊毛等黑灰产事件层出不穷,不仅破坏了正常的市场秩序,还给企业带来了巨大的经济损失.

为防范这些黑灰产行为,许多应用引入了登录设备数量管控机制,通过限制单个账号只能在有限台设备上同时使用,达到防止资源滥用的目的.然而,该机制的自身安全性尚未得到充分关注.特别是,现有的管控机制能否有效抵御设备模拟、身份伪造等攻击方式,仍是一个亟待探讨的问题.

针对这一研究空白,本文系统地分析了安卓应用中登录设备数量管控机制的安全性.首先,本文从实现原理上分析了该机制存在的两个主要风险点,并据此提出了基于登录凭证复用和设备标识伪造的绕过攻击.随后,本文结合了静态系统插桩和动态测试的方式,构建了一个分析系统,通过信息监控模块捕获目标应用的数据采集行为,通过模拟攻击模块伪造多种设备信息,对目标应用实施绕过攻击,从而评估管控机制的安全性.

为了评估现实应用中登录设备数量管控机制的安全性,本文分析了208款热门应用程序,结果表明它们的管控机制在面对绕过攻击时表现出脆弱性.进一步分析发现,其中有76款(36.5%)应用仅根据登录凭证进行认证,未采用更安全的设备标识策略;而在使用了设备标识的132款应用中,有48款(36.3%)仅依赖单一设备属性,其余采用多重设备信息的应用也只局限于特定的几种设备属性,这表明当前登录设备数量管控机制在安全性和可靠性方面存在明显不足,难以有效抵御绕过攻击.针对这一现状,本文提出了安全建议,从加强登录凭证的安全存储和提升设备标识的精度两方面提高登录设备数量管控机制的安全性.

收稿日期:2025-03-17 收修改稿日期:2025-04-14 基金项目:青年科学基金项目(62102091)资助. 作者简介:刘智晨,男,2000年生,硕士研究生,研究方向为移动安全;张 歆,女,2000年生,博士研究生,研究方向为移动安全;张晓寒(通信作者),男,1991年生,博士,青年副研究员,CCF会员,研究方向为移动安全、恶意软件检测、AI应用安全等.

1 背景与相关工作

1.1 登录设备数量管控机制

随着移动设备的普及,用户在多个设备上使用同一账户的现象日益增多,服务方需在满足用户需求的同时,保护自身资源免受侵害. 因此,登录设备数量管控机制应运而生,其核心目标是通过限制登录单个账户的设备数量,实现平台资源的合理分配,防止恶意账户共享等不当行为,保证平台的稳定运营. 若缺乏有效的管控机制,用户可能通过共享账户导致资源过度消耗,增加服务器与带宽压力,甚至对服务方的盈利模式造成影响.

登录设备数量管控机制在提供安全保障的同时,也面临着诸多挑战. 首先,尽管该机制所依赖的设备标识技术能够在大多数情况下准确识别设备,但仍存在标识伪造的风险[5],从而导致管控机制的有效性被破坏. 具体来说,攻击者可利用虚拟设备、模拟器等技术伪造设备标识,使服务方无法有效区分不同设备,从而绕过登录设备数量的限制. 其次,随着安卓版本的不断升级,谷歌官方对部分标识信息的采集权限进行了更严格的限制. 例如,自安卓 10 起,第三方应用已被完全禁止访问 IMEI,而设备序列号的获取则需要应用具备高级别的权限. 这些限制导致应用无法使用强唯一性的标识信息进行设备认证,对基于设备标识的管控机制提出了新的挑战.

1.2 相关工作

目前,与登录设备数量管控机制相关的研究主要聚焦于设备标识技术的实现方法. 现有技术主要通过收集设备的硬件配置、软件环境、网络状态等特征信息,生成唯一标识. Zhou 等人[6]首次提出利用设备的硬件动态特征,他们根据扬声器的频率响应对设备进行区分,验证了这种方法的可行性,随后,传感器特征也被用作设备标识[7]. 另外,Kondracki 等人[8]进一步探索了存储信息、配置信息和硬件特征的融合方式,扩充了设备标识信息的广度. 尽管对于设备标识技术的研究已有诸多进展,但随着攻击手段的演进,现有技术能否保持有效性仍然是一个亟需探索的问题.

此外,设备标识技术的实际应用也存在一定困难. 设备仿真技术的发展使得攻击者能够模拟或篡改设备的标识信息,从而绕过基于设备标识的安全机制[9]. 设备虚拟化技术的普及导致应用难以准确获取真实设备的软硬件信息,进一步增加了实施设备标识的复杂性[10-12]. 同时,隐私保护政策的日益严格限制了应用程序对设备信息的访问,使得可用于构建设备标识的数据范围进一步收缩[13,14].

在这一背景下,本文首次对真实世界中的登录设备数量管控机制进行了系统性研究,揭示了其安全现状,以帮助提升该机制的安全强度.

