软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(6):2047−2060 [doi: 10.13328/j.cnki.jos.006568]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Toast 重复绘制机制的口令攻击技术*

凌 振[1], 杨 彦[1], 刘睿钊[1], 张 悦[2], 贾 康[1], 杨 明[1]

[1](东南大学 计算机科学与工程学院, 江苏 南京 211189)

[2](暨南大学 网络空间安全学院, 广东 广州 510632)

通信作者: 杨明, E-mail: yangming2002@seu.edu.cn

摘 要: 移动终端在飞速发展的同时也带来了安全问题, 其中, 口令是用户信息的第一道安全防线, 因此针对用户口令的窃取攻击是主要的安全威胁之一. 利用 Android 系统中 Toast 机制设计的缺陷, 实现了一种基于 Toast 重复绘制机制的新型口令攻击. 通过分析 Android Toast 机制的实现原理和功能特点, 发现恶意应用可利用 Java 反射技术定制可获取用户点击事件的 Toast 钓鱼键盘. 虽然 Toast 会自动定时消亡, 但是由于 Toast 淡入淡出动画效果的设计缺陷, 恶意应用可优化 Toast 绘制策略, 通过重复绘制 Toast 钓鱼键盘使其长时间驻留并覆盖于系统键盘之上, 从而实现对用户屏幕输入的隐蔽劫持. 最后, 攻击者可以通过分析用户点击在 Toast 钓鱼键盘上的坐标信息, 结合实际键盘布局推测出用户输入的口令. 在移动终端上实现该攻击并进行了用户实验, 验证了该攻击的有效性、准确性和隐蔽性, 结果表明: 当口令长度为 8 时, 攻击成功率为 89%. 发现的口令漏洞已在 Android 最新版本中得到修复.

关键词: 口令攻击; Java 反射; Toast 重复绘制

中图法分类号: TP311

中文引用格式: 凌振, 杨彦, 刘睿钊, 张悦, 贾康, 杨明. 基于 Toast 重复绘制机制的口令攻击技术. 软件学报, 2022, 33(6): 2047–2060. http://www.jos.org.cn/1000-9825/6568.htm

英文引用格式: Ling Z, Yang Y, Liu RZ, Zhang Y, Jia K, Yang M. Repeating Toast Drawing Based Password Inference Attack Technique. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2047–2060 (in Chinese). http://www.jos.org.cn/1000-9825/6568.htm

Repeating Toast Drawing Based Password Inference Attack Technique

LING Zhen[1], YANG Yan[1], LIU Rui-Zhao[1], ZHANG Yue[2], JIA Kang[1], YANG Ming[1]

[1](School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

[2](College of Cyber Security, Jinan University, Guangzhou 510632, China)

Abstract: The mobile platform is rapidly emerging as one of the dominant computing paradigms of the last decades. However, there are also security issues that can work against mobile platforms. Being the first line of defense of various cyber attacks against mobiles, password protection serves an import role in protecting users' sensitive data. The offensive and defensive techniques related to passwords, therefore, gained a lot of attention. This work systematically studied the design flaws existing in the Android Toast mechanism and discovered a new type of vulnerability leveraging on Toast fade-in and fade-out animation, where malware can create a strategy of continuously displaying keyboard-like Toast views to capture the user's inputs stealthily, thereby stealing the user's password. The attackhas implemented, and extensive user experiments are performed to demonstrate its effectiveness, accuracy, and stealthiness. The results show that when the password length is 8, the attack success rate can reach up to 89%. It has also confirmed that the latest Android system has patched this vulnerability.

* 基金项目: 国家重点研发计划(2018YFB0803400); 国家自然科学基金(62022024, 61972088, 62072103, 62072098); 江苏省自然科学基金优青项目(BK20190060)

本文由"系统软件安全"专题特约编辑杨珉教授、张超副教授、宋富副教授、张源副教授推荐.

收稿时间: 2021-09-05; 修改时间: 2021-10-15; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

Key words: password attack; Java reflection; repeating Toast drawing

移动终端已经进入了大部分人日常生活的方方面面, 如社交、娱乐、购物、出行、移动办公、远程医疗等, 方便了人们的工作和生活, 促进了社会的进步. 根据 GSMA 实时智能数据分析[1], 截止到 2021 年 4 月, 全球总人口约为 78 亿, 约有 51 亿人拥有移动终端. 大部分移动终端基于 Android 系统, 该系统功能强大且易于使用, 给人们带来了很大的便利. 但 Android 系统在给人们带来便利的同时也带来了安全隐患.谷歌公司每年都会发现并修复数百个 Android 系统上的漏洞, 但仍有大量的零日漏洞存在, 这些漏洞可能会被恶意软件利用, 给社会带来巨大的经济损失. 根据 IDC 预测[2], 到 2023 年时, 全球安全软硬件问题导致的开支将达到 1 512 亿美元.

对移动终端进行的攻击有相当一部分是为了窃取用户的信息, 而获取用户的口令是一种有效的窃取用户信息的途径. 从用户输入口令到完成认证需要经过数个步骤: 用户与终端交互、应用处理、操作系统处理, 其中的每个步骤都可能被攻击. 例如: 在用户与终端交互时, 攻击者可借助各种传感器设备获取并利用人机交换过程中产生的光学、声学等侧信道信息, 以推断用户输入的口令[3−7]; 为了获得目标应用的用户口令, 攻击者会通过伪造感兴趣的目标应用[8,9]或篡改目标应用并注入恶意代码[10]等方式, 用户在此类应用中输入口令时, 口令会被窃取; 由于应用运行在系统中, 系统存在的设计漏洞[11−14]可能会导致口令被恶意应用截获. 其中, 针对人机交互过程中基于侧信道信息的口令猜测攻击技术需依赖于信号质量及口令猜测的成功概率, 攻击者在多个候选的口令集合中选择进行验证时, 也可能会触发登录限制, 攻击具有一定的局限性. 针对感兴趣目标应用的口令窃取攻击需要对具体目标应用定制攻击策略, 攻击的普适性较差. 而基于系统设计漏洞的口令攻击技术适用面较广, 威胁性更大, 受到研究人员的广泛关注.

