无法摆脱的追踪

当人们谈论“手机隐私”时，往往默认指的是是否授权摄像头、麦克风、定位。但在实际的移动互联网生态中，真正决定一个用户是否“可被识别”的，不是这些高敏感权限，而是藏于设备系统、网络协议、以及广告SDK背后的那些**身份锚点。**

Android系统看似不断提升隐私防护能力，从 Android 8 到 Android 15，多次更新都在强调限制非必要的设备信息访问、收紧权限申请流程，但依然未能解决一个核心问题：

> “即使你不登录、不授权、不使用账户，系统依然知道你是谁。”

这是因为，Android生态中存在多种设计与实现层级上的**“系统性追踪路径”**。本篇文章将以系统演化为起点，深入揭示用户在Android中为何“身份注定暴露”。

Android 隐私机制演进：表面进步，结构性留缝

让我们先从Android系统自身说起。Google并非对隐私毫无作为，事实上，Android系统从8.0开始便陆续进行多项隐私策略更新：

Android 版本	关键隐私更新	目标与效果
Android 8	引入Android_id、MAC地址随机化	防止通过网络信息、物理地址识别设备
Android 10	限制非重置型ID、后台定位限制	减少持久身份的泄露，提升定位透明度
Android 12	可以关闭广告id	限制广告追踪、提升用户主动权
Android 13	增加广告id权限	防止隐形读取敏感内容、增强感知能力

但这些提升主要集中在“显性权限”层面——即用户可以看到、能控制、能够理解的部分。而真正能够持续识别用户身份的，是更隐蔽、设计层面就存在的“身份锚点”。这些锚点，构成了用户在Android设备中**不可逃避的数字指纹体系。**

第一锚点：Android ID —— 被误解的“匿名身份”

Android ID 是设备在首次启动时生成的唯一标识符，原本意图用于替代MAC地址、IMEI等硬件ID，防止App通过不可更改的标识符持续追踪用户。

设计初衷

- 每个App空间内唯一；
- 同一开发者签名的App共享ID；
- 可通过恢复出厂设置重置。

实际问题

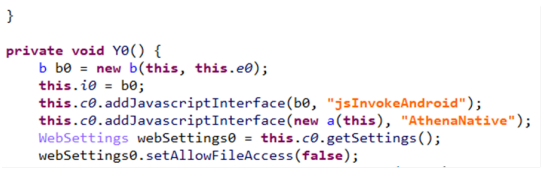
- 在部分设备实现中，系统App的Android ID可被普通App间接读取；
- 即便卸载App、重装，ID仍然相同；
- 用户对Android ID无感知、无入口查看、更无法控制。

这些问题导致了如果一个系统app的Android_id在设计上可以被其他app任意获取（android_id在很多场景被认为是无法有效的在系统层面实现全局追踪），由于系统app不可能被卸载，导致了这个系统app的Android_id可以当作一个非常有效的全局用户身份追踪唯一id。

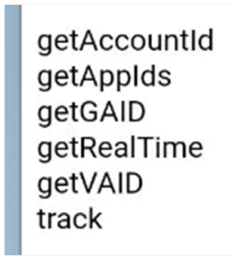
安全研究发现

系统App若开放特定JSB（JavaScript Bridge）、Binder接口，**Android ID 可被绕过权限体系读取**，成为事实上的“类永久ID”。

案例：某厂商B的系统App暴露JSB接口，任意App可通过WebView加载deeplink直接访问Android ID，绕过权限校验。



```
}
private void Y0() {
    b b0 = new b(this, this.e0);
    this.i0 = b0;
    this.c0.addJavascriptInterface(b0, "jsInvokeAndroid");
    this.c0.addJavascriptInterface(new a(this), "AthenaNative");
    WebSettings webSettings0 = this.c0.getSettings();
    webSettings0.setAllowFileAccess(false);
```



```
getAccountId
getAppIds
getGAID
getRealTime
getVAID
track
```

