

物联网设备固件安全分析现状研究综述

马晓成, 王允超, 于洪洋, 魏 强

信息工程大学 网络空间安全学院 郑州 中国 450000

摘要 随着物联网技术的发展及其应用场景的不断扩大, 越来越多功能各异的物联网设备接入网络。万物互联时代到来的同时, 大规模物联网设备暴露在互联网中, 使得设备安全受到严峻的考验。固件是物联网设备中运行的软件, 决定着设备的行为及功能, 因此固件的安全性对保护物联网设备免受威胁至关重要。由于物联网技术分支众多且迭代迅速, 针对物联网固件的安全测试研究也在不断的迭代更新。近年来, 针对物联网固件的安全分析涌现了大量的研究, 但缺乏专业的文献对研究现状进行及时的总结。本文首先梳理了物联网架构及当下设备的应用场景, 然后介绍了固件的分类和通用工作框架, 并讨论了固件可能面临的安全风险。基于物联网固件所面临的安全风险及其安全分析所面临的挑战, 本文归纳出固件的一般流程, 即总体分为固件获取、固件解包和文件提取、程序信息恢复和程序分析四个部分, 并分别进行相关介绍。其中, 程序分析部分是发现固件潜在问题必不可少的环节, 本文从静态分析、动态环境构建及动态分析的角度进行介绍, 对逆向分析、污点分析、符号执行、相似性检测、动态仿真环境构建及模糊测试等现有的固件程序分析技术进行详细分析和对比。最后总结当前工作的优势与不足, 结合物联网固件的发展趋势, 对未来的研究方向进行了展望。

关键词 物联网设备; 固件安全; 程序分析; 漏洞; 污点分析; 固件仿真; 模糊测试
中图法分类号 TP311 **DOI号** 10.19363/J.cnki.cn10-1380/tn.2025.07.07

An Overview of the Current Research on Embedded Device Firmware Security Analysis

MA Xiaocheng, WANG Yunchao, YU Hongyang, WEI Qiang

School of Cyberspace Security, Information Engineering University, Zhengzhou 450000, China

Abstract With the development of Internet of Things (IoT) technology and the continuous expansion of its application scenarios, an increasing number of diverse IoT devices are being connected to networks. As the era of ubiquitous connectivity unfolds, a large-scale deployment of IoT devices on the Internet exposes them to significant security challenges. Firmware, which is the software running on IoT devices, directly determines their behavior and functionality. Therefore, the security of firmware is crucial for protecting IoT devices from threats. Due to the numerous branches and rapid iterations of IoT technology, research on security testing for IoT firmware is also constantly being iterated and updated. In recent years, a large amount of research has emerged on the security analysis of IoT firmware, but there is a lack of professional literature to timely analyze and summarize the research status. This article first outlines the architecture of the Internet of Things and the current application scenarios of devices, then introduces the classification and general working framework of firmware, and discusses the security risks that firmware may face. Based on the security risks faced by IoT firmware and the challenges it faces in security analysis, this article summarizes the general process of firmware, which is divided into four parts: firmware acquisition, firmware unpacking and file extraction, program information recovery, and program analysis, and provides relevant introductions. Among them, the program analysis part is an essential step in discovering potential firmware issues. This article introduces from the perspectives of static analysis, dynamic environment construction, and dynamic analysis, and provides a detailed analysis and comparison of existing firmware program analysis techniques such as reverse analysis, stain analysis, symbol execution, similarity detection, dynamic simulation environment construction, and fuzz testing. Finally, the advantages and disadvantages of current work are summarized, and future research directions are discussed in combination with the development trend of IoT firmware.

Key words IoT devices; firmware security; program analysis; vulnerability; taint analysis; firmware emulation; fuzzing test

通讯作者: 魏强, 博士, 教授, Email: funnywei@163.com。

本课题得到国家重点研发计划(No. 2019QY0500)资助。

收稿日期: 2023-09-05; 修改日期: 2023-12-26; 定稿日期: 2025-06-26

1 引言

随着万物互联时代的到来,不同的应用需求将数以万亿计的新设备接入网络,推动生产生活和社会服务更加网络化、智能化和精细化,也促进了经济社会发展向着更为智慧高效的方向迈进。根据 GSMA 发布的《The mobile economy 2020(2020 年移动经济)》^[1] 报告显示,2019 年全球物联网设备总连接数达到 120 亿,预计到 2025 年左右,全球物联网设备总连接数将达到 246 亿。根据《阿里云物联网平台技术白皮书 V1.0》^[2],中国物联网设备连接数在 2016~2020 年间,年同比增长率为 46.1%,经历了高速增长,预计到 2025 年会达到 80 亿。物联网应用大规模走入人们的生活,其应用场景也在不断的扩大,物联网设备在生活中已经随处可见。

物联网设备的激增给我们的生活带来了巨大的好处,但出于保密性考虑,相当数量的物联网设备在产品投放前几乎不做安全测试,此外,物联网设备的系统架构与通用 PC 不同,往往缺乏安全防护机制。2016 年, Mirai 僵尸网络感染了数十万台物联网设备,包括家用路由器、摄像头以及家用空气质量检测仪等,并利用这些设备发起大规模 DDoS 攻击,导致 OVH、Dyn 等主流站点出现瘫痪现象。2016 年,一群安全研究人员发现了特斯拉 Model S 电动汽车的安全漏洞。他们成功地远程入侵了车辆的娱乐系统,并能够控制车辆的门锁、窗户和充电系统等功能。2017 年,在中东地区某能源系统里发现了 Trisis 病毒,攻击了施耐德电气的安全仪表系统。安全仪表系统是保障变电站控制设备运行安全的控制系统,依然会被黑客远程攻击。2019 年,安全研究人员发现多款智能家居设备存在严重安全漏洞,这些设备包括智能插头、智能相机、安防产品等,涉及 iHome、Wyze、Oco 和 Zmodo 等众多厂商。2019 年,领先的 MCU 操作系统 FreeRTOS 的 13 个关键漏洞使各种设备面临安全威胁,同年 7 月,另一款流行的 MCU 操作系统 VxWorks 被爆出了 11 个漏洞。2020 年,全球最大的信号灯制造厂商 SWARCO 被曝严重漏洞,该漏洞可以被利用破坏交通信号灯,造成交通瘫痪,甚至引发交通事故,威胁到人们的生命安全。2022 年,多款网络安全设备被曝出严重的安全漏洞,从防病毒、终端安全的防护产品,到安全网关、防火墙、入侵检测产品均有涉及,如飞塔(Fortinet)公司的安全操作系统 Fortinet FortiOS 被爆出的 CVE-2022-40684 已经被大肆利用,黑客利用该漏洞进入目标网络,从而加密文件进行勒索。2016 年以来, CNVD^[3]

公开收录的网络设备漏洞数量以每年 30%左右的速度增长,2022 年出现的网络设备、智能设备、工业控制系统等物联网设备的漏洞已经占到当年所收录总漏洞的 20%。

物联网设备的功能主要由固件提供,固件是控制设备硬件的底层软件,在确保设备按预期运行方面发挥着关键作用。随着越来越多的设备接入物联网,针对固件的攻击变得越来越普遍,一旦攻击者获得访问权限,就有可能远程控制设备,窃取敏感数据,或作为跳板做进一步攻击等。近年来,业界不断尝试将各种在通用计算系统上应用良好的漏洞分析与挖掘技术应用到嵌入式设备,应用过程中会受到以下几种阻碍^[4]:

1. 底层硬件平台复杂异构

物联网设备具有多样的种类和众多的厂商,其软硬件实现之间存在巨大的差异。底层硬件平台的复杂异构是物联网设备安全分析的一大障碍。不同设备使用不同处理器和内存架构,其固件的文件格式和装载规则也各不相同。因此,传统针对通用 x86 或 x86_64 平台的动态和静态分析方案无法直接应用于这些设备。

2. 文档/源码不公开

出于设备安全和知识产权的考虑,大多数厂商不会公开设备固件源码。此外,嵌入式固件通常是根据产品特性定制的程序,即使是相同的网络协议和硬件接口,不同厂商和开发人员的实现方式也各有差异。这种固件源码的封闭性和多样性给通用测试带来了很大的障碍。即使是基于开源的 OpenWrt 的固件,其中的定制部分也是保密的,以保护商业机密。对于研究[5]中分析的 5023 固件映像,作者无法找到与该固件相对应的源代码或文档。总之,固件映像通常缺乏用于进行静态分析的源代码。

3. 运行环境受限

出于成本和安全考虑,厂商在开发物联网设备的过程中通常只保留必要的功能和组件,不必要的组件会被移除,调试接口也会被封锁,很难直接对真实设备进行动态测试。即使可以使用二进制重写或插桩等方式,在真实设备上部署动态分析方法,但由于硬件资源的计算能力有限,分析性能会受到极大的限制,从而影响分析结果的准确性。

4. 固件程序的封闭性

固件通常会限制外部用户对固件的访问权限,只允许授权用户或系统管理员用户进行部分访问和修改。嵌入式设备程序通常通过外围设备与外部进行交互,比如 Wi-Fi 模块。为了让用户能够直观地操

作设备, 最常见的做法是提供具体的终端程序或 Web 界面。然而, 对于用户或分析人员来说, 这些在后台运行的处理程序如何运作以及相互交互, 往往是一个黑盒子, 难以被完全理解和分析。

由于固件在物联网设备中的重要性, 国内外针对嵌入式固件的安全分析涌入了大量研究, 并取得了一定的进展。2021 年, 文献[4]围绕嵌入式设备面临的安全风险, 对相关技术进行了综合分析和评估, 文献[6]总结了典型的固件安全缺陷类型及机理, 综述了固件安全缺陷检测技术与工具的优势与不足。但由于物联网技术分支众多且迭代迅速, 应用场景也在不断的扩大, 针对物联网固件的安全测试研究也在不断的迭代更新。本文结合物联网场景梳理了当前物联网固件的工作框架及面临的安全风险, 总结了固件分析的一般流程, 并对固件程序分析技术做了分析与探讨, 给出了当前发展趋势下未来的研究方向。

本文余下部分组织如下, 第二章系统的介绍了固件的发展现状, 第三章归纳了嵌入式固件暴露出的风险, 第四章详细的总结了固件分析的一般流程, 第五章系统的从静态分析和动态分析的角度对固件程序分析技术做了对比和探讨, 第六章给出了现有工作的简单总结和对未来工作方向的展望, 第七章总结全文。

2 固件发展现状

固件^[7]是嵌入在硬件设备中的一种软件, 负责控制硬件组件并为设备提供基本功能。本文的讨论对象是物联网设备中的嵌入式固件, 在后面内容中

简称固件。嵌入式系统固件通常会被集成到嵌入式设备中, 并存储在电可擦除只读存储器(EEPROM)或闪存芯片(FLASH)中, 用户可以使用特定的刷新程序来升级固件^[8]。