2 登录设备数量管控机制安全性分析

2.1 基本原理

登录设备数量管控机制的核心在于对登录凭证和设备标识进行双重验证,分为初始化阶段及后续使用阶段. 具体流程如图 1 所示,在初始化阶段中,当用户首次在设备上完成登录后,服务端会为其分配一个独特的登录凭证,并由客户端进行本地存储. 同时,服务端会基于客户端采集的设备属性生成唯一的设备标识以区分该设备,并将其与用户账号绑定. 在后续

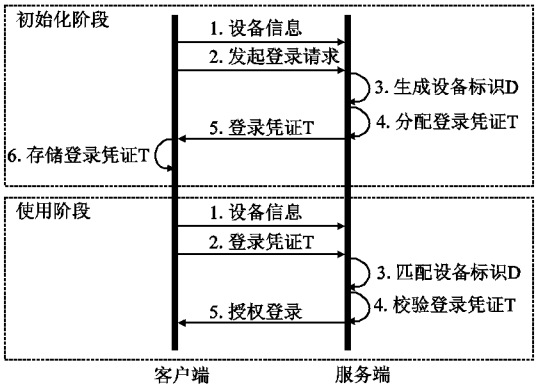

图 1 管控机制基本原理

Fig. 1 Basic principles of control mechanism

使用阶段中,用户无需重复手动登录,客户端会自动将登录凭证和设备属性作为身份凭据发送至服务端进行验证. 只有两项凭据的校验全部通过且账号的登录设备数量未超过上限时,服务端才会授权该登录请求. 若验证失败或设备数量超限,服务端将拒绝此次登录请求,或要求用户进行额外的身份认证,从而清除其他设备的登录状态,确保对在线设备数量的严格管控.

2.2 风险点分析

由于登录设备数量管控机制的目标是防止服务端资源被恶意滥用,因此本文的威胁模型假设攻击者是一名恶意用户,其攻击目标是绕过应用的管控机制,从而在超出限制的多台设备上进行非法访问. 在这种现实攻击场景下,攻击者具备对自身设备的完全控制权,能够访问全部本地数据,并能够利用系统插桩等技术手段篡改设备属性信息.

在该威胁模型下,基于登录凭证与设备标识进行设备识别的方式存在两个关键的风险点:登录凭证的易获取性和设备标识的可伪造性.

登录凭证的易获取性. 移动应用通常会在设备本地存储登录凭证,以便用户在重新启动应用时无需手动认证即可完成登录过程. 出于安全性考虑,应用通常将凭证数据存储于私有目录中,例如常见的/data/data/package_name 路径下的 SharedPreferences 或 SQLite 数据库等[15]. 然而,由于攻击者拥有对自身设备的完全控制权,其能够轻易访问这些存储位置,从而提取到有效的登录凭证,并可将其复用到其他设备的相同目录下,从而绕过服务端对凭证信息的校验.

设备标识的可伪造性. 设备标识通常用来识别唯一设备,以便服务端对发起请求的设备进行身份验证. 常见的设备标识信息包括 Android ID、MAC 地址等. 然而,这些标识信息均由应用客户端采集后发往服务端,其真实性难以完全保证,这一点在存在环境风险的设备上尤为突出. 对于高级攻击者而言,他们可以伪造应用所获取的自身设备标识数据,使得服务端无法区分攻击者的多台设备,从而伪装成另一台已登录的设备,绕过服务端对设备标识的校验.

2.3 攻击方式

基于上述两个风险点,本文提出了一种结合登录凭证复

用与设备标识伪造的绕过攻击,能够实现在多台攻击设备和一台原始设备上同时登录存在限制的同一账号. 首先,攻击者在一台原始设备上正常登录并使用目标应用,应用会对服务端分配的登录凭证进行本地存储,同时,客户端收集的设备信息也会传输至服务端以生成唯一的设备标识. 然后,由于登录凭证的易获取性,攻击者可以提取原始设备上的凭证数据,并将其复用到攻击设备中. 接着,由于设备标识的可伪造性,攻击者可以采集原始设备信息,并通过篡改攻击设备的对应信息项,完成设备标识的伪造. 最终,借助这两种手段的协同作用,攻击者能够欺骗目标应用将攻击设备识别为原始设备,进而可以在多台设备上以原始设备的身份完成登录,突破登录设备数量管控机制的限制.

3　登录设备数量管控机制分析系统

3.1　系统架构

为了检测真实应用中登录设备数量管控机制实现的有效性,本文提出了一个基于绕过攻击的分析系统,通过检验真实应用中登录设备数量管控机制的抗攻击能力评估该机制的安全性生态. 如图 2 所示,整个系统包括信息监控和模拟攻击两个主要模块.

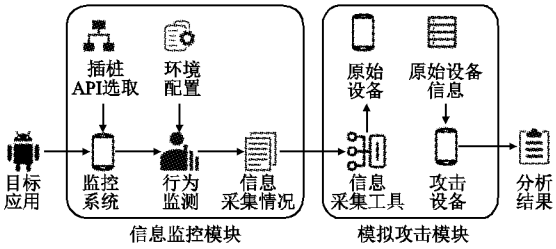

图 2　系统架构图

Fig. 2　System architecture

信息监控模块的核心功能包括两个方面:1)确定目标应用登录凭证的本地存储位置;2)监测目标应用的设备属性采集情况. 在具体实现上,该模块基于系统 API 的功能特性选取需要监控的 API. 通过对这些 API 进行插桩,构建了一个监控系统,对目标应用的数据收集行为进行监测,监控结果为模拟攻击提供了充分的先验知识和全面的数据支持.

模拟攻击模块的功能主要包括登录凭证的提取与复用以及设备标识的采集与伪造. 首先,针对登录凭证,该模块将从信息监控模块定位到的存储位置提取凭证数据,随后将这些有效凭证复用到攻击设备的相同路径下. 其次,对于设备标识部分,该模块基于信息监控模块捕获的设备属性项,采集原始设备的各项特征. 而后通过修改 AOSP 中信息采集 API 的实现逻辑,将攻击设备的特征修改为原始设备的值,从而实现设备标识的伪造. 在完成登录凭证的复用和设备标识的伪造后,模块将使用攻击设备与服务端交互,验证模拟攻击是否成功,进而分析目标应用管控机制的安全性.

