Software Engineering and Applications 软件工程与应用, 2025, 14(4), 948-959
Published Online August 2025 in Hans. https://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2025.144084

Hans汉斯

Android终端关键硬件环境可信评估方法研究

周云涛[1,2]，姚红静[1]，魏　琪[1]

[1]西北工业大学计算机学院，陕西　西安
[2]西北工业大学人力资源部，陕西　西安

收稿日期：2025年7月11日；录用日期：2025年8月19日；发布日期：2025年8月28日

摘　要

Android系统因其开放源代码的特性，面临日益严重的安全问题，移动智能终端应用已成为网络恶意攻击的主要目标。作为嵌入式系统，移动终端的底层硬件环境安全是确保其整体可信性和可靠性的关键。基于可信计算思想，针对影响Android终端软件性能的CPU、内存和电池等关键硬件进行实时监测，通过量化评估硬件环境可信水平，为移动终端平台的安全解决方案提供技术支撑，从而保障系统运行的稳定、安全和用户体验。

关键词

Android系统，移动智能终端，嵌入式系统，硬件环境，安全可信

Research on Trusted Evaluation Method for Key Hardware Environment of Android Terminal

Yuntao Zhou[1,2], Hongjing Yao[1], Qi Wei[1]

[1]School of Computer Science, Northwestern Polytechnical University, Xi'an Shaanxi
[2]Department of Human Resources, Northwestern Polytechnical University, Xi'an Shaanxi

Received: Jul. 11th, 2025; accepted: Aug. 19th, 2025; published: Aug. 28th, 2025

Abstract

Due to its open-source nature, the Android system is increasingly vulnerable to security threats, making mobile intelligent terminals a prime target for malicious attacks. As an embedded system, the security of the underlying hardware environment is critical to ensuring the overall trustworthiness and reliability of mobile terminals. This study employs the principles of trusted computing to

文章引用: 周云涛, 姚红静, 魏琪. Android 终端关键硬件环境可信评估方法研究[J]. 软件工程与应用, 2025, 14(4): 948-959. DOI: 10.12677/sea.2025.144084

monitor key hardware parameters in real time, such as CPU, memory, and battery, quantitatively evaluating the trustworthiness level of the hardware environment. The research provides technical support for enhancing mobile terminal platform security, thereby ensuring stable system operation and a secure user experience.

Keywords

Android System, Mobile Intelligent Terminal, Embedded System, Hardware Environment, Secure and Trustworthy

Copyright © 2025 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

移动终端的可信性已成为互联网时代的重要挑战。Android 系统基于 Linux 内核，其开放源代码的特性虽然促进了定制化开发，但也引入了诸多安全隐患。据统计，Linux 系统每年有近百个漏洞被 CVE 收录，攻击者常利用这些漏洞实施恶意行为。此外，Android 平台缺乏严格的第三方审查机制，进一步加剧了其安全风险。

移动终端存储了大量敏感信息，一旦被恶意代码入侵，可能导致用户隐私泄露和财产损失。尽管用户安全意识有所提升，但隐蔽性强的恶意软件仍能绕过传统防护手段。因此，从底层硬件环境入手，构建可信的移动终端架构，成为解决安全问题的关键。可信计算技术通过建立信任传递体系，为终端平台的可信性提供了理论支撑。

为此，论文基于可信计算思想，重点针对影响 Android 终端性能的硬件环境(如 CPU、内存和电池)进行实时监测与评估，为移动终端平台的安全解决方案提供技术支撑。

2. 移动可信计算技术

移动智能终端面对的是一个开放性的使用环境，受到的攻击风险种类繁多而且不断进化，针对移动智能终端不断出现的安全问题，在传统安全防护技术基础上，具备主动防御能力的可信计算技术成为了移动安全领域一个新的研究热点。