本文通过对 Android Toast 机制的分析, 发现 Toast 在不需要任何权限的前提下, 可覆盖在其他应用的上层, 且不触发任何系统警告. 针对 Toast 的这种设计特性, 本文结合无障碍服务设计了一种基于 Toast 重复绘制机制的新型口令攻击, 实现快速、隐蔽地获取用户口令输入. 在检测到用户点击目标应用的口令输入框时, 我们的恶意程序将构造一个定制的 Toast 钓鱼键盘视图, 并通过不断重复绘制该 Toast 视图, 达到钓鱼键盘长时间驻留屏幕并覆盖真实键盘的效果. 通过对 Android 源码分析与用户实验测试, 优化绘制策略使用户无法察觉到钓鱼键盘, 提高了该攻击的隐蔽性. 在此基础上, 本文利用 Java 反射技术绕过 Android 系统对 Toast 类中受限成员变量的访问控制, 实现了对 Toast 视图属性的修改, 使恶意应用所创建的 Toast 能够拦截用户点击事件, 从而有效地获取用户口令输入. 本文针对 Android 系统版本 10 及之前的版本有效. 谷歌公司在 Android 系统版本 11 中修复了该漏洞, 使攻击者无法再通过 Java 反射技术设置 Toast 拦截用户点击事件. Wang 等人[15]综合利用 Toast 和 Overlay 可对 Android 系统版本 11 以及之后版本进行攻击, 和本文攻击机制相异.

本文的主要贡献总结如下:

(1) 发现了 Toast 机制设计上的安全漏洞. 针对 Android 系统 Toast 机制进行分析, 发现 Toast 绘制和销毁时存在弹出和淡出动画, 通过重复绘制的方法能够使 Toast 长时间驻留在屏幕上, 并覆盖在其他应用上层, 在此过程中不需要任何权限, 且不触发任何系统警告;

(2) 提出了新型的口令攻击技术. 分析 Android 源码并发现: 可利用 Java 反射技术修改受限变量, 使 Toast 可拦截用户点击事件. 在此基础上, 本文利用 Toast 重复绘制机制, 结合无障碍服务设计新型口令攻击技术, 成功地实现了在用户无法察觉的情况下对用户口令进行窃取;

(3) 在实际环境中进行攻击验证实验. 本文首先通过实际 Android 应用验证攻击的有效性; 然后, 设计用户实验对攻击的成功率和隐蔽性进行测试. 实验结果表明: 本文所设计的新型口令攻击具有较高的成功率, 即使口令长度为 12 位时, 攻击成功率仍有 84.5%. 除此之外, 攻击具有良好的隐蔽性.

本文第 1 节主要介绍 Toast 机制和无障碍服务. 第 2 节描述基于 Toast 重复绘制机制的口令攻击基本思想. 第 3 节介绍攻击的具体工作流程. 第 4 节对基于 Toast 重复绘制机制口令攻击的有效性、准确性和隐蔽性进行实验分析. 第 5 节介绍相关工作. 第 6 节是总结以及对未来工作的展望.

1 背 景

本节主要介绍 Android 系统的 Toast 机制和无障碍服务功能. 首先介绍 Toast 机制的概念和 Toast 的工作流程, 然后介绍无障碍服务的概念和功能.

1.1 Toast机制

Toast 是 Android 系统提供的弹出消息提示框, 以实现对用户的快捷反馈, 其弹出时可覆盖于其他应用上方, 并遮挡部分屏幕内容, 但不会捕获用户对屏幕的点击. Toast 工作流程如图 1 所示. 本文将 Toast 消息提示框简称为 Toast. 为了在屏幕上显示一个 Toast, 需先构建一个 Toast 对象, 并调用它的 *show*(·)方法. 当 Toast 的 *show*(·)方法被调用时, 应用首先获取一个 NotificationManager 对象, 并调用其 *enqueueToast*(·)方法, 将要显示的视图传递给消息管理器(NotificationManager). 最后, 通过跨进程通信, 将该视图传递给窗口管理器(WindowManager), 并弹出 Toast 向用户展示消息, Toast 会在显示 2 s 或 3.5 s 后消失.

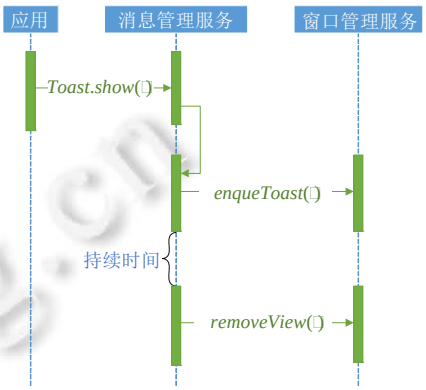


图 1 Toast 工作流程

Toast 除了能够显示文字提示外, 也可显示自定义视图.开发者通过 Toast 类静态方法 *Toast.makeText* (*context,text,duration*)来创建一个 Toast 对象. 设置参数 *text* 可设定 Toast 中需显示的内容, 设置参数 *duration* 可修改 Toast 在屏幕上的显示时间, 该参数只有两种取值: *LENGTH_LONG* 或者 *LENGTH_SHORT*, 分别对应的显示时间为 3.5 s 或 2 s. 图 2(a)是一个简单的例子, 通过编程在屏幕上弹出一个 Toast, 提示用户: “这是一个 Toast”. 此外, Toast 类也提供了 *setView*(·)方法可设置自定义的视图(view), 如图 2(b)所示.

(a) 普通 Toast (b) 自定义视图 Toast

图 2 弹出 Toast 示例

当 Toast 的显示持续时间达到设定的时间后, 消息管理器会调用 *removeView*(·)方法通知窗口管理器, 将

Toast 从屏幕上移除. 此时, 窗口管理器会启动一个持续时间为 500 ms 的淡出动画, 将 Toast 透明度逐渐降至 0. 在淡出动画持续的过程中, 透明度的降低速度由慢至快, Toast 随着动画逐渐退出屏幕. 在 *removeView*(·)方法被调用后, 即动画持续过程中, 若等待队列中存在下一个待显示的 Toast 对象, 则系统将显示新的 Toast, 新的 Toast 会直接覆盖于前一个正在执行淡出动画 Toast 的上层.

1.2 无障碍服务

无障碍服务是 Android 设计用来帮助残障人士使用 Android 设备的一种解决方案, 它可以帮助视觉、听觉、行动障碍人群方便地使用 Android 设备. Android 系统提供了若干个官方预装的无障碍服务应用, 如读屏应用 TalkBack, 用户使用 TalkBack 时无需注视屏幕, 仅需要使用手指触摸当前屏幕中显示的应用视图或文字即可听到语音反馈. 此外, 开发者可以使用 Android 提供的无障碍服务 API 创建和发布自定义的无障碍应用. 无障碍应用拥有比普通应用更高的权限, 而更高的权限往往意味着更有可能出现安全问题[12,16,17].

当无障碍服务被启用后, 用户和移动终端的交互过程会产生无障碍事件, 如点击事件、焦点变化事件等. 每个无障碍事件包含事件产生的时间、事件对应的应用、事件对应的控件节点 ID 等信息. 无障碍服务应用能够监听和处理系统中的无障碍事件, 通过无障碍事件对应的控件节点 ID, 应用可以获得无障碍事件所对应视图的信息, 如当无障碍事件发生于文本框时, 无障碍应用可获取当前文本框中的内容. 同时, 无障碍应用还可以通过控件节点对用户界面进行操作, 如设置文本框中的内容、模拟点击按钮等.

