智能家居组调研与技术报告

邓胜亮、王博

2019 年 1 月 12 日

目录

1 项目概述 **3**

2 当前成果 **3**

2.1 微服务相关调研 . . . 3

2.1.1 微服务的概念 . . . 3

2.1.2 微服务的实践与挑战 . . . 5

2.1.3 基于微服务开展工作的困难 . . . 5

2.2 物联网相关调研 . . . 5

2.2.1 物联网开发简介 . . . 5

2.2.2 物联网开发中的可视化工具 . . . 6

2.2.3 物联网开发相关云服务 . . . 6

2.3 安卓平台资源使用问题相关调研 . . . 6

2.3.1 论文调研：DefDroid 相关 . . . 6

2.3.2 论文调研：FlowDroid 相关 . . . 7

2.4 智能家具相关调研 . . . 8

2.4.1 智能家居的概念 . . . 8

2.4.2 智能家居的应用 . . . 8

2.4.3 潜在风险 . . . 9

2.4.4 调查与购置设备 . . . 10

2.4.5 相关平台 . . . 10

2.4.6 SmartThings 开发平台简介 . . . 11

2.4.7 现有的 SmartApp . . . 12

3 其他工作 **12**

4 接下来的计划 **12**

1 项目概述

我们的基本出发点是基于静态分析技术，对应用程序的 API 调用、资源使用等行为进行整理和分析，进而从中发现程序漏洞、性能缺陷、隐私泄漏等问题。

要进行这样的分析，首先需要选择一个可行的特定场景，针对这一场景调研相关问题，开发出可以迁移到其他应用场景的可行方案。最初我们设想在安卓或是微服务架构应用中进行，因此首先做了这方面的调研。在调研过程中我们发现，安卓平台上的工作已经有不少比较成熟的成果；微服务作为一种应用架构，本身构造灵活多变，不易于分析。因此我们将目标转向了目前尚没有广泛应用、有比较好的未来发展前景的智能家居平台。

一方面，由于本学期学习任务重、时间有限，另一方面，我们希望通过充分的调研，为接下来的工作明确方向、奠定基础。因此截至目前，我们所做的主要是调研工作。在本报告中，我们着重介绍目前的调研结果。除了这一课题，我们组的成员王博、邓胜亮还在其他课题或项目中做了一些工作，将在 §3 中进行介绍。

2 当前成果

2.1 微服务相关调研

2.1.1 微服务的概念

首先，微服务是一种架构。要理解微服务架构，可以首先考察一下传统的单体（monolithic）架构。在过去，一个整体的应用程序构成一个单独的单元。企业应用程序通常建立在三个主要部分中：一个客户端用户界面、数据库和一个服务器端应用程序。服务器端应用程序将处理 HTTP/HTTPS 请求，执行特定领域逻辑，通过数据库进行检索和更新数据，选择并填充要发送到浏览器的 HTML 视图。这个服务器端应用程序是一个庞然大物———一个逻辑上的可执行文件。[1]

[1] https://baijiahao.baidu.com/s?id=1600354904549354089&wfr=spider&for=pc

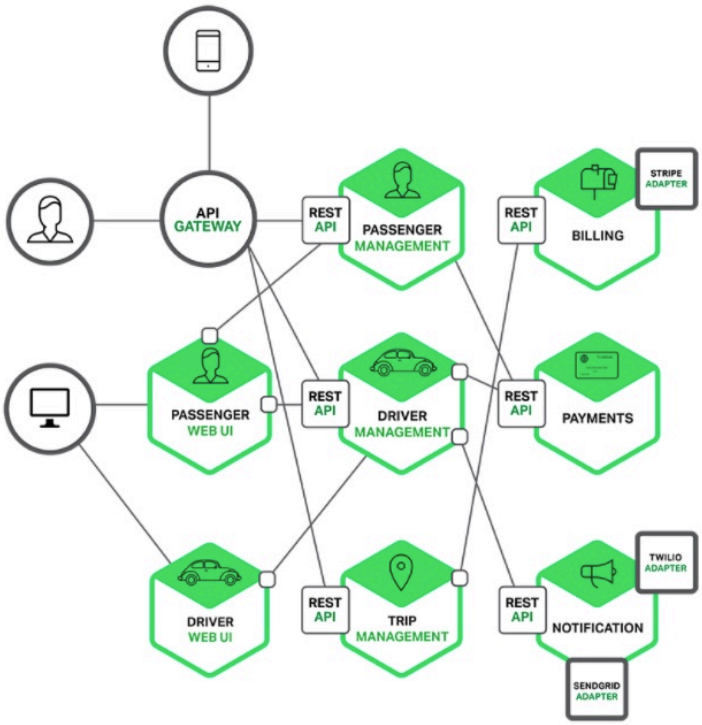

图 1: 微服务架构的打车软件示意图

作为一种传统的软件构建方式，单体应用很成功。尽管如此，它具有一些缺陷，尤其是当更多的组件被加入到应用中，每个小部分的更改都需要重新构建和部署新版本的服务器端应用程序，相互之间耦合度高，使得应用的模块化结构、合作开发变得困难和痛苦。