与普通计算机(PC)相比, 嵌入式设备存储空间小、计算资源有限, 设备运行所需的全部软件通常都被打包在固件之中, 包括操作系统内核、驱动、硬件抽象层、第三方库、应用程序等。固件存储在非易失性存储介质中, 在系统启动时被加载到内存中运行。在嵌入式系统启动的过程中, 最先运行的是 BootROM, BootROM 是系统中最底层的引导加载程序, 被厂商固化在只读存储器中, 无法修改, 负责最早的硬件初始化和引导工作, 并引导加载更高级别的引导加载程序(BootLoader)。BootLoader 是用来描述嵌入式系统中的引导加载程序的术语, 负责硬件初始化、加载和执行操作系统内核或其他应用程序, 并提供配置和调试功能。

2.1 工作场景

物联网是指将各种物理设备、传感器、网络结构和数据交互系统通过互联网连接起来, 以实现设备之间的智能互联和数据交换。如图 1 所示, 物联网架构中通常包括物联网设备、网络基础设施、云平台、边缘设备及应用程序等^[9]。其中应用程序是用于用户与设备交互的程序, 一般以 PC 客户端、Web 页面和手机 App 的形式存在; 网络基础设施主要提供了物联网设备之间的通信网络, 如局域网、无线网络、互联网等; 云平台是物联网数据的存储、处理和分析中心, 提供了大规模的数据存储和计算能力, 用于接收、存储和管理物联网设备产生的数据; 边缘

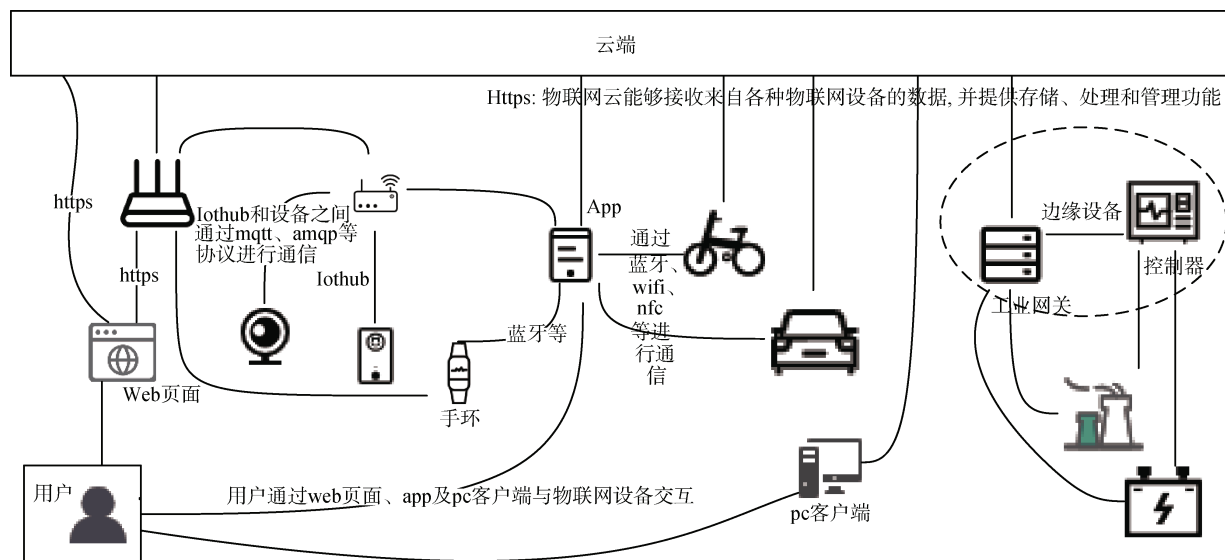


图 1 物联网应用场景架构图

Figure 1 IoT application scenario architecture diagram

设备是位于物联网边缘的设备,具有一定的计算和存储能力,可以帮助减少数据传输的延迟和带宽消耗,实现数据的本地处理和决策;物联网设备是物联网的基础组成部分,它可以是各种物理实体,包括传感器、控制器、智能设备等,这些设备能够感知环境、采集数据、执行命令,并通过网络连接与其他设备进行通信和交互。

目前比较主流的物联网应用场景主要有网络设备、智能穿戴、智能家居、车联网、工业互联网、云端设备等。网络设备是专门用于构建信息通信网络的硬件设备,起到将各种终端设备相互连接的作用,该类设备属于物联网设备中应用较早的一类。生活中常见的网络设备有路由器、交换机、调制解调器、网关、无线接入点、集线器、5G 基站等,除此之外,一些网络安全设备如防火墙、密码机、VPN 网关、入侵检测设备、入侵防御设备、网闸等也属于网络设备。智能穿戴设备是指能够穿着或佩戴从而与人体接触、采集人体信息,进而为用户提供个性化服务的设备,目前比较常见的智能可穿戴设备有 VR(虚拟现实)/AR(增强现实)设备、蓝牙耳机、智能手表/手环、智能眼镜、智能鞋袜、可穿戴医疗设备等。智能家居属于智能穿戴技术的扩展,其主要是以住宅为平台,通过先进的网络通信、嵌入式技术、云端控制等技术,将与家居生活相关的各单元如灯光控制、影音控制、安防保安、卫生清洁等有机的结合在一起,然后根据个人生活喜好进行定制,以此来提高人们生活环境的舒适度。车联网是指通过将车辆接入互联网,通过先进的传感器技术和通信技术,实现车辆、基础设施和云平台的实时通信,数据共享和协同行驶,以及车辆与道路基础设施的连接和交互。通过车联网,车辆可以实时获取导航指引、路况状况、交通信息等,为驾驶员提供远程车辆控制、智能导航、实时交通警示、智能驾驶辅助等服务。工业控制互联网是指将传统的工业控制系统与互联网技术相结合,以实现工业设备之间、设备与云平台之间的信息交换、远程监控和控制。工业控制互联网的应用范围非常广泛,涵盖了制造业、能源领域、交通运输、城市基础设施等行业。

2.2 工作框架

由于固件的应用场景非常广泛,不同固件的差异较大,但总体的工作框架可以划分为应用软件层、组件和服务层以及系统支撑层。如图 2 所示,其中实线代表通常情况下存在的,虚线代表因计算和硬件资源有限而可能不存在的。

应用软件层是位于固件架构的最高层,负责实

现具体的应用功能和业务逻辑,通常根据特定的需求和应用场景开发,利用底层的硬件和软件资源来实现特定的功能和任务。软件层主要包括用户界面的设计与实现、数据存储和处理、应用功能实现等。用户界面具体体现在与用户的输入输出进行交互,如显示屏、按键等设备控制和交互,以及 Web 界面的交互等。应用功能实现具体是指根据系统的要求和设计目标来开发特定的功能模块和业务逻辑,包括传感器数据处理、控制逻辑实现及通信协议处理和具体实现等。数据存储和处理则是对应用程序层需要的数据进行存储和处理,如存储配置信息、保存历史数据,数据分析等。除此之外,软件层还提供通信和网络功能的配置和远程管理,及可能发生的错误和异常处理等功能。

组件和服务层位于应用软件层之下,用于提供丰富的功能支持,以满足特定的应用需求和系统要求。组件和服务层通常以库的形式存在,提供可重用的功能和接口给开发人员使用,以达到模块化设计和代码复用的目的,提高开发效率。组件和服务层会提供用于支撑软件层的框架,如虚拟文件系统、设备框架、日志框架、通信协议栈、网络框架、图形用户界面等。开发人员可以基于这些框架进行开发,并根据具体的应用需求定制应用软件层,但在这些框架中会引入大量的第三方开源框架,从而容易引入未知的威胁。

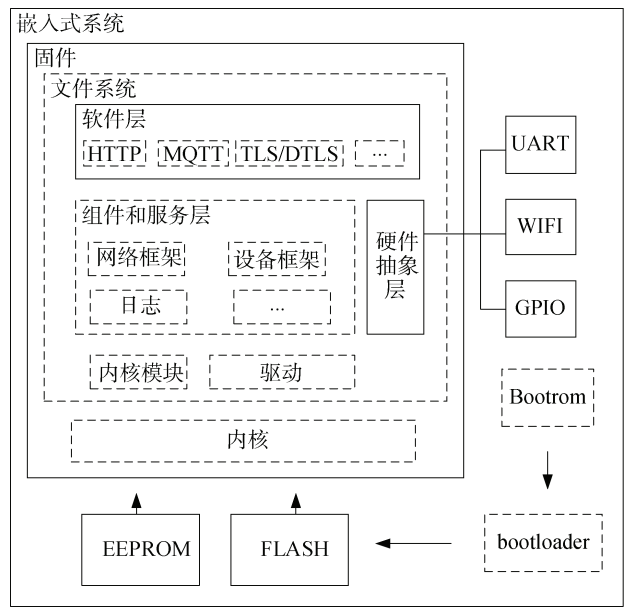


图 2 固件工作框架图
Figure 2 Firmware working frame

系统支撑层用于提供底层的功能和服务,为固件的其他组件和服务层提供支持和基础,其具体包

括操作系统内核及其内核模块、驱动以及硬件抽象层等。硬件抽象层是一种用于抽象和封装底层硬件细节的软件层, 它隐藏了底层硬件的具体细节, 提供一组统一的接口和功能, 以简化上层软件对硬件的访问和控制。驱动程序是用于控制和管理硬件设备的软件模块, 提供了与硬件设备交互的接口和功能, 使操作系统或其他软件能够与硬件设备进行控制, 它通常由硬件设备的制造商提供, 以适配特定的硬件设备和操作系统。内核是操作系统的基本构建块, 负责管理硬件资源的统一调度, 并为上层应用程序提供执行环境和系统调用接口, 其功能主要包括中断和异常处理、进程管理、内存管理、文件系统管理等。内核模块是指在操作系统内核中以插件形式加载和卸载的可执行单元, 用于扩展操作系统的功能, 增加新的设备支持或提供额外的功能。由于并不是所有固件都包含操作系统, 所以内核及内核模块并不是必须存在的, 但大多数固件都会存在驱动或硬件抽象层。

2.3 固件分类

嵌入式系统可以根据不同的标准进行分类, 如使用领域、计算能力、大小等。但我们关注的是嵌入式系统中的漏洞检测, 所以我们遵循文献[10]提出的, 依据嵌入式设备是否使用操作系统及使用操作系统的类型, 将其划分为以下三类并作简单介绍。