2 攻击概述

本节阐述 Android 口令攻击的基本思路. 首先介绍针对 Android 应用口令攻击方法的威胁模型, 然后分析 Toast 机制存在的设计缺陷, 在此基础上, 介绍基于 Toast 重复绘制口令攻击的基本思路.

2.1 威胁模型

本文口令攻击的威胁模型假设如下.

(1) 恶意应用被安装在用户的移动终端上, 并且用户为该应用开启无障碍服务权限. 攻击者可以将恶意应用伪装成游戏辅助应用、跳过广告应用等需要无障碍服务权限的应用, 提供这些应用具有的基本功能, 以此来诱导用户在应用市场下载恶意应用, 并为恶意应用开启无障碍服务功能. Diao 等人[18]对谷歌应用商店中 337 款需要无障碍服务权限的应用进行了调查, 发现超过半数以上(56.7%)的应用下载量超过 100 万. 而根据 Felt 等人[19]的调研, 只有约 17%的用户会关注应用申请的具体权限. Fratantonio 等人[16]在使用无障碍服务进行界面劫持攻击时也做了相同的假设;

(2) 攻击者可以预先获取被攻击应用的部分信息, 如被攻击应用的包名和输入口令界面上的文字. 应用信息可以通过预先分析被攻击应用的安装包来得到;

(3) 本口令攻击恶意应用是在用户登录被攻击应用时, 通过绘制钓鱼键盘以欺骗用户截获其口令输入, 因此本文假设被攻击应用在登录时用户需输入口令进行登录.

2.2 基本思路

通过对 Toast 机制的分析, 我们发现 Toast 存在以下特性.

(1) Toast 可以覆盖在其他应用的上层, 且不需要任何权限, 不会产生通知警告;

(2) Toast 在绘制时会执行弹出动画, 在销毁时会执行淡出动画. 这导致在连续绘制两个相同 Toast 时, 前一个还未完全淡出的 Toast 将被下一个新弹出的 Toast 迅速覆盖, 给用户造成同一个 Toast 长时间停留在屏幕上的假象;

(3) Toast 可展示自定义视图.

基于以上 Toast 的特性, 恶意应用可以通过多次重复绘制的方式使显示自定义视图的 Toast 持续覆盖于其他应用的上层. 此外, 无障碍服务也为恶意应用提供了监听用户输入口令时间的功能.

通过利用 Toast 设计上的特性, 本文提出了基于 Toast 重复绘制机制的口令攻击技术. 首先假设在被攻击

终端设备中已经安装了恶意应用, 当用户点击口令输入框后, 系统键盘会弹出以供用户输入口令. 此时, 恶意应用通过无障碍服务监听到用户的输入事件, 并重复绘制 Toast 展示自定义视图, 该视图布局和系统键盘布局相同且覆盖在系统键盘上, 并劫持用户的点击. 默认情况下, Toast 无法捕获用户的点击, 但通过对 Android 源码的分析, 我们发现使用 Java 反射技术设置 Toast 受限属性可使其具备拦截用户点击事件的功能. 根据用户点击的坐标可以推断出用户输入的字符. 由于用户的点击事件被拦截, 事件无法传递给底层真实的系统键盘, 因此用户的输入无法生效. 为使用户输入生效, 恶意应用可以使用无障碍服务将 Toast 截获的对应字符填入被攻击应用的输入框中, 令用户认为自己在操作真实系统键盘. 用户输入完成后, 恶意应用即可获得完整的用户口令.

3 基于 Toast 重复绘制机制的口令攻击

本节介绍基于 Toast 重复绘制机制的口令攻击, 首先描述口令攻击的总体流程, 然后介绍利用无障碍服务进行用户行为监听, 当监听到用户输入口令的事件后, 利用 Toast 重复绘制的技术进行钓鱼键盘的显示, 之后阐述如何利用反射获取用户在钓鱼键盘上的点击事件, 最后介绍通过用户点击位置对口令进行恢复.

3.1 攻击总体流程

如图 3 所示, 基于 Toast 重复绘制机制的口令攻击可以分为 4 个步骤, 分别为用户行为监听、钓鱼键盘显示、用户点击分析、口令恢复. 首先对用户行为进行监听, 当监听到用户的口令输入事件后, 恶意应用立即显示钓鱼键盘, 然后拦截用户的点击事件并进行分析. 当检测到用户输入结束时, 恶意应用对用户口令进行恢复, 从而完成整个口令攻击.

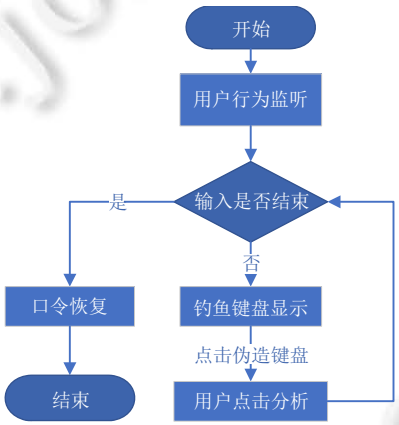


图 3 基于 Toast 重复绘制机制的口令攻击流程

接下来详细介绍口令攻击的这 4 个步骤.

3.2 用户行为监听

基于 Toast 重复绘制机制的口令攻击首先需要对用户的行为进行监听. 这需要将恶意应用安装在用户的移动终端上, 并为恶意应用开启无障碍服务权限. 获取无障碍服务权限后, 恶意应用就可以监听系统中所有应用产生的无障碍事件, 包括要攻击的应用.

在本文中, 恶意应用主要需要监听用户点击口令框开始输入的行为, 以此确定攻击开始的时机. 当用户点击口令框准备开始输入时, 系统会将用户点击事件发送给恶意应用, 事件中包括获得用户点击焦点的视图的类名以及该视图是否为口令框. 如果获得焦点的视图是口令框, 则视图类名为 EditText, 此时, 调用无障碍事件的 *isPassword*(·)方法会返回 true. 据此, 恶意应用可判断获得焦点的视图是否为口令框. 如果是口令框, 则开始显示钓鱼键盘, 即开始口令攻击.

3.3 钓鱼键盘显示

当监听到用户开始进行口令输入后，恶意应用开始显示钓鱼键盘. 在此过程中，恶意应用利用 Toast 重复绘制技术生成和系统键盘布局相同的钓鱼键盘，将其覆盖在系统键盘之上，并拦截用户的点击. 但使用 Toast 生成钓鱼键盘并拦截用户点击事件存在 3 个挑战: 第 1 个是需要使 Toast 能够捕获用户点击事件，第 2 个是如何使 Toast 长时间地停留并覆盖于系统键盘之上，第 3 个是如何根据不同用户的使用习惯切换钓鱼键盘的视图.