这些挫折引出了微服务架构风格：将应用程序构建为服务套件。除了服务是可独立部署和可伸缩的事实之外，每个服务还提供了一个严格的模块边界，甚至允许用不同的编程语言编写不同的服务，它们也可以由不同的团队来管理。总的来说，微服务具有下面的特点：

- 通过服务实现组件化；
- 根据服务能力进行管理；
- 产品不是项目；
- 分散治理。

微服务适用于边缘计算、雾计算这种计算资源分散，设备异构性大、计算存储资源小的应用场景，以及满足在确定的资源下，改变传统业务资源独立分配的模式，满足不同业务规格的弹性伸缩的需求。

2.1.2 微服务的实践与挑战

微服务为项目管理、产品开发和应用构建带来了新思路，被业界广泛实践和采用。

但是，在带来好处的同时，实践者们也发现了随之而来的新的问题和挑战。

首先，微服务架构对软件生命周期产生了不小的影响。由于微服务本身松耦合、分布式的特点，降低了应用实现和演化的难度，也对于整体应用的分析测试、调试和运维相比传统单体应用，提出了更高的要求。

其次，与微服务相配套的，需要一套相关的管理措施，包括服务注册、发现、负载均衡、灰度发布，以及避免功能性重复，明确架构，保障开发标准等的解决方案。

2.1.3 基于微服务开展工作的困难

通过查阅相关的资料，我们发现，在微服务应用中，工业界所关注的主要是良好的伸缩性、自动化部署、方便的调试，而应用出错往往是因为网络故障、硬盘故障、电源故障。相比之下，资源使用情况则不易定义且少有提及的方向。此外，由于微服务应用的灵活性和异构性，极有可能需要进行跨程序、跨语言的分析。这会使得相关的工作开展起来很困难。因此，我们暂停而转向了其他方向的调研。

2.2 物联网相关调研

2.2.1 物联网开发简介

物联网主要有三大部分[2]：

- 底层技术部分：传感器/设备，嵌入式处理器，网络器件。网络有蓝牙，Wifi，NFC，RFID 等。最近主流还有 NB-IoT 窄带物联网。
- 软件部分：终端设备驱动软件，服务器端软件，客户端软件。后两者是传统的应用开发，常见依托云服务。

  常见的数据流向：传感器/终端设备——边缘计算网关——服务器——客户端
- 应用部分：物联网的成果产出。

[2] https://www.zhihu.com/question/19751763

图 2: 物联网三大部分

隐私安全和资源有效利用都是物联网开发的重要问题。

物联网开发的不同部分，可能使用不同的语言。终端 IoT 设备常见 C 和 Java。服务器程序/客户端都是传统的软件开发，可以使用各种语言。

2.2.2 物联网开发中的可视化工具

终端硬件的程序开发，Node-RED, XOD, Wyliodrin, ReactiveBlocks

流程整体构建（前后端/UI），Zenodys, 阿里云 Link Develop 2.0[3]

2.2.3 物联网开发相关云服务

各大云服务厂商基本都有支持。如阿里云，亚马逊 AWS IoT, 微软 Azure IoT。

Azure 还有 Azure Sphere 云端安全服务[4]，及时保护监测响应各种危胁。

2.3 安卓平台资源使用问题相关调研

2.3.1 论文调研：DefDroid 相关

这篇论文于 2016 年发表在 MobiSys [3]。论文作者们分析发现：作为移动平台占有份最大的操作系统，安卓平台上并没有对应用的资源使用情况给出良好的管理和限制机制。由于移动应用平台的开发者相对年轻、开发经验不足、多数未受到严谨的软件开发教育、移动平台环境多变等原因，安卓平台上的应用程序多存在资源使用后不释放、频繁唤醒系统、过量使用移动

