E听说中学 App
完整技术分析报告

三大核心Bug · 四大后门/漏洞 · 渠道版本对比
基于APK反编译、运行时日志、动态抓包与注入测试的全方位取证分析

EXECUTIVE SUMMARY

报告摘要：发现总览

本次分析共发现13项问题，其中致命等级5项、高危4项、中危2项，涵盖性能缺陷、安全漏洞、隐私后门和权限滥用四大类别。

致命 Critical	高危 High	中危 Medium	总计 Total
5	4	2	13

问题发现汇总表（按严重等级排序）

序号	问题类别	严重等级	影响范围
1	录音评分超时	致命	核心评分功能不可用
2	FileProvider 全盘暴露	致命	任意文件可被读取
3	外部存储资源可篡改	致命	任意代码可被执行
4	JSBridge 接口暴露	致命	原生功能可被恶意调用
5	云端控制数据采集	致命	设备信息被静默上传
6	Service Worker API 可用	高危	持久化后门风险
7	蓝牙 SCO 权限滥用	高危	CPU 过热 60 ，硬件损耗
8	存储权限静默获取	高危	用户知情权被侵犯
9	明文 HTTP 传输	高危	中间人攻击风险
10	答题卡片大小异常	高危	UI 错乱、用户体验差
11	幽灵转圈（评分状态残留）	高危	评分状态混乱、误导用户
12	渠道版本差异	中危	修复不完整，漏洞残留
13	优秀榜数据重复	中危	UI 数据重复显示

⚠ **关键风险：**录音评分功能不可用的超时问题与全盘文件暴露的FileProvider漏洞形成叠加风险，配合JSBridge接口暴露和云端静默数据采集，构成完整的攻击利用链。

Security Analysis

分析目标：版本锁定与关键矛盾

核心洞察 / Key Insight

官网版与应用商店版版本号完全一致（7.0.65.2439），但存在"高编低运"的关键矛盾——编译SDK为34（Android 14），运行SDK仅为30（Android 11）。
这种配置强制启用了requestLegacyExternalStorage等兼容模式，为后续的存储权限滥用和FileProvider配置失控埋下伏笔，是系统性安全风险的根源性成
因。

01 · 版本信息

versionName

7.0.65.2439

versionCode

2439

minSdkVersion 21 · targetSdkVersion 30

02 · 高编低运矛盾

34

compileSdkVersion
Android 14

30

targetSdkVersion
Android 11

编译版本与实际运行版本严重错位

03 · 兼容模式后果

强制开启 requestLegacyExternalStorage标志位

复刻 Android 10 以前的旧版存储权限模型，绕过 Scoped Storage 沙箱机制

应用获得宽松的外部存储访问权限

04 · 风险传导

存储权限可被静默获取，应用在后台无需用户感知即可访问外部存储

FileProvider路径配置失控，exported属性暴露内部文件目录结构

攻击者可构造恶意Intent绕过权限校验，实现未授权文件读写

形成全盘文件暴露漏洞，隐私数据面临泄露风险

Bug 1：录音评分超时（现象描述）

录音评分超时是用户反馈最集中的核心功能缺陷，呈现四种典型复现场景：主动结束后的超时提示、滑动切换触发的超时、连续录音导致的回调丢失，以及蓝牙SCO搜索引发的CPU过热。这些现象表面分散，实则共同指向主线程阻塞与协议层信号缺失的双重根因，是交互设计、线程调度和网络协议三个层面的系统性缺陷。

场景一：主动结束后的超时

最常见场景 · 占比约45%

用户完成录音后，界面长时间显示"正在评分"，最终Toast提示"打分超时"，评分结果无法返回。此场景下用户操作路径完整，但服务端响应异常，是超时问题的典型表现。

场景二：滑动切换触发超时

交互设计缺陷 · 手势冲突

用户不点击"结束"按钮，直接左右滑动切换到下一张卡片，大概率触发评分超时。此场景暴露出手势交互与评分流程的耦合问题，滑动事件未正确触发评分终止逻辑。

场景三：连续录音回调丢失

竞态条件 · 状态机异常

上一单词评分结果返回前开始下一单词录音，上一单词的评分回调被系统丢弃。此场景揭示了评分状态机未正确处理并发请求，缺乏请求队列与回调保序机制。

场景四：蓝牙关联的过热

硬件关联 · SCO协议栈

蓝牙开启且未连接设备时，CPU温度可达60℃，与SCO协议栈反复搜索蓝牙设备相关。此场景表明评分超时与系统资源争用存在关联，蓝牙扫描占用主线程资源。

BUG ANALYSIS #1

Bug 1：日志证据（主线程阻塞 10.6 秒）

Logcat捕捉到主线程IO阻塞的致命证据：BpBinder binderTransact耗时10621毫秒，Looper消息分发延迟10676毫秒，均远超16毫秒的正常阈值，直接导致评分回调无法及时分发。

LOG FIELD ANALYSIS

日志字段解析

字段	含义	结论
time=10621ms	Binder IPC调用耗时	主线程IO阻塞10.6秒
interface=IContentProvider	ContentProvider查询接口	涉及存储卡扫描
Slow dispatch took 10676ms	消息分发延迟	远超正常阈值
android.bg	后台服务消息队列	MediaScanner连接阻塞

ⓘ 双重证据印证主线程被存储操作严重阻塞

Binder IPC阻塞： binderTransact耗时10621ms，interface=IContentProvider表明涉及ContentProvider同步查询

消息队列灾难： Looper Slow dispatch 10676ms，主线程消息处理严重滞后

阈值对比： 正常单帧16ms，当前阻塞超正常值600倍以上

因果链： 主线程阻塞 → 评分回调无法分发 → 超时计时器到期 → 用户感知"打分超时"

Bug 1 Evidence

日志证据（UI 渲染灾难性卡顿）

"Choreographer报告Skipped 7812 frames，按60fps计算累积卡顿达130秒，同时SurfaceFlinger报告DequeueBuffer timeout，表明系统合成服务已无法及时获取应用渲染输出。双重证据证实SyncPracticeActivity页面存在灾难性UI线程阻塞，与10.6秒的主线程IO阻塞形成叠加效应，共同导致评分流程的全面瘫痪。"

Choreographer

7812frames

帧跳过灾难：Skipped 7812 frames ÷ 60fps ≈ 130秒 累积卡顿，用户体验完全不可接受

SurfaceFlinger

DequeueBuffer timeout

系统合成超时：SurfaceFlinger等待应用渲染超时，系统层面已无法维持正常显示合成

Activity

SyncPracticeActivity

Performance

性能死亡螺旋

Bug Analysis 01

Bug 1：日志证据（停止录音但未发送结束信号）

核心发现：日志清晰呈现协议层信号缺失：AudioRecord.stop()成功执行且SpeexEncodeRelease完成本地资源释放，但全网检索不到任何endFlag=true的网络请求记录。讯飞SDK协议明确要求客户端在最后一个音频包设置endFlag=8为true，该信号缺失导致服务器持续等待流结束，客户端超时计时器先于服务器响应到期，形成"双端等待死锁"。