3.2　信息监控模块

为实现对目标应用信息收集情况的监测,本文结合静态分析与动态测试技术,设计并实现了一个自动化的监控模块,该模块主要包括 3 个步骤. 首先,基于系统 API 的功能选择合适的 API 进行静态插桩,以便在目标应用运行时能够定位登录凭证的存储位置并捕获其采集的设备属性. 其次,构建监控环境对目标应用进行动态监控,保证监控过程仅作用于目标应用,而不影响其他应用的正常运行. 最后,监控到的数据经过清洗和过滤后,将以统一格式存储,以供后续分析使用. 下面将详细介绍 3 个步骤的实现细节.

3.2.1　插桩 API 的选取

在整个分析系统的实现过程中,插桩 API 的选取是至关重要的一步. 为确保能够有效定位目标应用的登录凭证存储位置,并全面覆盖目标应用生成设备标识所需的设备属性,选取的 API 需严格遵循以下几个原则.

首先,有效性要求所选 API 能有效反映应用的设备属性采集行为或文件读写操作. 其中,设备标识通常依赖于 deviceID、MAC 地址等唯一信息,因此本文优先选择能够获取这类属性的 API. 同时,经由人工分析,本文发现操作系统信息、硬件信息等非唯一性属性常作为辅助识别信息. 此外,本文还重点关注了文件读写相关的 API,因为登录凭证的存储与使用常会通过这些 API 进行.

其次,全面性要求插桩的 API 集合应覆盖生成设备标识所必须的全部属性,包括但不限于设备型号、IMEI、操作系统版本等. 另外,API 集合还应涵盖所有文件读写接口,确保能够完整捕捉设备标识构建及登录凭证存取过程.

此外,精确性是保证监测质量的重要因素. 部分 API 受参数影响会返回不同的设备信息,因此,本文通过解析参数,精准定位采集的属性项. 例如,通过记录 open( ) 方法的文件路径参数,准确识别目标应用访问的具体属性.

最后,稳定性要求插桩操作不影响系统的正常运行,同时也不能被应用检测到异常. 为此,本文选择系统底层 API 作为插桩目标,避免修改应用层 API 导致插桩痕迹暴露.

表 1　关键 API 插桩示例

Table 1　Key API instrumentation examples

API	功能描述	插桩位置
getImei( )	获取 IMEI	TelephonyManager Service 实现类
Settings. getString( )	获取 Android ID 等标识符	getStringForUser 底层方法
SystemProperties. get( )	获取系统信息	__system_property_read 底层方法
open( )	文件读写操作	open 方法实现
popen( )	执行 shell 命令	popen 方法实现

基于上述原则,本文最终确定了 106 个插桩 API,能够监控 169 项设备信息. 与现有的前沿工作 VPDroid[16] 相比,本文的分析系统支持更广泛的设备信息项. 具体而言,VPDroid 支持 101 项设备信息,本文分析系统在涵盖全部 101 项信息的基础上,还支持了通过文件读取和命令执行获取的设备信息. 表 1 展示了几种关键 API 及其对应的插桩方法.

3.2.2　目标应用的动态监控

基于对上述 106 个 API 的插桩处理,本文对其调用情况进行动态监控,以捕获与设备标识生成相关的设备信息,并记录可能涉及私有文件访问的操作. 为确保监控结果的有效性并避免产生过多冗余信息对系统正常运行造成干扰,需要设定合理的监控范围. 为此,本文从监控环境、监控时机和动态

执行方式3个方面进行了设置.

1) *监控环境配置*. 为了保证监控过程的可控性,并避免影响其他应用的正常运行,本文基于AOSP定制了一个监控系统,使得监控仅对目标应用生效,减少了系统额外开销. 具体来讲,在对选取的API进行插桩时,除了注入实现监控功能的代码外,还添加了对目标应用的检测机制. 当应用调用被监控的API时,系统会读取预存的目标应用包名信息,并检查当前进程是否属于目标应用(应用进程的命名一般以包名为前缀). 只有通过进程检查后,监控代码才会生效,系统才会将应用采集的信息项记录到日志文件中.

2) *监控时间设定*. 动态监控的目标是捕捉应用在设备标识采集及登录凭证存取过程中的关键行为,因此实施监控的时间段要能够覆盖这些操作. 本文选择从目标应用的初次启动开始监控,直到用户完成登录为止. 因为在这段时间内,应用通常会在用户登录前收集设备信息以供生成设备标识,并且在用户登录后,应用会存储本次登录凭证. 该时间段能够保证关键数据的完整捕获,并且避免了因监控时间过长导致的大量无效信息.

3) *动态监控执行*. 由于监控时间段内涉及账号的登录操作,本文采用人工辅助的动态测试方案,需要人工完成目标应用的启动、注册、登录操作. 这一过程可以确保系统完整地记录目标应用的信息采集行为,并且避免了由于测试工具触发无关功能产生的大量冗余信息,在保证监控结果全面性的同时,极大地减少了无关数据的引入.

3.2.3 监控结果的处理

在动态监控过程中,系统产生了大量日志信息,这些日志详细记录了目标应用对监控信息的访问情况. 然而,监控到的原始数据往往包含一定的噪声数据和无效信息,直接使用可能影响分析的准确性. 为了确保监控结果的可靠性,本文对原始数据进行了预处理,主要包括数据去重与数据过滤两个部分,以提取具有实际分析价值的信息.

1) *数据去重*. 在监控过程中,某些API可能被目标应用多次调用,从而产生重复的日志信息,导致数据冗余. 而由于本文的关注点在于目标应用是否执行了信息采集操作,并不关注API的调用频次,因此,对于内容完全相同的API调用记录,仅保留首次调用的记录,减少无效数据的干扰,提高分析效率.