[3] https://www.liwei8090.com/6866.html

[4] https://azure.microsoft.com/zh-cn/services/azure-sphere/

数据、产生过多通知等行为。这些行为会导致过量的电池使用、流量消耗等问题，影响用户的使用。

尽管应用市场中有很多第三方工具，如 PowerTutor、WakeLock Detector、eDoctor 等可以用于检测和阻止这些行为。但是第三方 App 往往对应用行为理解有限，不能及时处理异常行为，有时还会影响用户的正常使用，降低了可用性。

该论文基于安卓 4.4.3 系统，对系统的定位服务、电源管理服务等模块进行了修改，加入监听机制，并添加了一个集中式的模块收集应用的资源使用情况，相应的架构图如下：

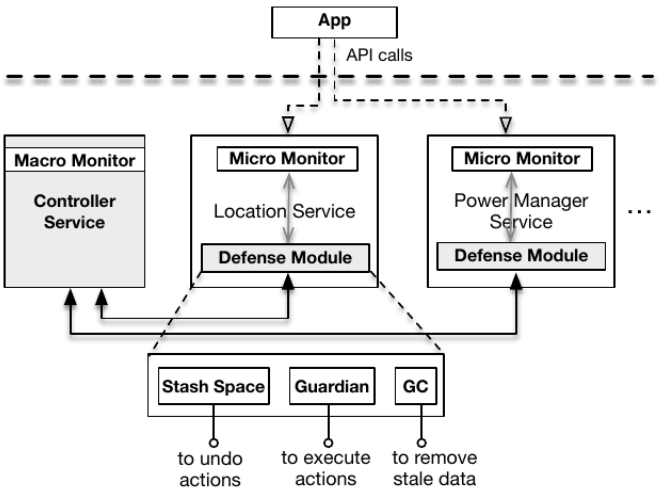

图 3: DefDroid 架构示意

监控模块对各应用的行为进行运行时的分析，并在恰当的时机采取诸如限制网络速率、降低定位精度等措施，有效减小应用的不合理行为的负面影响，同时又确保了可用性。

2.3.2 论文调研：FlowDroid 相关

FlowDroid [1] 是污点分析工具，可用于：程序编写失误检测，广告检测，恶意软件检测等。

静态检测的好处是，无动态检测的被探知现象，但是本身有难度：

- app 嵌入 Android 应用框架之中，回调较多，流程不单一
- 一部分执行功能实际在 manifest / 布局 XML 文件（UI）
- Android app 的 Java 代码中有其它 aliasing

FlowDroid 是字节码分析工具，on-demand 分析，效率较好。有关 Android 平台污点分析，恶意检测等工作，还有如下：

- EdgeMiner [2]: 静态分析辅助。用于分析 Android 应用框架代码，自动收集关于回调函数的信息。
- Amandroid [4]: 静态分析工具。支持组件间分析。
- AppContext [7]: 恶意软件识别。在静态分析基础上，使用机器学习方法 (SVM)
- IntentFuzzer: 随即白盒测试生成。基于 FlowDroid 的静态分析。
- CheckDroid [8]: 编程模式检查。从 Java 字节码出发，检查不良编程模式。
- DeepFlow [5]: 恶意软件识别。在静态分析基础上，使用深度学习。
- TASMAN [6]: 结合符号执行的数据流分析。基于 FlowDroid。

2.4 智能家具相关调研

2.4.1 智能家居的概念

对于智能家居，一个比较严格、完整的定义是：智能家居是在互联网影响之下物联化的体现。智能家居通过物联网技术将家中的各种设备（如音视频设备、照明系统、窗帘控制、空调控制、安防系统、数字影院系统、影音服务器、影柜系统、网络家电等）连接到一起，提供家电控制、照明控制、电话远程控制、室内外遥控、防盗报警、环境监测、暖通控制、红外转发以及可编程定时控制等多种功能和手段。与普通家居相比，智能家居不仅具有传统的居住功能，兼备建筑、网络通信、信息家电、设备自动化，提供全方位的信息交互功能，甚至为各种能源费用节约资金。

2.4.2 智能家居的应用

具体来说，智能家居的“智能”可以体现在包括路由器、摄像头、指纹门锁、插座、开关、灯具甚至下水管道等多种常见的家用物品、设备上。

抽象地去谈家居的智能化还是很难让人理解，三星 SmartThings 的宣传视频[5]中介绍了多种场景：

- 在门、窗打开时收到报警；
- 设定起床、睡眠时各种设备的自动开、关；
- 语音控制家用电器；
- 监控一些对小孩来说比较危险的地方，在小孩靠近时收到警报；
- 将 Water Sensor 放在水管下边，水管漏水时可以收到警报；