2.3.1 基于通用操作系统的固件

通用操作系统经常被改装用在嵌入式系统中, 通常基于通用操作系统的特定版本和配置, 在其基础上进行裁剪、优化和配置, 以满足嵌入式系统特定的需求。常见的嵌入式通用操作系统有嵌入式 Linux 系统、Android 和 Windows IoT Core 等。其中, 由于 Linux 具有开放的源代码、广泛的社区支持和成熟的生态系统, 以及适应性强、可定制性高等优点, 嵌入式 Linux 固件在嵌入式系统领域占据了相当大的比例。该类固件开发简单, 可移植性较高, 但是固件所需存储较大, 其主要应用于智能家居、网络设备等。Android 固件在嵌入式设备中应用也十分广泛, 其可以提供丰富的应用程序和多媒体功能, 最为熟知的应用场景就是智能手机和平板电脑, 目前被广泛用于车载娱乐系统和智能媒体设备等领域。相对而言, Windows 操作系统在嵌入式领域占比较小, 但在一些 NAS 设备及车载系统中也有使用。

2.3.2 基于定制的操作系统的固件

针对计算能力比较低的物联网设备, 通常会采用定制化的操作系统, 这些操作系统可能更注重轻量级设计和实时性能, 不存在复杂的虚拟内存机制,

但内核和应用代码之间仍旧存在逻辑隔离。为了满足嵌入式系统实时性要求和资源受限的环境问题, 该类操作系统大多被设计成实时操作系统(RTOS), 如常见的 VxWorks、uC/OS、QNX、Nucleus、ZephyrOS 和 FreeRTOS 等, 其中 FreeRTOS 和 ZephyrOS 是免费并且开源的 RTOS, 在小巧灵活的同时保证实时性能及高度可定制化, 被广泛用于工业自动化、智能家居、智能穿戴等领域。VxWorks、uC/OS、QNX、Nucleus 都是商业化操作系统, 由专门的技术团队开发和维护, 通常提供完整的开发工具链、调试器和仿真器以及技术支持, 被广泛应用于各种实时应用中。其中 VxWorks 和 QNX 以其强大的实时性能、可靠性、安全性及丰富的功能支持被广泛应用于工业控制、航空航天、汽车电子、医疗设备等领域, uC/OS 和 Nucleus 是两个小巧的商业 RTOS, 其中 uC/OS 注重可移植性和灵活性, 可以轻松移植到不同的硬件平台和编译器上, Nucleus 则更加注重易用性和开发效率, 提供了丰富的软件组件和工具。

2.3.3 单片机固件

单片机固件是指在单片机上运行的固件, 不依赖于任何操作系统, 通常被称为裸机固件。该类固件通常以单个可执行的二进制文件的形式存在, 大多基于一个控制循环和从外设触发的中断以处理外部事件。开发者需要直接编写所有的代码来控制单片机的功能和操作, 包括硬件初始化、处理输入输出、资源管理、任务调度、执行算法和逻辑任务等。该类固件的代码可以完全进行定制, 也可以基于操作系统库进行开发, 最终将库和代码编译在一起, 形成单一的软件。单片机固件的优点是简洁高效, 由于没有操作系统的开销, 它们通常具有更快的启动时间、更低的资源消耗及较好的实时性能。单片机固件能够监测和控制各种传感器和执行器, 被广泛应用于如洗衣机、冰箱、空调等的家电控制, 如汽车电子系统的引擎控制单元(ECU), 以及各种小型的智能设备和医疗设备等。

3 固件面临的安全风险

嵌入式固件的本质是可执行的二进制代码, 这些代码直接运行在嵌入式设备上, 控制其功能和行为。然而, 由于固件开发人员的安全意识不强和安全编码能力不足, 固件往往容易存在安全缺陷。此外, 由于固件通常长时间运行且很难更新, 这些安全缺陷可能存在较长的时间, 使得固件持续面临潜在的风险。

3.1 Web 管理端攻击

随着嵌入式设备逐渐成为接入云端的重要组件,

很多物联网设备都提供了 Web 管理程序, 常见的如路由器、摄像头、防火墙、交换机等。这些 Web 程序通常通过 Web 浏览器进行访问和操作, 使用户能够配置、监视和控制嵌入式设备的功能。因为它们直接面向互联网, 并且可能涉及敏感信息和设备控制, 因此, Web 端面临的各种攻击, 如命令注入、未授权访问、信息泄露、缓冲区溢出造成的拒绝服务及远程代码执行等, 对物联网设备造成严重威胁。在 Costin^[11]、KARONTE^[12]、Snipuzz^[13]、Zheng^[14]、Chen^[15]、Wang^[16]等的工作中, 均对嵌入式设备的 Web 程序进行了安全分析, 并发现了大量的漏洞。

3.2 协议解析攻击

物联网设备常采用各种网络协议与外界交互, 这些协议定义了设备如何连接、通信和共享信息, 以实现物联网的互联互通。如果设备固件在解析协议方面存在问题, 可能会导致缓冲区溢出、格式化字符串、命令注入、协议状态转换错误等漏洞。攻击者通过发送精心构造的数据包, 触发这些协议解析过程中的漏洞, 可能导致设备或系统发生不可预料的行为, 如导致设备崩溃、执行恶意代码、绕过安全控制、窃取敏感信息等。按照 TCP/IP 四层网络模型划分, 物联网中常见的协议如表 1 所示。在 IoTHunter^[17]、IoTFuzzer^[18]、Snipuzz^[13]、Sulley^[19]、Boofuzz^[20]、Eduard^[21]、Liu^[22]、魏晓^[23]、Zhou^[24]、Chen^[15]等的工作中, 均对固件协议做了大量安全分析。

表 1 物联网常见协议

Table 1 Common protocols for the IoT

通信层	协议
应用层	MQTT、CoAP、HTTP、FTP、AMQP、STOMP、XMPP
传输层	TCP、UDP
网络层	IPv4、IPv6、OSPF、BGP
网络接口层	Zigbee、Z-Wave、BLE、Wi-Fi、Ethernet、LIN、CAN、RFID、NFC、USB、NB-IOT

3.3 操作系统脆弱性

嵌入式设备通常具有有限的资源和空间, 并且为了满足特定的需求, 开发人员可能会对系统进行裁剪, 包括选择较低版本的内核。低版本的内核可能存在已知安全漏洞, 例如缓冲区溢出等问题。这些漏洞可能会随着固件的使用而长期存在, 因为更新内核需要额外的工作和资源, 并且可能破坏设备的兼容性。攻击者可以利用这些漏洞来执行恶意代码或获取未经授权的访问权限。此外, 操作系统中的驱动程序也是潜在的攻击目标, 驱动程序通常与硬件交互, 攻击者同样可以通过寻找驱动中的

漏洞来进行攻击。针对操作系统的攻击, 著名的有针对 Linux 内核的 Dirty COW (Copy-on-Write) 攻击, 以及可以篡改嵌入式设备中的 USB 驱动程序的 BadUSB 攻击。

物联网厂商在修剪内核的过程中可能会错误地删除必要的安全操作(DSO), 从而导致内存泄漏、NULL 指针解引用等各种错误。CPscan^[25]是用于自动检测物联网内核中代码修剪引起的错误的系统, 主要通过精确识别物联网内核中的 DSO, 比较与其关联的安全关键变量的有界使用链(在可能受影响的代码段中使用变量的位置和方式)来推断 DSO 的安全影响。另外, 在 Firmsolo^[26]中对内核模块的安全性进行了分析, 在 Periscope^[27]通过分析设备与驱动的交互过程来分析漏洞。

3.4 开源组件安全性

嵌入式设备固件的开发中通常会引入许多第三方组件, 如 Busybox、OpenSSL 等。嵌入式系统往往需要处理各种复杂的任务和功能, 而开发团队很难从头开始编写所有必要的软件模块。因此, 借助现成的第三方组件可以提高开发效率并减少重复工作。第三方组件包括各种软件库、驱动程序、通信协议栈等等, 它们由独立的开发者或组织创建。引入这些组件可以帮助开发人员实现常见功能, 如网络连接、数据存储、用户界面等等, 同时提供高质量的代码和可靠性。但开源组件在便捷开发的同时, 也引入了大量未知的威胁, 典型的开源组件漏洞有影响 OpenSSL 加密库的 Heartbleed(心脏出血)漏洞, 以及影响 Bash 命令解释器的 Shellshock 漏洞等。

3.5 后门

后门^[28-29]通常指的是在设备或软件中留下的特定接口或功能, 目的是方便设备调试人员、管理人员或开发人员进行操作和管理。这些后门可能是有意安排的, 也可能是无意遗留的。后门的存在可以带来一些便利性, 比如简化设备的调试过程、方便管理人员进行远程访问或维护, 或者为开发人员提供特殊的权限或功能以加快开发速度。然而, 如果这些后门被攻击者利用, 就会对系统的安全性构成威胁。此外, 有时开发人员在开发过程中可能会留下开发痕迹, 例如口令硬编码或认证绕过方法等, 这些都可能成为潜在的安全漏洞。因此, 在软件或设备的设计和开发过程中, 需要谨慎考虑后门的存在, 并采取适当的安全措施来保护系统免受潜在的滥用或攻击。在 HumIDIFy^[30]、Costin^[31]、Firmallice^[32]、忽朝俭^[33]、Gerbil^[34]、Thomas^[35]等的工作中对后门的检测进行了深入研究。

4 固件分析的一般流程

由于大多数嵌入式固件是针对特定产品个性化定制的, 厂商不会公开固件的源代码及专业文档,

针对固件的安全分析一般要结合逆向工程实现, 其主要包含由固件获取、固件解包和提取、程序信息恢复及利用程序分析技术寻找固件脆弱性四个阶段。固件分析的大致流程如图 3 所示。