2) *数据过滤*. 受安卓版本更新及权限机制变更的影响,部分API在较高版本的系统中无法正常获取设备信息. 例如,在Android 10及更高版本中,出于隐私保护的考虑,系统限制了对IMEI号的访问权限,普通应用调用getImei()方法时,仅能得到一个空值. 为提高数据的有效性,本文对这类无法采集真实设备信息的API进行过滤,避免错误数据影响后续实验结果.

经过预处理后,剩余数据将以格式化的形式存储,最终形成一个更准确且满足分析需求的设备信息项集合,为模拟攻击模块提供完备的前提条件.

3.3 模拟攻击模块

模拟攻击模块的目标在于通过模拟真实的攻击场景,评估目标应用的登录设备数量管控机制在面对此类攻击时的有效性,并通过动态调整模拟信息项深入分析其具体实现方式. 该模块的实现包括3个主要过程:设备标识采集与登录凭证提取、设备标识伪造与登录凭证复用、模拟攻击执行与测试结果分析.

3.3.1 设备标识采集与登录凭证提取

模拟攻击的第1步是获取原始设备的属性信息,为设备标识的伪造和登录凭证的复用提供真实的数据支持. 本文根据信息监控模块的结果,开发了一套专门的工具,实现对原始设备信息以及登录凭证信息的自动采集,并将采集结果以统一格式输出.

1) *设备标识采集*. 在移动设备中,应用程序可通过多种方式采集设备的标识信息,主要包括调用API返回属性值、读取成员变量值、访问文件信息以及执行系统命令. 针对不同的采集方式,本文设计了相应的采集策略,以确保全面获取原始设备中的重要标识信息.

对于API返回值方式,本文利用反射机制对目标API进行动态调用,捕获其返回的设备标识信息. 值得注意的是,即使在同一台设备中,不同签名的应用获取到的Android ID也是不同的. 对此,本文采用Frida[17]向目标应用注入代码,以确保采集到其实际的Android ID信息. 对于读取成员变量方式,本文直接访问目标类中的静态成员变量以提取相关数据. 对于文件访问方式,考虑到应用可能利用文件的全部或者部分内容信息,也可能利用文件的属性信息,本文采取完整保存文件的策略,以确保数据的完整性. 最后,对于执行系统命令方式,本文主动执行相应shell命令并捕获其输出的设备标识信息.

2) *登录凭证提取*. 基于信息监控模块,本文能够全面记录目标应用在监控时间段内的文件访问行为,从而识别出可能存储登录凭证的位置. 通常,登录凭证会存储于应用的私有目录中,而访问私有目录需要特定的权限,普通进程无法直接获取相关数据.

为解决这一问题,本文开发了一个基于root权限的自动化脚本,采集所有可能包含凭证信息的文件,确保登录凭证的成功提取.

3) *采集数据输出*. 在完成设备标识信息和登录凭证的采集后,需要对数据进行整理与格式化,以便后续的读取和解析. 由于采集到的数据类型多样,包括数值、文本和文件等形式,因此本文采用合适的存储策略,以确保数据的规范性和可用性,具体如下.

首先,对于通过API返回值、类成员变量读取以及系统命令执行方式获取的信息,本文将其统一转换为字符串格式,并采用键值对的方式存储,其中键为采集项的名称,值为对应的设备标识数据. 所有键值对数据将按类别存储在一个JSON文件中,因为JSON格式具有良好的可读性与扩展性,有助于后续对目标应用管控机制的分析. 其次,对于采集到的文件数据,本文按照其原始路径创建相同的目录结构,并将所有文件归并至统一的根目录下存储. 这种方式不仅保留了原始文件系统的层次结构,也便于在后续信息伪造过程中实现文件的重定向.

经过以上处理,所有采集到的数据将分两部分保存:一个是以JSON格式存储的结构化数据文件,另一个是包含全部采集文件的目录.

3.3.2 设备标识伪造与登录凭证复用

基于采集到的原始设备信息,本文通过修改系统底层 API 的实现逻辑,对攻击设备的标识信息进行伪造,同时对登录凭证进行复用,实现对原始设备环境的模拟,进而评估目标应用的登录设备数量管控机制的安全性. 此外,在这一过程中,模拟时机的选择至关重要,不恰当的时机可能导致攻击失效.

1) *模拟时机选择*. 采用管控机制的应用在启动后会立即收集设备标识信息,以验证设备身份. 因此,必须在应用执行采集操作之前完成模拟,否则目标应用可能获取到攻击设备的真实信息,影响实验结果. 在应用启动过程中,系统首先为其创建运行环境,加载必要的资源和库文件,然后再启动应用主线程,开始执行其功能. 基于这一流程,本文选择在目标应用获得执行权之前,实现设备信息的篡改,从而确保应用采集到的是伪造的原始设备信息.

此外,为避免模拟操作影响系统中的其他应用,本文设计了进程隔离机制. 具体来讲,在信息模拟之前,先获取系统中正在运行的进程信息,并判断当前进程是否属于待测试应用. 只有进程信息与目标应用匹配时,才执行篡改操作,从而确保伪造信息仅对目标应用生效,而不会干扰系统的其他应用的正常运行.

2) *设备标识伪造*. 为了有效地绕过应用对设备标识信息的检测,本文根据设备信息的不同采集方式设计了相应的伪造方法,实现设备标识信息的准确模拟.

