

基于差分隐私的 Android 物理传感器侧信道防御方法

唐奔宵 王丽娜 汪润 赵磊 王丹磊

(空天信息安全与可信计算教育部重点实验室(武汉大学) 武汉 430072)

(武汉大学国家网络安全学院 武汉 430072)

(tangbenxiao@whu.edu.cn)

A Defensive Method Against Android Physical Sensor-Based Side-Channel Attack Based on Differential Privacy

Tang Benxiao, Wang Lina, Wang Run, Zhao Lei, and Wang Danlei

(Key Laboratory of Aerospace Information Security and Trusted Computing (Wuhan University), Ministry of Education, Wuhan 430072)

(School of Cyber Science and Engineering, Wuhan University, Wuhan 430072)

Abstract The defensive research against Android physical sensor-based side-channel attacks mainly aims at the privacy leak which leverage mobile sensors as medium. The current defensive methods are malicious activity detection, virtual keyboards randomization, etc. However, these traditional methods can hardly protect user's privacy from sensor-based side-channel attacks fundamentally, for the unpredictable user decision and variety of novel attacks. In order to overcome the above problems, this paper presents a defensive method against physical sensor-based side-channel attacks based on differential privacy. This defensive method interferes the process of side-channel construction by injecting random noise coincident with the Laplace distribution which can obfuscate the original sensor data. The primary challenge of the proposal method is reducing the success rate of side-channel attacks as much as possible on the premise that ensuring normal operation of the sensor-based function and user experience. Taking the advantages of a sensor-based function extraction tool SensorTainter we designed, the sensor-based functions are analyzed detailedly and classified according to the types of based sensors and algorithms, thus we estimate the ranges of sensor data obfuscation for each category of sensor-based function. By analyzing 47 144 apps and 9 typical sensor-based side-channel attacks, the experiment proves that our defensive method can effectively defense against sensor-based attacks, which results in an accuracy decrease of 27 percent points at most in one attempt during key-event side-channel attacks and about 7 percent points in tracking side-channel attacks. Because of implementing in Android framework, this defensive method is completely user transparent and has great expansibility.

Key words Android; physical sensor; side channel; differential privacy; privacy protection

收稿日期:2017-12-29;修回日期:2018-05-02

基金项目:国家自然科学基金重点项目(U1536204);国家自然科学基金项目(61672394,61672393)

This work was supported by the Key Program of the National Natural Science Foundation of China (U1536204) and the National Natural Science Foundation of China (61672394, 61672393).

通信作者:王丽娜(lnwang@whu.edu.cn)

摘要 Android 物理传感器侧信道防御研究主要针对以移动设备传感器为媒介的隐私泄露攻击。当前的防御方案主要为预防检测、虚拟键盘随机化等。然而,防御过程中不可控的用户决策以及层出不穷的新型侧信道攻击,导致传统方案无法从根本上解决基于物理传感器的隐私泄露威胁。针对上述问题,提出了一种基于差分隐私的 Android 物理传感器侧信道防御方法。通过注入少量的特殊分布噪声,混淆传感器原始数据,进而干扰侧信道构建过程。如何在保证传感器相关功能正常运行与用户体验的前提下,尽可能降低侧信道攻击成功率是面临的最大困难。通过设计并实现传感器相关功能抽取工具 SensorTainter,对 APP 中传感器相关功能进行分析与分类,计算相关功能正常运行时能够承受的传感器数据混淆范围。依据对 47144 个 APP 以及典型传感器侧信道攻击的实验分析结果,证明该防御方案能够有效限制传感器侧信道攻击,单次点击事件攻击的准确率最高减少 27 个百分点。由于在 Android 应用框架层构建,该防御方案对于用户完全透明,具有很好的扩展能力。

关键词 Android; 物理传感器; 侧信道; 差分隐私; 隐私保护

中图法分类号 TP309; TP334

物理传感器已经成为智能手机中不可缺少的辅助硬件之一,传感器的应用为智能手机用户提供了更加丰富的功能。然而,用户在使用移动设备的过程中产生的行为特征可被物理传感器采集,并被利用于构建侧信道进行隐私泄露攻击^[1],攻击者可以轻易地对用户进行追踪^[2-3]或者窃取设备密码^[4-5]。近年来,移动设备的普及率持续增长,根据媒体市场研究公司 Zenith 的最新研究显示,全球智能手机的普及率在 2017 年已经达到 63%,其中 Android 系统市场占有率为 86.1%^[6],越来越多的用户暴露在隐私泄露的风险当中。

传感器隐私保护方面的研究已经具有一定积累^[7-8],但是针对以传感器作为媒介的隐私泄露攻击缺乏有效的防御机制。首先,Android 系统中物理传感器默认为低风险数据源,应用程序不需要任何申明即可直接访问传感器,因此基于敏感权限的恶意行为检测方案^[9-11]难以适用于传感器侧信道攻击。其次,传感器侧信道攻击方法在不断改进,尤其是结合了共谋攻击的新型侧信道^[12]让传统检测方案捉襟见肘。已有的传感器侧信道防御方案中,最直接的方法是降低传感器的采样频率,甚至禁止 APP 访问传感器^[13-14],然而,由于难以区分数据访问的恶意性,通过限制传感器性能的解决方案会严重影响 APP 的正常运行。随后,更加灵活的访问控制机制被提出^[15-16],但是访问控制无法防御伪装成合法 APP 的恶意攻击。键盘随机化机制^[17]可以破坏传感器数据与用户输入行为之间的映射关系,但对于以追踪为目的的侧信道攻击毫无办法,此外,修改用户已经非常熟悉的键盘布局对于用户体验而言也十分

不友好。文献[18]提出,通过嗅探程序向用户敏感操作时产生的传感器数据中注入大量噪声破坏传感器数据可用性,进而实现对侧信道的防御。然而,注入行为本身破坏了 Android 的沙盒机制,容易被攻击者利用,防御程序也无法有效地识别用户何时进行敏感操作。

通过对已知传感器侧信道的研究,我们发现传感器侧信道对于传感器数据精度的要求远高于正常应用程序中传感器相关功能的数据精度要求,换句话说,传感器侧信道与正常应用对于传感器数据中噪声的鲁棒性具有明显差异。基于上述发现,本文提出了一种基于差分隐私的传感器侧信道攻击防御方案,通过在传感器原始数据的数值与响应时间中注入随机噪声,从根本上降低传感器侧信道攻击(包括共谋攻击)的成功率。由于实施于 Android 应用框架层,本文防御方案能够对用户完全透明,不需要用户参与防御决策。

防御过程中面临 2 个困难:1) 如何混淆传感器原始数据能够有效地干扰侧信道攻击;2) 如何保证 APP 正常功能以及用户体验不受到数据混淆的影响。针对第 1 个问题,本文基于差分隐私技术,利用 Laplace 机制向传感器的原始数据中注入少量噪声,通过干扰侧信道学习过程,降低侧信道攻击成功率。为克服第 2 个问题,本文对各类 APP 能够承受的数据混淆程度进行详细分析。首先基于 FlowDroid^[19]与 SOOT^[20]框架,设计并实现了针对 Android 物理传感功能的抽取工具 SensorTainter,并对 Google Play 上 47144 个 APP 进行传感器相关的功能抽取、分析与分类。最后,通过详细的实验论证各种类型功能对于传感器数据混淆的承受上限,结合用户

体验实验与侧信道防御效果给出合理的传感器数据混淆范围建议。

本文具体贡献有 3 个方面:

1) 针对 Android 物理传感器侧信道防御不足的问题,提出了一种基于差分隐私的传感器原始数据混淆的防御框架。该防御框架实施于 Android 应用框架层,具有良好的扩展性,能够在对用户完全透明的情况下有效降低因传感器 API 管控不严而产生的敏感信息泄露风险。

2) 设计实现了 Android 传感器参与功能分析工具 SensorTainter,该工具基于 FlowDroid 静态污点分析思想,能够快速、精确地抽取出 APP 中传感器参与的程序片段。除物理传感器以外,该工具还可以用于其他低敏感 API 的功能分析。

3) 通过对 Google Play 中 47 144 个 APP 进行功能分析,并针对 9 种典型物理传感器侧信道进行

防御测试,权衡用户体验、功能完整性与防御能力,给出具体的传感器数据混淆的建议范围。

1 系统框架

定义应用程序按照预期设定进行工作的能力称之为功能完整性。应用程序的功能完整性依赖于算法的鲁棒性,其决定了该算法对于输入数据携带噪声的敏感程度,有些功能能够承受较大程度噪声干扰,有些则不然。如果将物理传感器侧信道看作是基于 Android 传感器实现功能中的一类,那么,挖掘侧信道攻击功能完整性与正常应用程序功能完整性之间对于传感器数据的鲁棒性差异,即可从根本上降低传感器侧信道攻击的威胁。本文根据上述思想提出了 Android 物理传感器安全防御方案,具体结构如图 1 所示:

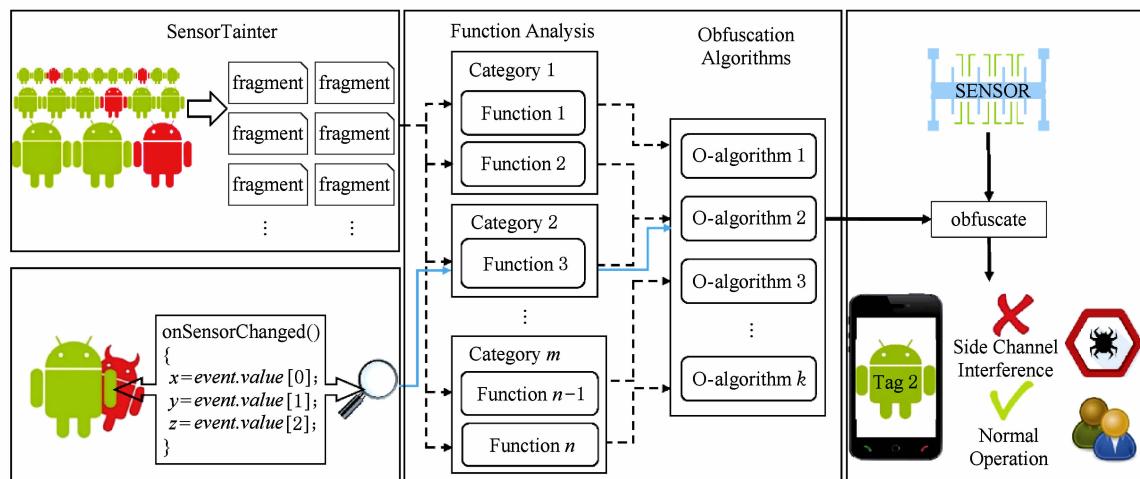


Fig. 1 Structure of Android physical sensor side-channel defensive method

图 1 Android 物理传感器防御方法结构

本节将从防御方法的构建与实施 2 个方面进行介绍。

构建过程中,首先分析市面上已有 APP 物理传感器使用情况,抽取传感器相关功能的程序片段。下一步,根据功能类型以及功能的具体算法进行可扩展的功能分类。基于差分隐私^[21]的思想,我们在传感器原始数据中注入特定噪声进行数据混淆,并通过实验分析各类正常功能以及物理传感器侧信道攻击对于数据混淆的承受能力。为部署传感器攻击防御框架,厂商需要在最新的系统版本中加入本文设计的传感器数据混淆模块,传感器数据混淆模块能够根据当前运行的 APP 中数据混淆标识进行不同程度的数据混淆。具体细节在第 4 节中说明。

应用超市需在 APP 上市之前加入传感器功能分析流程,将功能类别所对应的数据混淆标识(数据混淆程度)加入 APP 描述中,传感器相关功能对应的数据混淆建议范围在第 5 节中说明。若该 APP 的传感器相关功能在防御系统中没有记录,则将结果反馈,应用管理方单独对新功能重新进行数据混淆分析,并更新功能类别与对应数据混淆方案。具有数据混淆程度标识的 APP 运行时,系统将会根据标识在应用框架层截断传感器原始数据进行对应程度的混淆。