第二锚点：广告ID体系——GAID 与 OAID 的双重追踪网络

在当前的数字广告生态中，**广告ID（Advertising ID）**是支撑跨App、跨平台用户识别的核心机制。它以一种看似“可重置”的方式，被广泛用于追踪用户行为、构建画像，并为精准投放提供支撑。

在Android生态中，广告ID主要分为两类：**GAID**（Google Advertising ID）和 **OAID**（Open Anonymous ID），它们共同组成了一个冗余但强大的追踪体系。

一）GAID 与 OAID 的并存：设备中的双重追踪机制

- **GAID**：由Google提供，是Google Play生态中标准的广告ID。
- **OAID**：由部分国内设备厂商开发，用于在无Google服务环境中实现广告标识。

现实中，大多数国产Android手机中都**同时存在GAID与OAID**，也就是说，一个应用可以获取两个唯一ID，任何一个被限制，另一个都能“顶上”。

📌 图表 1：各主流厂商对广告ID的设置支持情况

	OAID			GAID			
brand	OAID settings	restriction tracing is allowed	user authorization required	GAID exists	GAID settings	restriction tracing is allowed	user authorization required
A	Yes-8	Yes	No	No	-	-	-
B	Yes-3	Yes	No	No	-	-	-
C	Yes-5	Yes	No	Yes	Yes-3	Yes	No
D	Yes-4	Yes（but not work）	No	Yes	Yes-5	Yes	No
E	Yes-4	Yes（but not work）	No	Yes	Yes-5	Yes	No
F	Yes-4	Yes（but not work）	No	Yes	Yes	No UI	No
G	Yes-4	Yes	Yes	Yes	Yes-5	Yes（but not work）	No
H	Yes-5	Yes	No	Yes	No	No UI	No

二）OAID的“不可见”与“不可控”

- 大部分设备**默认允许应用读取OAID**；
- 虽然设置中有“限制追踪”开关，但很多设备开启后**系统依旧返回真实OAID**；
- 设置路径极为复杂，有的需点击**8次菜单项**才能找到；
- 存在明显“黑暗模式UI”设计：刻意让用户难以操作关闭OAID。

三）GAID 的“绕过能力”与“控制缺失”

相较OAID，GAID理论上更合规，但现实依然问题重重：

- 部分设备中，GAID设置页面**完全不可见（No UI）**；
- 某些设备提供设置，但限制广告追踪后依旧能返回有效GAID；
- 开发者或第三方SDK可通过接口漏洞**绕过限制直接读取。**