[5] https://www.youtube.com/watch?v=p1RfW5X8I4w

- 使用 SmartThings Hub 连接各种设备，使用手机 App 来方便地遥控各种设备。

图 4: 家居智能化示意

可以想象，一套这样的智能家居将极大地方便日常生活。

2.4.3 潜在风险

尽管拥有美好的前景，智能家具所具有的潜在缺陷也不容忽视。下面是一个例子

一位亚马逊电商网站的德国用户向当地杂志《c't》爆料，自己在向亚马逊讨要自己的个人活动语音数据时，对方竟然发给他 1700 份陌生人对话录音。今年 8 月，他曾根据欧盟颁发的《通用数据保护条例》行使了自己的权利——要求亚马逊向自己开放所有存储在云端的数据。而两个月后，亚马逊才姗姗来迟地发给他一个可下载的 100MB 压缩文件。然而，除了他自己在亚马逊网站上的搜索记录，还有数百份音频文件以及一份解释 Alexa 语音命令的 PDF 分类记录。不过这位用户并没有任何嵌入 Alexa 的硬件设备，也没有使用过 Alexa 的相关服务；此外，文件录音里的声音主人也跟他没有任何关系。[6]

也就是说亚马逊拥有这些录音并将它们发给了其他用户，这无疑是一件很可怕的事情。

智能家居具有运动、视频、音频等多种信息的收集和处理能力，并且被放置于用户的家中。如果开发者在开发相关应用时未能正确处理设备中各种资源的申请、持有和释放，轻则应用行为不符合预期、增加额外的电力等消耗，重则造成用户隐私泄漏，甚至侵害用户财产和生命安全。

[6] https://news.baidu.com/share/detail/9185548158132136914

因此，针对智能家居平台上的应用的安全性分析、隐私保护及漏洞防范是非常重要的。我们接下来的工作也将基于此展开。

2.4.4 调查与购置设备

智能家居的设备、平台与厂商非常多样化。

当前涉足智能家居的国内厂商[7]有海尔、京东微联、华为 HiLink、小米等。

国外厂商[8]有 Nest、Samsun、Philips Hue、Ecobee、Amazon 等。

被列入智能家居行列的设备包括灯泡、门窗传感器、运动传感器、电源插座、运动传感器等，种类功能多样。

2.4.5 相关平台

智能平台设备众多，又要达到“智能”的目标，自然少不了一个统一的平台来管理和控制。这一方面，全球多家知名企业参与其中。

2014 年，苹果在 WWDC 上发布了 HomeKit，根植于苹果 iOS 系统，依靠 iPhone/iPad 等硬件优势，吸引第三方硬件厂商接入。能够支持语音控制等。但是限于手机系统，用户无法通过占移动平台市场份额更大的安卓系统来使用，难以拉拢大多数用户。

Nest 是被谷歌收购的品牌，最初只有两款智能单品。被谷歌收购后，加速了平台化转型，在 HomeKit 发布后不久宣布允许第三方公司访问其设备，进行通讯和互联。但是在后期发展中，Nest 本身出现了一系列的问题，光环逐步褪去[9]，创新停滞，丧失了发展势头。

SmartThings 的发展早于 HomeKit 和 Nest，后来被三星收购。SmartThings 的主要定位是三星智能家居平台，关键是内置 Z-Wave 和 ZigBee 通信模块，用于兼容基于这两种模块进行通信的设备。三星还提供了相对友好的开发者社区与平台，因此成为我们后续调研的重点。

HiLink 是华为开发的智能家居开放互联平台，目的是解决各智能终端之间互联互动问题，功能主要是连接基于不同协议的设备。HiLink 的核心是 HiLink 协议，华为将其定义为智能家居之间的“普通话”，用于连接基于 WiFi、ZigBee、Z-Wave 等标准的智能家居设备。