关键证据链

本地录音正常终止

AudioRecord.stop() mSessionID=21697执行成功，Speex编码器正常释放

网络信号完全缺失

无任何endFlag=true的POST请求记录，协议层结束信号未发送

服务器端持续等待

因收不到流结束信号，服务器保持连接等待，无法返回评分结果

超时机制触发

客户端本地超时计时器到期，触发"打分超时"Toast提示

事件序列分析（10:13:56 - 10:13:59）

时间戳	事件	状态
10:13:56.683	startBluetoothSco() 开始录音	□正常
10:13:56.847	SpeexEncode 编码执行	□正常
10:13:59.449	AudioRecord.stop() 停止录音	□正常
10:13:59.552	SpeexEncodeRelease 释放编码器	□正常
预期时间	endFlag=true 网络请求	□缺失

四次成功事件后关键网络信号缺失，形成完整证据链

ROOT CAUSE ANALYSIS

Bug 1：日志证据（监听器覆盖导致回调丢失）

核心发现：10:13时段的完整会话记录揭示第三个技术缺陷：会话周期内本地录音流程正常（startBluetoothSco至AudioRecord.stop约2.8秒），但endFlag信号缺失导致服务器无法返回评分。更严重的是，应用使用全局单例监听器存储评分回调，新会话启动时旧监听器被覆盖，即使服务器后续返回结果也因无法路由而丢弃，形成"协议信号缺失+回调路由失效"的双重故障。

会话周期

10:13:56.683至10:13:59.449，录音时长约2.8秒，本地流程无异常，音频采集完整。

信号缺失

与前一页一致，无任何endFlag=true网络请求记录，协议层终止信号未发出。

单例缺陷

采用全局单例模式存储评分回调，未绑定TaskID或SessionID，作用域与生命周期管理失控。

致命后果

新会话启动时全局监听器被替换，旧会话评分结果返回时无路由目标，回调被系统丢弃，数据永久丢失。

Bug 1：协议层铁证（edu_ai_api_ise.proto）

ProtoBuf协议定义文件明确规范了endFlag的必需性：IseRequestStreaming消息中bool endFlag = 8字段的注释为"音频结束信号"，类型为布尔值意味着必须显式设置true或false。该协议用于讯飞语音评测服务，是单句口语评分的核心通信协议。应用层完全未实现该字段的设置逻辑，构成协议合规性严重违规。

协议文件来源

APK反编译提取的 assets/unknown/edu_ai_api_ise.proto

消息类型定位

IseRequestStreaming 为流式语音评测请求消息

关键字段定义

bool endFlag = 8，注释明确为"音频结束信号"

协议合规性

该字段为必传字段，客户端必须在最后一个音频包显式设置为 true

PROTOCOL ANALYSIS

Bug 1：协议层补充（edu_ai_api_multi_spoken.proto）

多轮口语对话协议同样强制要求 endFlag 字段，双协议对比证实这是讯飞语音服务体系的通用必传规范。

核心结论

MultiSpokenDialogRequest 消息中 bool endFlag = 9 与 bool dialogLast = 8 并列定义，前者标记音频流结束，后者标记对话轮次结束。应用在全场景下的统一遗漏表明这是系统性开发缺陷，而非个别协议的实现疏忽。

协议覆盖扩展

edu_ai_api_multi_spoken.proto 用于多轮口语对话场景，支持连续多轮交互的
语音识别与评测，覆盖口语练习、情景对话等核心业务模块。

双结束信号

dialogLast = 8 标记对话轮次结束，endFlag = 9 标记音频流结束，两个字段协
同工作确保服务端准确识别会话边界。

通用规范确认

endFlag 在单句评测和多轮对话两类核心协议中均为必传字段，该设计模式体现了讯飞语音 SDK 统一的流式传输控制策略。

系统性遗漏

所有语音服务场景均未实现 endFlag 设置，该问题跨越多个业务线持续存在，指向代码层面的架构设计缺陷而非局部实现错误。

Bug 1：协议层大搜索（30+ 个文件命中 endFlag）

批量搜索形成压倒性证据：edu_ai_api_iat.proto（语音识别）、edu_ai_api_tts.proto（语音合成）、ChineseEvaluation.proto（中文评测）、Translate.proto（机器翻译）等30余个ProtoBuf文件全部包含endFlag字段定义。该字段贯穿讯飞语音云平台的全部服务类型，是平台级通用通信规范。应用在所有协议实现中的统一缺失，构成对讯飞SDK集成规范的全面违背。

搜索范围

APK内 assets/unknown 目录全部 .proto 协议定义文件

命中规模

30+ 个文件包含 endFlag 字段定义，覆盖讯飞全产品线

服务类型覆盖

语音识别（iat）、语音合成（tts）、口语评测（ise）、机器翻译（trans）等

规范统一性

所有服务均采用相同的流式结束信号机制，endFlag 为平台级基础设施

BUG ANALYSIS #01

协议 vs 行为直接对比

对比项	协议要求（.proto）	实际行为（日志）	结果
发送 endFlag=true	必须（30+文件证实）	无任何 endFlag=true 记录	✗缺失
停止本地录音	——	AudioRecord.stop() 正常	✓正常
告知服务端流结束	必须	完全没有	✗缺失

KEY INSIGHT

30余个ProtoBuf文件统一要求endFlag=true作为流式通信的必要终止信号，但运行时日志全网检索不到任何该信号的发送记录。本地录音流程（AudioRecord.stop）与网络信号流程（endFlag）的异步设计缺失同步机制，导致服务器端流状态机持续等待，客户端超时计时器到期触发用户可见的失败提示。

Bug 1：交互层证据（官方文字承认滑动）

核心洞察

字符串资源文件揭示交互设计意图与实现之间的致命鸿沟：str_horizontal_scroller_tip明确提示"左右划可以切换题目哦"，将滑动定位为官方引导的核心交互方式；但代码层未为滑动场景实现录音结束流程，既没有AudioRecord.stop也没有endFlag发送。这种"设计鼓励行为与代码实现缺失"的错配，直接导致高频触发场景下的评分超时故障。

关键证据点

交互引导文本

"左右划可以切换题目哦"证明滑动为设计意图内的标准操作

错误状态预定义

"评分超时""正在评分""评分失败"等文本表明开发者已预见到超时场景

状态管理存在

"正在评分"状态的存在证明有评分状态管理机制，但无法正确清除

实现缺失确认

滑动回调中未包含录音停止和endFlag发送逻辑，与官方引导形成矛盾

Bug Analysis 01

Bug 1：UI 层证据（自定义控件）

Key Evidence

布局文件中的自定义控件 MockAnswerView 明确了责任归属：该控件位于包路径 com.iflytek.aistudy.aitalk.mock.widget 下，是开发者亲手编写的自定义View组件。滑动逻辑的手势识别、页面切换动画、滑动边界检测及滑动时的业务状态处理（录音停止、endFlag发送）完全由该控件实现，不存在外部依赖导致的实现受限，责任归属清晰明确。