在本文防御方案中,用户始终没有参与安全决策,减少了由用户自身安全意识普遍较低而导致的额外风险。此外,数据混淆不会影响 APP 的正常工作,

保证用户体验没有受到影响。从攻击者的角度,由于获取到的传感器数据样本中加入了干扰噪声,通过不可信样本构建的侧信道攻击成功率随之下降,使得用户隐私得到保障。

实施物理传感器侧信道防御方法时,最关键的2个步骤为APP相关功能分析与传感器原始数据混淆。

1.1 Android物理传感器相关功能分析

无论在构建过程还是实施过程中,APP传感器相关功能分析都是最核心的步骤之一。为抽取APP传感器相关功能程序片段,本文基于FlowDroid静态污染分析思想,结合程序切片方法^[22-23],实现了SensorTainter工具。SensorTainter将传感器API调用看作污染源(source),通过追踪污染源SensorEvent在程序中的数据流动以及污染传播,挖掘出与传感器数据存在关联的语句,进而实现相关功能的抽取。SensorTainter核心算法与传播规则将在第2节中描述。

根据我们的调研,目前为止并没有研究对APP传感器API使用情况进行系统性分析。通过SensorTainter对47144个APP扫描,发现其中6408个APP调用了物理传感器API。我们对上述6408个APP进行详细的功能分析。首先依据APP所实现的功能类型进行分类,由于相同的功能之间可能存在调用传感器种类与算法的差异,相同功能类别中将根据具体算法以及传感器类别分类。物理传感器功能的分析结果将作为该APP数据混淆承受能力分析的参考依据,详细的实现过程和功能分析结果在第4节讨论。

1.2 数据混淆

通过总结近年来物理传感器侧信道领域的研究^[24]得出,侧信道主要利用用户行为习惯与传感器数据之间的映射关系实施隐私泄露,已有研究^[25-26]证明通过数据混淆的方式干扰映射关系能够降低隐私泄露的风险。本文基于差分隐私方法,通过注入随机噪声的方式对传感器的数值和响应时间进行数据混淆。实现过程中,考虑到Android传感器API为回调机制,我们在Android应用框架层嵌入数据缓存,每当对底层传感器轮询访问时,向实际传感器读数加入适当噪声。

数据混淆的程度由APP相关功能的承受能力以及对侧信道攻击的干扰预期共同决定。第4~6节中将详细讨论数据混淆方法。

2 Android传感器相关功能抽取

APP中传感器数据的使用情况可以通过分析传感器数据参与的语句获取,我们将Android传感器相关功能抽取近似地看作静态污点分析:传感器访问API被认为是污染源,传感器数据参与功能实现的过程则看作污染对象的流动与传播。但在细节上,传感器相关功能抽取又与传统的静态污点分析存在差异,例如进行功能抽取时不需要考虑路径上是否存在出口,两者的传播规则也不尽相同。

2.1 传感器功能抽取流程与算法设计

Android应用层中物理传感器API包含了请求和处理来自设备硬件信息的对象,API的入口点是SensorManager类,APP通过SensorManager注册准备使用的传感器后,传感器数据将会以SensorEvent的形式发送到监听类SensorEventListener中,监听类通过回调函数onSensorChanged接收更新的传感器数据。在进行程序切片时,该回调函数的SensorEvent对象被认为是传感器信息源。SensorTainter进行传感器数据参与功能的程序抽取过程中,首先对apk文件进行解压缩,搜索项目中的manifest等配置文件,然后反编译dex文件,转换成Jimple格式^[19]的中间件文件,并构建生命周期模型,在回调函数onSensorChanged处生成虚拟的入口函数。下一步,通过soot与hero^[27]构建函数调用图与内部控制流图。同时对源码进行扫描,寻找可能包含传感器类型的语句,例如带有字段sensor.getType的语句。最后,从虚拟主函数进入,根据传感器数据的流向与传播规则构建功能参与集,并将所有参与的语句进行输出作为功能描述。

SensorTainter中APP生命周期建模^[28]中的虚拟入口函数构建基于FlowDroid。传感器相关功能抽取时不考虑组件之间的异步操作,以及除自身以外的回调函数。附录A是对真实应用程序“不倒翁游戏”的抽取实例。

SensorTainter的分析过程分为向前功能参与分析(简称向前分析)与向后别名分析(简称向后分析)两部分,具体算法可参考文献[19]。向前分析负责传感器数据在向前流动的过程中,“污染”其他对象的关联传播,关联传播定义了对象之间的“污染”传递。定义 $x.f$ 表示对象 x 中的字段 f ,传播规则将在本节末尾描述。

向后分析负责新“污染”对象的别名查找。当与传感器数据相关联的变量被赋值给其他局部变量时,就会产生向后的别名分析,寻找具有相同引用的对象。若存在别名且发起别名分析的对象与传感器数据存在关联关系,则以别名对象处为起点生成1次新的向前分析,如果别名对象仅参与了算数的操作,则只记录当前状态。例如对于语句 $x = y + z$, y 是与传感器对象存在数据关联的变量, z 是一个常量,则只有变量 x 与传感器数据产生关联。

图2中展示了1次SensorTainter分析过程:

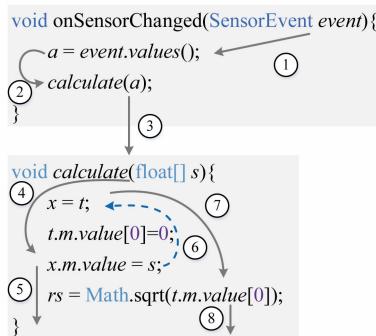


Fig. 2 Example of SensorTainter's algorithm

图2 SensorTainter 算法举例

图2中的实线(灰色)代表向前参与分析,虚线(蓝色)代表向后别名分析,标号代表分析步骤。

步骤①当设备的运动状态发生改变时,传感器回调函数被触发,变量 a 获得当前传感器读数;

步骤②~③函数 $calculate$ 被调用,变量 a 作为参数传入函数体内,完成实参 a 与形参 s 的映射;

步骤④~⑤变量 x 产生对 s 的引用,继续向前追踪变量 x 、 s 的数据流动;

步骤⑥对 x 进行向后别名分析,发现 x 与变量 t 引用同一个对象;

步骤⑦由 t 处产生新的向前数据流分析,发现 t 中对传感器数据的引用参与了变量 rs 的计算;

步骤⑧继续向前分析由变量 t 出发的数据流。

SensorTainter需要考虑被分析语句的上下文信息,考虑图3中场景:

当数据流之间出现交汇时,可能会影响“污染”的传播。图3中函数 $calculate$ 的传感器对象 $SensorEvent$ 被变量 d 引用后进行了加法操作,然而,并非每一次调用 $calculate$ 函数都会被看作有传感器数据的参与:图3中行②对 $calculate$ 的调用会产生传感器相关数据流,然而行③ $SensorEvent$ 对象被重新创建,因此行④处的函数调用将不与传感器数据产生关联。SensorTainter为每一条数据流分支创建单独的队列存储上下文信息。

```

void onSensorChanged(SensorEvent event){
    ① Data d1, d2;
    ② calculate(event, d1);
    ③ event = new SensorEvent();
    ④ calculate(new SensorEvent(), d2);
    ⑤ }

void calculate(SensorEvent e, Data d){
    ① d.value = e.value;
    ② d.value[0]++;
    ③ }

```

Fig. 3 Context sensitive scene

图3 上下文敏感场景

2.2 SensorTainter 传播规则定义

SensorTainter将APP中的语句分为4类:一般语句、函数调用语句、函数返回语句与其他语句,4种语句对应独立的参与传播规则。为了更精确表示对象中的字段信息,通过 $x.f^m$ 表示变量 x 中字段深度为 m 的访问路径。

2.2.1 向前分析传播规则

1) 一般语句。SensorTainter仅对传感器数据进行追踪,因此只有1个数据源sensor,普通的节点不能凭空产生传感器数据,只能被赋值(=)或者清除(new)。对于一个赋值语句 $x.f^n = y.f^m$,转移函数按照规则:

$$O \xrightarrow{s} \begin{cases} O \cup \{x.f^n.f^p\}, & \forall p: y.f^m.f^p \in O, \\ O \setminus \{x.f^n\}, & y.f^m.f^* \notin O \\ \quad \wedge \neg arrayElem(x.f^n), \\ O, & \text{otherwise,} \end{cases} \quad (1)$$

其中, O 表示参与集。对于普通节点的赋值语句,如果 $y.f^m$ 下有任何字段的来源于传感器数据有关,则把引用的 $x.f^n$ 下的相应访问路径加入参与集 O 中;如果 $y.f^m$ 下不包含任何与传感器数据相关的字段,而且 $x.f^n$ 也不在参与功能的数组之中,则将 $x.f^n$ 从 O 中去除;其他情况保持不变。需要额外说明右边是算数表达式的情况 $x.f^n = y.f^m(op)z.f^k$:

$$O \xrightarrow{op} O \cup \{x.f^n\} \wedge O_i \cup \{y.f^m\} z.f^k \in O, \quad (2)$$

其中, op 表示算数运算符。右边算数表达式中只要存在1个以上的变量与传感器数据之间存在关联,则左边的变量被认为直接参与功能实现,右边其他的操作数被认为间接参与相关功能实现,并加入间接参与集 O_i 中。间接参与节点不会产生新的向前参与分析,但会进行向后别名分析。

另一种特殊情况为 $x.f^n = \text{new } y.f^m$:

$$O \xrightarrow{\text{new}} O \setminus \{x.f^n.f^* \in O | \neg arrayElem(x.f^n)\}. \quad (3)$$

由于 Java 中 new 操作会将被赋值的对象重新初始化,因此不再认为该对象后续参与传感器相关的功能实现.

2) 函数调用语句. 调用代表了函数调用对象对于被调用对象函数体内部的影响,即通过函数调用事件进入到函数体内部,设函数调用的形式为 $c.m(a_0, a_1, \dots, a_n)$, 其中 c 表示对象实例, m 是对象包含的函数名, a 为函数实参参数, 函数调用流的处理规则可以表示为

$$O_{\text{callee}} \xrightarrow{s} \bigcup \begin{cases} \{this.f^m\}, c.f^m \in O_{\text{caller}}, \\ \{p_i.f^p\}, a_i.f^p \in O_{\text{caller}}, \\ \{y.f^q\}, x.f^q \in O_{\text{caller}} \wedge \\ \quad static(y.f^q), \end{cases} \quad (4)$$

其中, O_{caller} 是函数调用语句中被调用对象的参与集合, O_{callee} 是被调用函数的实际函数体的参与集合, s 为当前函数调用语句, p 是对应的形参. 如果被调用函数所在的对象中有成员参与了功能,则在被调用函数的实际函数体中标记为参与关系;如果函数调用中的实参参与了功能,则将被调用函数中对应形参加入 O_{callee} ;如果一个调用对象中的全局变量参与了功能实现,且这个全局变量也出现在了被调用函数体中,则将其加入到 O_{callee} 中.

3) 函数返回语句. 函数返回代表了被调用的函数体对于调用者的影响,即函数体内部处理结束后向外界返回的数据流. 函数体内参与数据之间的传递并不一定会影响到函数调用者,需要判断最后的返回值是否有与传感器数据有关. 对于函数调用 $b = c.m(a_0, a_1, \dots, a_n)$, 函数返回流的传递规则为

$$O_{\text{caller}} \xrightarrow{s} \bigcup \begin{cases} \{c.f^m\}, this.f^m \in O_{\text{callee}}, \\ \{a_i.f^p\}, p_i.f^p \in O_{\text{callee}} \wedge \\ \quad num(a_i.f^p), \\ \{y.f^q\}, x.f^q \in O_{\text{callee}} \wedge \\ \quad static(y.f^q), \\ \{b.f^v\}, r.f^v \in O_{\text{callee}}. \end{cases} \quad (5)$$

如果被调用函数存在参与字段,则将该路径加入 O_{caller} 中;如果被 O_{callee} 中形参存在参与字段,则调用返回后所对应的实参也存在对应参与字段,需要注意的是如果参与路径是基础类型,则必须是数值类型;如果被调用函数体中有 $x.f^q$ 参与功能实现,且与其对应的 $y.f^q$ 是静态全局变量,则 $y.f^q$ 需要加入 O_{caller} 中;如果被调用函数体的返回值参与了,则函数调用的结果接收对象 $b.f^v$ 也将被认为参与了功能实现.