对于通过 API 调用获取的设备标识,本文修改了这些 API 依赖的底层函数,使其返回存储于 JSON 文件中的预设信息. 对于成员变量形式的设备信息,本文采用反射机制直接修改目标类的成员变量值,例如调整 Build. MODEL 的值,伪装成原始设备的型号. 针对文件读取方式,由于安卓系统提供了 File 类、open( ) 和 fopen( ) 方法访问系统文件,而这些方法均依赖底层 libc 中的__open( ) 和__open_2( ) 函数,因此本文修改了这两个函数的实现,将文件访问重定向至采集目录中的对应文件,使目标应用读取到原始设备的文件信息. 最后,对于命令执行方式,本文修改了其底层依赖的 popen( ) 函数的实现,使其从 JSON 文件中读取伪造数据,并通过执行 echo 命令进行输出,从而确保系统命令执行的结果与原始设备保持一致.

3) *登录凭证复用*. 与设备标识信息的采集方式类似,应用在管理登录凭证时也采用文件读写的方式. 因此,本文同样通过修改底层文件访问函数 _open( ) 和_open_2( ) 实现登录凭证的复用. 具体而言,当目标应用调用这些函数时,系统首先检查其参数中的文件路径,判断是否指向应用的私有目录. 若检测到文件路径属于应用私有存储,则将文件访问请求重定向到预先准备的采集目录,并加载其中存储的原始设备登录凭证信息.

这一过程的关键是通过文件重定向,确保应用在执行登录操作时读取到预设的凭证,而非攻击设备的真实信息. 这种方法不仅实现了登录凭证的无缝复用,还最大程度地减少了对应用正常功能的干扰,保障了模拟攻击的成功进行.

3.3.3 模拟攻击执行与测试结果分析

在完成设备标识伪造和登录凭证复用后,可以开始模拟攻击过程,以评估目标应用在本文攻击场景下的鲁棒性. 对于能够成功绕过登录限制的应用,本文进一步动态调整模拟的设备信息项,以深入分析其管控机制的具体实现方式.

1) *模拟攻击执行*. 在设备标识伪造和登录凭证复用成功实施后,只需在攻击设备上启动目标应用,即可进行攻击测试. 启动后,目标应用会利用注入的原始设备信息进行身份验证. 如果其管控机制存在漏洞或未能有效识别伪造信息,便可绕过登录限制,在攻击设备上获得账号的使用权限. 然而,若目标应用检测到伪造的设备标识或登录凭证,通常会依据其安全策略采取防御措施,例如弹出安全警告并拒绝此次登录请求,或者强制用户重新认证等,这类情况判定为攻击失败.

为了进一步验证攻击的时效性,本文在突破目标应用的登录限制后,将持续观察其登录状态与功能使用是否稳定. 如果攻击持续有效,且登录状态在一段时间内未被重置或检测到异常,则表明目标应用的管控机制存在较大安全隐患,需要加强防护措施.

2) *迭代式测试分析*. 对于可被绕过登录限制的目标应用,本文通过修改 JSON 文件和采集目录中的设备信息项,动态调整模拟的设备信息,以深入分析其登录设备数量管控机制的具体实现方式. 具体而言,本文将根据应用对模拟攻击的反馈结果,逐步调整注入的设备信息,并在每次调整后重新启动目标应用,验证其登录行为,观察是否仍能成功绕过管控机制. 通过这种方式,本文可以识别出应用在设备认证过程中依赖的关键属性. 例如,如果注入某一信息项能够绕过目标应用的登录限制,而不注入该信息项时无法绕过,则表明该信息在管控机制的实现中起到了重要作用.

通过系统化的攻击测试与分析,本文不仅评估了目标应用的鲁棒性,还深入探讨了其管控机制的具体实现逻辑. 这些结果有助于揭示各类应用在登录设备管控方面的潜在安全缺陷,并为相关安全策略的优化提供参考.

3.4 系统实现

本文分析系统主要包括信息监控和模拟攻击两个模块. 在信息监控模块的实现过程中,本文基于 AOSP 10 源码对选定的 API 进行插桩,插桩的代码首先判断当前进程是否属于目标应用,如果是,则将与该 API 关联的设备信息输出到日志文件中;如果不是,则不做任何处理. 基于在监控系统上动态测试获得的目标应用的信息采集情况,模拟攻击模块首先利用本文开发的工具对原始设备信息进行自动化采集. 采集工具主要包含 3 部分:首先是使用 Android Studio 开发的第三方应用,用于采集无需特权的设备信息;其次是 Frida 脚本,用于采集 Android ID 等应用间不同的设备信息;最后是 shell 脚本,用于获取目标应用的私有文件. 随后,通过修改系统源码,在目标应用启动过程中,即其 Application 对象创建之后且应用获得执行权之前,注入实现设备信息篡改的代码,使得攻击设备上的目标应用获取到原始设备的信息,进而实现对目标应用登录设备数量管控机制的测试与分析.

4 实验与评估

本节首先评估了分析系统的性能,接着在真实世界的热门应用上模拟了绕过攻击,以揭示移动应用登录设备数量管控机制的安全性现状,并提出了针对性的安全建议.

4.1 实验环境与数据集

本文在一台运行 Ubuntu 18.04 操作系统的服务器上实现了上述分析系统,并完成系统镜像的构建.模拟攻击实验在 3 台设备上进行,其中一台 Pixel 3XL 作为攻击设备,运行基于 AOSP 10 开发的分析系统,另一台 Pixel 3XL 和一台 Pixel 2XL 作为原始设备,分别运行设备出厂的 Android 11 和 Android 10 系统.

实验过程中,所有测试均在个人账号上进行,确保不会对其他用户产生影响.此外,在分析应用管控机制的具体实现时,本文严格控制测试频率,避免过多消耗服务端资源.同时,对于成功绕过管控机制的应用,研究仅限于安全性评估,并不涉及任何恶意操作.