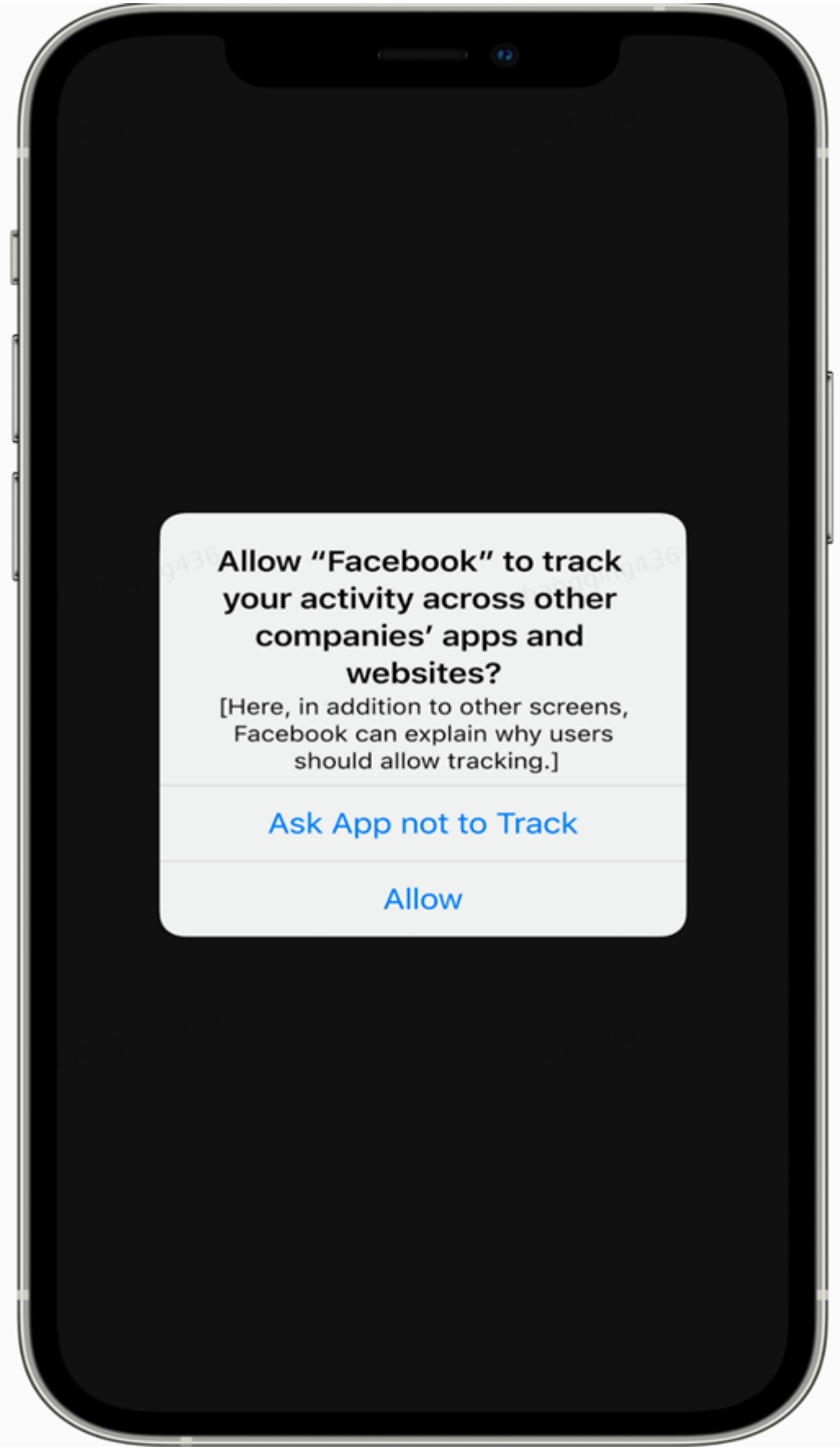
四）苹果的IDFA与ATT弹窗机制：一个对照样本

苹果自 **iOS 14.5** 起，强制要求应用获取IDFA前必须弹窗询问用户授权（ATT机制），未授权即返回空值。

- 实施后，IDFA可用率由70%降至不足10%；
- Meta等广告巨头损失高达百亿美元；

- 被称为广告行业的“隐私大地震”。

📌 **图表 2：苹果ATT授权机制弹窗示意图**



五）App Set ID：GAID的继任者，但尚未普及

Google为改善GAID的问题，引入了 **App Set ID**：

- 仅限同一开发者组织的应用共享；

- 不可跨开发者使用；
- 若用户开启“限制广告追踪”，ID将自动清零。

但App Set ID仍处推广初期，且无法在无GMS设备中使用，更无法替代OAID。

六）广告ID机制总结：表象合规，实则冗余

广告ID本应是“用户可控的标识符”，但因平台与厂商分裂、绕过路径众多，已成为结构性追踪工具。

简要总结如下：

指标	理想状态	实际情况
用户是否知情	应弹窗提示	多数不提示
用户是否可控	应可关闭	设置深藏或失效
是否唯一可识别	应可重置匿名	双ID共存冗余
是否被绕过	应受权限限制	SDK绕过普遍存在

第三锚点：数字内容版权保护——MediaDrm的另类使用

数字版权管理（Digital Rights Management，DRM）是一套用于保护数字内容版权的技术和方法。其主要目的是防止未经授权的复制、分发和使用数字内容，如音乐、电影、电子书和软件。DRM 通过加密、访问控制和许可证管理等技术手段来实现这些目标。