4) 其他语句. 对于函数调用 $b=c.m(a_0, a_1, \dots, a_n)$, 其他语句传递规则用于处理原生函数调用以及原始传感器数据源:

$$O_{\text{caller}} \xrightarrow{s} \begin{cases} O_{\text{caller}} \cup \{b.f^n\}, native(s) \wedge \\ \quad (a_i.f^p \in O_{\text{caller}}), \\ O_{\text{new}} \cup \{b.f^n\}, sensor(c.f^m), \\ O_{\text{caller}}, \text{otherwise}. \end{cases} \quad (6)$$

当前语句 s 是原生函数调用时,如果实参中包含有 O_{caller} 中的变量,则将其加入 O_{caller} 中. 在本文分析的结果中,并没有发现使用原生函数参与传感器数据相关功能的应用,因为绝大部分原生函数可以做到的功能都有更好的替代方法,但是考虑到切片程序的完整性与原生函数参与的可能性,还是加入了对原生函数的处理;如果函数调用是传感器 API 调用,则创建新的参与集用于记录新的传感器数据流;其他情况下不改变现有状态.

参与集中记录的只是参与传感器数据利用的对象,每当参与集发生变化时需要将当前包含新增参与对象的语句 s 加入当前函数体的抽取集合 PS 中:

$$PS_{\text{current}} \xrightarrow{s} \bigcup \{s\} \quad s \notin PS_{\text{current}} \wedge \\ ((contain(x.f^*)) \wedge x.f^* \in O) \vee \\ (contain(c.m^*)) \wedge c.m^* \in O. \quad (7)$$

每当进入新的函数体都将生产独立的程序抽取集合. 参与集发生变化时,如果当前节点处的语句不属于函数体内的程序抽取集合,且语句 s 包含参与集中的对象或者函数调用,则将 s 加入当前程序抽取集.

向前参与流分析中数据流的起点如果是某个参与对象 $x.f^m$ 的别名 $y.f^n$, 只有该数据流第 1 次通过了 $x.f^m$ 与传感器数据关联的语句时, $y.f^n$ 才会被认为是参与了传感器数据的使用.

2.2.2 向后分析传播规则

向后别名分析传播规则中,同样分为 4 种语句关联转移情况.

1) 一般语句. 对于语句 $x.f=y.g$, 一般语句的转移规则可以表示为

$$A \xrightarrow{s} \bigcup \begin{cases} A \setminus \{x.f.f^p\} \cup \{y.g.f^p\}, \\ \forall p: x.f.f^p \in A, \\ A, \text{otherwise}. \end{cases} \quad (8)$$

向后别名分析的本质上与向前参与分析规则类似,只是方向发生了转变. 如果 $x.f.f^p$ 是存在于别名集 A 中的已经记录的别名字段,则 $y.g.f^p$ 是 $x.f.f^p$ 的新别名,因此从当前别名集中去掉 $x.f.f^p$,并添加 $y.g.f^p$.

对于 new 语句 $x.f = \text{new}^*$, 别名集中关于变量 x 的所有引用的别名信息都需要被除去:

$$A \xrightarrow{\text{new}} A \setminus \{\forall m; x.f.f^* \in A\}. \quad (9)$$

2) 函数调用语句. 向后分析与向前分析中, 函数调用语句的传递方向正好相反, 对于函数调用, 参与传播从函数调用语句的调用方向返回方进行, 对于函数调用 $b=c.m(a_0, a_1, \dots, a_n)$:

$$A_{\text{callee}} \xrightarrow{s} \bigcup \begin{cases} \{r.f^m\}, b.f^m \in A_{\text{caller}}, \\ \{this.f^m\}, c.f^m \in A_{\text{caller}}, \\ \{y.f^q\}, y.f^q \in A_{\text{caller}} \wedge \\ \quad \text{static}(y.f^q). \end{cases} \quad (10)$$

若调用端 $b.f^m$ 是当前追踪变量的别名, 则在 caller 的函数体中返回值也应该是其别名; 若 callee 的对象实例是当前追踪变量的别名, 则 callee 中的对应成员对象也是被追踪变量 $x.f^m$ 的别名; 若某个全局对象是当前追踪变量的别名, 则该全局对象在 callee 中的对应对象也应是 $x.f^m$ 的别名.

3) 函数返回语句. 因为数据流是从 callee 函数体的起始位置向调用方流动, 所以返回流将 callee 内的别名映射到调用方的别名集中, 参数则由形参向实参方向流动:

$$A_{\text{caller}} \xrightarrow{s} \bigcup \begin{cases} \{c.f^m\}, this.f^m \in A_{\text{callee}}, \\ \{a_i.f^p\}, p_i.f^p \in A_{\text{callee}}, \\ \{y.f^q\}, y.f^q \in A_{\text{callee}} \wedge \\ \quad \text{static}(y.f^q). \end{cases} \quad (11)$$

如果在当前函数体内的成员对象在方法内部是对应别名, 则被调用对象的对应成员变量是调用方的别名; 如果被调用函数的形参是被追踪对象的别名, 则函数调用过程中所对应的实参同样是该对象的别名; 如果被调用函数体中的某个全局静态对象是被追踪对象的别名, 则这个静态对象在调用方也应该是别名.

4) 其他语句. 对于语句 $b=c.m(a_0, a_1, \dots, a_n)$, 其他类型数据流转移规则用于处理由原生调用产生的别名或者清除由 b 引用的别名:

$$A \xrightarrow{s} \begin{cases} A \cup \text{nativeAlis(s)} \wedge \text{native(s)} \wedge \\ \quad (a_i.f^m \in A \vee c.f^n \in A), \\ A \setminus \{b.f^n\}, b.f^n \in A, \\ A, \text{ otherwise.} \end{cases} \quad (12)$$

若当前语句为原生函数调用, 且满足 2 个条件之一: 实参中存在对象是当前被追踪对象的别名, 或者 callee 中相同深度的引用路径是被追踪对象的别名, 则将被 s 调用的原生函数所创建的别名加入当

前追踪的别名集合中; 由于函数调用语句 s 会将 b 的引用更新, 所以若调用函数的接收对象 $b.f^n$ 已经存在于别名集合中, 则该语句处对象 b 对应字段将不再是当前被追踪对象的别名; 其他情况下不改变别名集合 A .

向后别名分析中, 程序抽取集合不会记录产生别名的语句, 因为产生别名的语句(通常是赋值语句)将会作为新的向前参与流分析的起点, 并由向前分析记录.

3 物理传感器相关功能分析

Android 系统现支持 30 种以上的传感器. Android 物理传感器是指能够通过 SensorManager 对象进行调用与管理的传感器, 其中包含了 8 个基础传感器与约 22 个合成传感器. 基础传感器为直接由硬件设备提供数据的传感器, 合成传感器则是由基础传感器通过软件层面的组合得到. 本文中主要对基础传感器进行分析, 因为合成传感器可以看作由基础传感器实现的特殊功能, 基础传感器的数据混淆后也会连带影响合成传感器输出的结果.

3.1 传感器调用类型分析

本文通过 SensorTainter 对 47 144 个 Google Play 的 APP 进行分析, 发现 6 408 个 APP 具有物理传感器的调用行为, 调用率约为 13.59%, 文中所有对应用程序数量的讨论皆以上述 6 408 个 APP 为基础. 各类传感器的调用情况如图 4 所示:

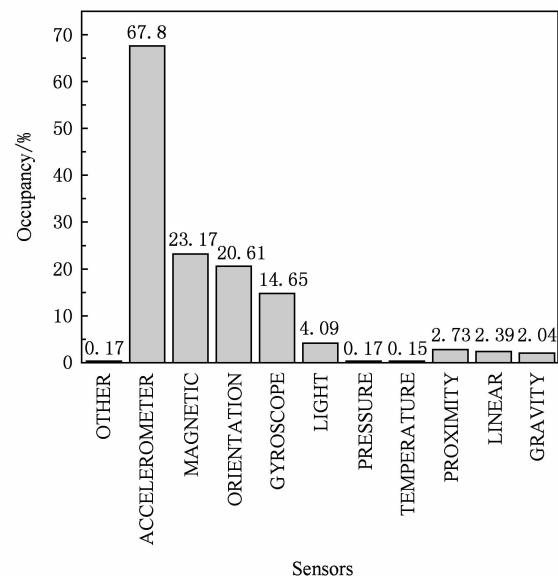


Fig. 4 Statistics of Android sensor distribution

图 4 传感器调用分布统计

加速度传感器的调用比重明显高于其他传感器,约 67.8% 的 APP 使用了加速度传感器,这是由于 Android 应用程序中大部分的常用功能都可以通过加速度传感器实现。磁强计、方向传感器与陀螺仪调用率比较接近,分别约为 23%, 21%, 15%。

磁强计除了作为电子罗盘的必要传感器,更多的时候是与加速度传感器一起组合计算方向。新的 Android 版本中方向传感器(ORIENTATION)已经不建议使用,但方向传感器的调用率仍然很高,主要原因在于直接调用方向传感器比通过加速计和磁强计组合更加方便,虽然在精度方面略有不足,但对于判断屏幕横屏竖屏等简单功能而言已经足够。陀螺仪通常用于游戏矢量旋转角度的计算,以及与加速度传感器一起实现惯性导航。光传感器(LIGHT)的调用率在剩余传感器中最高,主要用于实现屏幕的亮度调节。在本文的 APP 样本中,接近传感器(PROXIMITY)只用于实现接听电话时的锁屏功能。气压传感器(PRESSURE)理论上可以用于实现气压计小工具,但在被分析的样本中极少出现。图 4 中的其他传感器包含了除基础传感器与线性加速计、重力传感器以外的所有组合传感器,可以看出开发者对于官方提供的组合传感器并不习惯,更倾向于灵活的自定义开发方式。

APP 开发者会根据具体需求申请多个传感器。根据本文统计,约 67.75% 的应用程序仅调用了 1 种传感器,约 17.23% 的应用程序调用了 2 种传感器,剩下约 15.02% 的应用程序调用了 3 种及 3 种以上的传感器。具有多种功能的 APP 中,数据混淆上限由承受能力下限的功能决定。

3.2 传感器参与功能分析

本文使用基于关键词比较与词法子树的组合相似度判断方法,并结合内容特征^[29],对抽取出的程序片段进行分类。经分析发现,Android 物理传感器开发呈现明显的规律性。常用功能的核心算法具有很大的相似性,例如屏幕旋转、摇晃检测等,我们将其归咎于开源库的广泛应用。相对地,由于开发者无法在网络上找到开源的案例,一些并非常用的功能或者特殊场景下的功能算法之间则具备一定差异。此外,相同功能因预期要求不同,调用的传感器类型以及算法也会存在差异。为尽可能准确地对 APP 传感器相关功能进行分类,我们通过迭代扩展功能集的方式进行功能分类。功能分析结果如图 5 所示:

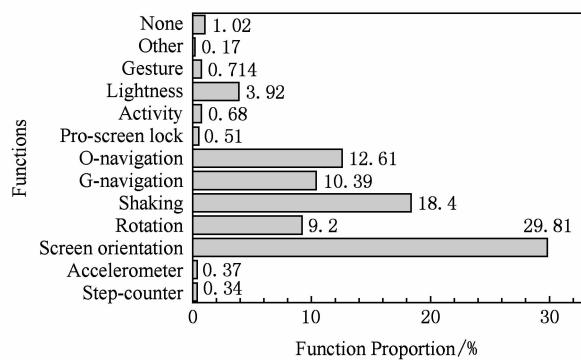


Fig. 5 Analysis result of sensor-based functions

图 5 传感器相关功能分析

首先筛选出具有明显功能标识的程序段,从中抽取功能对应的核心算法和传感器类型作为该功能的识别特征之一;然后利用关键 API、关键操作符以及操作顺序对其他功能片段进行初步匹配。对于难以识别的程序片段使用基于词法子树的相似度比较方法进行匹配,同时,人工经验被用于辅助特殊功能和新功能的识别。功能集会随着分析样本数量的增加而逐渐扩大。

图 5 为本文根据功能目标、功能算法以及传感器类别进行分类的统计结果。图 5 中,纵坐标表示分析得到的传感器相关功能,“None”表示无实际功能,“Others”表示其他功能,“Gesture”表示手势检测,“Lightness”表示亮度调节,“Activity”表示行为检测,“Pro-screen lock”表示距离锁屏,“O-navigation”表示基于方向传感器导航,“G-navigation”表示基于陀螺仪导航,“Shaking”表示摇晃检测,“Rotation”表示旋转角度计算,“Screen orientation”表示屏幕方向检测,“Accelerometer”表示加速度检测,“Step-counter”表示步数统计。传感器相关功能中所占比例最多的为判断屏幕横竖屏,占到所有功能实现中的近 30%。判断屏幕方向是大部分应用需要支持的功能,虽然 Android 提供了用于判断屏幕方向的 API,但是该 API 在实际应用上存在缺陷,无法判断一次性旋转 180 度以上的转动,因此相当一部分开发者选择利用加速度传感器实现屏幕旋转的检测功能。通过加速度传感器判断手机震动功能的比例排名第 2,约为 18.4%,我们在 1000 多个应用程序中发现了震动检测功能,除了摇一摇功能以外,常见的震动检测出现在需要抖动翻页、游戏等 APP 中。利用方向传感器和陀螺仪进行辅助导航功能主要出现在地图、娱乐搜索等提供导航的 APP 中,其中利用