可信计算的概念最早由美国国防部 1983 年在其制定的《可信计算机系统评价准则》中第一次提出。随后，Microsoft、Intel、IBM 等 IT 巨头公司于 1999 年领头发起和成立了可信计算平台联盟(TCPA)，并在 2003 年将其正式更名为可信计算组织 TCG (Trusted Computing Group) [1]，TCG 发布了一系列可信计算标准，介绍了如何利用可信平台模块(TPM)提供的各种功能来保护系统平台的安全。国内 1992 年正式在可信计算领域立项并开始研究。2004 年 10 月，武汉大学和瑞达公司合作研制出我国第一款自主研制的可信计算平台。2008 年中国可信计算联盟(CTCU)成立。随后各大公司推出了各自的可信产品，如兆日公司的可信计算机密码模块芯片、中兴集成电路公司的可信计算支撑平台等，并且都获得了国家密码管理局的认证。

与传统的安全防护技术相比，可信计算是指在运算的同时进行安全防护，使计算结果总是与预期相一致，计算全程可测可控，是一种运算与防护并存的主动免疫的计算模式。可信计算在移动智能终端安全研究中得到了极大的关注，取得了一系列成果。

2005 年 TCG 附属移动电话工作组 MPWG (Mobile Phone Work Group)发布了移动电话工作组白皮书，对可信计算应用在移动领域进行了展望和案例分析。欧洲国家于 2006 年开始了名为“开放式可信计算(Open Trusted Computing)”的研究计划，该计划基于 TPM 构建统一安全体系，实现安全的电子交易、家用协同计算以及虚拟数据中心等多个应用[2]。2007 年 MPWG 发布了移动可信模块 MTM (Mobile Trusted Module)规范，主要目的是为适应移动设备的特点，在移动平台上构建可信技术架构，为高端的移动设备提供安全保证。随着移动可信计算技术不断发展和成熟，许多外国学者开始研究可信移动平台的构建。

与依赖底层硬件隔离不同，另一类研究聚焦于通过系统级接口采集终端在运行过程中的性能状态，以此构建轻量级的安全评估与行为检测模型。该类方法强调部署简便性与对终端资源消耗的最小化开销，适用于资源受限的中低端移动设备场景。典型实现如 Google 官方提供的 Android Profiler 与 Battery Historian 工具，可分别用于监测 CPU 使用率、内存占用、电池放电速率等性能指标[3] [4]。第三方工具如 DevCheck 也被广泛用于实时采集/proc 与/sys 虚拟文件系统中的状态参数，便于开发者评估系统负载状态[5]。在学术研究方面，Zhao 等提出一种基于资源使用模式的行为识别方法，利用内存、CPU、I/O 访问特征构建轻量型分类模型，以识别潜在恶意后台任务[6]。Adhikari 等提出的 VolMemDroid 框架则进一步结合 volatile memory 特征，通过内存快照序列识别恶意任务执行痕迹，体现出运行态内存级可信性建模的可行性[7]。Bampatsikos 等将多属性决策(MADM)机制引入终端信任管理，综合考虑访问风险、可用性与安全级别等因素构建信任评分函数，并结合层次分析法实现权重调优与动态更新机制[8]。Nazir 等则结合 MITRE ATT&CK 攻击模型，构建了 ZTA (Zero Trust Architecture)架构下的恶意行为识别体系，实现了行为级别的动态风险建模与安全策略自动联动，拓展了可信性评估的实时响应能力[9]。尽管上述研究在信息采集与可信建模方面取得一定进展，但仍侧重于单一指标，缺乏多源资源联合分析，并依赖经验阈值，缺乏可信语义解释等问题。

Android 终端的可信性依赖于软硬件协同保障。硬件资源(如 CPU 和内存)的合理分配直接影响用户体验。研究表明，终端卡顿等问题多由硬件资源使用不当引起，例如低内存或高 CPU 负载运行[10]。

3. 移动终端硬件环境可信评估方法

硬件环境可信评估分为两步，即实时监测关键硬件参数，并将数据输入评估模型进行量化分析。

3.1. 硬件环境可信评估

论文选取了 Android 终端中 CPU、存储、电池等重要硬件为对象，自定义参数指标及其评估规则，通过后台服务的形式实时监测其状态变化，建立硬件的可信评估方法。评估规则分为两步，首先计算单项的评分，最后综合单项的评分计算出总的系统评分，评估流程如图 1 所示。