为全面评估登录设备数量管控机制的安全生态,本文从华为应用市场爬取应用数据,并基于触发管控机制时的语义特征筛选出候选应用.最终,本文选取下载量排名前 200 的候选应用作为主要实验数据.此外,考虑到谷歌应用在全球范围内的广泛使用及其成熟的管控机制,本文额外选取 8 款谷歌系列的应用对实验数据进行补充.

4.2 系统性能评估

为评估分析系统的性能开销,本文采用两台 Pixel 3XL 设备进行对照实验,其中实验组运行本文定制的分析系统,对照组运行标准的 AOSP 10 系统.通过对比两组设备在相同条件下的性能,评估分析系统的运行时间开销和内存开销.

1) *时间开销*.对于运行时间开销的评估,本文采用与 VPBox[18] 类似的实验设置:分别在无噪声和有噪声两种环境下运行同一组基准测试应用,并对比实验组和对照组的执行时间.其中,基准测试应用分别进行了复杂的计算任务、大规模的文件读写以及高频率的网络请求,它们分别重点考量了系统在 CPU 负载、文件 I/O 以及网络使用方面的性能开销.

为保证数据的可比性,本文以对照组为基准对实验结果进行了归一化处理.图 3 中的实验结果表明,在无噪声环境下,分析系统的运行时间开销均不超过 2.5%,对设备性能几乎没有影响;在有其他应用运行的噪声环境下,相对开销略有增加,这主要是由于噪声应用的引入增大了对照组的时间开销基数,而从具体数值来看,分析系统在两种环境下的额外运行时间开销基本保持一致.

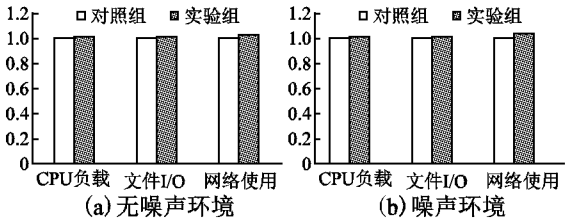

图 3 系统运行时间开销

Fig. 3 System runtime overhead

2) *内存开销*.对于内存开销的评估,本文在两组设备上启动多种应用模拟不同使用环境,并监测它们在各环境下的内存占用情况.具体来讲,本文采用内存监控工具进行实时记录,重点关注两组设备在执行相同功能时的内存差异.实验结果如图 4 所示,与对照组相比,实验组的内存消耗略有增加,这主要归因于设备信息模拟过程中数据的临时存储.但是,该额外开销相对较小,不会对设备运行产生明显影响,更未出现内存溢出或设备崩溃的情况.

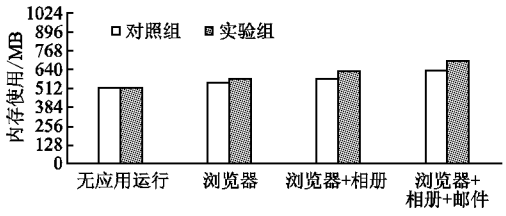

图 4 系统内存开销

Fig. 4 System memory overhead

4.3 管控机制安全性评估

4.3.1 总体结果

本文对实验数据中的 208 个应用进行了模拟攻击实验,以评估其管控机制的鲁棒性.实验结果表明,这些应用均未能有效抵御本文的攻击方式,揭示了现有管控机制存在严重安全缺陷.

进一步地,本文对各应用的管控机制实现方式进行了深入分析,在 208 款应用中,有 76 款仅根据登录凭证判定设备,而其余 132 款则在验证登录凭证的基础上,引入了设备标识校验机制.

设备标识的使用情况如图 5 所示,在采用该策略的 124 款国内应用中,有 46 款仅依赖 Android ID,有 62 款在 Android ID 的基础上结合了设备的 MODEL 信息进行判断,有 14 款采用 Android ID、MODEL 和 BRAND 信息的组合方式进行设备验证,另有 2 款仅仅采用唯一性较差的 MODEL 信息作为设备标识.以上结果表明,现有应用的设备标识策略主要依赖 Android ID 这一特殊标识,它能有效实现设备的区分,但由于依赖的设备属性较为有限,容易受到针对性攻击.

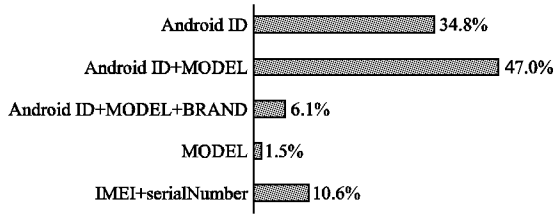

图 5 设备标识使用情况

Fig. 5 Use of device identification

对于谷歌系的 8 款应用,本文发现它们均基于 GMS (Google Mobile Services)进程实现设备的管理与验证.由于 GMS 作为系统预装应用,具备开机自启的特性,因此能够访问系统层级的特权数据.分析结果表明,GMS 依赖 IMEI 和 serialNumber 作为设备标识,这些特权数据具有较强的唯一性和稳定性,能够有效区分不同设备.

4.3.2 不同实现方式的评估

基于对 208 款应用的分析结果,本文将登录设备数量管控机制分为 3 类:仅校验登录凭证、结合登录凭证与普通设备标识、结合登录凭证与特权设备标识.其中,对于采用普通设备标识的方式,根据标识信息的数量进一步分为单一标识和复合标识两类.各实现方式的使用情况详见表 2,下面将对不同实现方式进行详细分析.