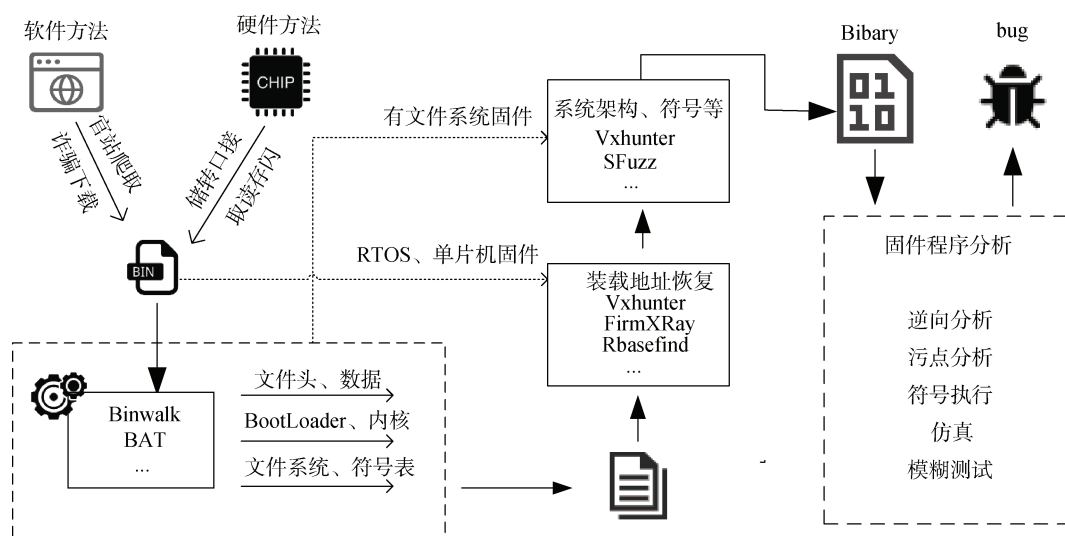


图 3 固件分析流程图

Figure 3 Firmware analysis flowchart

4.1 固件获取

固件获取是固件分析的第一步, 然而大部分厂商为了保证设备的安全与知识产权, 不会公开固件的源码^[6], 仅提供固件下载方式以供更新使用, 但提供方式不尽相同, 主要有以下几种方式: 在官网公开固件的所有历史版本; 仅提供最新版本固件, 无法获取其历史版本; 更新固件以增量包的方式提供; 隐藏固件下载, 仅设备发送更新请求获取, 通信流量可能加密; 固件以加密后的形式提供。因此, 针对不同的固件提供方式, 所采用的固件获取方式也不尽然相同, 以下分别从软件获取和硬件获取的角度分别介绍几种方式可供灵活采用。

通过软件方式获取固件。软件方法无须物理访问真实设备, 部分可以通过厂商的 Web 网站或 FTP 站点公开获取固件, 或者通过分析设备的网络流量获取更新连接。例如在 Costin^[31]和 Chen^[36]的工作中, 通过 Web 爬虫从各大设备厂商的官方网站获取了数万个物联网设备固件。

通过模拟设备行为诱骗下载。这算是广义上的软件获取方式, 它模拟真实设备与云端进行通信的数据格式和交互协议, 通过修改设备型号、版本号等参数, 从而欺骗云端以获取固件下载链接。Wei^[24]提出的幻影设备详细介绍了这种行为, Zhu 等人^[37]通过这种方式收集固件数据集。

直接从设备中转储固件。嵌入式设备固件通常

存储于焊接在设备主板上的闪存芯片中, 因此如果无法使用软件方式获取固件, 还可以通过拆解设备来获取。直接进行固件转储需要拿到物理设备, 并对其进行拆解, 然后通过 UART(Universal Asynchronous Receiver/Transmitter)接口或者 JTAG(Joint Test Action Group)接口对固件进行转储, 或者找到闪存芯片, 使用编程器对闪存进行读取。

4.2 固件解包和文件提取

供应商使用专用的包装器和文件格式打包固件, 或者使用私钥进行加密^[38]。目前的固件解包工具主要针对有文件系统的固件, 即基于通用操作系统的文件, 而无文件系统固件(如 RTOS 或单片机)整体就是一个可执行程序, 对该类固件的解包分析仍处于起步阶段^[91]。有文件系统固件可能包含六个部分: 文件头、被压缩的数据、加载引导程序(BootLoader)、文件系统、符号表、系统内核。在文件头部通常会包含特征字符串, 可以采用签名匹配的方式对固件内容进行识别, 进而解压缩并提取需要分析的文件和程序。在实践中, 可以利用不同的解包工具, 如 Binwalk^[39]、BAT^[40]和 firmware-mod-kit^[41]来提取固件。Costin^[31]等人通过开发 BAT 脚本解包固件, Costen^[36]、Tien^[42]、Cheng^[5]、Chen^[18]、Thomas^[30]及 Redini^[12]等在 Binwalk 固件提取工具的 API 的基础上构建了新的解包工具, 其中, Chen 等人^[36]针对收集的 23,035 个固件镜像进行解包, 仅解包了其

中的 8,617 个固件, 因此并不是所有的嵌入式设备都可以被解包分析。

目前的解包工具并不完善, 容易产生误判及解包失败, 固件解包失败通常有三种因素: 使用了工具不支持的文件系统、压缩包或固件被加密、固件中魔术值被修改。对于解包工具不支持的文件系统, 通常需要手动提取和解压, Zaddach 等人^[43]介绍了 SquashFS、CramFS、SquashFS、CramFS、ROMFS 等文件系统及 Tar、GZip、Ar、CPIO 等归档和压缩包的解压方法。对于加密的压缩包, 可以使用 PkCrack^[44]、ARCHPR^[45]等工具来进行破解, 对于加密的固件, 则可以通过动态读取内存, 逆向边界版本固件的升级程序、逆向 BootLoader 获取解密算法和密钥。对于魔术值被修改, 可以尝试寻找可疑的特征字段, 手工替换魔术。一旦固件被提取, 需要对内容进行过滤以获取所有相关信息, 这可以包括二进制文件、配置文件、数据文件识别等。为此, 可以使用不同的工具执行文件签名匹配, 如 Signsrch^[46], file^[47], Binwalk^[39]。

4.3 程序信息恢复

由于不同设备的固件运行环境复杂异构, 差异较大, 在没有专用文档的情况下, 需要尽可能的分析出固件程序分析所依赖的各种信息, 如系统架构、装载地址、符号信息等。

从有文件系统固件中提取出来的可执行文件一般为 ELF 文件格式, 其文件头中包含了处理器架构、装载地址和段信息。针对单片机固件和定制操作系统固件, 目前的研究思路通常是先分析固件在存储器中的装载基址, 然后将其转换为可供分析的镜像文件^[48]。因为固件中(尤其是 RTOS 系统)的许多数据引用或函数调用操作依赖于基址, 错误的地址将导致错误的数据引用或控制流跳转, 只有正确的基址才能将大多数数据引用指针链接到预期的目标。目前针对单片机固件和定制操作系统固件, 如何高效识别装载基址已经有了一些解决方案, 朱瑞瑾^[49]利用二进制文件中字符串的偏移量以及二进制文件中 LDR 指令加载字符串地址之间的联系来分析装载基址, Vxhunter^[50]通过数据指针的绝对地址与预期目标进行匹配来判定 VxWorks 的装载地址, FirmXRay^[51]在 Vxhunter 的思想提出了一种针对 BLE 固件基址识别算法, SFuzz^[52]将 Vxhunter 的思想在 RTOS 固件基址识别上做了实现。固件装载基址爆破工具 rbasefind^[53]集成了基于数据手册串口信息、固件常量、跳转表、魔术值、指令等多种装载基址识别方法。

在嵌入式固件分析过程中, 符号信息缺失是需

要面对的另一个问题^[54]。在固件开发过程中, 为了压缩固件体积和提高代码运行效率, 开发者倾向于使用静态链接的方式链接二进制文件, 或者剥离二进制文件的符号表, 增加了固件反编译的难度。符号表缺失可能导致静态分析工具无法准确识别函数和变量的边界及作用域, 使静态分析技术来分析固件受到一定的阻碍。研究人员在针对固件程序分析之前往往会尝试恢复符号表, Vxhunter^[50]通过识别内嵌符号表后修复函数名, SFuzz^[52]根据不同的场景实现了四种自动恢复函数语义的方法。SymLM^[55]利用深度学习, 通过上下文敏感执行感知代码嵌入预测条带二进制中的函数名。

4.4 固件程序分析

程序分析技术是用于分析计算机程序的方法集合, 旨在理解、验证和优化程序性能。在对固件漏洞挖掘的过程中, 通常使用各种程序分析技术对固件进行分析, 以找到潜在漏洞。固件分析面临多种挑战, 需要选择合适的程序分析技术单独或结合使用来对其进行安全分析。由于缺少源码和专业文档, 针对固件程序分析通常结合逆向工程, 在二进制代码层面进行分析。

二进制程序分析可以通过分析时是否执行代码分为静态分析方法和动态分析方法。静态分析是指在不运行程序的情况下对给定的二进制代码进行检查。它通常被设计为对整个程序进行推理, 并具有探索代码的所有潜在执行路径的能力。动态分析对象为可执行代码, 在动态运行程序的过程中分析寄存器、内存、输入输出等各类信息, 从而定位程序漏洞位置。针对嵌入式固件的动态分析可分为基于真实设备的分析和基于仿真环境的分析, 前者只适用于盲目性较大的黑盒测试, 后者则高度依赖于仿真技术的发展。静态分析技术主要解决固件底层平台复杂异构及文档、源码不公开的难题, 针对固件的静态分析技术主要有逆向分析、污点分析、符号执行、相似性检测等。针对固件的动态分析主要以动态仿真环境构建及模糊测试技术为主, 仿真环境构建以及基于仿真环境的动态模糊测试技主要解决固件运行环境受限及程序运行封闭性的问题。在后面的章节中我们将回顾这些方法, 以及它们在嵌入式系统固件分析中的应用。

5 固件程序分析技术

5.1 静态分析技术

5.1.1 逆向分析技术

针对固件的逆向分析, 通常是指在缺乏设计文

档和源代码的情况下, 为了获得固件内部模块功能和实现细节以及模块间的关系, 而对其内部二进制程序及其他相关文件进行分析的过程。逆向分析的目的在于尽可能多地获取程序内部状态信息从而更好的分析和理解程序。当前常用的固件逆向分析工具主要有 IDA Pro^[56]、Ghidra^[57]、Radare2^[58]、Capstone^[59]、Keystone^[60]等, 大多数固件的二进制程序逆向分析工作基于这些工具展开。针对 VxWorks 设备的逆向分析工具 VxHunter^[50], 就是基于 IDA Pro 及 Ghidra 这两个逆向工具开发的插件脚本。

表 2 固件程序分析技术对比

Table 2 Comparison of firmware program analysis techniques

分析方式	分析技术	解决缺陷
静态分析	逆向分析	
	污点分析	底层硬件平台复杂异构;
	符号执行	文档/源码不公开
	相似性检测	
动态分析	仿真	运行环境受限
	模糊测试	固件程序运行的封闭性

物联网设备提供的功能大多需要基于无线协议进行通信, 除了主流通信协议外, 越来越多的物联网厂商基于自身需求定制私有通信协议, 这些私有协议一般不对外开放协议细节。近年来, 有不少研究人员针对物联网私有协议展开了逆向分析工作。Eduard 等人^[21]逆向分析了一款市场占有率极高的胰岛素泵及其外围设备的通信协议, Liu 等人^[22]展示了一款基于嵌入式 Linux 系统的通信协议手工分析框架, 魏骁等人^[23]提出了一种基于静态二进制分析的工控协议解析方法。Zhou 等人^[24]和 Chen 等人^[61]也对 TP-Link、阿里智能、小米米家等主流物联网厂商的私有通信协议进行逆向分析, 证明私有通信协议在被逆向破解后会暴露设备的逻辑缺陷, 造成严重的后果。Yu 等人^[62]提出了一种利用车辆专业诊断工具自动逆向工程诊断协议的新框架, 该框架设计和开发了一种新的网络物理系统, 以自动触发和捕获诊断协议的消息, 并对其格式、语义含义、用于处理响应消息的专有公式进行逆向工程。