(1) CPU 可信性评估

CPU 频率和温度是两项至关重要的性能指标，一般来讲，频率越高则处理速度越快，但同时也会造成温度的上升，长时间在高温下运行会对 CPU 造成危害，此时则需要降频以达到保护 CPU 的目的。基于此，本文将频率和温度作为 CPU 可信评估的两项主导性指标。从内核文件中读取其最大和最小频率，取最小频率作为 CPU 频率的基准值，定为 100 分。CPU 温度工作范围为 25~75 度之间，50 度以上即为不安全温度区域，因此，本文将人体恒温 37 度作为温度指标的基准值，定为 80 分。

确定了指标的基准值后，在此基础上进行增减计算得到各指标的评分。由于目前的移动终端均支持多个处理核心，因此本模块对每个核心单独计算，对多个核心的评估值总和再取均值，最终得到 CPU 的可信性评估值。

CPU 频率的评估规则如公式(1)所示：

$$P_{cpuFreq} = 100 - \left(P_i - P_{\min}\right) \times \frac{100}{P_{\max} - P_{\min}} \tag{1}$$

式(1)中，P_{max} 表示当前内核的最高频率，P_{min} 表示最低频率也即为频率的基准值，P_i 表示第 i 个内核当前的实时频率。

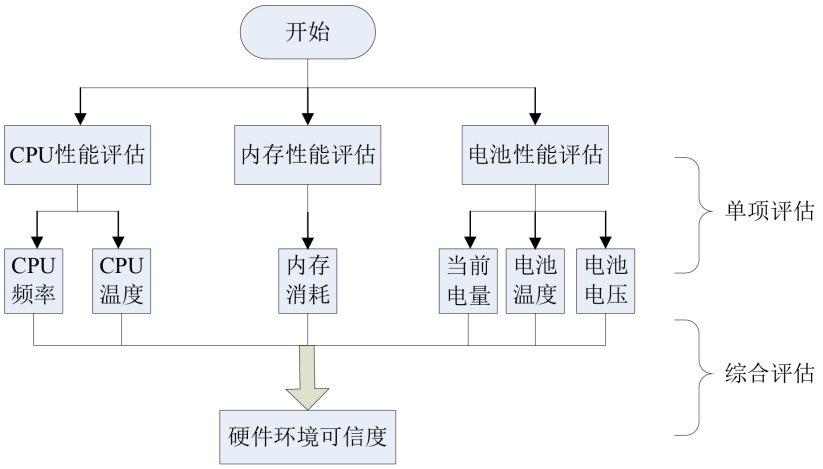

Figure 1. Hardware environment trust evaluation process
图 1. 硬件环境可信评估流程

CPU 温度的评估规则如公式(2)所示：

$$P_{cpuTemp} = \begin{cases} 80 + \left(S - T_i\right) \times \dfrac{100 - 80}{S - 25} & 25 \le T_i \le S \\ 80 - \left(T_i - S\right) \times \dfrac{80}{55 - S} & S < T_i < 55 \\ 0 & T_i \ge 55 \end{cases} \tag{2}$$

式(2)中，T_i 表示内核 i 当前的温度，S 为温度的基准值，本系统中取值为 37。

(2) 内存可信性评估

Table 1. Average performance metrics of top 10 android application categories
表 1. Android 平台十大主流分类应用性能指标平均情况

应用分类	内存消耗
浏览器	35.21 MB
图书阅读	25.17 MB
社交网络	24.25 MB
机票出行	28.35 MB
新闻资讯	28.28 MB
影音播放	30.27 MB
系统工具	22.22 MB
通话通讯	23.14 MB
拍摄美化	25.72 MB
网购支付	26.36 MB

根据 Testin2013 年 8 月发布的一份《中国好应用质量排行》中[11] (如表 1 所示)，通过计算这十大主流分类的内存的平均值作为评分规则的 80 分的基准。通过应用内存消耗进行统计，得出主流应用平均消耗内存为 26.90 MB，因此，论文将内存消耗为 26.9 MB 的应用内存评分定为 80 分。

为反映当前系统的内存消耗状态，选取当前系统中运行进程中内存消耗最大的应用作为系统当前内存的评分。内存可信性评估规则如下：

$$P_{memory}=\begin{cases}80+(S-M)\times\dfrac{100-80}{S} & M\le S\ \&\ lowMem=false\\ 80-(M-S)\times\dfrac{80}{M_{\max}-S} & S<M\le M_{\max}\ \&\ lowMem=false\\ 0 & lowMem=true\,\|\,M>M_{\max}\end{cases}\tag{3}$$