3.3.1 用户点击捕获

由于默认情况下 Toast 无法捕获用户的点击事件，因此需要通过反射来完成捕获用户点击事件的功能. 钓鱼键盘最重要的功能是捕获用户点击事件且不被用户察觉. 为达成不被用户察觉的目的，Toast 可以显示和系统键盘相同的视图，并将其覆盖在系统键盘之上. 但默认情况下, Toast 无法捕获点击事件，这时，当用户点击屏幕时，该点击会穿过 Toast 到达系统键盘上并被其捕获. 通过 Android 源码的分析我们得知: 显示 Toast 的视图是存放于 Toast 类成员变量 *mTn* 中的 *mNextView* 对象，但该视图的点击捕获属性存放于 *mParams* 对象中，被系统默认设置为 *FLAG_NOT_TOUCHABLE*[20]，因此该视图无法捕获点击事件，具体代码如代码片段 1 所示.

代码片段 1. Toast 初始化的部分代码.

```
1:  final WindowManager.LayoutParams params=mParams;
2:  params.type=WindowManager.LayoutParams.TYPE_TOAST;
3:  params.setTitle("Toast");
4:  params.flags=WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
5:          |WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE
6:          |WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE;
```

为了使 Toast 可以捕获点击事件，恶意应用可以在 Toast 初始化之后、调用 *show*(·)方法之前重新设置 *mNextView* 对象属性，即修改 *mParams* 对象的值，将 *FLAG_NOT_TOUCHABLE* 参数去除，之后，Toast 即可捕获点击事件. 但 *mTn* 对象和 *mParams* 对象在系统代码中均未被 public 修饰，恶意应用无法直接访问这些对象，并修改其属性. 为此，恶意应用可利用 Java 反射机制访问受限的 *mTn* 对象，然后通过 *mTn* 对象访问其中的 *mParams* 对象并修改其值，将 Toast 设置为可捕获用户点击，具体代码如代码片段 2 所示.

代码片段 2. 使用反射修改 Toast 属性.

```
1:  Toast toast=new Toast(MainActivity.this);
2:  toast.setView(keyboard);   //设置钓鱼键盘视图
3:  Field f=toast.getClass(·).getDeclaredField("mTN");
4:  f.setAccessible(true);   //去除 Java 对 Toast 类的 mTN 成员变量访问控制的检查
5:  Object mTn=f.get(toast);   //获取 toast 对象中的 mTN 对象
6:  Field f2=mTn.getClass(·).getDeclaredField("mParams");   //获取 mTn 对象的 TN 类的 mParams 成员变量
7:  f2.setAccessible(true);   //去除 Java 对 TN 类的 mParams 成员变量访问控制的检查
8:  Object mParams=f2.get(mTn);   //获取 mTn 对象中的 mParams 对象
9:  mParams.flags=WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE;   //设置为点击可捕获
10: toast.show(·);
```

3.3.2 基于 Toast 重复绘制的钓鱼键盘驻留

在口令攻击过程中，钓鱼键盘需要长时间驻留并覆盖于真实系统键盘之上. 一个 Toast 最长只能停留 3.5 s，因此我们使用重复绘制的方法保证钓鱼键盘长时间驻留在系统键盘之上. 当恶意应用检测到攻击完成后，可将当前钓鱼键盘设置为透明，造成键盘已经退出的假象. 接下来将重点介绍重复绘制技术的理论基础和实现方法.

在第 1.1 节中, 通过实验观察 Toast 存在显示过程的弹出动画和退出过程的淡出动画, 两个动画均是对 Toast 视图透明度变化的调整. 若重复绘制多个 Toast, 在第 1 个 Toast 缓慢淡出的过程中, 系统会同时将 Toast 等待队列中待显示的下一个 Toast 快速弹出, 相邻两个 Toast 在屏幕中存在互相重叠的部分, 用户很难察觉 Toast 提示框的重复绘制过程, 从而达到了使 Toast 长时间覆盖在真实系统键盘上方的目的.

Toast 重复绘制技术的具体实现流程如图 4 所示.

- 步骤 1. 恶意应用在监听到用户输入口令的无障碍事件后, 通过 *Toast.show*(·)的方法通知系统的通知管理器来创建一个 Toast;
- 步骤 2. 系统中的通知管理器在接收到恶意应用发起的创建 Toast 请求后会生成 token, 并将其通过 *enqueueToast*(·)方法放入 Toast 队列中, 该 token 将会标识唯一的 Toast;
- 步骤 3. 系统的通知管理器会从队列中获取一个 token, 并通知系统服务中的窗口管理器来将 Toast 绘制到屏幕上. 窗口管理器一次只会处理一个 token, 其他 Toast 所对应的 token 将会在队列中等待, 以此来保证系统不会同时显示多个 Toast[21];
- 步骤 4. 当 Toast 的持续时间达到设定值(即 2 s 或 3.5 s)时, 系统的通知管理器会调用 *removeView*(·)方法通知窗口管理器将 Toast 从屏幕上移除;
- 步骤 5. 窗口管理器接收到移除 Toast 的通知后, 开始执行淡出动画. 在此期间, 如果队列中仍然有未显示的 Toast 所对应的 token, 窗口管理器将同时执行下一个 Toast 的弹出动画.

恶意应用将以选定的时间 D 为间隔重复步骤 1–步骤 5, 以使得 Toast 长时间驻留在真实系统键盘上方.

在图 4 中, 前后两个 Toast 消失和出现之间存在一段时间间隔 T_a, 在这段时间内, 屏幕上不存在任何 Toast, 如果用户在此时点击屏幕, 则恶意应用无法拦截点击事件. 假设用户输入一次口令所用总时间为 T_s, Toast 在屏幕上时间为 T_d, 则切换次数可定义为 $T_s/(T_d+T_a)-1$. 用户输入捕获的概率为 Toast 在屏幕中显示时长与相邻两个 Toast 弹出时间间隔的占比, 即 $T_d/(T_d+T_a)$. 由此可以分析出: 当 T_d 更大时, 切换次数更少, 捕获概率更高. 这在第 4.3.2 节的实验中得到验证, 所以攻击时, Toast 的持续时间应设置为 3.5 s.