FirmSolo^[26]通过对物联网固件内核配置进行逆向分析, 实现对 Linux 固件内核模块的动态分析。作者利用固件映像中的内核模块中的信息(即导入导出符号和内核的 version magic)来逆向推导出内核配置文件, 并构建一个可以在模拟环境中加载模块的内核, 此功能允许下游分析将其范围扩展到以特权模式执行的代码中。iDEA^[63]是一款针对 Apple 操作系

统驱动程序错误的自动分析工具, 对于 Apple 独特的编程模型, 作者从 Apple 的驱动程序编程和管理模型中找到了可靠的信息源来恢复类, 并确定了 Apple 驱动程序与用户空间程序交互的独特范例。在 iDEA 的支持下, 作者针对两种常见类型的安全风险(竞态条件和越界读/写)和主要 Apple 操作系统的驱动进行了测试。Tsang 等人^[64]尝试定义和扩展一类以电源管理集成电路(PMIC)恶意配置为特征的固件漏洞, 并提出了一种在电源管理 IC 固件上构建和部署攻击的方法(FANDEMIC), 攻击过程中涉及对裸机物联网固件二进制文件进行逆向工程, 并识别与其 PMIC 交互的功能。

逆向分析技术本身不能发现漏洞, 它主要是帮助研究人员更好的理解二进制程序及相关文件本身的功能和实现细节, 从而可以配合代码审计或者其他程序分析技术来发现固件程序的脆弱点。

5.1.2 污点分析技术

污点分析技术^[65]是一种用于检测和跟踪敏感数据在程序中的流动的程序分析技术, 基本原理是给程序中的敏感数据(如用户输入、密钥等)打上标记, 然后跟踪这些数据在程序内部的传播路径, 从而找到潜在的漏洞点或不安全操作。在嵌入式固件中, 污点分析在实践中会面临一些挑战。首先, 由于间接的内存访问, 很难在多个函数之间找到准确的数据流。其次, 指针别名会阻止发现污点类型漏洞的数据路径。此外, 间接调用通常出现在固件中。这些障碍导致二进制固件中的漏洞检测性能较差。

污点分析常用于分析污点类漏洞, 污点类的漏洞是对用户的输入没有进行有效过滤, 使其进入敏感函数而导致的漏洞, 如命令注入、缓冲区溢出漏洞等。DTaint^[5]通过识别指针混叠、过程间数据流和数据结构布局的相似性来减少固件的影响, 然后自底向上生成过程内和过程间的数据流图, 并基于数据流图, 生成从 sinks 到 sources 的路径, 并检查路径上污点数据约束条件判断是否存在污点类型漏洞。Saint^[66]通过跟踪敏感 sources 的信息流, 如设备状态(灯开/未开)和用户信息(在家/外出)到外部接收器(如互联网连接和短信), 在物联网应用程序中发现敏感数据流。REDINI 等人^[12]针对物联网设备的 Web 服务提出边界二进制程序概念及查找方法并设计 KARONTE, 通过跟踪边界二进制文件与其他组件交互的数据流, 执行跨二进制污点分析。然而, 这种方法没有考虑到 Web 前端因素的影响, 导致识别的准确率相对较低。CHEN 等人^[15]在 KARONTE 的基础上充分考虑前端因素, 提出基于前后端共享关键

表 3 固件污点分析现有工作对比

Table 3 Firmware stain analysis comparison of existing work

文献	年份	研究对象	漏洞类型	优点	缺点	开源
Dtaint ^[5]	2018	IoT 固件	缓冲区溢出	支持指针分析和间接调用分析, 可以自底而上快速生成数据流图	只支持基于 Linux 的固件	否
SAINT ^[66]	2018	IoT App	敏感数据流, 数据泄露	测试物联网设备配套 app 是否存在数据泄露	仅测试 SmartThings 应用程序, 可能存在过污染	否
KARONTE ^[12]	2020	IoT 固件	污点类漏洞	提出 web 服务边界二进制程序概念及查找方法, 可以执行跨二进制污点分析	没有考虑到 Web 前端因素的影响, 识别的准确率相对较低	是
SaTC ^[15]	2021	IoT 固件	污点类漏洞, Web 服务漏洞	充分利用前后端关联关系来定位边界二进制文件和 Source 点	没有考虑边界二进制文件中的外部数据引入函数, 使用粗粒度污点分析, 导致误报率较高	是
SUTURE ^[68]	2021	操作系统内核	污点类漏洞	基于摘要的高阶污染流构建方法来有效地枚举交叉入口污染流	无系统和原则性的假警报过滤工作	是
FBIR ^[67]	2022	IoT 固件	污点类漏洞, Web 服务漏洞	在 SaTC 的基础上对外部数据引入函数进行识别	针对经过 NVRAM 中转的 Web 输入数据, 存在一定的局限性	是
IOTFUZZER ^[18]	2018	IoT 设备关联 App	内存损坏漏洞	从设备配套 app 获取通信协议的丰富信息指导真实设备的黑盒模糊测试	依赖于逆向修改配套 app, 需要大量的逆向知识, 受到配套应用程序内输入净化的阻碍	否
Zheng ^[14]	2019	IoT 固件	污点类漏洞	使用二进制静态分析为模糊测试生成有用的输入	受到仿真框架 Firmadyne 的限制, sink 定义为库函数, 未考虑供应商自定义库	否
SFuzz ^[52]	2022	RTOS	污点类漏洞	结合调用图分析与污点分析对固件程序进行切片, 然后使用指令级仿真引擎仿真探索空间的代码, 减轻仿真难度和分析复杂性	仅支持 RTOS 固件, 数据入口点由人为提供	是

字的漏洞分析方法 SaTC, 通过识别前后端同名字符串定位边界二进制程序, 同时将识别的字符串在程序中的引用点作为污点分析的 source 点, 进行粗粒度的污点分析来检测污点型漏洞。刘翎翔等人^[67]分析了 SaTC 误报率较高的原因, 并在其基础上进行修改以提高分析的准确性。SUTURE^[68]是一款能够发现操作系统内核中的高阶漏洞的静态分析工具, 发现了主流供应商(如谷歌、三星、华为)的多个安卓内核中的高阶污点漏洞。它采用基于摘要的高阶污点流构建方法来有效枚举跨条目污点流, 同时结合现有工具中未见的多种创新的分析精度增强功能, 从而实现高度精确的过程间流、上下文、字段、索引和可能路径敏感的静态污点分析。

污点分析对查找固件程序中污点类漏洞有很强的适用性, 并且大多数针对固件程序的污点分析采用的是静态污点分析, 不需要运行固件程序, 对固件程序分析带来极大的便利性。但由于静态分析缺乏程序运行时的细粒度信息, 再加上固件污点分析所面临的挑战, 单纯的采用污点分析技术往往会有很大的局限性, 通常表现为漏报误报率较高, 因此很多研究尝试采用污点分析结合其他动态分析技术(如模糊测试等)展开工作。FIoT^[37]基于 Angr^[69]开发, 通过遍历固件中的二进制程序控制流图(Control

Flow Graph, CFG)并执行反向代码切片, 构建从输入源到敏感函数调用的危险代码片段, 然后利用动态符号执行与模糊测试对该代码片段进行分析。Zheng 等人^[14]通过对固件二进制程序进行污点分析获取有用关键字从而指导 fuzz 种子生成。IoTFuzzer^[18]通过对被测设备配套的 APP 进行污点分析, 跟踪与应用程序协议相关领域的传播, 获取其与设备进行通信的协议的丰富信息, 用于指导在真实设备上的模糊测试。CHEN 等人^[52]提出了 SFuzz, 通过粗粒度的污点传播, 修剪独立于外部输入的路径, 在调用图中执行正向切片, 然后对切片进行模糊测试。

5.1.3 符号执行技术

符号执行技术旨在通过生成满足所需路径约束的输入来达到特定的程序状态。与动态分析中使用的具体值不同, 符号执行输入的是符号值, 因此与具体执行相比, 符号执行可以探索所有潜在的路径。符号执行可以解决嵌入式系统硬件兼容性挑战和外设建模困难的问题, 符号执行不是导出固件以在模拟器或虚拟机中执行, 而是将固件在符号执行引擎内象征性地执行, 同时将外设访问的结果视为符号数据。常用的符号执行引擎有 KLEE^[70]、Angr^[69]等。

FIE^[71]最早将符号执行引入嵌入式固件程序分析, 它建立在 KLEE^[70]符号执行引擎的基础上, 通过

状态修剪和内存弄脏来增强符号执行的覆盖率，可以识别与分析流行的 MSP430 系列微控制器运行的固件的内存安全相关的 Bug。Firmalice^[32]受 KLEE^[70]等启发提出的一个二进制框架，用于分析嵌入式设备固件是否存在后门，它建立一种新的身份绕过缺陷模型，认为从固件入口点派生出的任何路径到特权操作都应

该经过一个可靠的输入验证过程，然后通过数据依赖图对入口点到特权操作位置等程序进行切片，最后通过符号执行引擎找到入口点通往特权位置的可能成功路径。Gerbil^[34]在 Angr 框架上进行扩展，通过对固件程序进行加载信息提取、库函数识别和间接数据流恢复来探索物联网固件相关的权限分离漏洞。

表 4 固件符号执行现有工作对比
Table 4 Firmware symbols perform existing work comparisons

文献	年份	研究对象	漏洞类型	优点	缺点	是否开源
FIE ^[71]	2013	MSP430 固件	内存安全性	最早将符号执行引入固件程序分析，并通过状态修剪和内存清除来增加代码覆盖率	仅适用于微控制器的小型、简单的固件程序	是
Firmalice ^[32]	2015	设备固件	后门	提出一种新的身份绕过缺陷模型，分析嵌入式设备固件是否存在后门	无法对抗代码混淆，无法对抗基于数学的后门等涉及复杂约束求解的情况	否
Gerbil ^[34]	2019	IoT 固件	后门	首次对物联网固件相关的权限分离漏洞进行深入分析	无法处理代码签名冲突问题，符号执行前需要人工干预进行切片	是
FloT ^[37]	2019	单片机固件	内存损坏漏洞	在固件切片上进行符号执行和模糊测试，无须访问源码或完全仿真固件	只针对单片机固件，智能检测堆栈溢出漏洞，需要人工预处理固件	否
Inception ^[72]	2018	IoT 固件	内存破坏	突破了物理设备的计算资源限制，支持和真实设备外设交互和中断	会出现外围设备和代码的状态不一致问题，还需支持更多的二进制码和 C/C++	否
Ai ^[73]	2020	设备固件		采用动态测试生成方案，在物理设备上具体执行，在调试主机上进行符号执行	只能探索程序，发现漏洞能力较弱	是