方向传感器(包括使用加速计与磁强计合成方向传感器)的程序约占 12.6%,比使用陀螺仪进行惯性导航高了约 2 个百分点。游戏矢量旋转检测主要用于射击类游戏等 APP 中,该功能对于数据精度与响应时间都有一定的需求。

进一步分析得出,相比于传感器侧信道攻击,APP 中正常功能对传感器原始数据的敏感程度较低,算法多以阈值比较的形式实现。APP 中有 4 种传感器建议响应频率: NORMAL, UI, GAME, FASTEST, 分别对应 200 ms, 67 ms, 20 ms 和 0 ms, 其中,NORMAL 和 UI 的使用次数最多,FASTEST 几乎没有出现。少数 APP 中存在 1 种以上的传感器相关功能,例如,在最近市面上出现的某款约车软件中分析出了摇晃检测与辅助导航功能。

根据上述分析结果与当前已知的传感器侧信道攻击方法,后续主要讨论加速度传感器、方向传感器、磁强计和陀螺仪的原始数据混淆对于应用的影响以及隐私泄露攻击成功率。

本文在分析过程中发现了一些需要引起重视的现象:通过加速度传感器与简单机器学习方法对用户手势改变进行推测的游戏应用已经出现在应用市场中,无论开发者处于何种目的,该行为对于用户的隐私安全具有非常大的潜在危险。在我们之前的研究^[30]中已经证实手势变化可以泄露 PIN 等用户敏感信息。更为严重的是多数 APP 中调用运动传感器的同时,也会调用 TouchEvent 与 MotionEvent 等用于触控的 API,约为总数的 41.87%,导致用户使用应用程序的同时,触碰行为与运动传感器之间的关联轻而易举的被 APP 得到,而该行为是大多数基于传感器侧信道的构建前提。

4 基于差分隐私的传感器数据混淆

传感器侧信道符合相似的构建过程,即利用机器学习技术挖掘用户行为与传感器数据之间的映射关系,因此,如果能够在不影响用户体验的前提下条件下干扰侧信道的学习阶段,则可以抑制侧信道的攻击能力。我们借鉴差分隐私的思想对传感器数据进行模糊,使得传感器数据在反映用户行为的过程中失准。

4.1 数值噪声

差分隐私用于保证一定数据精度下,有效确保数据的隐私性。我们将侧信道学习阶段的特征提取

过程映射为差分隐私的数据查询行为,通过 Laplace 机制向当前响应的传感器读数 $x_o(n)$ 中注入随机噪声 $noise(n) \sim Lap(0, \beta_d)$:

$$x_o(n) = x(n) + noise(n), \quad (13)$$

其中, $\beta_d = S_d / (\epsilon/d)$, $d \in [1, 3]$ 表示传感器的数据维度, S 为当前维度上的敏感度,其值等于窗口范围内传感器读数的最大值与最小值的差值。混淆因子 ϵ 表示数据混淆程度,由 Laplace 的概率分布函数可知, ϵ 越大概率密度的尺度参数越小,产生高值噪声的概率越大。

为了使防御对攻击者以及用户的完全透明,数据混淆机制实施于 Android 的框架层,通过在传感器管理对象 android.hardware.SensorManager 文件中添加过滤器的方式实现噪声注入,具体修改方法见附录 C。

4.2 时间噪声

基于时间的隐私泄露攻击通过识别用户在不同行为时产生的时间间隔进行隐私窃取。绝大部分传感器侧信道攻击都需要时间信息参与学习过程,因此必须考虑响应时间混淆。

文献[31]证明在系统时钟中加入随机延迟噪声可以有效降低该类攻击的威胁。本文通过随机添加响应时间补偿的方式对传感器响应时间进行混淆。

差分隐私的思想同样适用于时间噪声。时间序列是 1 维序列,因此参数调整为 $\beta = S/\epsilon$ 。因为时间具有顺序性,每个时间点添加噪声后不能超过前后相邻时间点,因此 $S = 1/f$,其中 f 为传感器采样频率。实时对传感器的响应时间进行混淆会使得实际读数存在最大 1 位的延迟,所以在对传感器要求很高的情况下(例如射击游戏矢量旋转角度检测)不对响应时间进行混淆。传感器数值混淆与响应时间混淆的核心算法设计如算法 1 所示。

算法 1. 应用框架层传感器数据混淆算法。

输入:当前传感器读数 v_{in} 、传感器响应系统时间 T_{in} 、混淆程度 ϵ_{value} 、 ϵ_{time} 、传感器轮询频率 f 、传感器维度 d 、缓存窗口 w ;

输出:混淆数据 v_{out} .

$LaplaceDistribution(\beta)$; 根据参数 β 随机生成 Laplace 噪声

- ① $S_{value} = Max_w - Min_w$; /* 计算敏感度参数 */
- ② $\beta_{current} = S_{value} / (\epsilon_{value}/d)$;
- ③ $noise_{value} = LaplaceDistribution(\beta_{current})$;
- ④ $v_{out}[i] = v_{in}[i] + noise_{value}$; /* 加入数值噪声进行数据混淆 */

```

⑤ push  $v_{out}$  into cache; /* 时延缓存 */
⑥  $\beta_{time} = 1/(f \times \epsilon_{time})$ ;
⑦  $noise_{time} = LaplaceDistribution(\beta_{time})$ ;
⑧  $T_{out} = T_{in} + noise_{time}$ ; /* 加入时间噪声进行
    数据混淆 */
⑨ while  $T_{out} < T_{last}$  and  $noise_{time} > 1/f$ 
⑩    $noise_{time} = LaplaceDistribution(\beta_{time})$ ;
⑪    $T_{out} = T_{in} + noise_{time}$ ;
⑫ end while /* 时间噪声范围修正 */
⑬ while true
⑭   if  $T_{out} \leqslant$  current system time then
⑮     pop  $v_{out}$  from cache;
⑯     break;
⑰   end if
⑱ end while

```

4.3 数据混淆承受范围分析

本节将讨论不同程度数据混淆对于 APP 传感器相关功能的实际影响情况以及各功能所对应的数据混淆承受上限。

4.3.1 屏幕方向检测

根据第 3 节中的分析结果,物理传感器相关功能中应用范围最广的是屏幕旋转检测。首先构造屏幕旋转检测试 APP,该 APP 根据接收到的加速度传感器读数计算当前的屏幕方向,为比较分析加速度混淆程度对于屏幕方向判断影响,不同混淆程度的传感器数据将会通过多个线程同时传递到数据收集 APP 中。

噪声程度随 ϵ 的增加而增强,时间开销上,实验设备 Meizu max5 中对加速度传感器读数进行 1 次混淆的平均时间为 0.024 ms,相对于传感器响应时间可忽略不计(即使 FASTEST 下也存在 3 ms 左右停顿)。

屏幕方向检测功能影响实验中,共有 5 名参与者进行了约 1 000 次测验,每次测试的屏幕旋转的方向、方式以及时间等因素均为随机,实验结果基本一致,本节中随机选取其中 1 次实验结果进行举例说明。图 6 表示 1 次实验中,屏幕方向随时间的变化,当 Y 方向数值为 1 时表示屏幕当前方向为横屏,数值为 -1 时表示当前方向为竖屏,0 表示设备朝向其他方向转动或者没有屏幕转动的情况,图 6 中实线(蓝色)表示利用未经混淆的原始数据计算得到的屏幕方向,虚线(红色)表示利用 $\epsilon=10$ 混淆的数据计算的屏幕方向,传感器读数的时间混淆均为 $\epsilon=2$ 。从图 6 中看出,进行混淆后的加速度传感器数在 $\epsilon=$

10 的情况下依然能正确地判断屏幕方向,仅因时间混淆而在时间为 7 400 ms 处产生了轻微的延迟。

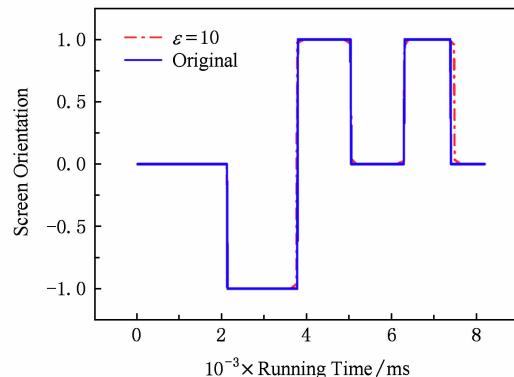


Fig. 6 Screen orientation detection with $\epsilon=10$
图 6 $\epsilon=10$ 屏幕方向检测

将混淆程度提高到 $\epsilon=50$ 后,如图 7 所示,被混淆的加速度读数依然可以正确地判断屏幕方向,但是出现了少量的波动情况。实际使用过程中只要不出现旋转方向判断错误,小幅度的波动是被允许的。

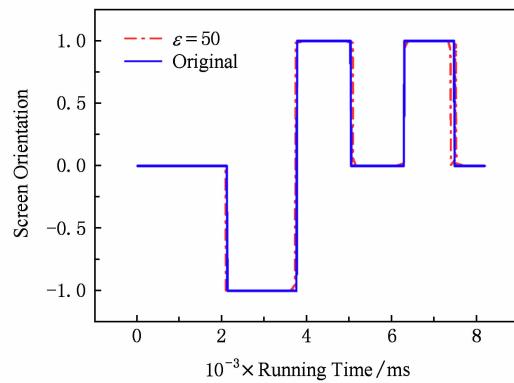


Fig. 7 Screen orientation detection with $\epsilon=50$
图 7 $\epsilon=50$ 屏幕方向检测

实验结果证明:利用加速度传感器实现的屏幕方向检测功能对于传感器数据的鲁棒性较高,可以容忍高程度的数值混淆。仅考虑用户体验与功能完整性,针对屏幕方向检测功能的传感器数据混淆范围为 $1 \leqslant \epsilon \leqslant 10$,时间混淆上,时延范围为 -4.28 ~ 3.33 ms,平均时延为 -0.06 ms。

4.3.2 摆晃检测功能

揆晃检测功能不仅被广泛使用于揆一揆的实现上,还被用于相册、阅读等 APP 中实现快速翻页。根据对本文 APP 数据集的统计分析结果,揆晃检测功能均依赖于加速度传感器实现,具体实现算法主要分为 3 类,算法样例见附录 B。

实验过程中,所有参与者在不受任何限制的环境下以随机幅度揆晃设备,实验程序将分别通过 3 种

摇晃检测算法统计不同加速度混淆程度的摇晃次数。图8显示了3种摇晃检测算法在混淆程度 $1 \leq \epsilon \leq 100$ 的摇晃次数分布情况,实验中的时间混淆 $\epsilon=2$ 。3种算法对于摇晃的判定依据不同,因此相同数据下的平均摇晃统计次数存在差异。从图8可看出,摇晃检测算法1(s-algorithm 1)摇晃统计分布的上下边缘距离最宽,这是由于摇晃检测算法1判断方法最简单,因此受到数据混淆的影响程度最大。相较而言,摇晃检测算法2(s-algorithm 2)与摇晃检测算法3(s-algorithm 3)受到加速度数据混淆的影响较小,没有出现较大幅度的误差。

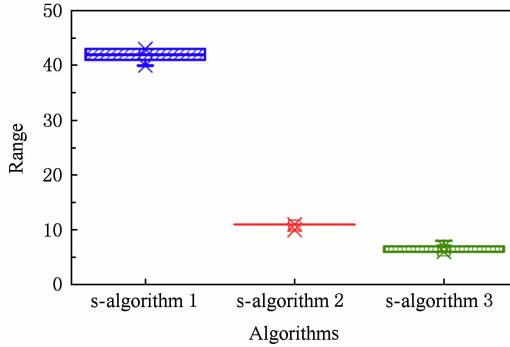


Fig. 8 Result of impact on shaking detection of obfuscation

图8 3种摇晃算法测试计数统计结果

3种算法随数据混淆程度的平均摇晃次数统计情况如图9所示,点划线(蓝色)、短划线(红色)与实线(绿色)分别表示算法1(s-algorithm 1)、算法2(s-algorithm 2)与算法3(s-algorithm 3)。所有摇晃检测算法在混淆程度 $\epsilon=20$ 之前都具有很好的鲁棒性,实际摇晃检测次数与未混淆情况时保持一致。当混淆程度超过20后摇晃检测算法1与摇晃检测算法2对应的检测效果开始出现小幅度偏差。随着混