这部分信息主要从`https://www.jiemian.com/article/1234481.html` 总结而来。

[7] `https://www.jianshu.com/p/493e67c8e555`

[8] `https://www.makeuseof.com/tag/smart-home-brands-can-trust/`

[9] `https://36kr.com/p/5118488.html`

2.4.6 SmartThings 开发平台简介

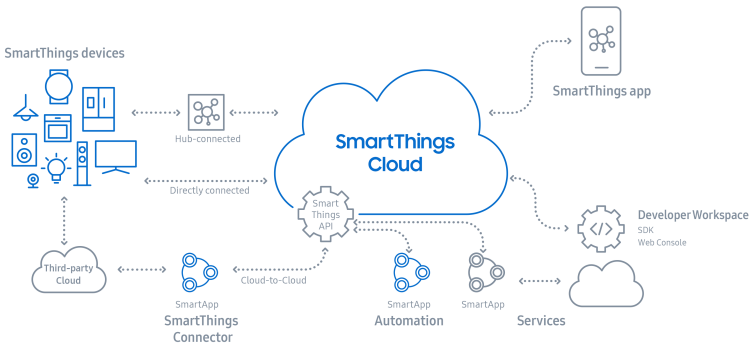

图 5: SmartThings 平台整体架构

上图是 SmartThings 智能家居平台的整体架构，该平台将智能家居的设备与“智能”进行了分离。每个设备直接或间接地连接到 SmartThings Cloud，向云端发送传感数据、从云端接收控制命令。

在云端，开发者通过编写 SmartThings App 来中心化地管理各种设备。SmartThings App 通过 WebHook 或 AWS Lambda 与开发者编写的应用交互，发送数据、接收命令。

用户在移动设备上安装 SmartThings 应用，通过该应用来管理各种设备和各种 App，方便地为家居添加各种自动化的功能。

作为一个例子，SmartThings 演示了如何使用 node.js 实现一个根据天气控制灯泡变化颜色的应用。

基于 WebHook 的调用使得开发者可以用任何语言、任何方式来实现 SmartThings App，只需要利用规定的接口与 SmartThings Cloud 交互即可。相关的接口定义（REST API）在这里可以看到。

SmartThings API 将设备进行了高层次的抽象，使得开发者不需要关心底层究竟是什么设备，只需要考虑他需要设备支持的功能。把这种抽象叫做 Capability。从这里可以看到所有 Capability 的列表。

举其中一例：能够设置色调的设备具有名叫 Color Control 的 Capability，具有 color、hue、saturation 三种属性，应用可以获取这三种属性的值，也可以通过 API 向该设备发送 setColor、setHue、SetSaturation 三种指令，来调节设备的颜色。

可以看到这种抽象大大简化了应用的开发，同时也清晰地定义了设备与自动化机制的界限。我们接下来的研究很有可能在这一系列 API 上进行。

2.4.7 现有的 SmartApp

我们在 GitHub 上搜索到了一些相关的开源项目，包括：

- tonesto7/nest-manager，将 Nest 集成进 ST；
- krlaframboise/SmartThings，Kevin LaFramboise's SmartThings Repo，将近 60 个；
- a4refillpad/Xiaomi，小米 device handler；
- codersaur/SmartThings，有一些 device handler 和 SmartApp；
- statusbits/smartthings，一些 SmartApp。

这其中包含了一些第三方开发者编写的应用，之后或许可以作为研究案例。

3 其他工作

邓胜亮同学最初在量子编程组，学习了量子计算基本原理，并做了一些关于线路映射问题的调研；参加了 NASAC2018 中的软件原型竞赛，负责了五个模块中两个模块的设计和实现，以及比赛现场的工具展示。

王博同学最初在图形组，调研了程序语言在 3D 渲染中的应用，主要是 Slang 语言以及其实现着色器特化的原理。

4 接下来的计划

- 深入了解当前业界的研究热点和主要方法；
- 学习 SmartThings API 相关应用案例，调查有关缺陷的具体表现形式；
- 初步实现缺陷定义、设计检测方法；
- 实现和完善一个实用的缺陷检测工具。

参考文献

[1] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *SIGPLAN Not.*, 49(6):259–269, June 2014.

[2] Y. Cao, Y. Fratantonio, A. Bianchi, M. Egele, C. Kruegel, G. Vigna, and Y. Chen. Edgeminer: Automatically detecting implicit control flow transitions through the android framework. In *NDSS*, 2015.

[3] P. Huang, T. Xu, X. Jin, and Y. Zhou. Defdroid: Towards a more defensive mobile os against disruptive app behavior. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 221–234, New York, NY, USA, 2016. ACM.

[4] F. Wei, S. Roy, X. Ou, and Robby. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1329–1341, New York, NY, USA, 2014. ACM.

[5] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1385–1392, 2013.

[6] K. Yang, J. Zhuge, Y. Wang, L. Zhou, and H. Duan. Intentfuzzer: detecting capability leaks of android applications. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 531–536. ACM, 2014.

[7] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 303–313, Piscataway, NJ, USA, 2015. IEEE Press.

[8] S. Yovine and G. Winniczuk. Checkdroid: a tool for automated detection of bad practices in android applications using taint analysis. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, pages 175–176. IEEE Press, 2017.