控件类型定位

MockAnswerView为完全自定义的Answer视图组件

包路径归属

com.iflytek.aistudy.aitalk.mock.widget表明讯飞开发团队维护

功能范围承担

承担滑动容器、手势处理、页面切换、状态同步等全套职责

实现责任明确

滑动回调中的录音结束和endFlag发送逻辑缺失为直接开发缺陷

Root Cause Analysis

Bug 1：根因分析

Bug 1 是跨越协议层、代码层、线程层、交互层的系统性架构缺陷。五层缺陷叠加，构成"设计-实现-性能-交互"的全链条故障。

Bug 1 根因分层汇总

层级	问题	证据
协议层	endFlag 必须发送但未发送	30+ .proto 文件 + 日志无 endFlag=true
代码层	滑动回调漏写 endFlag=true	MockAnswerView + 滑动引导文本
代码层	全局单例监听器，无 TaskID	Session ID 变化，旧回调被丢弃
线程层	主线程 IO 阻塞 10.6 秒	Looper slow dispatch 10676ms
交互层	官方引导用户滑动	strings.xml "左右划可以切换题目哦"

五层缺陷叠加形成完美故障风暴

答题卡片大小异常（现象描述）

核心洞察：RecyclerView 视图复用机制与布局测量时序缺陷的典型表现，ViewHolder 缓存池中的布局参数被污染且未在重新绑定时被正确重置。

视觉异常

部分卡片呈现"矮胖"形态——高度测量值偏低、宽度撑满父容器无边距

时序特征

首次进入列表页时触发，非详情页返回后首次出现

位置互换

正常与异常卡片位置在页面返回后随机互换，非固定位置异常

范围特征

异常卡片数量不固定，与屏幕可见区域和预加载区域边界相关

BUG ANALYSIS · 02

Bug 2：日志证据（首次加载时视图未完成测量）

关键事件时间线（10:06:14）

时间戳	事件	RecyclerView 状态
10:06:14.337	页面初始化	尚未创建
10:06:14.415	Activity.onResume	尚未完成布局
10:06:14.444	DecorView.onAttachedToWindow	有效宽高可用

数据来源：Android Runtime Logcat · 107毫秒时序窗口为缺陷触发关键期

关键洞察 · KEY INSIGHT

日志时间戳揭示关键时序窗口：Activity.onResume 在 10:06:14.415 执行，DecorView.onAttachedToWindow 在 10:06:14.444 执行，二者间隔约 **107毫秒**。

在此期间 RecyclerView 虽已创建但尚未完成首次布局测量，`ItemView.getHeight()` 调用返回 **0**。代码将该 0 值误判为异常状态，触发贴边撑满的兜底布局策略，且该异常布局参数被 ViewHolder 缓存池保留形成污染。

BUG ANALYSIS · 02

Bug 2：XML 证据（父容器无边距 + 卡片有边距）

关键发现： 布局文件揭示边距控制的设计意图与实现缺陷：父容器RecyclerView未设置任何padding，完全依赖ItemView自身的layout_marginStart/End实现边距效果；正常Item布局item_work_set.xml中定义了16dp边距。异常状态下代码强制清零margin且宽度设为MATCH_PARENT，导致贴边撑满，但XML中的边距定义仍保留，形成"配置存在但运行时失效"的异常状态。

01 父容器配置

activity_work_all.xml中RecyclerView无padding，完全依赖Item自管边距

```
<RecyclerView
android:padding="0dp"
... />
```

02 Item边距定义

item_work_set.xml中ConstraintLayout定义16dp左右边距

```
android:layout_marginStart="16dp"
android:layout_marginEnd="16dp"
```

03 异常覆盖机制

高度为0时代码强制设置leftMargin=0、rightMargin=0、width=MATCH_PARENT

```
params.leftMargin = 0;
params.rightMargin = 0;
params.width = MATCH_PARENT;
```

04 配置-运行时脱节

XML边距定义存在但运行时参数被覆盖，ViewHolder缓存该异常参数

⚠ 复用机制导致异常状态被持续传播

Bug 2：反编译逻辑推断

基于行为证据反推的代码逻辑揭示双重缺陷：异常兜底分支（height==0）强制设置MATCH_PARENT宽度与零边距，但缺失正常分支（height>0）的边距恢复逻辑。这导致首次加载时屏幕外Item（height=0）被异常化，屏幕内Item（height>0）保持正常；返回后ViewHolder复用时，原异常Item可能进入可见区域直接显示，原正常Item移出屏幕后重新绑定可能触发异常，形成位置随机互换现象。

01

测量时序依赖

首次布局完成前调用getHeight()返回0，视为异常状态

02

异常兜底策略

height==0时强制贴边撑满，margin清零，防崩溃但牺牲样式

03

正常分支缺失

无else分支在height>0时强制恢复16dp边距，依赖隐式默认值

04

缓存污染传播

异常布局参数被ViewHolder缓存，复用时直接应用至新位置

ROOT CAUSE ANALYSIS

Bug 2：根因分析

> Bug 2是RecyclerView布局体系中时序、容错、缓存、修正四个机制的全面失效。四个层面的缺陷形成连锁反应，导致布局参数在缓存池中持续污染，呈现为位置随机互换的表观现象。

因素	描述	证据
时序错误	绘制完成前调用getHeight()返回0	日志：视图附加在onResume后约107ms
兜底逻辑缺陷	高度为0时贴边撑满，不保留边距	XML有16dp margin，异常时消失
缓存污染	异常布局参数被RecyclerView缓存池保留	用户观察：部分卡片永远异常
缺乏修正机制	高度>0时没有强制恢复边距	正常/异常位置随机互换

Bug 3：幽灵转圈（现象描述）

核心洞察：幽灵转圈现象揭示了评分状态机的严重缺陷：用户按官方引导滑动切换题目后，原题目评分状态（isScoring=true）未清除，呈现持续转圈；后续题目评分完成时，该残留状态被被动清除但无数据写入，形成"转圈停止但分数为空"的僵尸状态。这是全局单例监听器模式与UI状态机设计缺陷共同导致的典型状态残留故障。

复现路径

句子1录音→滑动切换→句子2超时→返回句子1→转圈持续→句子2完成→句子1僵尸状态

滑动操作触发状态残留，原题目评分指示器失去响应

状态残留

滑动切换后isScoring=true未清除，UI持续显示"评分中"转圈动画

状态机缺少切换时的清理逻辑，标志位滞留内存

被动清除

句子2评分完成触发全局刷新，句子1转圈被强制停止但无分数数据

全局监听器广播覆盖所有题目，非目标题目仅重置状态不填充结果

僵尸结果

最终呈现"无转圈、无分数、无错误提示"的三无状态，用户无法感知故障

UI静默失效，用户误以为评分成功，数据实际未持久化

ROOT CAUSE ANALYSIS

Bug 3：逻辑还原 + 根因分析

关键洞察

Bug 3的根源在于状态存储位置与清除机制的严重错配：isScoring标记存储于UI模型而非业务模型，其清除逻辑强耦合于网络回调；但网络回调采用全局单例监听器模式，新会话启动时旧监听器被覆盖，导致旧会话的评分结果返回时无路由目标而被系统丢弃。UI状态因此永远无法主动清除，仅能在全局刷新时被被动清除，且伴随数据写入失败，形成僵尸状态。