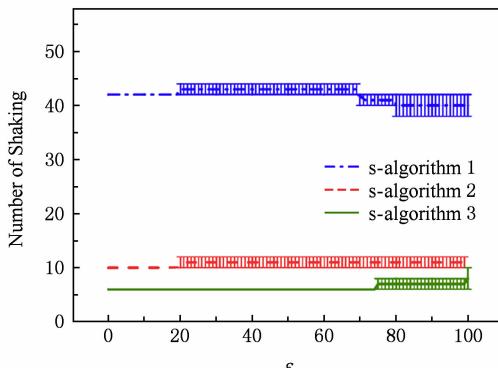


Fig. 9 Statistics of 3 shaking detection algorithms with ϵ increasing

图9 3种算法计数随混淆程度变化统计

淆程度继续增大,摇晃检测算法2误差范围稳定,而通过摇晃检测算法1实现的摇晃检测功能表现出明显异常。摇晃检测算法3在 $\epsilon=75$ 时开始出现小幅度偏差。

从功能的实际应用场景角度分析,摇晃检测算法1与摇晃检测算法2主要在摇一摇类功能中应用,用于判断短时间内智能手机是否发生大幅度摇晃,只需在多次摇晃中检测要设备摇晃即可满足功能要求,因此,虽然摇晃检测算法1与摇晃检测算法2相比摇晃检测算法3鲁棒性略低,但算法本身应用场景对于算法的精度要求不高。摇晃检测算法3通常被用于相册、阅读中的翻页功能,这类功能需要较高的鲁棒性与低精度(精度太高容易引起频繁翻页),对于混淆数据的要求反而更高。

摇晃检测功能验证中时间混淆范围为 $-3.58 \sim 2.84$ ms,平均时延为 -0.084 ms。综合考虑用户体验与具体功能要求,本文建议利用摇晃检测算法1与摇晃检测算法2的摇晃检测所对应的传感器数据混淆范围为 $1 \leq \epsilon \leq 20$,摇晃检测算法3相关功能的数据混淆范围为 $1 \leq \epsilon \leq 70$ 。

4.3.3 地理方位检测

地理方位检测功能主要发现于地图导航、电子罗盘等应用中,参与功能实现的传感器为陀螺仪和方向传感器。我们比较相同方位时不同程度混淆的数据在计算方位时的误差,分析方位检测功能中算法鲁棒性。实验过程中,方位被映射到{正北,东北,正东,东南,正南,西南,正西,西北}八个方向,分别以数字0~7表示,数值混淆范围为 $1 \leq \epsilon \leq 100$,时间混淆为 $\epsilon=2$ 。随机抽取的某次实验结果如图10与图11所示。

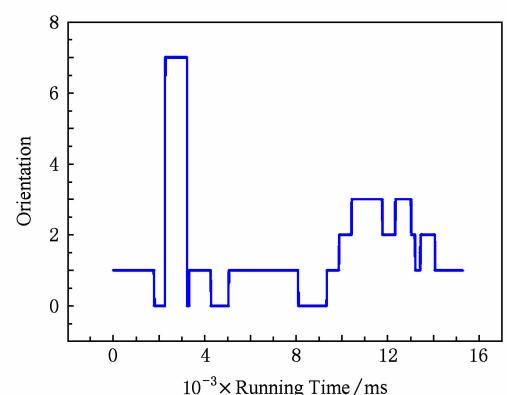
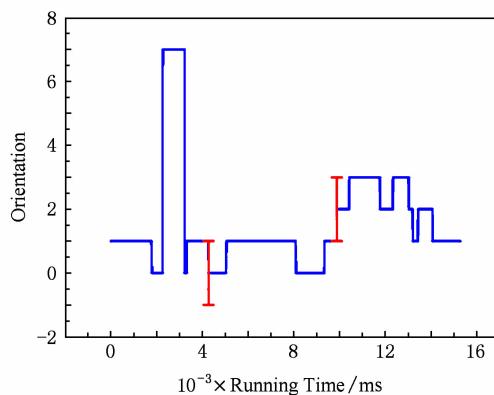


Fig. 10 Geographic orientation detection with $\epsilon=10$

图10 $\epsilon=10$ 地理方位检测

图10与图11分别为混淆程度 $\epsilon=10$ 与混淆程度 $\epsilon=100$ 时的方向检测结果,其中线段(蓝色)表示

Fig. 11 Geographic orientation detection with $\epsilon = 100$ 图 11 $\epsilon = 100$ 地理方位检测

准确方位,误差棒(红色)表示通过混淆数据计算的方位与准确方位之间的误差。从比较结果中不难看出,方位检测算法对于传感器数据中的噪声具有很强的鲁棒性,根据我们对实验结果的统计,混淆程度平均到达 $\epsilon = 100$ 后方向才开始出现明显偏差。混淆程度为 $\epsilon = 10$ 以下时,所有实验中的方向计算结果均与准确方向一致,数据混淆对于方位检测功能没有产生任何负面影响。实验中时间混淆的平均范围为 $-3.75 \sim 5.15$ ms,平均时延为 -0.0156 ms,方位检测功能的数值混淆的理论承受范围为 $1 \leq \epsilon \leq 10$ 。

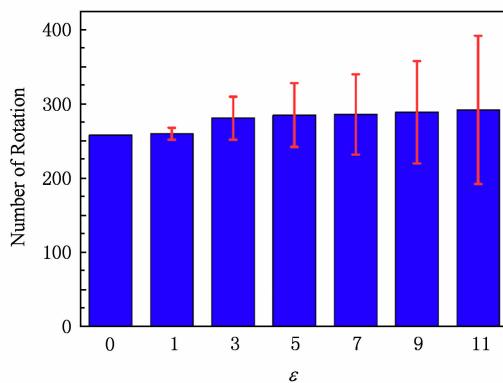
4.3.4 设备旋转方向检测

设备旋转方向检测主要存在 2 种实现方式:1) 方向传感器,根据一段时间间隔后设备在某个方向上的角度差计算设备的旋转方向;2) 陀螺仪,APP 首先判断设备姿态,而后依据某一维度上的旋转角速度判断设备旋转方向。通过方向传感器实现设备旋转方向检测的实现原理与 4.3.3 节中相似,本小节中针对第 2 类算法进行分析。

参与人员被要求以任意姿态随机旋转实验手机,APP 将根据不同混淆程度的陀螺仪读数计算当前旋转方向,并统计方向改变的次数。通过比较错误的旋转方向次数对算法鲁棒性进行评估。实验结果如图 12 所示。

图 12 中,X 轴为数据混淆程度,Y 轴为平均旋转次数,误差棒(红色)表示旋转检测的误差情况。旋转方向检测算法对于数据的敏感程度远超出预期,数值混淆程度达到 $\epsilon = 3$ 时开始出现明显误差,虽然平均方向改变次数并没有随着混淆程度增加而大幅度改变,但是误差范围逐渐扩大。

实际应用时,用户可以容忍短暂的响应停顿,但通常无法接受 APP 的错误响应。我们定义检测到旋

Fig. 12 Experimental result of smartphone rotation with ϵ range from 0 to 11图 12 设备旋转计数分布随 ϵ 增长统计

转事件但没有响应的情况为轻微误差,例如屏幕旋转停顿。定义旋转方向判断错误为严重误差,例如向右旋转判断为向左旋转,2 种误差随混淆程度变化的统计结果如图 13 所示:

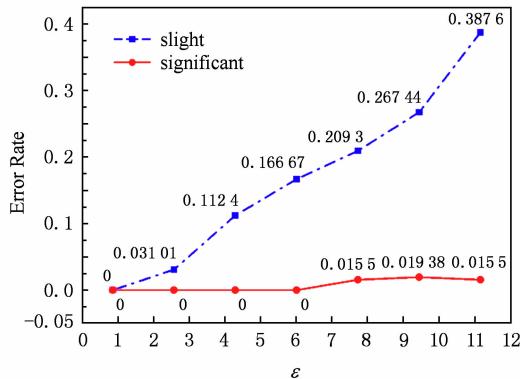
Fig. 13 Statistics of slight error and significant error with ϵ range from 1 to 11图 13 轻微误差率与严重误差率随 ϵ 增长统计

图 13 中,由实线(红色)表示的严重误差的错误率在混淆程度为 1~5 之间时始终保持为 0,而点划线(蓝色)表示的轻微错误率随着混淆程度增加接近线性增长,可知轻微的数据混淆并不会产生影响功能性的错误。实验中时间混淆的平均范围为 $-3.93 \sim 3.06$ ms,平均时延为 -0.0965 ms。为了保证用户的体验不受到影响,设备旋转方向检测的数值混淆最大范围为 $1 \leq \epsilon \leq 3$ 。

4.3.5 游戏矢量旋转检测

游戏矢量旋转角度功能在所有的传感器相关功能中对于数据的敏感程度最高,实现该功能的 APP 通常不会对传感器数据进行太多的运算,因此不会产生由复杂运算而造成的精度损失。依据应用场景,

我们将游戏矢量角度旋转角度分为 2 种类型: 1) 射击、驾驶类游戏, 对于旋转角度以及响应时间要求严格, 通过陀螺仪实现; 2) 不倒翁等简单游戏, 通常由加速度传感器实现.

首先讨论第 1 类应用场景. 实时的旋转角度由旋转速度与时间间隔的乘积计算, 陀螺仪数值的混淆误差会与时间混淆误差累积. 我们比较设备旋转情况下由混淆数据计算的旋转角度与准确角度之间的误差. 图 14 显示了 10 种混淆程度下的角度误差统计情况:

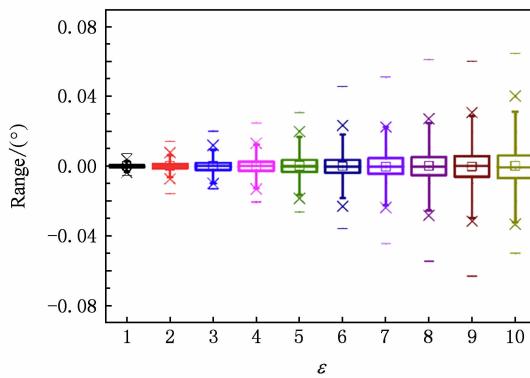


Fig. 14 Error of vector rotation angle with ϵ range from 1 to 10 based on gyroscope

图 14 基于陀螺仪的矢量旋转角度误差与 ϵ 关系

从图 14 可以看出, 第 1 类游戏矢量角度算法由于积累了数值与时间 2 方面的误差, 算法对于数据混淆的改变非常敏感, 混淆程度为 $\epsilon=1$ 时的矢量角度几乎与真实矢量角度相同, 但是, 当 $\epsilon=3$ 时开始出现较大范围的误差, 最大误差为 0.02° . 随着 ϵ 的加大, 旋转角度的误差程度明显增加, 误差范围逐渐扩大, 最大误差接近 0.07° . 仅观察误差数值, 混淆后的数据对于矢量旋转角度的计算的影响似乎并不严重, 在混淆程度 $\epsilon=10$ 时, 最大的误差也没有超过 0.1° , 为具体分析误差的影响程度, 图 15 中显示了 10 种混淆程度下的计算误差与实际矢量角度之间的误差比率.

根据图 15 中的结果, $\epsilon=1$ 时矢量角度的误差率接近于 0, 最大误差率约为 0.2%, 在 $\epsilon=5$ 以下时最大误差率稳定在 6% 以下, 平均误差率同样趋近于 0. ϵ 增加到 6 之后, 误差率上限突然上升至 14%, 虽然整体误差率均值仍然保持较低水平, 但是小概率出现的大误差依然会给应用程序功能完整性以及用户体验造成不良影响. 该项实验中的时间混淆范围为 $-3.26 \sim 5.25$ ms, 平均时延 0.0024 ms. 由陀螺

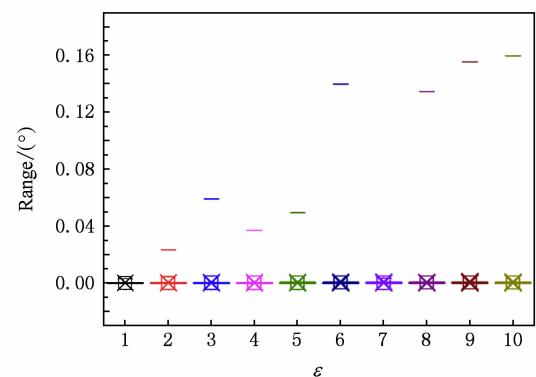


Fig. 15 Error rate of vector rotation angle with ϵ range from 1 to 10 based on gyroscope

图 15 基于陀螺仪的矢量旋转角度误差率与 ϵ 关系

仪为基础的第 1 类矢量旋转检测功能的混淆程度的承受范围建议为 $1 \leq \epsilon \leq 3$.