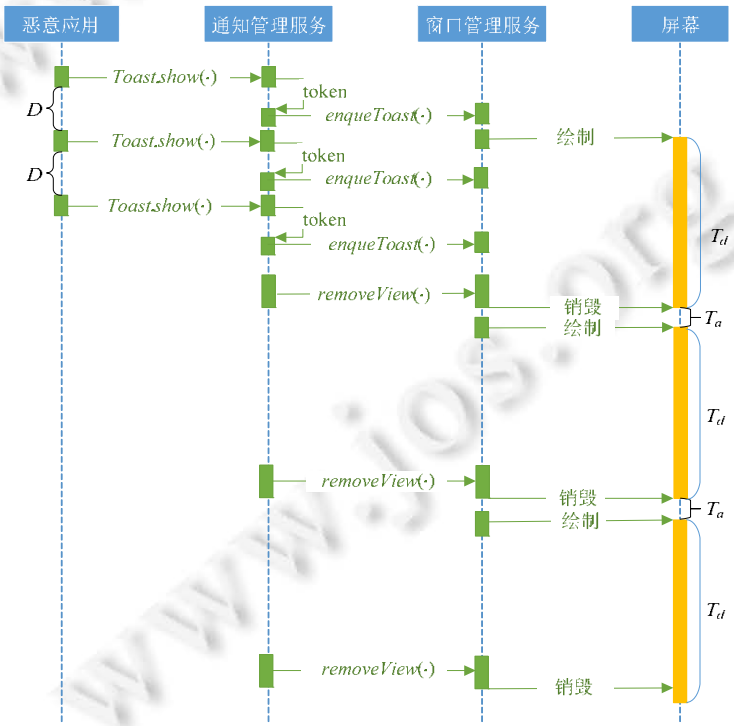


图 4 Toast 重复绘制流程图

在 Toast 重复绘制的过程中，恶意应用需要选择合适的时间间隔 D 和 Toast 创建策略，以提高 Toast 长时间驻留的隐蔽性. 由于 Android 系统会将所有当前待显示 Toast 对应的 token 放入一个队列中逐个显示，但同一时间只能有一个 Toast 显示在屏幕上，因此如果时间间隔 D 过短，恶意应用就会在队列中短时间内添加过多的 token，即使攻击结束后，用于显示钓鱼键盘的 Toast 仍会不停地出现，攻击就很可能会被用户察觉. 相反，如果 D 过长，两个 Toast 的显示间隔时间过长，后一个 Toast 将无法在前一个 Toast 完全消失之前显示在屏幕上，容易被用户察觉. 因此，需要将时间间隔 D 设为一个稍小于 Toast 持续时间的值，通过设置不同的 D 值进行攻击尝试，我们最终发现将 D 设置为 3.4 s 时攻击效果最好.

3.3.3 键盘视图切换

钓鱼键盘显示需要考虑不同输入类型的切换. 用户的口令可能不仅包括小写字母，而且包括大写字母、数字等字符. 因此，恶意应用需要根据用户的点击将 Toast 所显示的钓鱼键盘的样式进行替换，以免攻击被用户察觉. 大多数输入法应用中，输入口令时，小写字母与数字位于同一个类型的视图中，如图 5 左侧所示，因此无需额外考虑输入数字的情况. 当用户输入大写字母时，键盘视图如图 5 右侧所示. 在用户点击大小写切换键后，恶意应用需要将钓鱼键盘从小写切换到大写. 在 Toast 重复绘制的过程中，恶意应用可以使用 *setImageBitmap*(·)方法修改 Toast 中显示的内容，由此可以完成不同输入类型的切换.

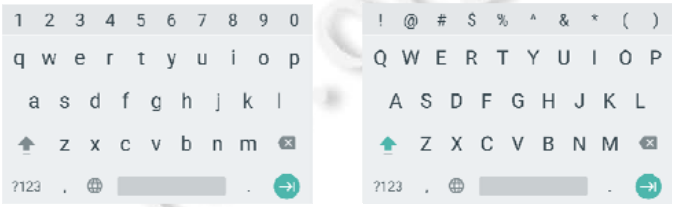


图 5 两种键盘视图

除此之外，部分用户不使用系统默认的输入法，而是使用第三方输入法，这就要求恶意应用需要将钓鱼键盘的视图显示为对应输入法的视图，以免攻击被用户察觉. 通过调用系统接口[22]可以获取用户当前使用的输入法，攻击者将用户常用输入法的键盘视图预置在恶意应用中，在攻击开始时，恶意应用根据当前输入法获取对应的键盘视图并加以显示. 由此，大部分情况下，恶意应用都会显示出与用户真实键盘相同的钓鱼键盘. 如果用户使用小众输入法，恶意应用没有预置该输入法对应的键盘视图，则选择该机型默认输入法的键盘视图. 如 Pixel 机型出厂时默认的输入法是 Gboard，如果用户购买设备后将默认输入法设置为小众输入法，攻击时，恶意应用将生成 Gboard 所对应的键盘覆盖在小众输入法上. 部分应用为了口令输入安全使用动态布局键盘，对此需要对相应的应用进行提前分析，了解其动态布局键盘的视图，之后对键盘视图进行伪造，并将其覆盖在真实键盘之上.

3.4 用户点击处理

钓鱼键盘显示完成并捕获用户点击后，需要根据用户的点击进行处理，以识别出用户输入的字符. 攻击时，钓鱼键盘所捕获的点击事件中，有效信息为用户点击屏幕的坐标，恶意应用需要根据用户点击的坐标推断出用户输入的字符. 以 Google 官方输入法 Android Keyboard 为例，输入口令时，其键盘布局为标准的 QWERTY 键盘与数字 0–9 的结合. 通过阅读源码[23]以及实际测量可知：当屏幕宽度为 X 像素时，所有按键宽度与高度见表 1，由此可计算出每个按键的中心坐标. 当用户点击事件发生时，首先计算每个按键中心坐标与用户点击坐标的欧拉距离，然后选取欧拉距离最小值所对应的按键为用户当前按键.

表 1 键盘内字符的宽高

按键	宽(像素)	高(像素)
小写字母、逗号、句号	0.1×X	0.13×X
DELETE, SHIFT	0.15×X	0.13×X
空格	0.5×X	0.13×X

在获取到用户输入字符后, 恶意应用在被攻击应用的口令框中需填入相应的用户输入, 以免用户察觉攻击行为. 由于用户的点击事件被钓鱼键盘拦截, 无法传递到下层的真实系统键盘, 因此用户点击的字符不会被添加到口令框中, 很可能被用户察觉攻击行为. 恶意应用在识别出用户点击的字符后, 可利用无障碍服务提供的 *performAction*(·)方法将用户输入填充至口令框, 从而使用户难以察觉攻击行为.

3.5 口令恢复

在监听到用户点击完成按钮后, 将此前获取的输入进行拼接恢复出用户口令. 由于在捕获用户点击时, 已经根据用户点击坐标识别出用户点击输入的每一个字符, 因此将得到的字符拼接就可以得到用户的口令. 但口令恢复得到的结果并不一定完全准确, 因为两个 Toast 切换时会存在短暂的间隙, 导致屏幕上不存在任何 Toast. 如果用户在此时进行输入, 则恶意应用无法拦截到用户点击, 也就无法获取用户输入的字符, 因此恢复的口令中可能会遗漏用户的输入, 在第 4.3.2 节中将进行实验测试口令恢复的成功率.