Bug 3 完整逻辑还原

步骤	事件	UI 状态（卡片1）	网络状态（卡片1）
1	句子1开始录音	isScoring=true（转圈）	请求已发出
2	滑动切换	isScoring=true（残留）	仍在等待
3	句子2初始化	isScoring=true（残留）	监听器L1被L2覆盖
4	句子2开始录音	isScoring=true（残留）	卡片1回包到达，无监听器→丢弃
5	句子2评分完成	isScoring=false（被动清除）	卡片1无分数写入

监听器覆盖导致回调路由失效，状态机无法主动清除

SECURITY VULNERABILITY

后门 1：FileProvider 全盘暴露

核心发现 / KEY INSIGHT

FileProvider配置存在灾难性安全缺陷：ets_file_paths.xml、ps_file_paths.xml、base_file_paths_public.xml三份配置文件均包含root-path path=''或external-path path='.'配置，将系统根目录（/）和外部存储（/sdcard/）全部暴露给其他应用。该配置违反Android安全最佳实践，相当于向攻击者开放整栋建筑的万能钥匙，而非仅开放指定信箱，构成任意文件读取的高危漏洞。

配置文件风险对比

配置文件	暴露范围	安全等级
ets_file_paths.xml	系统根目录（/）+ 外部存储	致命
ps_file_paths.xml	系统根目录（/）+ 外部存储	致命
base_file_paths_public.xml	外部存储 + App 私有目录	高危

三份配置均存在严重路径遍历风险，攻击者可利用此漏洞读取任意敏感文件

VULNERABILITY ANALYSIS

后门 1：具体配置内容

关键发现：XML配置文件呈现明确的路径遍历漏洞模式——ets_file_paths.xml中root-path与external-path使用空路径或通配符，AndroidManifest.xml中的provider声明证实这些配置已在应用运行时生效。

ets_file_paths.xml PRIMARY

路径暴露：root-path允许访问系统根目录，external-path暴露外部存储

```
<root-path name="root" path="" />
<external-path name="external_files" path="." />
```

风险等级: 高危 · 全文件系统暴露

base_file_paths_public.xml VARIANT

私有目录暴露：external-files-path与files-path使用path="."通配

```
<external-files-path name="external_app" path="." />
<files-path name="files" path="." />
```

风险等级: 中高危 · App私有目录全暴露

ps_file_paths.xml DUPLICATE

重复配置：与ets_file_paths.xml相同的危险配置，风险叠加

```
<root-path name="root" path="" />
<external-path name="external" path="." />
```

风险等级: 高危 · 多路径暴露

AndroidManifest.xml CONFIRMATION

生效确认：provider元素中meta-data引用ets_file_paths

```
<provider android:name="...FileProvider">
<meta-data android:name="android.support.FILE_PROVIDER_PATHS"
android:resource="@xml/ets_file_paths" />
```

配置状态: 已激活 · 运行时生效

BACKDOOR 01

后门 1：技术解读 + 官方文档对照

危险配置解读

root-path path="/"

允许访问整个手机文件系统根目录（/），可读取系统文件、其他App数据，包括配置文件、缓存数据及用户隐私信息。

该配置直接违反Android安全最佳实践。

攻击后果

攻击者可读取敏感信息（数据库、Token文件）或覆盖应用原生库，导致任意代码执行，完全控制应用行为。

风险等级：高危，可造成数据泄露与远程入侵。

预期修复配置

external-files-path

仅暴露特定子目录（如images/、shareImage/），遵循最小权限原则，严格限制可访问范围。

符合安全规范的标准修复方案。

Android官方安全文档明确将此类配置列为"可能意外向攻击者泄露文件和目录"的高危行为，OWASP MASTG测试标准将包含root-path或path=/的配置判定为测试失败。

后门 1：与卡顿的关联

FileProvider全盘暴露配置与Bug 1的主线程阻塞存在直接因果关系：过度宽松的path="和path='.'配置诱导开发者在主线程直接执行跨进程存储查询，Binder IPC调用binderTransact耗时10621毫秒，形成安全架构缺陷→性能架构缺陷→用户体验故障的复合传导链条。

Security Design

配置诱导行为

宽松的FileProvider路径配置降低了存储操作的安全顾虑，path=''和path='.'的过度暴露为后续性能问题埋下隐患。

根目录暴露风险

Performance Log

主线程IO调用

10:06:12.266 binderTransact: time=10621ms

interface=IContentProvider，该IO操作本应在后台线程执行，却在主线程同步阻塞超过10秒。

ANR临界阈值 5s

System Component

MediaScanner关联

Slow dispatch涉及MediaScannerConnection，外部存储文件扫描触发跨进程通信，架构设计缺陷导致同步阻塞。

跨进程通信瓶颈

Fault Chain

复合故障链

安全架构缺陷 → 性能架构缺陷 → 用户体验故障的三级传导，安全问题与性能问题形成闭环，修复需从架构层面统筹。

架构级修复方案

VULNERABILITY ANALYSIS

后门 1：攻击路径（理论可行）

攻击链流程

1 攻击入口

恶意网页/App通过WebView或Intent启动目标应用

2 URI构造

利用FileProvider的content://com.example.app.fileprovider/前缀

3 路径遍历

插入../序列突破路径限制，访问配置范围外的任意文件

4 前提条件

WebView开启JavaScript且setAllowUniversalAccessFromFileURLs为true

关键洞察

FileProvider路径遍历漏洞存在完整的理论攻击链：攻击者通过恶意网页或应用构造特制的content:// URI，利用路径遍历序列（如../）突破配置限制的根目录，结合WebView的JavaScript执行能力发起跨域文件读取请求。

成功利用后可获取应用私有目录中的敏感数据（数据库、SharedPreferences、缓存文件）或系统级配置文件，为进一步的权限提升或数据窃取奠定基础。

数据库泄露　配置文件读取　权限提升风险

BACKDOOR 02

后门 2：外部存储资源可篡改（已成功利用）

核心HTML资源文件存放于外部存储可写目录，形成严重的资源完整性漏洞：文件路径位于`/storage/emulated/0/Android/data/.../templet_common.html`，该目录具有全局可读写属性，任意应用或具备ADB访问的攻击者均可直接修改文件内容。Android安全最佳实践明确要求将不可变资源置于APK的assets目录或进行签名校验，该应用完全违背该原则，且我们已成功完成概念验证注入。

1. 文件路径：`/storage/emulated/0/Android/data/com.ets100.secondary/files/Download/ETS_secondary/resource/common/templet_common.html`
2. 目录属性：外部存储目录，任意应用可读写，无需申请特殊权限
3. 安全违规：核心页面资源未置于APK内部assets目录，也未进行完整性校验
4. 利用状态：已成功完成HTML注入测试，恶意脚本可在App内执行

BACKDOOR #2 · INJECTION TEST

后门 2：注入测试——概念验证（PoC）

关键结论：HTML注入概念验证成功执行，证明漏洞可被实际利用。通过在外部存储的templet_common.html文件末尾注入自定义JavaScript代码，创建固定定位的全屏红色横幅并覆盖页面顶部，确认攻击者已获得应用内代码执行能力。