第 2 类场景中矢量旋转角度基于加速度传感器计算, 附录 A 中不倒翁游戏为典型样例. 通过比较不同混淆程度的加速度传感器数据对于旋转角度计算的误差, 分析各类算法对于数据混淆的敏感程度. 混淆程度 $1 \leq \epsilon \leq 10$ 所对应的误差统计结果如图 16 所示:

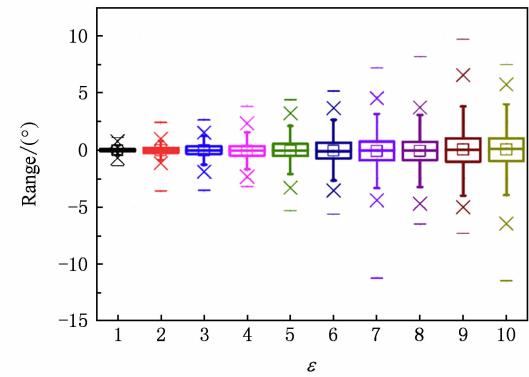


Fig. 16 Error of vector rotation angle with ϵ range from 1 to 10 based on accelerometer

图 16 加速度传感器矢量旋转角度误差与 ϵ 关系

通过与图 14 中的误差统计结果对比, 由加速度传感器计算矢量旋转角度的算法在鲁棒性上明显低于由陀螺仪实现的算法, 即使在 $\epsilon=2$ 的情况下的最大误差达到了 $\pm 2^\circ$ 左右, 该功能实现算法在计算过程中放大了由数据混淆带来的噪声.

考虑到第 2 类场景用户体验对响应时间以及精度要求与第 1 类场景相比较低, 定义超出 $\pm 1^\circ$ 量化范围以外的计算结果为错误结果, 进行进一步分析, 统计错误率以分析误差的分布情况, 结果如图 17 所示.

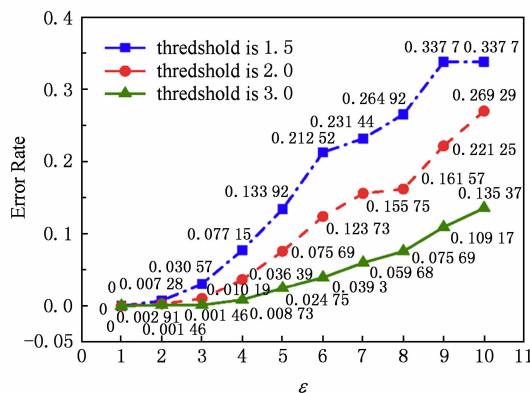


Fig. 17 Error rate of vector rotation angle based on accelerometer with 3 thresholds with ϵ range from 1 to 10

图 17 矢量旋转角度的 3 种阈值对应误差与 ϵ 关系

阈值分别选择 1.5° , 2° 和 3° . 阈值 3° 所对应的错误率在 $\epsilon < 4$ 时增长不明显, 说明混淆程度在 4 以下时绝大部分误差不超过 $\pm 3^\circ$, $\epsilon > 4$ 后错误率缓慢上升, 误差范围逐渐扩大, $\epsilon = 9$ 时误差超过 $\pm 3^\circ$ 的样本已经超过 10%. 根据阈值为 1.5° 与 2° 的统计分布, 由数据混淆而导致的矢量旋转角度偏差在混淆程度 $\epsilon \leq 2$ 以下时普遍低于 1.5° , $\epsilon > 3$ 后数据混淆对算法造成较大的干扰. 综上分析, 虽然第 2 类场景中用户体验对于旋转角度的要求较低, 但是算法本身的鲁棒性弱于基于陀螺仪的矢量旋转算法, 数据混淆承受范围建议为 $1 \leq \epsilon \leq 2$, 实验中的时间混淆范围为 $-4.59 \sim 4.02$ ms, 平均时延 0.017 ms.

5 传感器侧信道对抗效果分析

5.1 传感器侧信道防御实验

第 4 节分析了传感器数据混淆对于功能实现以及用户体验的影响, 本节中将讨论传感器数据混淆方案对抗防御传感器侧信道的实际效果.

我们从代表性的传感器侧信道中筛选出 9 种典型攻击方案进行复现, 其中, 前 7 种传感器侧信道为“输入侧信道”, 即以传感器数据做为媒介窃取用户的敏感输入信息, 后 2 种侧信道为“追踪侧信道”, 追踪侧信道通过提取移动设备物理传感器的固有误差实现对设备的识别, 进而达到用户追踪的目的. 实验过程中所涉及的侧信道详细信息如表 1 所示.

考虑到 Android 碎片化和传感器硬件差异的影响, 我们选用了 5 部不同的智能手机重复进行实验, 覆盖了多种芯片、厂商、设备型号和系统版本. 实验中所使用的设备信息如表 2 所示.

Table 1 List of Side Channels in Defence Experiment

表 1 防御实验涉及侧信道列表

Side Channel	Sensor
Accessory ^[14]	Accelerometer
Signature ^[32]	Linear-Accelerometer Gyroscope Orientation
Pinlogger ^[33]	Accelerometer Orientation
Textlogger ^[34]	Accelerometer Gyroscope
Tapprints ^[35]	Accelerometer Gyroscope
Taplogger ^[5]	Accelerometer Gyroscope
InputExtraction ^[36]	Linear-Accelerometer Orientation
MobileTracking ^[25]	Accelerometer
TrackingExploring ^[37]	Linear-Accelerometer Accelerometer Gyroscope

Table 2 List of Experimental Devices

表 2 实验设备列表

Device	CPU	Sampling/Hz	Android
Meizu m3x	Helio P20	50	5.0
Meizu max5	Helio X10	50	4.4
MI 4c	Snapdragon 808	50	4.4
MI note3	Snapdragon 650	50	4.4
Pro 5	APQ 8064	50	4.1

首先进行输入侧信道的防御对抗实验, 我们邀请了 15 位人员参与数据收集, 每位参与者进行相同操作: 以习惯握姿在分配设备的虚拟数字键盘上随机点击按键, 实验程序会实时记录下点击操作过程中的传感器数据, 以及对应时间信息, 重复操作保证 10 个类别(数字 0~9)均包含足够样本. 由同一参与者收集的样本做为独立数据集, 分别在每个数据集上进行 7 种输入侧信道攻击, 以平均的单次点击输入预测成功率作为评价指标, 实验结果如表 3 所示.

表 3 中, 纵向表示侧信道类型, 横向表示不同混淆程度的数值噪声背景下单次点击输入的预测准确率. 从表 3 中结果看出, 当进行数据混淆后, 所有输入侧信道攻击的单次准确率立刻出现不同程度的下降, 并且预测准确率随混淆程度的增大而呈现稳定的下降趋势, 初始数据集中表现较优异的侧信道下降幅度更明显. 仅观测数值可能会认为数据混淆方案对抗侧信道的防御效果不足, 然而, 输入侧信道的单次点击预测误差会随着预测序列长度的增加而累积, 以单次准确率最高的 Textlogger 侧信道为例, 当攻击长度为 4 的 PIN 时, 初始情况下侧信道的平均成功率约为 32%, 即有约 9 成的概率在 6 次之内尝试攻破 PIN, 当进行了混淆程度为 $\epsilon = 1$ 的混淆

后,Textlogger 攻击 PIN 的平均成功率为 21.5%,为达到相同的攻击效果需要至少 10 次尝试,然而大

多数智能设备在自动锁屏前仅允许 5 次失败。因此,数据混淆能够非常有效地对抗主流输入侧信道攻击。

Table 3 Input Side Channel Accuracy under Data Obfuscation Environment

表 3 数据混淆环境下输入侧信道准确率

%

Side Channel	ϵ										
	0	1	2	3	4	5	6	7	8	9	10
Accessory	63	59.5	56.6	52.8	48.4	47.2	47.1	43.8	40.7	40.9	39.3
Signature	68.4	60.6	57	55	51.4	55.1	52.6	50	52.1	50	50.7
Pinlogger	64.6	59.5	58.2	56.1	51.9	51.1	49.8	50.3	50.4	48.7	47.8
Textlogger	75.1	68.1	66.1	60.6	59.8	55.3	55	55	52.7	51.4	48.3
Tapprints	71.8	65.2	63.3	59.9	55.1	53.7	53.1	51.3	50.4	49.9	48
Taplogger	32.6	32.4	33.2	28.6	25.2	24.9	25.2	24.9	23.8	23.5	19.1
InputExtraction	64.6	55.8	53.5	54.9	50.4	48.4	51.3	48.7	50.4	48.7	49.6

Note: Data obfuscation can immediately produce significant effects.

追踪侧信道的对抗实验中,使用 2 种追踪侧信道分别对 5 台实验设备进行识别,平均识别准确率如表 4 所示。

由表 4 中结果看出,数据混淆对抗追踪侧信道的情况与输入侧信道对抗实验结果相似,实施数据混淆后能够立即抑制追踪侧信道的设备识别准确率,在混淆程度为 1 的情况下,2 种侧信道攻击识别成功率分别下降了约 5 个百分点和 2.5 个百分点,

随着混淆程度逐渐增加,追踪侧信道的预测准确率持续下降。需要说明的是,实际攻击场景中攻击者需要从大量设备中识别目标设备^[25],其数量远远超过实验中的 5 台设备,因此可以预测基于数据混淆方式的防御机制能够在实际应用中有效抵抗追踪侧信道攻击。综上所述,本文提出的数据混淆防御机制在实际对抗多种类型传感器侧信道攻击时均具有良好的防御能力,防御效果随混淆程度的增加而增强。

Table 4 Tracking Side Channel Accuracy under Data Obfuscation Environment

表 4 数据混淆环境下追踪侧信道准确率

%

Side Channel	ϵ										
	0	1	2	3	4	5	6	7	8	9	10
MobileTracking	96.7	92.3	92.2	92.2	91.5	91.4	91.3	91.3	91.2	90.6	90
TrackingExploring	98	95.5	95.5	95.1	95.2	95	94.8	94.5	94.3	94	93

5.2 传感器相关功能建议混淆范围

综合考虑第 4 节中传感器相关功能对于数据混淆的承受能力与用户体验,以及本节中数据混淆对

于典型侧信道攻击的防御实验结果,本文给出了平衡功能完整性、用户体验与侧信道攻击防御能力的建议混淆方案,如表 5 所示:

Table 5 Proposed Scope of Sensor Data Obfuscation

表 5 传感器数据混淆程度 ϵ 建议范围

Function	Value Range	Time Range	Function	Value Range	Time Range
Screen Orientation	$\epsilon \leq 10$	$\epsilon \leq 2$	Accelerometer	\times	\times
Shaking-algorithm1	$\epsilon \leq 10$	$\epsilon \leq 2$	Pedometer	$\epsilon \leq 3$	$\epsilon \leq 2$
Shaking-algorithm2	$\epsilon \leq 10$	$\epsilon \leq 3$	Action	$\epsilon \leq 10$	$\epsilon \leq 2$
Shaking-algorithm3	$\epsilon \leq 15$	$\epsilon \leq 3$	Brightness	\times	\times
Device Orientation	$\epsilon \leq 5$	$\epsilon \leq 3$	Gesture	$\epsilon \leq 5$	$\epsilon \leq 2$
Vector Rotation-Gyroscope	$\epsilon \leq 2$	\times	Rotational Direction	$\epsilon \leq 3$	$\epsilon \leq 2$
Vector Rotation-Accelerometer	$\epsilon \leq 3$	$\epsilon \leq 3$	Others	\times	\times

Note: “ \times ” means no data obfuscation.