混合执行技术将符号执行与具体执行相结合。在嵌入式系统上应用混合执行是一个具有挑战性的问题，第一，嵌入式设备计算资源的限制无法满足符号执行的需要；第二，纯符号执行中缺乏运行时信息不够准确，很难分析真实的程序；第三，现有的动态解决方案很难移植，只能针对特定的设备进行。为了解决这些问题，Inception^[72]从高级源代码、手工编写的程序集、二进制库和部分处理器硬件行为中生成并合并 LLVM 位代码，利用基于 KLEE^[70]改进的符号虚拟机运行生成的位代码，并使用不同的分析策略处理外围设备的内存和中断，分析过程中引入了 JTAG 调试器，确保内存访问和真实硬件之间的实时转发，保持事件同步以获得有意义的准确结果。Ai 等人^[73]将混合执行引入到嵌入式系统测试中，同时支持多种体系结构(如 x86、ARM 和 PPC)。为了实现他们的目标，将固件程序在物理设备上实现具体执行，可以获得准确的运行时信息，以及与硬件相关的操作，在调试主机上，将二进制代码转换 VEX^[74]中间语言(IR)，并对转换后的 IR 进行符号执行，解决了嵌入式设备计算资源不足的限制。

符号执行由于存在路径爆炸的问题，很难分析体量较大的真实程序，因此当前现有工作很少以符号执行为主体对固件程序进行脆弱性分析，符号执

行通常对其他分析技术起到辅助作用，以解决其在分析固件程序中局限。在 SFuzz^[52]中，对固件程序切片进行模糊测试之后，通过符号执行来评估漏洞，以消除切片后丢失上下文带来的误报。DTaint^[5]使用静态符号执行分析每个函数的立即数和符号变量，约束表达式以及数据类型。FloT^[37]在进行动态模糊测试之前执行符号执行，由符号执行来仿真固件的行为，进而再使用变异数据替换源数据，进行动态模糊测试。Avatar^[75]、Laelap^[76]、uEmu^[77]等仿真技术的实现也用到了符号执行技术，这在后面仿真技术小节详细介绍。

5.1.4 相似性检测

相似性分析旨在根据已有的二进制代码(如已揭露的漏洞等)在代码库中搜索出词法、语法、语义等相似的二进制代码，从而探测出潜在的漏洞，维护程序的安全。基于代码相似性的分析方法可以用来检测固件中的已知缺陷，且可以跨越不同平台进行检测。在本节中，我们将回顾识别固件中漏洞的代码相似性检测方法。

传统的相似性分析方法。discovRE^[78]基于相应控制流图(CFG)来计算函数之间的相似性，实现了跨架构固件中的脆弱函数检测，并通过一种基于数字特征的高效预滤波器来快速识别缩小候选函数，使得其能在

大型代码库高效地搜索相似函数。Genius^[79]受discovRE^[78]启发,认为基于CFG漏洞搜索技术的瓶颈不在于图匹配算法,而在于搜索方案,并提出将CFG转化为高级数字向量,然后使用哈希技术实现实时搜索的漏洞搜索方案,解决了跨平台漏洞搜索技术的可扩展性挑战。BinArm^[80]针对基于ARM架构的智能电网IED(智能电子设备)固件,提出了一个多级检测引擎来检测固件中脆弱函数,以最大限度降低功能匹配的计算成本,FirmUp^[81]通过考虑函数之间的关系来识别

固件中的脆弱函数,通过提出的来回博弈算法对上下文相邻过程的进行部分匹配,提高了二进制代码搜索技术的准确性。XMATCH^[82]通过构建条件公式作为二进制代码跨平台搜索的语义特征,一个条件公式由一个If子句和一个术语组成,每个子句都是一个符号公式,描述在什么条件下(在If子句中说明)将发生给定的动作(在术语中),其中的跨平台指令使用McSema^[83]提升到IR,条件公式通过提取上下文语义(包括触发条件和数据流关系)以提高搜索精度。

表 5 固件代码相似性检测现有工作对比
Table 5 Firmware code similarity detection compares existing jobs

文献	年份	粒度	特征	工作	跨架构	是否开源
discovRE ^[78]	2016	函数	结构特征 数字特征	实现高效预滤波器	是	是
Genius ^[79]	2016	基本块	CFG	使用哈希技术实现高效搜索方案	是	否
BinArm ^[80]	2018	函数	函数形状、分支、邻域	多级检测引擎和模糊哈希,降低函数匹配成本	否	否
FirmUp ^[81]	2018	基本块	DFG、上下文	使用周围可执行文件中的信息进行更精确匹配	是	否
XMATCH ^[82]	2017	函数	上下文语义(包括触发条件和 DFG)	提取条件公式作为搜索的语义特征	是	否
Gemini ^[84]	2017	基本块	ACFG	基于 DNN 生成嵌入向量,测量嵌入距离	是	否
VulSeeker ^[85]	2018	函数	LSFG	定制语义感知 DNN 模型	是	是
αdiff ^[86]	2018	函数	原始字节、CFG	采用三个语义特征来解决跨版本相似性检测的挑战	是	否
Trex ^[88]	2020	函数	汇编指令	开发了一种新的神经架构,即层次转换器,它可以在预训练阶段从微痕迹中学习执行语义	是	是
Codee ^[87]	2021	函数	汇编指令、CFG	一个无监督的张量嵌入方案	是	是
XBA ^[89]	2022	基本块	二进制反汇编图(BDG)	使用表示学习方法自动学习不同平台的语义对齐,并将不同平台的二进制代码转换为相同的嵌入向量表示	是	是
FirmSec ^[90]	2022	基本块	语法、控制流	利用语法特征和控制流图特征来检测固件中版本级别的第三方组件(TPC),然后识别相应的漏洞	是	是

基于深度学习的代码相似性分析方法。传统的相似性分析方法主要依赖于近似的图形匹配算法,这些算法不可避免地很慢,有时不准确,并且很难适应新的任务。近年来,深度学习已应用于许多应用领域,包括二进制分析,它可以将二进制分析任务(例如,为二进制函数生成嵌入向量)表示为可以端到端训练的神经网络的参数,并依赖于尽可能少的领域知识。Gemini^[84]引入了深度学习,通过修改神经网络,使其可以将从固件中提取的函数控制流图转化为数字特征向量-嵌入向量(Embedding Vectors),然后通过测量两个函数的嵌入向量的之间的距离来有效的进行相似性检测。VulSeeker^[85]受到Genius^[79]的启发,提取了将CFG和DFG相结合的标签语义流图(LSFG),然后将基本块的数字向量馈送到定制的语义感知DNN模型来生成整个二进制函数的嵌入向量,最后基于余弦距离测量两个函数的相似性。

αdiff^[86]使用深度神经网络直接处理每个函数的原始字节,首先提取函数内特征,然后进一步分析跨版本二进制中相对稳定的函数调用图,并提取函数间和模块间特征,然后基于这三个特征计算距离,实现二进制代码相似性检测。Codee^[87]是一种无监督的张量嵌入方案,它提出了一种基于网络表示学习模型的高效基本块嵌入生成算法,使用基于NLP的神经网络来生成语义感知令牌嵌入,然后使用函数中的所有基本块嵌入来获得可变量长度函数的特征向量,最后使用局部敏感哈希方法来查找存储库中的前k个相似匹配函数。Trex^[88]提出可以通过分析函数的执行语义来检测相似代码,并设计了一个新的神经结构-分层转换器,它可以在预训练阶段自动从函数的微跟踪(一种受限的动态跟踪)学习执行语义,然后将学到的知识用于匹配语义相似的函数。XBA^[89]通过一种跨平台的二进制共同表示,将不同平台的二

进制代码转化为相同的嵌入向量表示, 来实现跨平台的二进制代码相似性分析。具体来说, 通过将二进制文件抽象为二进制反汇编图(BGD), 该图编码了控制流和其他在反汇编二进制文件中发现的不同实体的丰富上下文信息, 包括基本块、调用的外部函数和引用的字符串字头, 然后将二进制代码表示法学习表述为一个图排列问题, 即寻找从两个为不同平台编译的二进制文件中提取的 BDG 之间的节点对应, 最后 XBA 以半监督的方式学习两个 BDG 之间的语义排列。FirmSec^[90]利用语法特征和控制流图特征来检测固件中版本级别的第三方组件(TPC), 然后识别相应的漏洞, 首次对 TPC 的使用以及固件中的相应漏洞进行大规模分析, 发现固件中使用的 TPC 平均落后了 5 年, 作者在研究过程中构建了迄今为

止最大的数据集, 并将其开源。
相似性检测对检测第三方开源代码的漏洞能提供良好的支持, 但其只能检测已经存在的漏洞, 并且需要高可信度的公开测试集提供统一的测试数据, 因此对个性化定制的应用层代码适用性较差。

5.2 动态仿真环境构建

由于固件运行环境受限, 并且物联网设备通常采用 ARM、MIPS 等精简指令集体系结构, 无法直接在通用计算机上运行与调试设备中的程序。此外, 针对测试物联网真机可能需要频繁重启, 以确保每个测试用例都有一个清晰的状态, 这是代价昂贵的。固件动态仿真是用来实现在各种体系结构的计算平台上仿真执行异构或不同系统平台下软件的技术, 是目前很多针对设备固件的动态模糊测试研究所依赖的基础技术^[91]。

表 6 动态仿真环境构建相关工作概述

Table 6 Overview of related work on building dynamic simulation environment

类型	描述	优点	缺点	相关工作
全系统仿真	将系统固件或其部分软件转移到一个设计好的虚拟环境来运行	实现固件的更好可见性	理想状态, 可能因设备而异遇到各种问题	QEMU ^[92] 、Unicorn ^[93] 、Qiling ^[94] 、EMUX ^[95] 、PANDA ^[96]
混合仿真	提取并修改固件使其能在模拟器内执行, 同时视将 IO 访问转发给外设, 收集反馈注入模拟器	解决外设缺乏文档和建模复杂的问题	可扩展性受到外设频繁交互以及多设备同时测试的影响	PROSPECT ^[97] 、Avatar ^[75] 、Avatar2 ^[98] 、SURROGATES ^[99] 、Kammerstetter ^[100] 、PRETENDER ^[101] 、Frankenstein ^[102]
托管	通过对外设进行建模或者功能修改替换等方法, 可以在设备外重新托管固件	解决混合仿真可扩展性的问题	可能与固件真实执行环境产生差异	Firmadyne ^[36] 、FirmAFL ^[103] 、FirmAE ^[104] 、ECMO ^[105] 、HALucinator ^[106] 、Para-rehosting ^[108] 、SAFIREFUZZ ^[109] 、P2IM ^[110] 、DICE ^[111] 、Fuzzware ^[112] 、Laelaps ^[76] 、Jetset ^[113] 、μEmu ^[77]