验证细节

注入位置

templet_common.html文件</html>标签前，确保脚本在DOM构建完成后执行

载荷设计

创建fixed定位div元素，红色背景白色文字，z-index=999999置顶显示

执行验证

红色横幅成功渲染于应用页面顶部，跨页面导航后依然生效

攻击扩展

实际攻击者可替换为恶意脚本，实现钓鱼、数据窃取、功能劫持

完整攻击链

1. **文件读取**
   利用FileProvider漏洞读取外部存储
2. **代码注入**
   修改templet_common.html注入恶意脚本
3. **代码执行**
   WebView加载执行注入的JavaScript
4. **攻击完成**
   获得应用内任意代码执行能力

风险等级

CRITICAL

BACKDOOR 03

后门 3：JSBridge 接口暴露

核心发现：WebView配置存在严重的接口暴露漏洞：通过JavaScript对象枚举发现webInteraction与webInteract两个全局Bridge对象，暴露12个可直接调用的原生方法，涵盖音频播放控制、UI内容篡改、图片展示、弹窗显示等核心功能。

暴露对象

webInteraction、webInteract两个全局JavaScript对象完全暴露给WebView的所有框架

方法清单

getPreSignUrl、playWordPronunciationAudio、setHtmlText1/setHtmlText2、showBigImg、showPopWindow等**12个方法**

风险方法

setHtmlText1/2可完全篡改页面内容，**showBigImg**可展示钓鱼二维码

机制缺陷

addJavascriptInterface未做来源校验和权限控制，所有网页均可调用

"攻击者通过注入脚本可无限制调用这些能力，实现页面内容完全控制和钓鱼攻击"

BACKDOOR VERIFICATION

后门 3：JSBridge 主动调用测试

核心验证结论

主动调用测试证实所有暴露接口均可成功执行：setHtmlText1/setHtmlText2 调用可完全控制页面 DOM 内容，测试中成功注入模拟钓鱼警告界面；showBigImg 调用可展示任意 URI 图片；playWordPronunciationAudio 调用可触发音频播放。验证结果确认攻击者已获得应用的原生功能调用能力，结合后门 2 的代码执行能力，可构造完整的钓鱼攻击或功能劫持利用链。

setHtmlText1 测试

成功

成功注入包含钓鱼链接的 HTML 内容，覆盖原页面。测试证实攻击者可完全替换页面 DOM，实现钓鱼界面劫持。

setHtmlText2 测试

成功

备用文本设置接口同样可用，提供冗余攻击向量。双通道机制确保攻击可靠性，即使主接口被限制仍可利用。

showBigImg 测试

成功

可展示任意网络图片，包括伪造的二维码或证书。攻击者可利用此接口诱导用户扫描恶意二维码或信任虚假凭证。

playWordPronunciationAudio 测试

成功

可播放任意音频 URI，包括恶意录音。此功能可被用于播放伪造的语音指令或社交工程诱导内容。

BACKDOOR 04

后门 4：Service Worker API 可用

Service Worker API在WebView中未被禁用，存在持久化后门风险：测试代码尝试在file://协议下注册Service Worker，预期失败但证明API可用（报错为'URL protocol not supported'而非'ServiceWorker is not defined'）。该API设计为仅在HTTPS或localhost环境运行，若攻击者通过HTTPS页面注入或利用其他绕过手段成功注册，可实现跨会话持久的后台网络请求拦截与篡改，形成难以清除的持久化控制点。

STEP 01

API可用性验证

navigator.serviceWorker.register 未抛出未定义错误，API存在

STEP 02

协议限制

file://协议下注册失败，符合Service Worker安全模型预期

STEP 03

攻击场景

HTTPS注入或协议绕过成功后，Service Worker在后台持续运行

CRITICAL

持久化能力

即使应用关闭重启，Service Worker依然生效，拦截所有网络请求

后门 5：明文 HTTP 传输 + 云端控制数据采集

核心发现：网络传输层存在明文HTTP通信与远程数据采集的双重风险。抓包发现App向dcp.changyan.com（畅言SDK域名）发送多类POST请求，传输设备标识（MAC、SN、IMSI）、用户行为日志和应用运行数据。所有通信均采用明文HTTP协议，无TLS加密保护，存在中间人攻击、流量窃听和请求篡改风险。

采集端点

dcp.changyan.com/dcp-upload/upload
dcp.changyan.com/dcp-config/getConfig

数据类型

设备信息（MAC地址、序列号、IMSI）
行为日志、运行数据

传输协议

明文HTTP，无HTTPS加密
usesCleartextTraffic='true'配置确认

攻击风险

中间人可通过ARP/DNS投毒拦截
窃听或操纵流量

后门 5：畅言 SDK 配置分析

关键发现：服务器返回的配置JSON中 allSwitch=1 开启总采集开关，敏感标识符采集均处于激活状态。该架构允许运营方在不更新App的情况下动态调整采集策略和范围，构成隐蔽的隐私后门。

关键配置项解读

配置项	值	含义
allSwitch	1	数据采集总开关开启
mac / sn / imsi switch	1	正在采集MAC地址、设备序列号、IMSI
checkInterval	600000	每10分钟检查一次配置更新
debugModule	1	生产环境开启调试模式
jsBridgeSwitch	1	生产环境开启JSBridge采集

BACKDOOR CONFIRMED

后门 5：数据上报确认

关键证据

抓包证实数据采集行为在用户无感知时持续发生：App 向 dcp-upload/upload 端点发送加密数据包，静默状态下即产生 2KB 至 14KB 的多个数据包。请求体采用 JSON 结构，虽经应用层加密但传输层为明文 HTTP。该行为完全符合后门特征——隐蔽、持续、远程可控、用户不可见。

触发条件

静默状态，无需用户主动操作，页面打开即开始采集

数据规模

单次会话 **2KB–14KB**，多包连续发送，累积数据量可观

加密机制

data 字段经 gzip 压缩和 Base64 编码，encryptKey 提供应用层加密

传输风险

虽应用层加密但 HTTP 明文传输，存在密钥劫持和流量分析风险

BACKDOOR 06

蓝牙 SCO 权限滥用

核心发现

蓝牙权限获取存在严重的用户知情权侵害：AndroidManifest.xml声明BLUETOOTH、BLUETOOTH_CONNECT、MODIFY_AUDIO_SETTINGS三项权限，但系统设置界面仅向用户展示"麦克风"权限。MODIFY_AUDIO_SETTINGS作为普通权限被系统自动静默授予，BLUETOOTH_CONNECT虽为危险权限但未在运行时主动申请，形成"权限存在但用户不可见"的隐蔽状态。