6 相关工作

6.1 物理传感器侧信道攻击

Cai 等人于 2011 年首次详细地提出了利用 Android 方向传感器实现以窃取用户点击屏幕位置的隐私泄露攻击方案^[4]. 作者利用点击过程中设备转动的细微角度变化作为点击事件的特征. 随后, Xu 等人在 TouchLogger 的基础上加入了加速度传感器, 实现了一种更加稳定的 PIN 攻击方法 TapLogger^[5]. 作者发现用户在常用设备上进行点击输入时, 除了设备的旋转角度以外, 加速度变化上也存在一定规律, 通过统计点击时间内加速度的波动以及时间间隔, 可以使侧信道攻击准确性上升. 上述侧信道攻击方案只能应用于通用数字键盘, ACCessory^[14] 通过提取用户点击过程中加速度传统统计特征, 实现针对英文全键键盘的点击定位攻击. 研究证明, 除加速度传感器与方向传感器以外, 陀螺仪也可以被用于定位用户点击屏幕的具体位置, 例如 TapPrints^[35]. 由于点击过程十分短暂, 针对单一点击事件的攻击方法对于设备的性能要求较高, 鲁棒性有所欠缺, 文献[38]将研究的对象从单一的点击事件转移到了完整的输入序列, 作者认为用户对于习惯的输入序列具有独特的输入规律, 这种规律可以由输入过程中的加速度反映. 与其思想类似, 文献[39]利用动态时间规整(dynamic time wrap, DTW)对相同输入序列进行识别, 仅需要极少训练样本即可达到可观的攻击效果.

除了以输入信息为目标侧信道, 物理传感器还被用于身份与地理位置信息泄露. 不同厂商生产的传感器芯片存在独有的误差分布, 可以被用于对设备进行追踪. AccelPrint^[40] 利用加速度传感器中的固有缺陷实现了用户追踪. 类似的, 文献[25]与文献[41]分别从不同角度阐述了如何利用 Android 物理传感器的硬件误差构建设备的指纹信息, 以达到用户定位的目的. 除追踪以外, 物理传感器也被用于身份验证. Shahzad 等人^[42] 根据用户在触摸屏上执行的操作产生的加速度特征进行用户认证, 动作既可以是手势, 也可以是简单交互.

6.2 Android 应用恶意行为分析

Android 应用恶意行为分析主要依赖于敏感 API 调用以及特殊权限申请. Zhang 等人^[11] 基于 APP 中敏感 API 调用构建基于上下文的行为图, 以此作为依

据判断 APP 中是否存在威胁操作. AndroidLeaks^[43] 是一款检测 APP 潜在隐私泄露的静态分析工具, 通过追踪敏感数据的流动判断隐私信息是否被泄露. 研究者通过软件静态分析方法, 对 APP 中敏感 API 的调用过程进行分析, 自动生成目标应用敏感 API 调用的自然语言描述, 该工作解决了 APP 权限申请信息生涩难懂的问题, 但最终选择权仍然保留在安全意识较为薄弱的用户手中. FlowDroid^[19] 实现了上下文敏感的精确静态污点分析, FlowDroid 在 soot 的基础上, 克服了 Android 应用程序生命周期复杂等难题, 通过探索污染源与信息泄露出口之间可达路径的方式分析应用程序中可能存在的威胁. 类似的静态污点分析研究包括文献[44-46]. 为了降低系统开销, 提高检测准确率, Backes 等人^[22] 将程序切片^[47] 引入 Android 恶意行为分析并取得理想效果. 作者在传统的静态分析之前加入程序切片过程, 将感兴趣的程序片段抽取出来, 进而降低了完整应用程序分析时的系统开销. 本文综合借鉴静态污点分析与程序切片的思想, 抽取应用程序中与传感器相关的程序片段, 分析传感器数据的详细利用过程.

7 总 结

本文针对当前 Android 物理传感器侧信道防御效果不理想的问题, 提出了一种对于用户完全透明的基于差分隐私的隐私防御方案. 由于已知的 Android 物理传感器侧信道攻击大多基于机器学习, 构建信道时非常依赖于目标设备中传感器数据质量, 通过在 Android 框架层对传感器原始数据进行特殊的混淆, 从而达到降低攻击成功率的效果. 进行传感器数据混淆面临 2 个困难: 1) 如何保证数据混淆行为不会影响用户体验; 2) 如何混淆原始数据以有效对抗传感器侧信道攻击. 对于第 1 个问题, 本文基于静态污点分析与程序切片的思想, 设计了一款针对 Android 物理传感器的功能分析工具 SensorTainter. SensorTainter 将传感器对象视为污染源, 追踪传感器数据流流向, 抽取出与传感器数据相关的程序片段, 分析相关功能实现算法对于输入数据的敏感程度, 通过 SensorTainter 对 47 144 个应用程序进行分析, 本文统计出了应用程序中物理传感器的使用情况以及功能分类. 针对第 2 个困难, 本文借鉴差分隐私方法, 引入 Laplace 噪声对传感

器原始数据的数值与时间进行混淆，并根据功能分类的结果提出最优的数据混淆建议范围。实验证明：本文提出的基于数据混淆的安全框架可以有效降低主流 Andorid 物理传感器攻击，并且不会对用户体验和 APP 功能完整性造成干扰。

参 考 文 献

- [1] Nahapetian A. Side-channel attacks on mobile and wearable systems [C] //Proc of IEEE CCNC'16. Piscataway, NJ: IEEE, 2016: 243–247
- [2] Das A, Borisov N, Caesar M. Exploring ways to mitigate sensor-based smartphone fingerprinting [J]. arXiv preprint, arXiv: 1503.01874, 2015
- [3] Crager K, Maiti A, Jadliwala M, et al. Information leakage through mobile motion sensors: User awareness and concerns [C] //Proc of the EuroUSEC'17. Reston, VA: ISOC, 2017
- [4] Cai Liang, Chen Hao. TouchLogger: Inferring keystrokes on touch screen from smartphone motion [C] //Proc of the 6th USENIX Workshop on HotSec. New York: ACM, 2011: 9–15
- [5] Xu Zhi, Bai Kun, Zhu Sencun. TapLogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors [C] //Proc of the 15th ACM Conf on Security and Privacy in Wireless and Mobile Networks. New York: ACM, 2012: 113–124
- [6] Kantar Worldpanel. Grocery market share [EB/OL]. [2017-12-01]. <https://www.kantarworldpanel.com/global/grocery-market-share>
- [7] Dai Hua, Yang Kang, Xiao Fu, et al. An energy-efficient and privacy-preserving range query processing in two-tiered wireless sensor networks [J]. Journal of Computer Research and Development, 2015, 52(4): 983–993 (in Chinese)
(戴华, 杨庚, 肖甫, 等. 两层传感网中能量高效的隐私保护范围查询方法[J]. 计算机研究与发展, 2015, 52(4): 983–993)
- [8] Dai Hua, Yang Kang, Qin Xiaolin, et al. Privacy-preserving top- k query processing in two-tiered wireless sensor networks [J]. Journal of Computer Research and Development, 2013, 50(6): 1239–1252 (in Chinese)
(戴华, 杨庚, 秦小麟, 等. 面向隐私保护的两层传感网 Top- k 查询处理方法[J]. 计算机研究与发展, 2013, 50(6): 1239–1252)
- [9] Pistoia M. Program analysis for mobile application integrity and privacy enforcement [C] //Proc of ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2015: 1698–1699
- [10] Zhang Mu, Duan Yue, Feng Qian, et al. Towards automatic generation of security-centric descriptions for Android apps [C] //Proc of ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2015: 518–529
- [11] Zhang Mu, Duan Yue, Yin Heng, et al. Semantics-aware Android malware classification using weighted contextual API dependency graphs [C] //Proc of ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2014: 1105–1116
- [12] Qi Wen, Ding Wanfu, Wang Xinyu, et al. Construction and mitigation of user-behavior-based covert channels on smartphones [J]. IEEE Trans on Mobile Computing, 2017, 17(1): 44–57
- [13] Maiti A, Jadliwala M, Bilogrevic I, et al. (Smart) watch your taps: Side-channel keystroke inference attacks using smartwatches [C] //Proc of the 17th ACM Int Symp on Wearable Computers. New York: ACM, 2015: 27–30
- [14] Owusu E, Han Jun, Das S, et al. ACCessory: Password inference using accelerometers on smartphones [C] //Proc of the 12th Workshop on Mobile Computing Systems & Applications. New York: ACM, 2012; Article No. 9
- [15] Conti M, Nguyen V T N, Crispo B. CrePE: Context-related policy enforcement for Android [G] //LNCS 6531: Proc of the 13th Int Conf on Information Security. Berlin: Springer, 2010: 331–345
- [16] Lei Lingguang, Jing Jiwu, Wang Yuewu, et al. A behavior-based system resources access control scheme for Android [J]. Journal of Computer Research and Development, 2014, 51(5): 1028–1038 (in Chinese)
(雷灵光, 荆继武, 王跃武, 等. 一种基于行为的 Android 系统资源访问控制方案[J]. 计算机研究与发展, 2014, 51(5): 1028–1038)
- [17] Ryu Y S, Koh D H, Aday B L, et al. Usability evaluation of randomized keypad [J]. Journal of Usability Studies, 2010, 5(2): 65–75
- [18] Shrestha P, Mohamed M, Saxena N. Slogger: Smashing motionbased touchstroke logging with transparent system noise [C] //Proc of the 9th ACM Conf on Security & Privacy in Wireless and Mobile Networks. New York: ACM, 2016: 67–77
- [19] Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android APPs [J]. ACM Sigplan Notices, 2014, 49(6): 259–269
- [20] Lam P, Bodden E, Lhoták O, et al. The Soot framework for Java program analysis: A retrospective [C] //Proc of CETUS'11. New York: ACM, 2011
- [21] Dwork C, McSherry F, Nissim K, et al. Calibrating noise to sensitivity in private data analysis [G] //LNCS 3876: Proc of Theory of Cryptography. Berlin: Springer, 2006: 265–284
- [22] Backes M, Bugiel S, Derr E, et al. R-droid: Leveraging Android APP analysis with static slice optimization [C] //Proc of the 11th ACM Asia Conf on Computer and Communications Security. New York: ACM, 2016: 129–140