式(3)中，M 代表当前应用的内存消耗，S 为内存消耗的基准值，这里取 26.9，M_{max} 代表进程可分配的最大内存，lowMem 为低内存运行标志。

(3) 电池可信性评估

电池可信性评估考虑的因素主要包括电量、电压及电池温度。对于电量的评估，只要获得当前电池的电量值即为电量参数的评估值。电池的工作温度为 0~40 度，但实际中 20~30 区间为正常使用时的温度范围，一般超过 50 度即为危险区间，因此，将 30 度作为电池温度的基准值，评分为 80。正常电压为 3.6 v~4.2 v，充电电压可达 4.3 v~5.5 v，随着电量的消耗电压会逐渐降低，当电压低于 3.6v 时会自动关机。因此，本文选择将 3.9 v 作为电压的基准值参与电压的评估。

在电池可信性评估过程中，当电池健康状态属性异常时，评分结果为 0，在健康状态属性为良好时，电池温度的可信性评估规则如公式(4)所示。

$$P_{BatTemp}=\begin{cases}80+(P-T)\times\dfrac{100-80}{P-20} & 20\le T\le P\\ 80-(T-P)\times\dfrac{80}{50-P} & 30<T<50\\ 0 & T\ge 50\end{cases}\tag{4}$$

式(4)中，T 为电池的当前温度，P 为基准温度，取值为 30。

电池电压的可信性评估规则如公式(5)所示。

$$P_{BatVoltage}=\begin{cases}80-(P-V)\times\dfrac{80}{P-P_{\min}} & P_{\min}\le V\le P\\ 80+(V-P)\times\dfrac{100-80}{P_{\max}-P} & P<V\le P_{\max}\\ 0 & V<P_{\min}\ \|\ V>P_{\max}\end{cases}\tag{5}$$

式(5)中，P 表示基准电压取值为 3.9 v，P_{min} 表示正常运行的最低电压取值为 3.6 v，P_{max} 表示正常运行的最高电压取值为 4.2 v，V 表示当前的电池电压。

一般地，将单个指标的评估结果累加并取均值，即可得到当前硬件环境的可信度评估值。

3.2. 硬件状态监测

Android 内核文件系统 Linux 提供了 sys 文件，可以通过访问目录/sys 实时查看系统运行时的内部数

据和一些重要的系统信息。

(1) CPU 状态监测

Android 四层架构的最底层即是 Linux 内核，Android 系统的一切服务都是以 Linux 为基础[12]。Android 文件系统如图 2 所示。

移动设备会由于 CPU 负载过高导致发热，从而造成整体性能下降，甚至 ANR (Application Not Responding)等一系列问题。图 3 为使用 ADB [13]工具通过访问/sys/devices/system/cpu/cpuNum/cpufreq/目录下的 scaling_cur_freq 文件得到的 CPU 频率信息。

CPU 状态参数是以后台服务的形式实时采集，由于 CPU 状态监测是一个耗时的任务，使用普通的 Service 会造成主线程阻塞，导致应用的崩溃问题。因此，本文选用 IntentService 代替 Service 实现状态监测任务，具体的状态监测过程如表 2 所示。