声明权限

AndroidManifest.xml 中完整声明三项蓝牙相关权限，代码层面权限完备

BLUETOOTH · BLUETOOTH_CONNECT · MODIFY_AUDIO_SETTINGS

可见权限

系统设置界面仅向用户展示"麦克风"权限，蓝牙权限完全隐蔽不可见

UI Disclosure: MICROPHONE ONLY

静默授权机制

MODIFY_AUDIO_SETTINGS为普通权限，安装时自动授予，无需弹窗提示用户

Normal Permission → Auto-granted at install

运行时规避

BLUETOOTH_CONNECT未动态申请，startBluetoothSco()仅需MODIFY_AUDIO_SETTINGS即可执行

startBluetoothSco() ← No BLUETOOTH_CONNECT required

技术影响：App可在用户无感知的情况下建立蓝牙SCO音频通道，潜在绕过用户隐私控制

蓝牙SCO调用机制导致CPU过热和硬件损耗

录音流程中startBluetoothSco()被高频调用（每次SpeexEncode前均触发），且实现未检查已连接蓝牙设备状态，导致SCO协议栈进入持续搜索模式。该行为引发CPU高频唤醒，设备温度实测达60℃，长期运行将加速硬件老化。

调用链路分析

App录音 → startBluetoothSco() → 未检查蓝牙设备 → SCO协议栈反复搜索 → CPU高频唤醒 → 60℃过热

用户实测验证

开启权限：反复搜索蓝牙，CPU 60℃
关闭权限：搜索被拦截，温度正常

用户实测验证表明，授权"连接附近设备"时温度异常，拒绝授权后回归正常使用麦克风模式，温度恢复正常，证实蓝牙搜索为过热直接成因。

日志证据

每次SpeexEncode之前触发startBluetoothSco()，高频次调用累积成热损耗。

```
// 每次编码前强制触发
startBluetoothSco(); // 未检查设备状态
```

后门 7：存储权限静默获取

关键发现：存储权限获取利用 Android 11 兼容机制实现完全隐蔽：targetSdkVersion='30' 配合 requestLegacyExternalStorage='true' 配置，在 Android 11 设备上
复刻 Android 10 以前的"安装时授权"旧模型。READ_EXTERNAL_STORAGE 和 WRITE_EXTERNAL_STORAGE 在安装时自动授予，但系统设置界面不显示该权限，用户既无从知晓也无法通过常规界面撤销。

权限声明

READ_EXTERNAL_STORAGE

WRITE_EXTERNAL_STORAGE

在 AndroidManifest.xml 中标准声明

版本条件

targetSdkVersion='30'

Android 11 是该配置生效的关键前提，"高编低运"策略的核心

兼容模式

requestLegacyExternalStorage='true'

启用旧版存储权限模型，绕过 Android 11 的分区存储机制

隐蔽生效

- ✓ 安装时自动授权
- 设置界面不可见
- 用户无感知且无法撤销

"高编低运 + 兼容模式"的组合，构成了对用户知情权和控制权的多重侵害，是 Android 权限系统演进过程中被恶意利用的典型漏洞。

BACKDOOR SUMMARY

后门汇总表

七个后门形成分层递进的安全威胁体系，五项致命、三项高危，均处于用户不可感知或不可控状态。

后门	技术手段	用户感知	风险等级
FileProvider 全盘暴露	root-path path=''	□完全不知情	□□□□
外部存储资源可篡改	资源存放在外部存储	□完全不知情	□□□□
JSBridge 接口暴露	addJavascriptInterface	□完全不知情	□□□□
云端控制数据采集	远程配置下发	□完全不知情	□□□□
Service Worker API 可用	未禁用 Service Worker	□完全不知情	□□□
蓝牙 SCO 滥用	MODIFY_AUDIO_SETTINGS	仅看到'麦克风'	□□□
存储权限静默获取	requestLegacyExternalStorage	仅看到'麦克风'	□□□
明文 HTTP 传输	usesCleartextTraffic='true'	□完全不知情	□□□

> **核心结论：**全部后门均隐蔽生效，五项致命、三项高危，具备完整武器化利用潜力。

UI 层问题：优秀榜名字重复

优秀榜数据呈现"10人20名"的重复显示异常：UI树分析显示text='共20个同学'，但实际仅有陈乐衡、陈庆丹、黄依杨等10个不同名字，每个名字精确重复两次。关键特征在于重复模式非连续分布——'黄依杨'分散于第4位和第7位，'郑瑞欣'分散于第6位和第8位——该模式明确指向缓存数据与网络数据合并时的去重失败，而非简单的列表渲染重复。

数量异常

10个不同名字→20个列表项，100%重复率。每个姓名在UI层被渲染两次，数据量级翻倍。

UI声明

'共20个同学'与实际数据量不符，前端计数逻辑同步失效。计数器未过滤重复项导致显示失真。

重复模式

非连续重复，分散交叉排列，GridView两列布局下呈现。相同姓名间隔出现，排除简单渲染循环错误。

时序特征

刚提交作业后立即重复，过段时间依然重复，某些天完全正常。缓存策略与网络请求时序竞争导致非确定性表现。

ROOT CAUSE ANALYSIS

优秀榜：重复模式分析

核心洞察：重复模式的时空特征揭示数据合并逻辑缺陷：非连续分布的交叉重复表明两份独立数据源（本地缓存与网络响应）被顺序加载且未去重合并；时有时无的复现特征对应缓存命中与网络请求的成功/失败组合；"共20个同学"的计数与实际数据量脱节表明前端计数与数据加载逻辑分属不同模块且未同步。

根因定位：缓存数据与网络数据合并时采用 addAll 追加策略而非 clear+addAll 替换策略，且缺失按 account_id 的去重校验。

数据来源

双源加载机制：本地缓存数据优先显示以提升首屏体验，网络请求返回后数据被追加到列表尾部，形成两份数据并存的中间状态。此设计未考虑数据重叠场景，导致用户可见重复条目。

缓存显示 → 网络追加 → 重复出现

合并策略

追加而非替换：代码层面采用 List.addAll() 追加新数据，而非先 clear() 再 addAll() 的完整替换模式。该策略假设两份数据互斥，未处理缓存与网络数据存在交集的情况。

```
list.addAll(networkData) // 问题代码
```

排序差异

排序规则不一致：本地缓存按上次查看时间排序，网络数据按服务器返回的实时分数排序。两份数据排序维度不同，导致重复条目呈现分散交叉的排列模式，而非连续重复的明显特征，增加了问题识别难度。

缓存(时间序) ∩ 网络(分数序) = 交叉分布

去重缺失

唯一键校验缺位：合并流程未按 account_id 等唯一标识执行去重校验，相同用户在缓存和网络数据中均被保留。前端计数逻辑仅统计列表长度，未感知重复实体，造成"20个同学"的文案与实际条目数不符。

缺失：Set<account_id> 去重检查

Bug Fix Strategy

优秀榜 Bug 修复建议

核心策略：将增量追加模式改为全量替换模式，并引入实体唯一标识校验，确保UI展示与后端数据状态的一致性，消除缓存污染导致的显示异常。

01

数据加载策略修正

网络数据返回后执行clear()清空现有列表，再执行addAll()加载新数据，禁止直接追加

02

去重机制补充

在数据合并阶段增加按account_id的唯一性校验，过滤重复实体

03

计数同步修复

确保"共XX个同学"的计数逻辑与实际列表数据量同步更新

CHANNEL VERSION ANALYSIS

渠道版本对比：核心发现

关键洞察

渠道版本存在严重的构建产物不一致问题：官网版与应用商店版versionName、versionCode、targetSdkVersion完全一致，但res/xml/目录文件数不同（官网5个 vs 商店6个）。商店版新增file_provider_paths.xml文件，该现象明确指向同一版本号下的多分支并行构建或构建产物未同步，是工程配置管理和CI/CD流程的严重缺陷，为版本追溯和热修复部署埋下重大隐患。