攻击视频演示可见 https://reurl.cc/WEzEM7. 在视频中, 我们使用 Google Pixel 2 机型进行演示, 首先为恶意应用开启无障碍服务权限, 然后切换到被攻击应用进行口令输入操作, 此时, 恶意应用运行在系统后台. 在用户开始进行口令输入时, 所弹出的键盘为恶意应用通过 Toast 伪造的钓鱼键盘. 当用户在被攻击应用的口令框中输入完毕后, 恶意应用会通过 Toast 将获取到的口令内容进行展示.

4 实验验证

本节将在实际环境中评估基于 Toast 重复绘制机制的口令攻击技术. 首先, 在多款实际应用中验证攻击的有效性; 然后设计用户实验测试攻击的准确性, 获取口令恢复成功率等数据; 最后, 通过用户实验来测试攻击的隐蔽性.

4.1 实验环境

为验证攻击的有效性、准确性和隐蔽性, 本节根据以上技术开发恶意应用并进行实验. 将所开发的恶意应用安装到移动终端并启动后, 攻击程序会在后台监听无障碍事件, 并在用户开始输入口令时使用 Toast 拦截用户输入, 分析点击坐标, 并将所获口令记录到本地日志文件中.

在攻击有效性分析的实验中, 使用的终端型号为 Google nexus6p, Android 系统版本 8, 选取的被攻击应用为谷歌应用市场中较为流行的 5 款应用, 分别为 Skype、脸书、印象笔记、推特、Instagram, 以及两款下载量较大的金融类应用, 分别为美国银行和支付宝. 攻击隐蔽性分析的实验中, 所选用的应用为美国银行应用. 在攻击准确性分析的实验中, 采用的测试应用为本文定制开发的应用, 因为需要将恶意应用得到的口令与用户真实输入的口令进行对比, 由此判断攻击是否成功.

攻击准确性分析和隐蔽性分析实验均为用户实验. 用户实验选取了 20 名志愿者进行实验, 其中包括 3 名女性和 17 名男性, 志愿者年龄在 22–33 岁之间, 平均年龄 25 岁, 其中, 20 名志愿者实验使用的终端型号和操作系统版本见表 2.

表 2 实验设备列表

厂商	型号	系统版本
Samsung	s8	8
Samsung	SMG9	9
Google	nexus6p	8
Google	pixel 2xl, pixel 4	9
Vivo	v1813A, x21iA, v1816A, v1813BA	9
Oppo	PMEM00	9
Xiaomi	mi5	8
Xiaomi	mix 2s, mi6, mi8	9
Xiaomi	mix3	10
Huawei	EML-AL00, mate20	9
Huawei	nova3	9.1
Huawei	mate20 x, HMA	10

4.2 攻击有效性分析

本实验分别测试攻击在不同应用中的表现. 测试者预先将上文提到的 7 款测试应用安装到移动终端上, 安装恶意应用并为其开启无障碍服务权限. 然后分别打开这7款测试应用, 操作应用跳转至输入口令界面, 在该界面输入长度为 10 的口令, 并点击登录按钮. 待 7 款应用全部测试完成后, 从测试终端的本地存储中取出日志文件进行分析. 实验结果见表 3.

表 3 测试应用列表

应用名	包名	版本	能否攻击
Skype	com.skype.raider	8.45.0.43	能
脸书	com.facebook.katana	196.0.0.16.95	能
印象笔记	com.evernote	8.4.1	能
推特	com.twitter.android	7.68.1	能
Instgram	com.instagram.android	69.0.0.10.95	能
美国银行	com.infonow.bofa	8.1.16	能
支付宝	com.eg.android.AlipayGphone	10.1.65	需要特殊处理

经过分析日志文件发现: 除了需要对支付宝应用定制用户行为监听方法外, 恶意应用对 7 款应用均可获取用户输入的口令. 在采用默认的恶意软件攻击时, 我们发现用户在支付宝中输入口令时没有记录到任何输入. 其原因是恶意程序通过检测到口令框, 并获取到焦点事件后, 开始显示 Toast 进行攻击, 但支付宝的口令框在获取焦点时并不会发出任何事件, 导致恶意程序无法检测到攻击发起时机. 对此情况虽然无法直接检测到用户开始输入口令的动作, 但是通常用户在输入口令前需要输入账号. 在用户输入账号后, 会点击口令框, 此时账号框失去焦点, 口令框获得焦点. 恶意应用虽然无法检测到口令框获得焦点的事件, 但是可以检测到账号框失去焦点的事件. 因此可以在账号框失去焦点后, 恶意程序检测当前界面的根节点是否发生变化, 进而推断出用户是否仍在输入口令的界面, 如果用户仍然停留在输入口令界面, 则可认为用户开始向口令框中输入口令, 此时可以开始显示 Toast 实施攻击.

由该实验可以得知: 虽然在小部分情况下需要针对特定的应用设计特定的攻击策略, 但是基于 Toast 重复绘制机制的口令攻击技术具有较好的有效性.

4.3 攻击准确性分析

本文介绍的口令攻击主要是通过捕获用户的键盘输入来实现, 所以在攻击中, 对用户点击事件的捕获率决定着攻击的成功率. 由前文可知, 攻击中生成的单个 Toast 持续时间(2 s 或 3.5 s)对用户点击事件捕获率存在影响. 因此在攻击准确性分析实验中主要包含两个实验, 分别测试在两种 Toast 持续时间下用户点击事件的捕获率和在最佳的 Toast 持续时间下不同长度口令恢复的成功率.

4.3.1 点击事件捕获率

在本实验中, 分别设置恶意应用构建的单个 Toast 持续时间为 2 s (*LENGTH_SHORT*)和 3.5 s (*LENGTH_LONG*)来测试攻击中点击事件的捕获率. 志愿者将会在两种条件下分别向测试应用的输入框中输入 100 个字符来测试其中的点击事件捕获率. 在实验中, 统计 Toast 捕获到点击事件的次数为 C_t, 没有捕获到点击事件的次数为 C_p, 则可得捕获率为 $C_t/(C_t+C_p)$.

表 4 中的数据显示: 对每个移动终端, Toast 持续时间越长, 触摸事件的捕获率越高. 由实验数据可知: 由于硬件和系统版本不同, 不同机型的捕获率会有所区分, 但 Toast 持续时间为 3.5 s 时, 一般会比 Toast 持续时间为 2 s 的捕获率更高, 这也与前文的理论相符合.