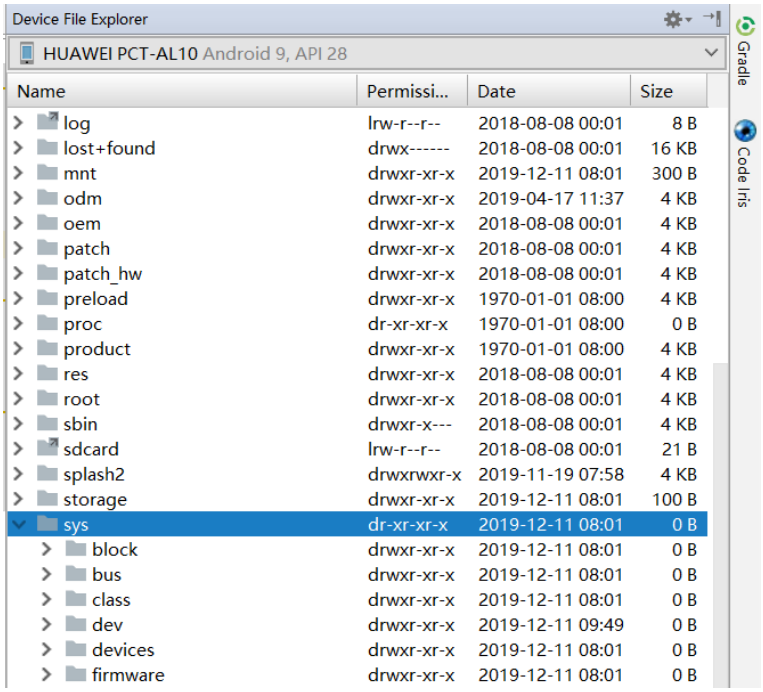

Figure 2. Android file system structure
图 2. Android 文件系统

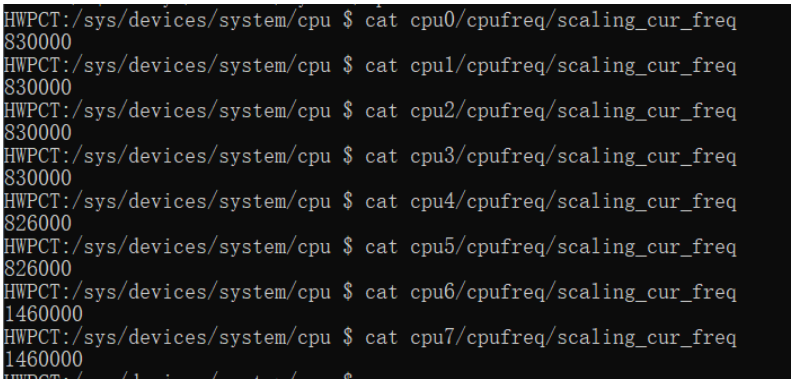

Figure 3. CPU frequency information extraction
图 3. CPU 频率信息

Table 2. CPU status monitoring process
表 2. CPU 状态监测流程

```
1. 首先获取当前设备 CPU 内核数量
int cpuCores = Runtime.getRuntime().availableProcessors();
2. 遍历/sys/devices/system/cpu 目录，读取内核实时频率信息
FileReader fileReader = new FileReader("/sys/devices/system/cpu/" +
                    cpuNum + "/cpufreq/scaling_cur_freq");
    BufferedReader bf = new BufferedReader(fileReader);
    Cpufrequence = bf.readLine().trim();
3. 遍历/sys/class/thermal 目录，读取当前内核温度信息
FileReader fileReader = new FileReader("/sys/class/thermal/thermal_zone/" +
                    coreNum + "/temp");
    BufferedReader bt = new BufferedReader(fileReader);
    long temperature = Long.parseLong(bt.readLine()) / 1000.0;
4. 对获取到的数据进行格式化处理，保存并通知 UI 刷新显示。
```

(2) 内存状态监测

内存消耗是影响移动平台应用性能的一个重要指标。Android 为设备的每个应用都设定了一个最高的可用内存阈值，当一个应用的内存超过该阈值就会发生 OOM (Out Of Memory)错误，进而引发程序卡死及界面崩溃等问题。为此，论文选用 Android 原生 API 实现对系统及进程的内存状态进行实时监控。Android 系统不仅包含了 C/C++语言实现的 nativ 进程，还有 Java 语言实现的 dalvik 虚拟机进程。因此，Android 内存分析包含 native heap 和 dalvik heap 两部分。

Android 程序框架层提供的 ActivityManager 类，可以获得当前系统中正在运行的应用进程、服务、任务和内存等信息。对于系统内存使用情况，首先通过 Context 的 getSystemService 获取 ActivityManager 对象，然后调用其 getMemoryInfo 函数可以获得系统的内存状态信息，该函数需要传递 MemoryInfo 对象作为参数，而且将返回的内存信息存放到该 MemoryInfo 对象中。MemoryInfo 中包含的内存相关的重要属性如表 3 所示。