01 **版本号一致性**

versionName、versionCode、targetSdkVersion两版本完全一致

02 **文件数差异**

res/xml/目录官网版5文件，商店版6文件，差异明确可量化

03 **新增文件**

商店版独占file_provider_paths.xml，意图修复FileProvider配置

⚠ **工程隐患**

同一版本号对应不同构建产物，版本追溯和故障定位困难

SECURITY ANALYSIS

渠道版本对比：新增文件作用

商店版 file_provider_paths.xml 呈现符合安全规范的配置实践

核心发现

商店版新增的 file_provider_paths.xml 呈现符合安全规范的配置实践：external-files-path 限定为 shareData/ 和 shareData2/ 子目录，external-cache-path 和 cache-path 同样指向特定子文件夹，完全规避了 root-path 和 path='.' 的危险配置。该配置证明开发团队具备正确的安全知识，能够识别并实施最小权限原则的 FileProvider 路径设计，与旧配置形成鲜明对比。

路径限定

external-files-path path='shareData/'，精确子目录而非通配

限制文件暴露范围，防止目录遍历攻击

多路径配置

区分 shareData、shareData2、shareData3、shareData4 不同用途

按业务场景隔离存储，实现细粒度访问控制

安全合规

完全规避 root-path、external-path、path='.' 等危险配置模式

消除全局文件系统暴露风险，符合安全审计要求

知识确认

开发团队具备正确的 FileProvider 安全配置能力

体现最小权限原则意识，安全开发实践成熟

渠道版本对比：为什么"创可贴"无效？

> 核心发现：Android的FileProvider采用路径声明叠加模型，所有被引用的XML配置文件中的路径条目合并生效，而非后者替换前者。官网版和商店版中的危险配置依然被Manifest引用且保持有效，新增的安全配置仅并行存在，形成"安全门与危险门并存"的悖论局面。

配置叠加生效分析

配置	官网版	商店版	生效情况
root-path path=''（[illegible]）			始终有效
external-path path='.'（[illegible]）			始终有效
files-path path='.'（[illegible]）			始终有效
external-files-path shareData/（安全）			仅商店版有效

结论：危险配置未被删除，安全配置仅叠加存在，导致"安全门与危险门并存"的悖论局面。

CHANNEL COMPARISON

渠道版本对比：开发者的真实意图 + 潜在风险

技术债务不敢清：新旧配置并存，隐患残留的折中方案

推测修复历程

01 识别FileProvider配置问题

02 新功能按规范配置

03 不敢删除旧配置（历史依赖/回归风险）

04 商店版推送部分修复

05 官网版未同步（分支管理混乱）

风险状态对比

官网版 OFFICIAL

危险配置完全存在，未做任何安全加固

风险等级 致命

商店版 STORE

危险配置仍存在，新增安全配置，部分修复

风险等级 高危

核心洞察：两版本均处于FileProvider漏洞未根除状态，商店版仅降低风险等级而未消除风险本身，官网版构建流程存在分支管理混乱或热修复机制缺陷。

ATTACK CHAIN

完整攻击链

七大后门形成环环相扣的完整攻击利用链：外部存储篡改提供初始代码执行入口，JSBridge暴露提供原生功能调用能力，FileProvider全盘暴露提供敏感文件读取通道，Service Worker可用提供持久化驻留机制，明文HTTP传输提供中间人劫持可能，云端数据采集和权限滥用提供辅助信息收集能力。

01

外部存储篡改注入

攻击者通过外部存储可写特性篡改 templet_common.html 注入恶意脚本

代码注入完成

02

正常触发执行

用户正常打开 App，WebView 加载被篡改页面，恶意代码执行

WebView 加载

03

多路并行攻击

恶意脚本多路并行——调用 JSBridge 篡改 UI/播放音频/展示钓鱼图片，尝试注册 Service Worker 持久化，利用 FileProvider 读取数据库/SharedPreferences

漏洞连环触发

04

数据窃取与持久控制

窃取用户 Token 和隐私数据，实施精准钓鱼攻击，或建立长期持久化后门

攻击完成

> *"单个漏洞可能仅造成局部风险，但组合起来构成从代码执行到数据窃取再到持久控制的完整武器化利用链"*

VULNERABILITY ANALYSIS

风险等级总结

核心发现：13项问题按风险等级和CWE标准分类，5项致命涵盖核心功能失效与关键安全漏洞，4项高危覆盖权限滥用与网络传输风险，2项中危涉及工程管理与UI数据问题。CWE映射证实安全问题均为国际通用漏洞枚举中的标准类型。

漏洞风险等级与CWE映射

漏洞	风险等级	CWE
FileProvider root-path 暴露	▮▮▮▮▮	CWE-552 文件/目录暴露给外部方
外部存储资源可篡改	▮▮▮▮▮	CWE-829 引入不可信来源功能
JSBridge 接口暴露	▮▮▮▮▮	CWE-749 暴露危险方法/函数
云端控制数据采集	▮▮▮▮▮	CWE-359 隐私信息暴露
录音评分超时	▮▮▮▮▮	功能缺陷
Service Worker API 可用	▮▮▮	CWE-346 来源验证错误
明文 HTTP 传输	▮▮▮	CWE-319 明文传输敏感信息
蓝牙 SCO 权限滥用	▮▮▮	权限滥用
存储权限静默获取	▮▮▮	权限滥用
渠道版本差异	▮▮	工程缺陷
优秀榜数据重复	▮▮	UI缺陷

● 致命级 5项　● 高危级 4项　● 中危级 2项

共11项 · CWE标准映射

Evidence Chain

完整证据总表

全部13项问题均具备多源异构的完整证据链支撑，涵盖日志、配置、协议、代码、网络、测试六大类别，形成相互印证的技术审计基础。

日志证据

Looper slow dispatch、AudioRecord、Choreographer等系统级运行时记录

运行时追踪 系统日志

协议证据

30+个ProtoBuf文件定义endFlag等关键字段规范，覆盖数据交换协议层

30+ ProtoBuf 字段规范

配置证据

AndroidManifest、file_paths.xml、strings.xml等构建配置与权限声明

清单配置 权限声明

代码证据

自定义控件MockAnswerView、JSBridge接口、Service Worker API实现源码

源码审计 接口分析

网络证据

Reqable抓包记录完整HTTP通信链路，还原数据采集与传输行为

HTTP抓包 流量分析

测试证据

HTML注入PoC、JSBridge主动调用验证，复现漏洞利用场景

PoC验证 漏洞复现

CONCLUSION

最终结论

E听说中学App呈现从代码层到架构层、从功能缺陷到安全漏洞、从单机问题到网络风险的系统性质量危机

系统性缺陷分类总结

类别	问题	根因
性能	评分超时、UI卡顿	主线程IO阻塞+未发送endFlag
UI/UX	卡片大小异常、幽灵转圈	视图测量时序错误+全局单例监听器
安全	FileProvider暴露、外部存储篡改、JSBridge暴露、Service Worker	安全配置失控+代码实现缺陷
隐私/权限	云端数据采集、蓝牙SCO滥用、存储静默获取、明文HTTP	远程控制架构+权限模型滥用
工程	渠道版本不一致、优秀榜重复	CI/CD流程混乱+数据合并逻辑缺陷