1. 全系统仿真

全系统仿真是指将系统固件从它本身硬件移出或转移它软件的一部分到一个设计好的虚拟环境来运行固件。从理论上讲, 这种方法实现了对固件执行的更好可见性。不幸的是, 没有任何现有的工作能够精确地支持未知的设备, 因此, 在目前的状态下, 这种方法只存在于理想主义的环境中。

QEMU^[92]是目前广泛使用的支持全系统仿真开源工具, 基于动态二进制翻译以实现对不同架构的程序的仿真。QEMU 支持用户仿真和系统仿真两种模式。用户模式下, QEMU 会将系统调用直接传递给宿主机系统, 当遇到宿主机不支持的系统调用或不具备的硬件设备, 程序就会发生崩溃。系统模式能够模拟一个包括内核和外部设备在内的完整计算机系统, 需要软件模拟内存管理单元对所有内存访问进行地址转换, 并模拟程序产生的系统调用和系统运行需要的硬件设备, 因此仿真执行异构程序的兼容性好, 但产生的开销较大。目前流行的全系统仿真框

架底层大多是基于 QEMU 实现, 如 Unicorn^[93]、Qiling^[94]、EMUX^[95]、PANDA^[96]等。

2. 混合仿真

在虚拟和可扩展的环境中分析嵌入式系统是一项具有挑战性的任务, 因为固件运行需要不时访问相关外设, 但这些外设缺乏文档并且建模复杂, 很难针对外设进行模拟。为了解决这些问题, 提出了部分仿真, 即提取并修改固件使其能在模拟器内执行, 同时视需要将 IO 访问转发给外设, 同时在设备上收集反馈注入模拟器。

PROSPECT^[97]实现了一个代理, 将 QEMU 虚拟机内部运行的固件发起的每个外设访问隧道传输到设备, 实现虚拟化执行环境与被测系统实际外围硬件组件的完全透明连接。Avatar^[75]是一款与 PROSPECT^[97]类似但可扩展性更强的动态分析框架, 用于将实际硬件与基于通用处理器的模拟器集成, 实现了对基于 ARM 的嵌入式系统上进行评估。除了对嵌入式系统进行部分仿真以及与其外接设备协调执行之外,

Avatar 的后续工作 Avatar2^[98]实现了协调与更多的物理设备、调试器及二进制分析框架的交互, 如 Angr^[69]、PANDA^[96]等, 并支持 x86、x86-64 和 ARM 硬件架构。由于仿真过程缓慢, 一般的仿真解决方案并不适合需要近实时响应的嵌入式设备。针对任意基于 ARM 的嵌入式系统, SURROGATES^[99]实现了一个能够近实时仿真和检测嵌入式系统的系统, 利用定制的 FPGA 低延迟硬件将嵌入式系统与主机的 PCI Express 总线连接起来, 允许模拟器完全访问嵌入式系统的外围设备。

Kammerstetter 等人^[100]提出通过足够的训练来学习被访问的外设行为, 将用于外围设备通信的缓存与运行时程序状态近似结合起来, 以避免通过物理设备进行重复测试, 但只适用于小型固件并且会遭受与符号执行类似的状态爆炸影响。PRETENDER^[101]通过反复运行固件, 并记录与相关固件的真实交互, 然后利用机器学习和模式识别技术为每个可用外设建立模型。Frankenstein^[102]实现了一款针对蓝牙固件的仿真框架, 通捕捉物理设备快照, 然后在虚拟环境中仿真, 从而提供模糊测试环境。

3. 托管(Rehosting)

部分仿真仅提供了一种对目标系统及其所有外设进行完全仿真的假象, 可扩展性受到与连接的外设频繁交互的影响, 跨多个设备同时测试也可能出现问题。托管是指通过对外设进行建模或者功能修改替换等方法, 以至可以在设备之外重新托管固件, 而不需要访问任何外设。在实践中, 基于仿真的 Rehosting 技术经常修改固件以简化过程, 并在此过程中从虚拟环境删除一些固件。托管只适用于目标系统外设能够成功仿真的情况, 如基于 Linux 的嵌入式系统, 或者目标系统硬件的文档完整可用等。

Chen 等人^[36]针对基于 Linux 的嵌入式固件系统, 引入了自动化仿真及动态分析框架 Firmadyne, 它通过 QEMU 运行一个通用的 Linux 插件内核来实现自动分析数千固件二进制文件的可扩展性。Firmadyne^[36]可以自动化解析固件提取文件系统、判断系统架构、修复常见的设备文件、获取网络配置等, 完全实现一键式仿真。但由于真实固件执行环境和仿真固件执行环境中的差异, Firmadyne 的成功率较低。FirmAFL^[103]基于 Firmadyne 模拟方式, 使用一种称为增强过程仿真的新技术来解决系统模式仿真引起的性能瓶颈。FirmAE^[104]在 Firmadyne 的基础上提出仲裁模拟, 并将几种启发式方法系统化为仲裁技术来提高仿真成功率。ECMO^[105]提出了一种外周移植的技术, 其主要思想是将指定外设的设备驱动程序移

植到 Linux 内核二进制文件中。通过这样做, 它可以用支持的外设替换内核中当前 QEMU 不支持的外设, 从而使 Linux 内核可重新托管。

HALucinator^[106]通过提供固件库函数的通用实现来替换固件硬件抽象层(HAL), 以实现硬件与固件的解耦并可以成功在 QEMU 中运行, 最终还结合了 AFL 实现了对固件的模糊测试。Para-rehosting^[108]系统的探讨了 MCU 软件的可移植性问题, 通过对 MCU 的通用功能进行建模, 使用可移植操作系统接口抽象并实现了一个便携式 MCU。对于特定于外围设备的逻辑, 则像 HALucinator^[106]一样, 基于 HAL 进行外围功能替换。SAFIREFUZZ^[109]是 ARM Cortex-M 固件的吞吐量优化的重新托管和模糊测试框架, 它主要通过在与目标设备共享指令集系列的高性能系统上将嵌入式固件作为 Linux 用户空间进程运行。

P²IM^[110]根据处理器和外设之间接口的设计文档, 抽象了各种外围设备, 并基于自动生成的寄存器模型实时处理固件 I/O 操作。DICE^[111]在 P²IM 基础上实现了一种模拟并生成 DMA 输入的方法, 通过识别和模拟抽象的 DMA 输入通道, 来取代模拟种类繁多的外设和控制器。Fuzzware^[112]提出了一种细粒度访问建模方法, 使用模糊测试尝试对硬件的 MMIO 行为进行精确建模。Laelaps^[76]利用符号执行辅助外设仿真来推测固件的预期行为, 并生成相应的输入以动态地引导具体模拟执行过程。Jetset^[113]改进了 P²IM 的 MMIO 寄存器模型, 利用符号执行技术来引导 MIMO 寄存器的数据输入, 同时基于上下文敏感度选择条件转移路径, 提高 QEMU 的保真度。μEmu^[77]同样利用符号执行技术对 P²IM 的寄存器模型进行了改进, 在符号执行期间, 它推断出响应未知外设访问的规则并存储在知识库中, 在动态固件分析期间参考该知识库。

5.3 动态模糊测试技术

模糊测试技术最早由 Miller^[107]于 20 世纪 90 年代针对 UNIX 系统进行安全测试时提出。其关键思想是向目标程序提供大量可能触发程序运行异常的测试用例, 并监控程序崩溃、错误和内存泄露等异常。针对固件的模糊测试包含针对真实设备的模糊测试和基于仿真的模糊测试, 前者主要是利用设备采用的通信协议进行模糊测试, 后者则依赖于对仿真框架的高度定制来实现。

1. 针对真实设备的模糊测试

物联网设备通过各种网络协议来进行数据传输, 针对真实设备的模糊测试主要围绕设备使用的通信

协议展开, 以黑盒测试为主。Sulley^[19]与 Boofuzz^[20]都是优秀的协议模糊测试框架, Boofuzz 是 Sulley 的分支和后续版本, 修复了一些错误, 扩展了新的特性。IoTFuzzer^[18]是一种没有协议规范而进行引导式的基于协议的模糊测试框架, 不需要访问固件, 通过对固件配套应用程序进行分析, 推导出用于通信的丰富协议信息, 通过识别和重用程序特定逻辑来变异测试用例,

以发现内存损坏漏洞。针对有状态的网络协议, Snipuzz^[13]实现了一种轻量级的高效黑盒模糊测试方法。Snipuzz 作为一个与设备通信的客户端运行, 并根据响应推断出变异的消息片段。这种基于消息片段的变异策略大大缩小了搜索空间, 以更改探测消息。此外, Snipuzz 利用相似度得分以及层次聚类策略, 减少了响应因随机性和固件内部机制导致的错误分类。

表 7 动态模糊测试相关工作概述
Table 7 Overview of dynamic fuzzing related work

测试环境	优点	缺点	相关工作
真实设备	通过协议传输数据, 测试方便	大多是黑盒测试, 功能覆盖范围有限, 代价昂贵	Sulley ^[19] 、Boofuzz ^[20] 、IoTFuzzer ^[18] 、HUBFUZZER ^[114] 、TEEzz ^[115] 、μAFL ^[116]
仿真环境	能够进行覆盖率反馈的灰盒模糊测试	需要对仿真框架高度定制, 仿真环境构建困难	FirmAFL ^[103] 、FirmFuzz ^[117] 、Firmwire ^[118] 、Fuzzware ^[112] 、IoTHunter ^[17] 、Periscope ^[27] 、TriforceAFL ^[119] 、Frankenstein ^[102] 、P ² IM ^[110] 、DICE ^[111] 、uEmu ^[77] 、Laelaps ^[76] 、Para-rehosting ^[108] 、HALucinator ^[106] 、RVFuzzer ^[120] 、PGFUZZ ^[121] 、PatchVerif ^[122] 、MORPHUZZ ^[123] 、WMIFuzzer ^[16]