Table 3. Attribute description of MemoryInfo
表 3. MemoryInfo 属性信息

属性	含义
totalMem	可访问的总内存容量
availMem	系统当前可用内存容量
threshold	低内存门限值
lowMemory	低内存运行标志

Table 4. Memory status monitoring process
表 4. 内存状态监测流程

```
1. 获取 ActivityManager 对象
ActivityManager manager = (ActivityManager)getSystemService(ACTIVITY_SERVICE);
2. 获取系统内存信息
ActivityManager.MemoryInfo info = new ActivityManager.MemoryInfo();
manager.getMemoryInfo(info);
sumMem = formatSize(info.totalMem);
availMem = formatSize(info.availMem);
thread = formatSize(info.threshold);
isLow = info.lowMemory;
```

续表

3. 检测当前系统中运行的进程，并查看各个进程的内存占用情况
List<ActivityManager.RunningAppProcessInfo> **appProcessList** = manager.getRunningAppProcesses();
Debug.MemoryInfo[] **memoryInfo** = manager.getProcessMemoryInfo(memoryPid);
totalPss = memoryInfo[0].getTotalPss();
dalvikPss = memoryInfo[0].dalvikPss;
nativePss = memoryInfo[0].nativePss;
4. 对内存使用情况进行汇总，通知主线程在 UI 刷新显示。

进程的内存使用情况的原理是调用 ActivityManager 类的 getProcessMemoryInfo 函数，并传入运行进程 ID，得到 Debug.MemoryInfo 数组对象，Debug.MemoryInfo 继承 Parcelable，用于统计进程的内存信息。通过调用 Debug.MemoryInfo 相应方法可获取该进程的堆栈内存信息。

内存状态监测被设计为一个服务，在后台实时对系统中内存使用情况进行统计，内存监测的过程如表 4 所示。

(3) 电池状态监测

电池对于系统的威胁性相对较低，但移动终端的运行必须得到电池的能量供给。因此，电池状态也是可信评估的重要指标。

电池监控采用监听系统广播的方式。当终端电池状态发生变化时，Android 系统会对外发送一条常量系统广播，其 Intent 对象携带标志电池状态的属性，通过设置监听对应 Intent 的 BroadcastReceiver，即可获取终端电池相关信息。因此，电池监控定义一个监听系统广播的广播接收器 BroadcastReceiver，用来接收及处理电池状态的信息。表 5 列出了系统广播携带的有关电池状态的重要属性信息。

Table 5. Attributes of battery status
表 5. 电池状态属性

属性	状态
电池健康状态	BATTERY_HEALTH_OVERHEAT (电池温度过高)
	BATTERY_HEALTH_DEAD (无电)
	BATTERY_HEALTH_OVER_VOLTAGE (电压过大)
	BATTERY_HEALTH_COLD (电池温度过低)
	BATTERY_HEALTH_GOOD (电池状态良好)
level	电池当前电量
scale	电池最大容量
voltage	电池电压
temperature	电池温度

电池监控的实现过程包括广播接收器的注册、广播接收器处理逻辑的定义和监测结果的发送，实现的监测流程如表 6 所示。

4. 硬件环境可信评估方法验证

Android 是基于 Linux 内核实现的，在内核文件中记录了系统的硬件运行参数，同时 Android 系统本身也提供了相应的系统广播和类库，通过访问内核文件及调用系统 API 的方式即可获取系统硬件状态参数。

为此，硬件环境可信评估方法的验证选用基于 Android 操作系统的荣耀 V20 手机作为移动终端测试设备，负责终端硬件环境监测、软硬件环境可信评估及用户间的交互。

Table 6. Battery status monitoring process
表 6. 电池状态监测流程

1. 注册接收电池状态的广播接收器
BroadcastReceive receiver = **new** BatteryBroadCastReceiver();
registerReceiver(**receiver**, **new** IntentFilter(Intent.**ACTION_BATTERY_CHANGED**));
2. 定义广播接收器处理逻辑，计算电池状态信息
class BatteryBroadCastReceiver **extends** BroadcastReceiver {
level = intent.getIntExtra(BatteryManager.**EXTRA_LEVEL**, 0);
scale = intent.getIntExtra(BatteryManager.**EXTRA_SCALE**, 1);
battery = (level * 100) / scale;
power = intent.getIntExtra(BatteryManager.**EXTRA_VOLTAGE**, 0);
technology = intent.getStringExtra(BatteryManager.**EXTRA_TECHNOLOGY**);
health = getHealth(intent.getIntExtra(**"health"**, BatteryManager.**BATTERY_HEALTH_UNKNOWN**));
}
3. 向主线程发送状态广播，通知 UI 显示