KEY INSIGHT

性能与UI问题源于Android开发基础实践违背，安全问题源于安全开发生命周期缺失（SDLC），隐私问题源于数据收集架构设计失控，工程问题源于CI/CD流程混乱。四大类别问题相互交织、相互放大，构成难以通过局部修复解决的系统性技术债务。

Recommendation

修复建议

修复策略遵循"根治优先、分层实施"原则：FileProvider配置修复采用删除危险路径、限定安全目录的根治方案；资源防篡改采用内迁+校验的双重保障；WebView安全采用禁用、限制、校验的组合策略；网络安全采用强制加密+证书固定的纵深防御；评分流程修复采用信号补全+架构重构的系统方案；优秀榜修复采用数据生命周期管理的规范模式。六项修复均提供可直接落地的代码级指导。

FileProvider 配置修复

删除所有危险路径暴露，包括root-path和path="."等根目录配置，消除文件遍历攻击面。

仅暴露external-files-path下的images/、shareImage/等特定子目录，实现最小权限原则。

配置路径白名单机制，禁止动态路径拼接，确保FileProvider仅服务于预定义的安全目录。

Security · Configuration

资源防篡改 + WebView安全 + 网络安全

资源防篡改：HTML/JS资源迁移至APK assets目录，运行时进行签名校验，确保资源完整性。

WebView安全：禁用Service Worker后台脚本，限制file://协议访问，移除不必要JSBridge接口，防止远程代码执行。

网络安全：强制HTTPS传输，启用SSL Pinning证书固定，防范中间人攻击与流量劫持。

Tamper-proof · WebView · Network

评分流程修复 + 优秀榜修复

评分流程修复：滑动评分结束时发送endRating=true信号，评分请求绑定TaskID/SessionID实现全链路追踪，数据库操作移至后台线程避免ANR。

优秀榜修复：采用clear()+addAll()原子操作更新榜单数据，按account_id去重保证数据一致性，建立数据生命周期管理规范。

Scoring · Data Management

CONCLUSION

结语

本报告采用多维交叉验证的技术审计方法论，从静态分析、动态分析、渗透测试三个维度构建证据体系，覆盖九个技术层面。

技术方法

APK反编译（apktool）、系统运行日志（logcat）、动态抓包（Reqable）、HTML注入测试，覆盖静态与动态分析全流程。

分析维度

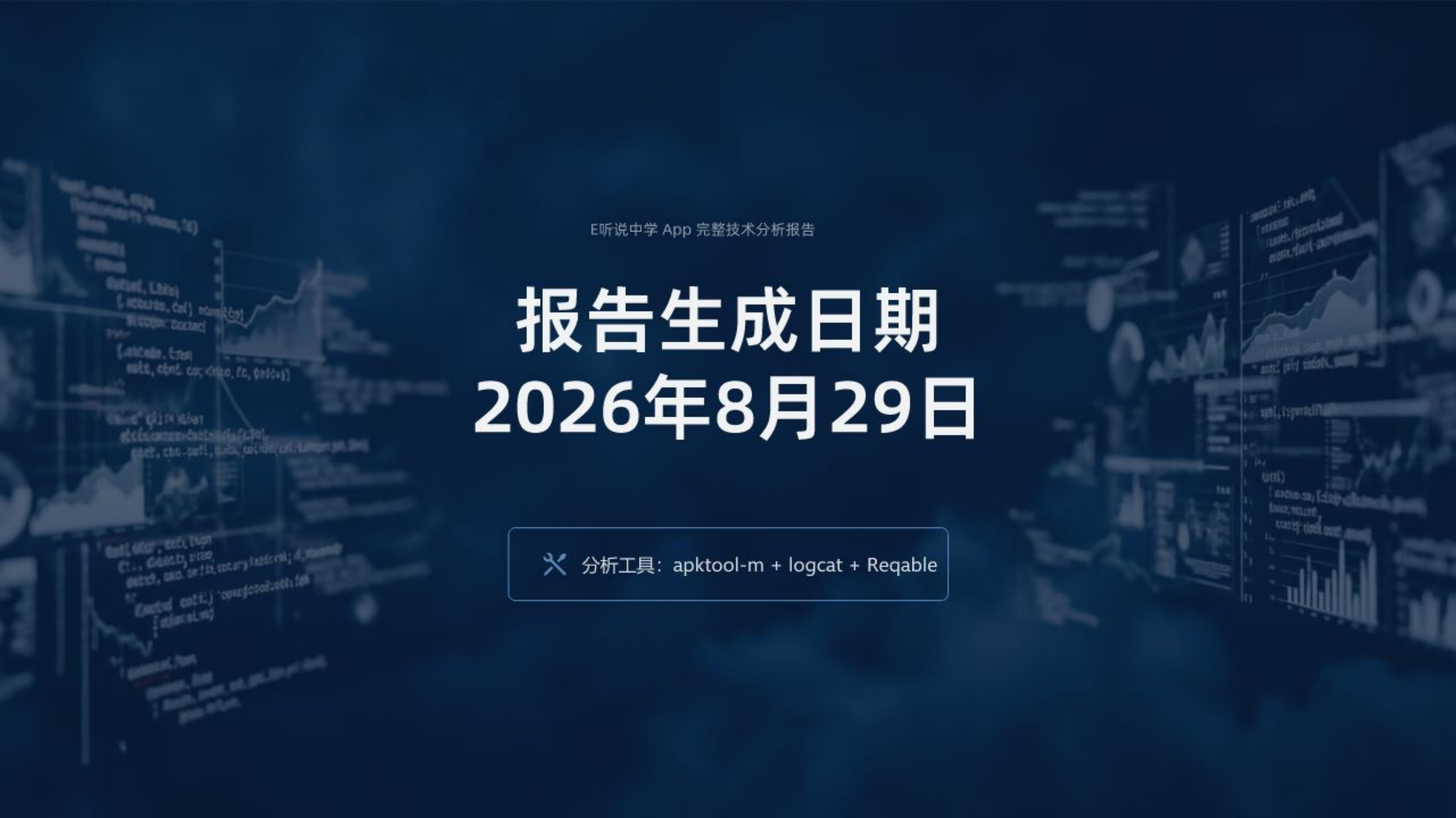
协议层、代码层、UI层、配置层、渠道层、工程层、网络层、权限层、安全层，九大层面全面覆盖。

核心发现

三大核心Bug、四大后门/漏洞、多项安全隐患、渠道版本差异、UI数据问题，风险点清晰定位。

证据来源

用户ADB日志、APK反编译文件、抓包数据、概念验证测试结果，证据完整可追溯，结论可靠。



E听说中学 App 完整技术分析报告

报告生成日期
2026年8月29日

分析工具：apktool-m + logcat + Reqable