表 4 Toast 持续时间和事件捕获率

型号	捕获率 2 s (%)	捕获率 3.5 s (%)
mix3	0	0
mix 2s	98	98
mi5	99	99
mi8	93	97

表 4 Toast 持续时间和事件捕获率(续)

型号	捕获率 2 s (%)	捕获率 3.5 s (%)
mi6	93	97
s8	0	0
SMG9	0	0
nova3	0	0
EML-AL00	96	99
nexus6p	98	99
mate20	98	100
HMA	0	0
mate20 x	0	0
pixel 2x1	85	93
pixel 4	96	99
v1813A	0	0
x21iA	0	0
v1816A	99	99
v1813BA	100	100
PMEM00	0	0

4.3.2 口令恢复成功率

在本实验中, 设置恶意应用构建的单个 Toast 持续时间为 3.5 s, 每个志愿者在被攻击应用中分别输入长度为 4、6、8、10、12 的口令各 10 个, 口令由大写字母、小写字母、数字组成. 实验结果如图 6 所示. 可以看出: 随着口令长度的增加, 攻击成功率逐渐降低. 这是因为部分情况下, 用户输入时 Toast 恰好消失, 未能捕获用户输入. 但即使在口令长度为 12 位时, 攻击成功率仍有 84.5%. 这说明该攻击具有较好的准确性.

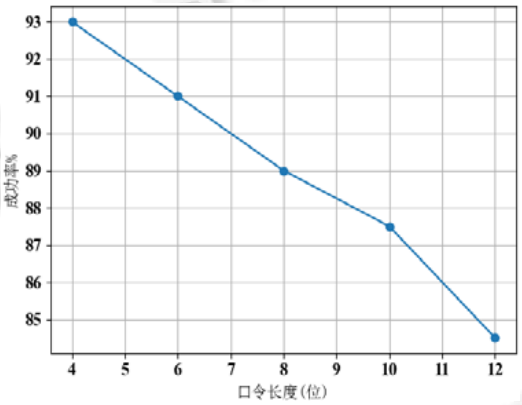


图 6 口令长度-攻击成功率关系

4.4 攻击隐蔽性分析

我们进行了两组对照实验以测试攻击的隐蔽性. 在实验中, 志愿者将会在终端上打开美国银行应用, 输入所给口令, 最后点击登录按钮. 我们选择了两种场景进行测试: 一种为正常使用场景, 另一种为系统后台运行恶意应用的场景. 当志愿者在两种场景下完成操作后, 我们对其进行访谈, 询问是否观察到两种场景下的操作存在差异, 只有一名志愿者反映在操作时终端出现反应滞后的情况. 除此之外, 没有人观察到可疑情况. 由此可知: 该攻击具有良好的隐蔽性, 可应用于真实场景中.

5 相关工作

移动终端的用户在进行口令认证时需要数个步骤, 用户一般需要在触摸屏的虚拟键盘上进行点击, 按键所对应的字符会被输入到应用的口令框中, 应用由此可以得知用户要输入的口令. 由于应用运行在操作系统上, 操作系统同样可以得知用户的口令. 在用户输入口令的过程中, 攻击者可以通过以下几种方式进行口令攻击.

(1) 攻击用户与移动终端的交互. Maggi 等人[3]首次提出触摸屏的虚拟键盘被拍摄后可能导致口令泄露, 他们设计的程序可以自动提取视频中包含用户点击屏幕的视频帧, 并通过图像识别算法可以识别出这次点击所对应的字符, 由此可以恢复用户输入的口令. Yue 等人[4]假设攻击者可以拍摄用户的输入过程, 在看不到屏幕上任何信息的情况下, 设计多种计算机视觉技术自动地定位用户在屏幕上的点击位置, 接着使用平面映射将屏幕点击位置转换为虚拟键盘的键值, 以获取用户口令. Xu 等人[5]提出即使远距离拍摄或光源经过反射, 仍可从视频信息中分析出用户的口令. Zhou 等人[6]使用用户解锁设备时, 手指产生的声学信号来推测用户的图形口令. Aviv 等人[7]指出: 用户在输入图形口令后屏幕上会留有污渍, 根据这些污渍可以推断出用户的图形口令. 攻击用户与移动终端的交互大多依靠侧信道的方式来实现, 这一般要求攻击者在物理上接近被攻击者, 这是一个比较强的攻击假设. 并且, 侧信道攻击得到的一般不是准确的口令, 而是多个候选口令, 需要经过多次尝试才能得到准确的口令;

(2) 攻击目标应用. Jung 等人[8]指出: 当用户在伪造的应用中输入口令时, 口令可能会被窃取. 他们首先分析数个银行应用的安装包, 绕过其签名及完整性验证, 然后修改其字节码以添加恶意代码, 最后进行重新打包和签名, 以实现对目标应用的恶意篡改. Aonzo 等人[9]使用应用包名来欺骗口令管理器, 误导用户将口令输入到伪造的应用中. Luman 等人[10]提出在 Virtual App 等应用容器中运行应用时, 用户的口令可能会被窃取, 他们通过分析应用市场中的应用后发现, 已经有部分恶意容器上架到应用市场, 在运行时会窃取用户的口令. Ren 等人[24]提出了 Android 多任务机制存在漏洞, 恶意应用可以通过任务劫持的方式发起伪装欺骗攻击. Wang 等人[17]设计了 ActivityHijacker, 使用 Activity 劫持攻击, 并设计机制规避通知栏通知和近期任务的显示, 以提高攻击的隐蔽性. 攻击目标应用一般需要对目标应用进行预先分析, 然后通过伪造应用或是在应用中注入恶意代码的方式来得到用户的口令. 目前, Android 应用反调试及反篡改技术飞速发展, 对这种攻击方式形成了新的挑战;

(3) 攻击操作系统. Niemietz 等人[12]提出了包括传统 Toast 攻击在内的多种针对 Android UI 的攻击, 大多数漏洞已被修复. Kraunelis 等人[13]首次指出, Android 无障碍服务框架可能被用于伪造攻击. Kalysch 等人[14]发现, Android 无障碍服务可能导致口令泄露. 他们发现: 虽然无法直接得到用户口令, 但用户输入的当前一位短时间内会被无障碍服务获取, 由此可以推断出完整的口令. Fratantonio 等人[16]使用无障碍服务和 Overlay 来进行点击劫持, 通过同时弹出多个 Overlay 覆盖于键盘上, 来获取用户输入行为, 从而窃取用户口令. 攻击操作系统一般利用操作系统中的安全问题来设计攻击. 分析操作系统中的安全问题需要较高的技术门槛, 并且可能需要一些系统权限. 目前, Android 系统对权限的申请越来越严格, 为实现这些攻击带来新的挑战.

6 总结与展望

本文主要研究了 Android 系统 Toast 机制在设计逻辑和实现中存在的安全问题, 并基于该安全问题设计口令攻击. 通过重复绘制恶意 Toast, 使其长时间保持在前景并覆盖在其他应用之上. 同时, 使用 Java 反射技术设置 Toast 对象相关成员变量, 使恶意 Toast 可以拦截用户点击, 进而显示钓鱼键盘劫持用户输入. 最后, 配合无障碍服务完成整个口令攻击. 在攻击设计完成后, 本文设计实验在实际环境中对所设计攻击进行评估, 实验结果表明, 本文提出的攻击技术具有较好的有效性、准确性和隐蔽性.