1)仅依赖登录凭证的实现方式. 为全面评估该策略的应用现状,本文从应用分布和流行度两个维度进行了分析. 从应用分布来看,社交、金融及购物类应用由于交互频繁或涉及资金交易,通常不采用单一凭证认证,在本文研究的 76 款应用中,仅有 3 款社交应用、2 款金融应用和 2 款购物应用采用了这种方式. 相比之下,教育和办公等工具型应用更倾向于使用单一凭证认证方式,分别有 21 款教育应用和 14 款办公应用采用了该认证方式. 这表明,功能相对简单、交互频率较低的应用优先考虑功能实现,而对安全性的关注相对不足. 从流行度来看,仅依赖登录凭证的应用平均下载量约为 1200 万次,显著低于采用登录凭证与设备标识组合策略的应用(约 3800 万次). 这一差异表明,随着用户规模的增长,开发者更注重安全需求,从而引入更完善的验证机制,以保障用户及平台资源的安全. 此外,单一凭证认证方式不仅容易受到本文的绕过攻击,也难以抵御其他常见攻击类型. 例如,中间人攻击可截获应用与服务端之间的通信数据,劫持有效的登录凭证,从而冒充合法用户进行访问. 在这种情况下,攻击者不仅可以突破应用的管控策略,甚至能完全接管受害者账号,造成严重的安全威胁.

表 2　管控机制实现情况

Table 2　Implementation of control mechanisms

管控机制实现方式	应用数量(百分比)
仅登录凭证	76(36.5%)
登录凭证 + 单一普通标识	48(23.1%)
登录凭证 + 复合普通标识	76(36.5%)
登录凭证 + 特权标识	8(3.9%)

2)结合登录凭证和普通设备标识的实现方式. 相较于仅依赖登录凭证的方式,不少应用已引入设备标识以增强身份验证. 然而,分析结果表明,这些应用在设备标识的选择上较为局限,且部分应用的实现存在明显的安全隐患.

首先,设备标识的信息范围存在局限性. 在采用设备标识策略的 124 款普通第三方应用中,绝大多数(98.4%)使用 Android ID 作为核心标识. 其中,约 62.3% 的应用进一步结合其他设备属性形成了复合标识. 然而,这些附加属性的唯一性通常较弱,在同一型号或同一厂商生产的设备间存在较高的碰撞概率,从而造成了设备区分能力的削弱. 更令人担忧的是,个别应用仅依赖设备型号信息进行设备标识. 这不仅导致无法对设备进行有效区分,还由于市面上有限的设备型号数量,使其容易受到暴力攻击,攻击者可通过遍历所有设备型号实现伪造,导致安全防护的失效.

其次,设备标识的采集方式也存在不足. 以 Android ID 的为例,系统提供了多种采集途径,包括:1)调用 Settings. Secure. getString()方法;2)通过反射获取 Settings 类的 sNameValueCache 或 MOVED_TO_GLOBAL 变量;3)借助 ContentResolver 的 call()方法间接获取;4)使用 query 命令查询. 然而,如表 3 所示,本文分析的这些应用中,大部分(80.3%)仅采用 getString()这种简单的方法. 由于该 API 暴露于系统上层,容易遭到攻击者的劫持,导致采集结果可被篡改. 另外,有 24 款应用从多个渠道采集了 Android ID,并对各方式的采集结果进行了一致性校验. 这些应用大多来自知名互联网厂商,且拥有庞大的用户基础,主要集中在社交、影音和购物等涉及高敏感信息的领域. 通过这种策略,确保了在单一采集方法遭到篡改时,仍能检测到异常,从而增强了管控机制的抗攻击能力.

表 3　Android ID 的采集情况

Table 3　Collection of Android ID

采集方式	应用数	采集方式种类	应用数
调用 getString 方法	122	1 种	98
内存反射	9	2 种	18
ContentResolver 间接获取	17	3 种	6
执行 query 命令	4	4 种	0

3)结合登录凭证和特权设备标识的实现方式. 厂商预装应用通常具备更高的权限,能够获取普通第三方应用无法访问的特权设备信息. 这些特权标识符具有强唯一性,可以避免普通设备标识可能存在的碰撞问题,在设备身份验证中具有更高的可靠性.

尽管特权标识符能够提供精确的设备识别能力,但其安全性仍面临严重挑战. 特别是在设备 root 后,攻击者可轻易获取并滥用这些信息. 鉴于特权标识几乎不可更改,一旦被提取,攻击者可长期伪造设备身份,绕过应用的认证机制,进而实施恶意操作.

4.4　安全建议

本文提出的攻击方式要求目标应用可以在 root 设备上正常运行,并且攻击者能够完全掌握应用本地数据. 这导致攻击存在一定局限性,因为应用可以通过环境检测,强制要求使用 root 设备的用户在每次登录时进行身份验证. 然而,随着环境检测对抗手段的不断发展,这种保护机制也可能会被绕过. 因此,为了进一步增强应用的安全性,本文从登录凭证和设备标识两方面提出了安全建议,以提高应用抵御特权攻击者的能力.

4.4.1　登录凭证安全

防范针对登录设备数量管控机制的攻击的最佳方案是避免在本地存储登录凭证. 然而,此方法要求用户每次启动应用时均需重新验证身份,影响用户体验,因此仅适用于无需频繁交互的应用.

另一种可行方案是利用可信执行环境(如 TrustZone)对凭证数据进行加解密处理. 由于解密密钥存储于可信执行环境中,攻击者无法在另一台设备上同时复用该密钥与加密的凭证数据,从而避免登录凭证的复用. 相关研究 IMVisor[19] 和 TruApp[20] 探讨了 TrustZone 在保护应用敏感数据方面的可行性,而 Rubinov 等人的研究进一步验证了 TrustZone 对关键应用的强大保护能力[21],为可信执行环境在登录设备数量管控机制的安全性应用提供了参考.

4.4.2　设备标识安全

为了提高设备标识的安全性,本文提出以下几点建议,在确保其有效性的同时,提高攻击难度,从而提升系统的整体安全性.