4.1. 移动智能终端的可信评估系统

采用 C/S 架构设计移动智能终端的可信评估系统用于硬件环境可信评估方法验证。客户端负责从 Android 内核获取硬件状态参数并实时监测硬件环境状态变化，服务器端对大量的 APK 样本进行模型训练，并对客户端提交的待测 APK 进行权限预测，反馈预测结果给客户端，使其作为安全决策的依据。客户端与服务器之间通过网络进行通信。系统总体架构如图 4 所示。

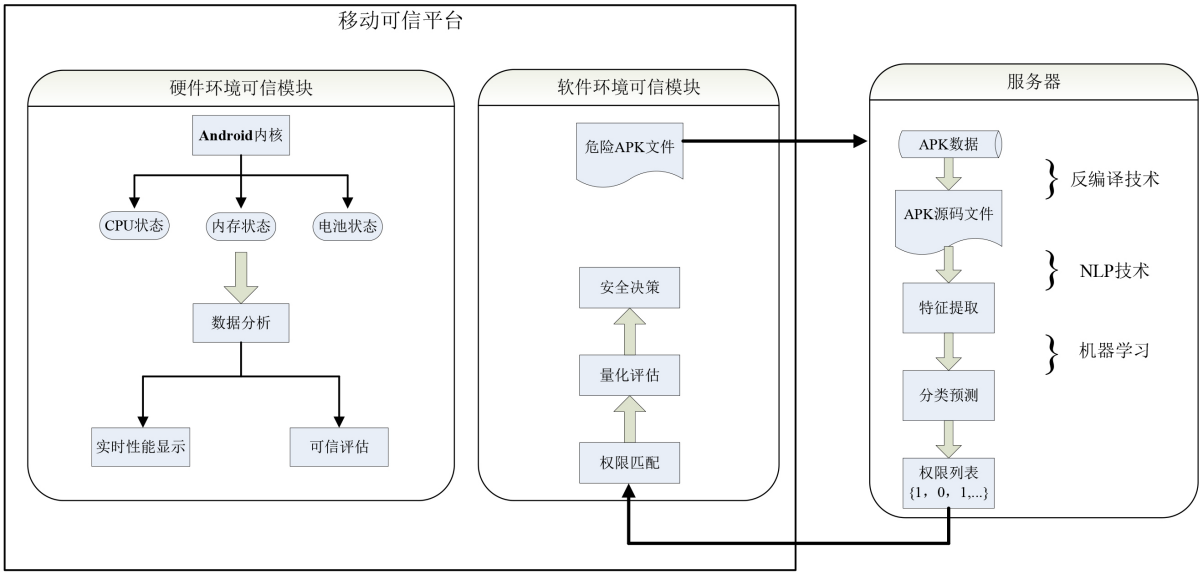

Figure 4. Overall architecture of the trusted system
图 4. 可信系统总体架构

部署移动终端可信系统后，便可对当前的硬件环境各项指标进行性能监测，并对检测结果进行实时分析。图 4 中，硬件环境可信评估时，客户端实时从系统内核获取硬件状态参数，进行量化评估后得到当前硬件环境可信等级，将其作为当前系统硬件环境的性能评估指标并以此为应用软件的运行提供参考

依据，为系统应用层提供可信的底层硬件运行环境。

4.2. 硬件环境监测

监测项按照性能指标分为 CPU 性能、内存使用和电池状态。Android 可信监测软件的性能监测任务都是以服务的形式在后台实时采集数据，再将性能数据以广播的形式通知给 UI 显示。移动终端可信系统的运行效果如图 5 所示。

图 5 中检测到的数据均为系统运行所产生的实时数据。从数据分析可知，选用的测试设备包含 8 个 CPU 内核，当前运行频率及温度均在正常阈值内，同时电池及内存消耗均表现正常，系统当前的整体硬件性能良好。