- [23] Hoffmann J, Ussath M, Holz T, et al. Slicing droids: Program slicing for smali code [C] //Proc of the 28th Annual ACM Symp on Applied Computing. New York: ACM, 2013: 1844–1851
- [24] Spreitzer R, Moonsamy V, Korak T, et al. SoK: Systematic classification of side-channel attacks on mobile devices [J]. arXiv preprint, arXiv:1611.03748, 2016
- [25] Das A, Borisov N, Caesar M. Tracking mobile Web users through motion sensors: Attacks and defenses [C] //Proc of the 22nd NDSS'16. Reston, VA: ISOC, 2016
- [26] Bojinov H, Michalevsky Y, Nakibly G, et al. Mobile device identification via sensor fingerprinting [J]. arXiv preprint, arXiv:1408.1416, 2014
- [27] Bodden E. Inter-procedural data-flow analysis with ifds/ide and soot [C] //Proc of ACM SOAP'12. New York: ACM, 2012: 3–8
- [28] Ling Zhen, Luo Junzhou, Chen Qi, et al. Secure fingertip mouse for mobile devices [C] //Proc of IEEE INFOCOM'16. Piscataway, NJ: IEEE, 2016: 376–385
- [29] Jiao Sibei, Ying Lingyun, Yang Zhi, et al. An anti-obfuscation method for detecting similarity among Android applications in large scale [J]. Journal of Computer Research and Development, 2014, 51(7): 1446–1457 (in Chinese)
(焦四辈, 应凌云, 杨轶, 等. 一种抗混淆的大规模Android应用相似性检测方法[J]. 计算机研究与发展, 2014, 51(7): 1446–1457)
- [30] Tang Benxiao, Wang Zhibo, Wang Run, et al. Niffler: A context-aware and user-independent side-channel attack system for password inference [J]. Wireless Communications and Mobile Computing, 2018; Article ID 4627108
- [31] Martin R, Demme J, Sethumadhavan S. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks [J]. ACM SIGARCH Computer Architecture News, 2012, 40(3): 118–129
- [32] Mehrnezhad M, Toreini E, Shahandashti S F, et al. Touchsignatures: Identification of user touch actions and pins based on mobile sensor data via javascript [J]. Journal of Information Security and Application, 2016, 26: 23–38
- [33] Mehrnezhad M, Toreini E, Shahandashti S F, et al. Stealing pins via mobile sensors: actual risk versus user perception [J]. International Journal of Information Security, 2018, 17 (3): 291–313
- [34] Ping Dan, Sun Xin, Mao Bing. Textlogger: Inferring longer inputs on touch screen using motion sensors [C] //Proc of the 8th ACM Conf on Security & Privacy in Wireless and Mobile Networks. New York: ACM, 2015
- [35] Miluzzo E, Varshavsky A, Balakrishnan S, et al. Tapprints: Your finger taps have fingerprints [C] //Proc of the 10th ACM Int Conf on Mobile Systems, Applications, and Services. New York: ACM, 2012: 323–336
- [36] Shen Chao, Pei Shichao, Yang Zhenyu, et al. Input extraction via motion sensor behavior analysis on smartphones [J]. Computers & Security, 2015, 53(c): 143–155
- [37] Das A, Borisov N, Chou E. Every move you make: Exploring practical issues in smartphone motion sensor fingerprinting and countermeasures [J]. Proceedings on Privacy Enhancing Technologies, 2018, 2018(1): 88–108
- [38] Aviv A J, Sapp B, Blaze M, et al. Practicality of accelerometer side channels on smartphones [C] //Proc of the 28th Annual Computer Security Applications Conf. New York: ACM, 2012: 41–50
- [39] Liu Jiayang, Zhong Lin, Wickramasuriya J, et al. uWave: Accelerometer-based personalized gesture recognition and its applications [J]. Pervasive and Mobile Computing, 2009, 5 (6): 657–675
- [40] Dey S, Roy N, Xu Wenyuan, et al. AccelPrint: Imperfections of accelerometers make smartphones trackable [C] //Proc of the NDSS'14. Reston, VA: ISOC, 2014
- [41] Amerini I, Becarelli R, Caldelli R, et al. Smartphone fingerprinting combining features of on-board sensors [J]. IEEE Trans on Information Forensics and Security, 2017, 12 (10): 2457–2466
- [42] Shahzad M, Liu A X, Samuel A. Behavior based human authentication on touch screen devices using gestures and signatures [J]. IEEE Trans on Mobile Computing, 2017, 16 (10): 2726–2741
- [43] Gibler C, Crussell J, Erickson J, et al. AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale [G] //LNCS 7344: Proc of the 5th Int Conf on Trust and Trustworthy Computing. Berlin: Springer, 2012: 291–307
- [44] Mann C, Starostin A. A framework for static detection of privacy leaks in Android applications [C] //Proc of the 27th Annual ACM Symp on Applied Computing. New York: ACM, 2012: 1457–1462
- [45] Lu Long, Li Zichun, Wu Zhenyu, et al. Chex: Statically vetting Android APPs for component hijacking vulnerabilities [C] //Proc of the 12th ACM Conf on Computer and Communications Security. New York: ACM, 2012: 229–240
- [46] Gordon M I, Kim D, Perkins J H, et al. Information flow analysis of Android applications in droidSafe [C] //Proc of the NDSS'15. Reston, VA: ISOC, 2015
- [47] Liang Donglin, Harrold M J. Slicing objects using system dependence graphs [C] //Proc of IEEE ICSM 1998. Piscataway, NJ: IEEE, 1998: 358–367



Tang Benxiao, born in 1991. PhD candidate. Student member of the CCF. His main research interests include mobile privacy, program analysis, and machine learning.



Wang Lina, born in 1964. PhD, professor, PhD supervisor. Member of the CCF. She has authored 50 research papers and won 12 authorized patents. Her main research interests include system security and steganalysis.



Wang Run, born in 1991. PhD candidate. Student member of the CCF. His main research interests include mobile security and privacy, machine learning security.



Zhao Lei, born in 1985. Received his BE degree and PhD degree both in computer science at Wuhan University. Associate professor at the School of Cyber Science and Engineering, Wuhan University. His main research interests include software security, software analysis, and system protection.



Wang Danlei, born in 1992. Received his ME degree in information security at Wuhan University. His main research interests include machine learning and information content security.

附录 A. 传感器相关功能程序片段抽取样例.

1) 不倒翁 APP 部分源码

```
public class BudaowActivity extends Activity
    implements SensorEventListener{
```

```
private GSensitiveView gsView;
private SensorManager sm;
```

```
@Override
public void onCreate(Bundle savedInstanceState{
    State){
```

```
.....
gsView=new GSensitiveView(this);
setContentView(gsView);
sm=(SensorManager) getSystemService(
    SENSOR_SERVICE);
sm.registerListener(this,sm.getDefault-
    Sensor(Sensor.TYPE_ACCELEROMETER),
    SensorManager.SENSOR_
    DELAY_NORMAL);
```

```
}
```

```
.....
public void onSensorChanged(SensorEvent
    event){
if (Sensor.TYPE_ACCELEROMETER !=
    event.sensor.getType()){
    return;
}
float[] values=event.values;
float ax=values[0];
```

```
float ay=values[1];
double g=Math.sqrt(ax*ax+ay*ay);
double cos=ay/g;
if(cos>1){
    cos=1;
} else if(cos<-1){
    cos=-1;
}
double rad=Math.acos(cos);
if(ax<0){
    rad=2 * Math.PI - rad;
}
int uiRot=getWindowManager().
    getDefaultDisplay().getRotation();
double uiRad=Math.PI/2 * uiRot;
rad-=uiRad;
gsView.setRotation(rad);
}

.....
private static class GSensitiveView extends
    ImageView{

private Bitmap image;
private double rotation;
private Paint paint;
.....
@Override
protected void onDraw(Canvas canvas){
    double w=image.getWidth();
    double h=image.getHeight();
```

```

Rect rect=new Rect();
getDrawingRect(rect);
int degrees=(int)(180 * rotation/
Math.PI);
canvas.rotate(degrees,rect.width()/2,
rect.height()/2);
canvas.drawBitmap(image,(float)
((rect.width()-w)/2),
(float)((rect.height()-h)/2),
paint);
}

public void setRotation(double rad){
rotation=rad;
invalidate();
}
}
}

```

2) 功能抽取结果:

```

$r1:=@parameter0; android.hardware.
SensorEvent
$r3=$r1.<android.hardware.SensorEvent:
    android.hardware.Sensor sensor>
$i0=virtualinvoke $r3.<android.hardware.
Sensor:int getType()>()
$r2=$r1.<android.hardware.SensorEvent:
    float[] values>
if 1==$i0 goto $r2=$r1.<android.
hardware.SensorEvent:float[] values>
$f0=$r2[0]
$f1=$r2[1]
$f2=$f0*$f0
$f3=$f1*$f1
$f2=$f2+$f3
$d1=(double)$f2
$d1=staticinvoke<java.lang.Math:double
sqrt(double)>($d1)
return
$d0=(double)$f1
$d1=$d0/$d1
$b1=$d1 cmpl 1.0
$b1=$d1 cmpg -1.0

```

```

$d1=1.0
$d0=staticinvoke<java.lang.Math:double
acos(double)>($d1)
$d1=-1.0
$d1=$d0
$b1=$f0 cmpg 0.0F
$d1=6.283185307179586-$d0
$r4=virtualinvoke $r0.<com.example.
budaow.Budaow:android.view.WindowManager
getManager()>()
if $b1≥0 goto $r4=virtualinvoke
$r0.<com.example.budaow.Budaow:
    android.view.WindowManager getWindow-
Manager()>()
$r5=interfaceinvoke $r4.<android.view.
 WindowManager:android.view.Display
getDefaultDisplay()>()
$i0=virtualinvoke $r5.<android.view.
Display:int getRotation()>()
$d0=(double)$i0
$d0=1.5707963267948966*$d0
$d1=$d1-$d0
if $b1≤0 goto $b1=$d1 cmpg -1.0
$r6=$r0.<com.budaow.Budaow:com.
example.budaow.Budaow$GSensitiveView
gsView>
virtualinvoke $r6.<com.budaow.Budaow$GSensitiveView:
void setRotation(double)>(
    $d1)
if $b1≥0 goto $d0=staticinvoke<java.
lang.Math:double acos(double)>($d1)

```

附录 B. 摆晃检测算法示例。

- 1) 摆晃检测算法 1(s-algorithm 1)

```

boolean issshake=false;
if (Math.abs(sensor.values[0])>threshold ||
    Math.abs(sensor.values[1])>threshold ||
    Math.abs(sensor.values[2])>threshold)
    issshake=true;

```

- 2) 摆晃检测算法 2(s-algorithm 2)

```

boolean issshake=false;
float deltaX=sensor.values[0]-lastX;
float deltaY=sensor.values[1]-lastY;

```

```

float deltaZ=sensor.values[2]-lastZ;
if ((Math.sqrt(deltaX * deltaX+deltaY *
deltaY+deltaZ * deltaZ)/diffTime *
10 000)>shakeThreshold)
isshake=true;

```

3) 摆晃检测算法 3(s-algorithm 3)

```

boolean isshake=false;
float tp=(float)((x * x+y * y+z * z)/
96.170387);
if(tp>shakeThreshold)
isshake=true;

```

附录 C. Android 系统框架层传感器平移混淆部分核心算法示例

```

android_hwre_SensorManager.cpp
class LaplaceNoise{
public:
    LaplaceNoise(double b):mu(0.0),beta(b){
        srand((unsigned)time(NULL));
    }
    LaplaceNoise(double m,double b):mu(m),
    beta(b){
        srand((unsigned)time(NULL));
    }
    LaplaceNoise(unsigned int seed,double m,
    double b):mu(m),beta(b){
        srand(seed);
    }
    double getLocation(){return mu;}
    double getScale(){return beta;}
    double noise(){
        double p=rand()/double(RAND_MAX);
        double x=(p>0.5)? -log(2.0-2.0 *
        p):log(2.0 * p);
        return mu+beta * x;
    }
private:
    double mu;
    double beta;
};
class MixLayer{
public:
    MixLayer(unsigned int bs,unsigned int dv,

```

```

        unsigned int dt,unsigned int di,unsigned
        int f):
    bufferSize(bs),start(0),end(0),deltaV
    (dv),deltaT(dt),dimenton(di),
    frequency(f)
    {
        buffer=new ASensorEvent[bufferSize+1];
        max=new double[dimenton];
        min=new double[dimenton];
    }
    ~MixLayer(){
        delete []buffer;
        delete []max;
        delete []min;
    }
    unsigned int getBufferSize(){return bufferSize;}
    unsigned int getDeltaV(){return deltaV;}
    void setDeltaV(unsigned int dv){deltaV=dv;}
    unsigned int getDeltaT(){return deltaT;}
    void setDeltaT(unsigned int dt){deltaT=dt;}
    unsigned int getFrequency(){return frequency;}
    void setFrequency(unsigned int f){frequency
        =f;}
    bool isEmpty(){return start==end;}
    bool isFull(){return (end+1)% (bufferSize+
        1)==start;}
    bool push(ASensorEvent se){
        if(isFull())return false;
        buffer[end]=se;
        end=(end+1)% (bufferSize+1);
        return true;
    }
    bool pop(){
        if(isEmpty())return false;
        start=(start+1)% (bufferSize+1);
        return true;
    }
    void calcM(){
        for(unsigned int i=0;i<dimenton;i++){
            max[i]=0.0;
            min[i]=0.0;
            for(unsigned int j=start;j != end;j =
            (j+1)% (bufferSize+1)){

```

```

if(j==start){
    max[i]=buffer[j].data[i];
    min[i]=buffer[j].data[i];
}
else{
    if (buffer[j].data[i]>max[i])
        max[i]=buffer[j].data[i];
    if (buffer[j].data[i]<min[i])min
        [i]=buffer[j].data[i];
}
}

ASensorEvent * retMixedData(){
if(isEmpty())return NULL;
ASensorEvent * out=buffer+start;
calcM();
for(unsigned int i=0;i<dimention;i++){
    double betaV=(max[i]-min[i])*dimention/deltaV;
    LaplaceNoise lpV(betaV);
    double noiseV=lpV.noise();
    out->data[i]+=noiseV;
    double delay=1.0/frequency;
    double betaT=delay/deltaT;
    LaplaceNoise lpT(betaT);
    double noiseT=lpT.noise();
    double tout=out->timestamp+noiseT;
    double tlast=buffer[(end+bufferSize)%bufferSize+1].timestamp;
    while(tout<tlast&&noiseT>delay){
        noiseT=lpT.noise();
        tout=out->timestamp+noiseT;
    }
    if (tout<time(NULL))out->timestamp=tout;
}
return out;
}

private:
unsigned int bufferSize;
unsigned int start,end;
ASensorEvent * buffer;
unsigned int deltaV,deltaT;
unsigned int dimention;
double * max, * min;
unsigned int frequency;
}

```