在 Android 平台上，DRM 的实现主要通过 MediaDrm 框架。这个框架为应用程序提供了一个接口，用于与 DRM 系统交互，保护和管理数字媒体内容的版权。

一）版权保护的需要

MediaDrm ID 最初用于识别设备，确保数字内容的许可证与特定设备绑定。这意味着只有拥有正确 MediaDrm ID 的设备才能访问和播放受保护的内容。

最初，Google收购了Widevine Technologies，随着 Android 平台的不断发展，Google 将 Widevine DRM 逐渐集成到 Android 操作系统中，使其成为保护数字内容的核心组件。Widevine 的集成使得 Android 设备能够支持多种流媒体协议和加密标准，如 MPEG-DASH、HLS、CENC 和 CMAF，从而为开发者和内容提供商提供了一种强大的工具来保护视频和音频内容。

除Widevine 方案外，其他厂商也有对应的DRM技术保护数字内容。如Microsoft PlayReady，由微软开发的 DRM 方案，广泛用于 Windows 设备和 Xbox 游戏机；苹果上的FairPlay主要用

于保护 iOS 和 macOS 设备上的内容。

二）指纹功能使用的演化

Widevine不仅提供内容加密和许可证管理，还能确保内容只能在授权设备上播放。Widevine 提供的设备唯一 ID（如 MediaDrm ID）成为识别设备的重要手段。这种唯一性确保内容许可证与特定设备绑定，防止未经授权的访问。

在 Android 中，Widevine 的 UUID 是 `-0x121074568629b532L, -0x5c37d8232ae2de13L` 。通过

```
mediaDrm = new MediaDrm(WIDEVINE_UUID);
byte[] widevineId =
mediaDrm.getPropertyByteArray(MediaDrm.PROPERTY_DEVICE_UNIQUE_ID);
```

就可以获取到对应的DEVICE_UNIQUE_ID。

现如今drmid已经与androidid，oaid等逐渐作为设备指纹中重要的组成部分在设备风险识别中使用。

三）ID的生成与唯一性

学习源码，我们会发现，Media DRM的初始化逻辑，这个初始化以及后续的流程，是由native_setup进行的，而该函数接收到的是包名。

```
public MediaDrm(@NonNull UUID uuid) throws
UnsupportedSchemeException {
        /* Native setup requires a weak reference to our object.
         * It's easier to create it here than in C++.
         */
        mAppPackageName = ActivityThread.currentOpPackageName();
        native_setup(new WeakReference<MediaDrm>(this),
                getByteArrayFromUUID(uuid), mAppPackageName);

        mCloseGuard.open("release");
    }
```

跟踪后续流程，MediaDrm只是对于每个包名，也就是一串字符串做了生成。这种对于app自身的ID很像android id，但是有很大区别。

Android id是基于应用密钥，但是MediaDrm 的生成仅仅基于字符串。

那么我们就可以通过setup一串特定字符串，让不同的app对同一特殊字符串生成指纹，从而可以对应上同一手机，也就变相达成了OAID的效果。

结语：逃无可逃的用户识别机制，技术的原罪？

Android系统的设计初衷，并非恶意侵犯用户隐私。但在现实环境中，平台管控不足、设备厂商行为分化、广告市场的强大诱因，以及SDK经济链条的背后推力，**共同构成了一个“结构性用户追踪体系”**。

无论是从Android ID到广告ID，再到系统配置、网络层的手机号识别，普通用户已经**几乎无法通过系统设置真正切断身份锚点。**

政策建议 & 技术改革方向

行动方	建议
平台方	限制系统App ID共享行为；
OEM厂商	统一广告ID控制逻辑，杜绝“限制追踪失效”等表面设置
开发者/SDK方	明示ID使用方式，禁止写入系统表用于追踪
监管机构	建立ID治理标准，纳入广告ID、衍生ID、系统行为等维度
用户（有限能力）	关闭广告id的获取