Figure 5. Runtime performance of the trusted system
图 5. 可信系统运行效果

4.3. 硬件环境可信评估

硬件环境可信评估完成 6 类性能指标监测结果的量化，并根据性能综合得分情况，给出当前系统硬件环境的可信度。图 6 为硬件环境监测功能的执行结果。

结合图 5 中监测的性能实时数据进行分析评估表明：

1) 由于当前系统前台进程只有监测软件自身，CPU 频率及温度均处于比较低的运行状态，评分也就相对较高。

2.当前系统运行进程中内存消耗最高的为 29 MB，超出了内存消耗的基准值 26.9，评分低于其基准分 80 分。

3) 电池模块，其性能指标均达到了基准值以上，评分高于基准值。

由此可见，论文提出的硬件环境可信评估方法实现了对移动终端硬件环境的实时监测和可信评估。同时，实验验证对于硬件环境的主观评价与系统评估结果几乎一致，证明了方法的有效性。

Figure 6. Evaluation results of hardware trustworthiness
图 6. 硬件环境可信评估结果

5. 小结

移动智能终端应用信息领域的安全成为亟待解决的问题。作为一个嵌入式系统，论文提出实时监测CPU、内存和电池等物理硬件参数的变化，进行硬件环境的综合可信评估。实验表明，研究的方法可以有效地评价系统的安全状态，支撑面向异常性能指标解决方案建立，保障移动终端运行的正确性和良好的用户体验。未来研究可扩展至更多硬件指标，进一步提升评估的全面性。

参考文献

[1] Trusted Computing Group. https://trustedcomputinggroup.org/

[2] 张焕国, 罗捷, 金刚, 等. 可信计算研究进展[J]. 武汉大学学报(理学版), 2006, 52(5): 513-518.

[3] Google. Profile Your App Performance. https://developer.android.com/studio/profile?hl=zh-cn

[4] Google (2021) Battery Historian Tool. https://github.com/google/battery-historian

[5] Duvvur, V. (2023) Modernizing with Confidence: Strategies for Enhancing Cybersecurity and Compliance in Legacy System Upgrade. *International Journal of Emerging Trends in Computer Science and Information Technology*, **4**, 41-48. https://doi.org/10.63282/3050-9246.ijetcsit-v4i4p105

[6] Zhao, X., Huang, G., Jiang, J., Gao, L. and Li, M. (2021) Research on Lightweight Anomaly Detection of Multimedia Traffic in Edge Computing. *Computers & Security*, **111**, Article 102463. https://doi.org/10.1016/j.cose.2021.102463

[7] Khalid, S. and Hussain, F.B. (2024) Volmemdroid—Investigating Android Malware Insights with Volatile Memory Artifacts. *Expert Systems with Applications*, **253**, Article 124347. https://doi.org/10.1016/j.eswa.2024.124347

[8] Bampatsikos, M., Politis, I., Bolgouras, V. and Xenakis, C. (2023) Multi-Attribute Decision Making-Based Trust Score Calculation in Trust Management in IoT. *Proceedings of the* 18*th International Conference on Availability, Reliability and Security*, Benevento, 29 August-1 September 2023, 1-8. https://doi.org/10.1145/3600160.3605074

[9] Nazir, A., Iqbal, Z. and Muhammad, Z. (2025) ZTA: A Novel Zero Trust Framework for Detection and Prevention of Malicious Android Applications. *Wireless Networks*, **31**, 3187-3203. https://doi.org/10.1007/s11276-025-03935-1

[10] 苏敏. Android 系统 UI 性能测试方法的研究[D]: [硕士学位论文]. 武汉: 武汉理工大学, 2018.

[11] Testin: 2013 年 8 月中国好应用质量排行[EB/OL]. https://www.docin.com/p-703974880.html, 2025-07-25.

[12] 文伟平, 梅瑞, 宁戈. Android 恶意软件检测技术分析和应用研究[J]. 通信学报, 2014, 35(8): 78-85.

[13] Votipka, D., Vidas, T. and Christin, N. (2013) Passe-Partout: A General Collection Methodology for Android Devices. *IEEE Transactions on Information Forensics and Security*, **8**, 1937-1946. https://doi.org/10.1109/tifs.2013.2285360