1)多重属性联合标识. 单一设备标识易被伪造,建议结合多重设备属性进行标识. 具体来讲,可以将 Android ID、Drm ID 这种强标识性信息与系统配置、蓝牙和传感器等硬件

信息结合使用,这种组合方式使得攻击者难以同时伪造所有特征,提高了设备标识的抗篡改能力.

2) 多渠道采集设备标识. 本文建议采用多种方式来获取同一设备属性,并检验各采集结果的一致性,可有效检测部分采集渠道遭到篡改的情况. 这种方式增加了攻击者获取完整标识数据的难度,增强了设备标识的鲁棒性.

3) 设备状态检查. 本文建议通过分析传感器、电池等设备状态信息,辅助设备身份的验证. 这些状态信息的变化具有一定规律性,攻击者难以在短时间内精准模拟,并且模拟过程中设备状态可能已经发生变化,从而增加了伪造难度. 相比静态标识,基于动态信息的认证方式更为复杂,需要对设备状态信息的变化进行全面的计算与衡量,但同时它也提供了更强的抗攻击能力.

5 结 语

本文围绕安卓应用中的登录设备数量管控机制展开系统性实证分析,揭示了其在安全性方面的潜在风险. 通过构建分析系统并模拟攻击场景,本文对 208 款热门应用进行了评估与分析. 实验结果表明,这些应用的管控机制普遍存在可被绕过的安全隐患,尤其是基于设备标识的策略,在标识信息的采集深度和广度方面均存在局限性. 针对这些问题,本文提出了多项改进建议,以增强登录设备数量管控机制的鲁棒性.

References:

[1] Lam M S. Omlet: a revolution against big-brother social networks (invited talk) [C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014: 1-1, doi: 10.1145/2635868.2684426.

[2] Sherman M. An introduction to mobile payments: market drivers, applications, and inhibitors [C]//Proceedings of the 1st International Conference on Mobile Software Engineering and Systems, 2014: 71-74.

[3] Ruiz E, Avelar R, Wang X. Protecting remote controlling apps of smart-home-oriented iot devices [C]//Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, 2018: 212-213.

[4] Threat Hunter. Research report on the black and gray industries of the Internet in the first half of 2024 [EB/OL]. https://ww.threathunter.cn/reportDetail/56abbc1209f472c16a33cd19ac82519e, 2024.

[5] KANG N H. Research on identification technology of internet terminal devices [D]. Fuzhou: Fuzhou University, 2018.

[6] Zhou Z, Diao W, Liu X, et al. Acoustic fingerprinting revisited: generate stable device id stealthily with inaudible sound [C]//Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2014: 429-440.

[7] Bojinov H, Michalevsky Y, Nakibly G, et al. Mobile device identification via sensor fingerprinting [J]. arXiv preprint arXiv: 1408.1416, 2014.

[8] Kondracki B, Azad B A, Miramirkhani N, et al. The droid is in the details: environment-aware evasion of android sandboxes [C]//Proceedings of the 29th Network and Distributed System Security Symposium (NDSS), 2022, doi: 10.1145/3460120.3484765.

[9] Kertész A, Pflanzner T, Gyimóthy T. A mobile IoT device simulator for IoT-fog-cloud systems [J]. Journal of Grid Computing, 2019, 17: 529-551, doi: 10.1007/s10723-018-9468-9.

[10] Chen W, Xu L, Li G, et al. A lightweight virtualization solution for android devices [J]. IEEE Transactions on Computers, 2015, 64 (10): 2741-2751.

[11] Zhang L, Yang Z, He Y, et al. App in the middle: demystify application virtualization in android and its security threats [J]. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2019, 3 (1): 1-24.

[12] Wei H J, Lin L K, Lin C Y, et al. Measuring and optimizing the performance of the android virtualization framework [C]//Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, 2024: 1533-1535.

[13] Nyoni P, Velempini M. Privacy and user awareness on facebook [J]. South African Journal of Science, 2018, 114 (5-6): 1-5.

[14] Soumelidou A, Tsohou A. Towards the creation of a profile of the information privacy aware user through a systematic literature review of information privacy awareness [J]. Telematics and Informatics, 2021, 61: 101592, doi: 10.1016/j.tele.2021.101592.

[15] Dmrfcoder. Android data storage methods [EB/OL]. https://github.com/DmrfCoder/interview/blob/master/Android, 2025.

[16] Song W, Ming J, Jiang L, et al. App's auto-login function security testing via android os-level virtualization [C]//IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021: 1683-1694.

[17] Frida. A prevalent dynamic instrumentation toolkit [EB/OL]. https://frida.re/, 2025.

[18] Song W, Ming J, Jiang L, et al. Towards transparent and stealthy android os sandboxing via customizable container-based virtualization [C]//Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2021: 2858-2874.

[19] Tian C, Wang Y, Liu P, et al. Im-visor: a pre-ime guard to prevent ime apps from stealing sensitive keystrokes using trustzone [C]// 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2017: 145-156.

[20] Yalew S D, Mendonca P, Maguire G Q, et al. Truapp: a trustzone-based authenticity detection service for mobile apps [C]//IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2017: 1-9.

[21] Rubinov K, Rosculete L, Mitra T, et al. Automated partitioning of android applications for trusted execution environments [C]//Proceedings of the 38th International Conference on Software Engineering, 2016: 923-934.

附中文参考文献:

[4] 威胁猎人. 2024 年上半年黑灰产互联网黑灰产研究报告 [EB/OL]. https://ww.threathunter.cn/reportDetail/56abbc1209f472c16a33cd19ac82519e, 2024.

[5] 康年华. 互联网终端设备识别技术的研究 [D]. 福州: 福州大学, 2018.