本文在基于 Toast 重复绘制机制的口令攻击中, 使用 Toast 来显示钓鱼键盘拦截用户的输入, 在显示钓鱼键盘时, 根据用户当前输入法生成对应的键盘. 虽然能够知道用户使用的输入法类型, 但是无法得知用户习惯使用的键盘布局为 9 键还是 26 键, 生成的键盘可能并不是用户平时使用的键位键盘, 即使在这之后可以模拟键盘的操作, 进行相关的切换, 但是警觉的用户可能察觉到配置发生了变换; 当用户自定义键盘皮肤时, 伪造的键盘会与用户设置的皮肤不一致, 这可能导致攻击被用户发觉. 在未来的工作中, 可以结合用户的输入法配置文件读取输入法各项设置, 进而生成与用户自定义键盘相同的键盘, 提高攻击的隐蔽性. 且在攻击

中使用了无障碍服务来监听用户点击口令框的事件, 这是一个较强的攻击模型, 无障碍服务所需要的权限与其他权限不同, 需要用户手动授予, 这增加了攻击实施的难度. 在未来的工作中, 可能考虑以其他方式监听用户点击口令框事件, 以减少攻击所需要的权限.

References:

[1] GSMA. Representing the worldwide mobile communications industry. 2021. http://www.gsma.com

[2] IDC. Solid growth ahead for security products and services. 2019. https://www.idc.com/getdoc.jspcontainerId=prUS45591619

[3] Maggi F, Volpatto A, Gasparini S, Boracchi G, Zanero S. A fast eavesdropping attack against touchscreens. In: Proc. of the 7th Int'l Conf. on Information Assurance and Security (IAS). IEEE, 2011. 320−325.

[4] Yue Q, Ling Z, Fu X, Liu B, Ren K, Zhao W. Blind recognition of touched keys on mobile devices. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security. 2014. 1403−1414.

[5] Xu Y, Heinly J, White AM, Monrose F, Frahm JM. Seeing double: Reconstructing obscured typed input from repeated compromising reflections. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. 2013. 1063−1074.

[6] Zhou M, Wang Q, Yang J, Li Q, Xiao F, Wang Z, Chen X. Patternlistener: Cracking Android pattern lock using acoustic signals. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. 2018. 1775−1787.

[7] Aviv AJ, Gibson KL, Mossop E, Blaze M, Smith JM. Smudge attacks on smartphone touch screens. Woot, 2010, 10: 1−7.

[8] Jung J H, Kim JY, Lee HC, Yi JH. Repackaging attack on Android banking applications and its countermeasures. Wireless Personal Communications, 2013, 73(4): 1421−1437.

[9] Aonzo S, Merlo A, Tavella G, Fratantonio Y. Phishing attacks on modern Android. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. 2018. 1788−1801.

[10] Shi L, Fu J, Guo Z, Ming J. "Jekyll and Hyde" is risky: Shared-everything threat mitigation in dual-instance apps. In: Proc. of the 17th Annual Int'l Conf. on Mobile Systems, Applications, and Services. 2019. 222−235.

[11] Lee J, Chen A, Wallach DS. Total recall: Persistence of passwords in Android. In: Proc. of the NDSS. 2019.

[12] Niemietz M, Schwenk J. UI redressing attacks on Android devices. In: Proc. of the Black Hat Abu Dhabi. Abu Dhabi, 2012. 1−7.

[13] Kraunelis J, Chen Y, Ling Z, Fu X, Zhao W. On malware leveraging the Android accessibility framework. In: Proc. of the Int'l Conf. on Mobile and Ubiquitous Systems: Computing, Networking, and Services. Cham: Springer, 2013. 512−523.

[14] Kalysch A, Bove D, Müller T. How Android's UI security is undermined by accessibility. In: Proc. of the 2nd Reversing and Offensive-oriented Trends Symp. 2018. 1−10.

[15] Wang S, Ling Z, Zhang Y, *et al*. Implication of animation on Android security. In: Proc. of the 42nd IEEE Int'l Conf. on Distributed Computing Systems (ICDCS). IEEE, 2022.

[16] Fratantonio Y, Qian C, Chung SP, Lee W. Cloak and dagger: From two permissions to complete control of the UI feedback loop. In: Proc. of the 2017 IEEE Symp. on Security and Privacy (SP). IEEE, 2017. 1041−1057.

[17] Wang Z, Li C, Guan Y, Xue Y, Dong Y. Activity hijacker: Hijacking the Android activity component for sensitive data. In: Proc. of the 25th Int'l Conf. on Computer Communication and Networks (ICCCN). IEEE, 2016. 1−9.

[18] Diao W, Zhang Y, Zhang L, Li Z, Xu F, Pan X, Liu X, Weng J, Zhang K, Wang X. Kindness is a risky business: On the usage of the accessibility APIs in Android. In: Proc. of the 22nd Int'l Symp. on Research in Attacks, Intrusions and Defenses ({RAID} 2019). 2019. 261−275.

[19] Felt AP, Ha E, Egelman S, Haney A, Chin E, Wagner D. Android permissions: User attention, comprehension, and behavior. In: Proc. of the 8th Symp. on Usable Privacy and Security. 2012. 1−14.

[20] Android Developer. Windowmanager.Layoutparams. 2020. https://developer.android.com/reference/android/view/WindowManager.LayoutParams

[21] Lim B. Android tapjacking vulnerability. arXiv preprint arXiv: 1507.08694, 2015.

[22] Android Developer. Default input method. 2021. https://developer.android.com/reference/android/provider/Settings.Secure#DEFAULT_INPUT_METHOD

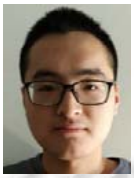
[23] Android Open Source Project. Latin IME source code. 2020. https://android.googlesource.com/platform/packages/inputmethods/LatinIME

[24] Ren C, Zhang Y, Xue H, Wei T, Liu P. Towards discovering and understanding task hijacking in Android. In: Proc. of the 24th USENIX Security Symp. (USENIX Security 2015). 2015. 945–959.

凌振(1982－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为匿名网络, 移动终端安全, 物联网安全, 可信计算.

张悦(1989－), 男, 博士, 主要研究领域为 IoT 安全.

杨彦(1998－), 男, 硕士生, CCF 学生会员, 主要研究领域为系统安全.



贾康(1995－), 男, 硕士, 主要研究领域为安卓系统安全.

刘睿钊(1998－), 男, 硕士生, 主要研究领域为安卓系统安全.

杨明(1979－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为网络安全, 隐私保护.