HUBFUZZER^[114]针对通过集线器协调物联网设备和云后端通信的场景, 提出一种基于集线器来模糊物联网设备的方法, 该方法无须对任何配套应用程序进行逆向分析, 它利用集线器和物联网设备之间交换的消息来自动发现所有功能, 然后启动面向功能的消息语义引导模糊测试。TEEzz^[115]是第一个能够在商用智能手机上对可信应用(TA)进行 fuzzing 的可信执行环境(TEE)感知模糊框架, 由于模糊器只能观察与 TA 的交互, 不能控制 TA 的代码或数据, TEEzz 通过 TA API 的交互来推断其字段类型和消息依赖性, 设计状态和类型感知的模糊修改器, 以完成对 TA 的黑盒模糊测试。μAFL^[116]使用硬件在环的方法, 其利用现有的嵌入式系统开发中的调试工具, 构建了一个与 AFL 兼容的模糊框架, 实现对微控制器固件的覆盖率反馈的灰盒模糊测试, 其通过 ARM ETM 硬件调试功能收集指令跟踪数据, 在保留与原始 AFL 算法相同的路径敏感性的基础上, 直接处理原始 ETM 数据来表示代码覆盖率。

由于真实设备提供的调试接口有限, 因此基于真实设备的模糊测试多是黑盒模糊测试, 现有的黑盒模糊测试工作要么依赖于逆向修改配套 app 进行测试, 要么依赖于厂商披露的 API 测试程序或者固件使用协议的相关文档, 测试盲目性较大, 并且对测试设备的功能覆盖范围有限。

2. 基于仿真的模糊测试

由于针对真实设备的进行模糊测试比较受限, 并且代价昂贵。大量研究围绕仿真技术展开。Firm-AFL^[103]基于 Firmadyne 实现了对基于 Linux 的物联网固件的高性能灰盒模糊测试, 该系统是首个

面向固件的灰盒模糊测试解决方案, 并且成功提升了固件模拟执行的效率。FirmFuzz^[117]通过静态分析收集信息来指导固件模拟和 Fuzzing 生成, 通过监控固件运行时生成的日志来检测漏洞。Firmwire^[118]是一个专为三星和联发科基带固件设计的分析平台, 它提供了专用的 API, 可用于对基带固件进行模糊测试、模拟和调试。Fuzzware^[112]是一个专门用于对原始单片机固件进行模糊测试的系统。该系统引入了一种细粒度的访问建模方法, 利用准确的内存映射 I/O(MMIO)建模, 能够捕捉固件逻辑中的所有路径, 帮助 Fuzzing 变异集中在重要的输入上, 从而增加了代码覆盖率和 Fuzzing 效率。IoTHunter^[17]使用多阶段消息生成技术, 根据给定的状态序列依次切换协议状态, 进行基于反馈的状态探索, 并通过多阶段消息生成进行覆盖率引导的灰盒模糊测试。Periscope^[27]提出了 Fuzzing 框架 PeriFuzz, 这是一个专门用于检测硬件-操作系统边界上出现的驱动程序漏洞的漏洞发现工具。TriforceAFL^[119]在 QEMU 全系统模拟的基础上可以对操作系统内核进行模糊测试。Frankenstein^[102]、P²IM^[110]、DICE^[111]、uEmu^[77]、Laelaps^[76]、Para-rehosting^[108]、HALucinator^[106]等以仿真为主的研究也都集成了模糊测试技术, 以实现对固件的漏洞挖掘。

随着仿真技术的发展, 涌现了一批对新兴场景的模糊测试工作。RVFuzzer^[120]通过控制引导的输入突变来查找机器人车辆(RV)控制程序中的输入验证错误, 使用各种 RV 控制模型的通用理论模型来提供语义指导。其具体包含一个控制不稳定检测器, 通过控制模型观察 RV 的物理操作来检测控制程序的不

当行为, 将其作为反馈来引导输入生成, 以更有效地查找输入验证错误。PGFUZZ^[121]是一种策略引导的模糊测试框架, 用于验证 RV 是否遵守涵盖用户命令、配置参数和物理状态的已识别安全和功能策略。它通过具有时间约束的时态逻辑公式来表达所需的策略, 作为对分析系统进行模糊测试的指导。PatchVerif^[122]在 PGFuzz 基础上实现了一款自动化补丁分析框架, 用于评估修补的补丁是否在 RV 控制软件中引入了错误。PatchVerif 使用专用的输入突变算法来生成 RV 输入, 以最大化原始代码和修补代码之间的行为差异。使用收集到的有关补丁引入的行为修改的信息, PatchVerif 使用支持向量机(SVM)来推断补丁是有故障还是正确。MORPHUZZ^[123]利用有关虚拟机管理程序设计的见解与覆盖引导的模糊测试相结合来查找虚拟设备实现中的错误, 是第一种自动引出现代云中真实虚拟设备的复杂 I/O 行为的方法。WMIFuzzer^[16]将模糊测试应用于商用现货(COTS)物联网设备的 Web 固件界面, 通过尽可能捕获 GUI 消息作为最初的种子, 并通过构建抽象语法树进行突变, 可以在不需要预定义数据模型的情况下测试正在运行的物联网固件。

6 未来展望

随着物联网技术的不断发展, 固件程序体量和复杂度的也在不断攀升, 单纯的污点分析、符号执行、模糊测试等程序分析技术弊端不断显现, 污点分析因内存间接访问等问题导致准确性较差, 符号执行因路径爆炸的问题而无法分析大体量真实程序, 模糊测试要么针对真实设备盲目性较大, 要么依赖于高度定制仿真框架。幸运的是, 学术界和工业界对固件安全的重视程度在不断加强, 研究也在不断的增多, 并在不同分析技术上都取得了一定进展, 特别是仿真技术的发展为动态分析技术, 尤其是模糊测试技术的发展提供了良好的前提, 污点分析等静态分析技术开始与动态分析技术结合使用以提高分析精度。

物联网设备数量的持续爆发式增长和应用场景的不断扩大, 针对物联网设备的攻击面也在不断增大, 固件面临着 web 管理端攻击、协议解析攻击、操作系统的脆弱性、开源组件的安全性及未知的后门等等安全风险。物联网场景架构之下, 暴露出来的移动端 APP、PC 客户端、WEB 管理端、云服务、边缘设备、Iothub 及联网设备均会遭受到攻击。尤其是新兴技术的引入, 如 ai 模型与 5G 通信技术等, 及新兴场景如车联网等的发展, 更是为未来固件安全领

域提供了丰富的研究方向。

1. 基于固件程序局部仿真的动态程序分析技术

随着仿真技术的发展, 业界涌现出了一批优秀的仿真框架, 其中 cpu 指令级仿真框架 Unicorn^[93], 和基于 Unicorn 的高级二进制分析、仿真框架的 Qiling^[94]的出现, 使得仿真二进制代码片段得到简化, 无需对仿真框架进行深度定制。因此, 未来可以着手于固件程序分析本身, 从大体量的真实固件程序甚至内核程序中剪切出更小更关键的控制流范围, 通过仿真代码片段, 使用动态模糊测试技术或其他动态数据流追踪技术, 可以更加快速的发现潜在漏洞。

2. 针对固件程序的多目标定向模糊测试技术

随着仿真技术的发展, 针对固件的灰盒模糊测试技术也在逐步发展, 大多数灰盒模糊测试都是基于覆盖率引导的, 目的是在有限时间尽可能多地覆盖分支路径。然而固件程序代码量日渐增加, 盲目地扩展代码覆盖率范围来检测程序漏洞可能会浪费大量资源。固件程序中存在大量的污点型漏洞, 通常能确定目标危险函数的位置, 并且通常存在多个目标位置, 因此可以利用现有的定向模糊测试技术, 通过设计合适的测试用例生成算法来实现针对固件程序的多目标定向模糊测试技术。

3. 针对物联网固件内核层面的安全检测技术研究

为了满足嵌入式设备有限的资源和空间, 固件所采用的内核大多基于现有的操作系统进行裁剪, 在修剪内核的过程中可能会错误地删除必要的安全操作。并且嵌入式设备通常选用内核版本较低, 而且更新缓慢, 已知的安全漏洞会随着设备的使用长期存在。目前针对内核模块的安全研究仍然较少, 固件中的内核模块中存在大量的定制化网络服务、驱动、加密程序等, 但静态分析内核模块难度较大, 随着逆向分析及仿真技术的发展, 使脱离真实设备对内核模块的动态分析成为可能。

4. 物联网设备云服务接口的安全检测技术

随着云计算和 5G 通信技术的发展, 越来越多的物联网设备通过云服务平台对数据进行存储、处理和分析, 甚至用户和物联网设备通过云平台作为桥梁进行交互。随着云平台在物联网场景中的应用越来越多, 大量的云服务 API 暴露在公网之下, 带来了巨大的攻击面。一旦云平台和物联网设备的交互出现问题, 会导致设备向不可预知的方面运行, 给用户带来隐私或者人身安全上的隐患。因此, 需要设计合适测试方法, 针对物联网设备的云平台 API 的安全性进行测试, 如何设计测试方法来检测用户-云平

台-设备三者通信过程中的存在的问题,也是未来需要解决的一个问题。

5. 物联网固件中 AI 模型组件的安全性研究

实现生活智能化是物联网技术发展的主要目标之一,物联网设备想要实现智能化,必然少不了人工智能技术的引入。智能化设备依赖人工智能技术对感知数据进行分析处理,根据不断变化的应用场景实时进行决策。如车联网中的自动驾驶系统,通过传感器对周围环境进行感知,从而决定对车辆当前行为的控制。固件自动化决策离不开人工智能模型的引入,因此,评估 AI 模型的安全性,也是未来需要重点关注的问题。

6. 面向新兴场景的安全问题研究

随着物联网技术的飞速发展,新的应用场景也在不断涌入,其中具代表性的有车联网、星网等。针对新兴场景的固件安全研究分析仍有较大空间,如其中涉及的协议的逆向分析,固件仿真研究,针对固件系统漏洞和协议漏洞挖掘等等。

7 结语

本文围绕着嵌入式设备安全分析技术发展现状展开调查研究,首先对嵌入式固件的发展状况进行介绍、然后对固件存在的安全风险和固件分析的一般流程进行了总结,接着对嵌入式固件分析用到的技术进行了全面的分析比较与评估,最后给出了未来需要关注的一些研究方向。

参考文献

- [1] GSMA. <https://www.gsma.com/>. Jun. 2020.
- [2] aliyun. <https://static-aliyun-doc.oss-cn-hangzhou.aliyuncs.com/file-manage-files/zh-CN/20220318/ynbf/阿里云物联网平台技术白皮书 V1.0.pdf> Mar. 2022.
- [3] China National Vulnerability Database. <https://www.cnvd.org.cn/>. 2022.
- [4] Yu Y C, Chen Z N, Gan S T, et al. Research on the Technologies of Security Analysis Technologies on the Embedded Device Firmware[J]. *Chinese Journal of Computers*, 2021, 44(5): 859-881.
(于颖超, 陈左宁, 甘水滔, 等. 嵌入式设备固件安全分析技术研究[J]. *计算机学报*, 2021, 44(5): 859-881.)
- [5] Cheng K, Li Q, Wang L, et al. DTaint: Detecting the Taint-Style Vulnerability in Embedded Device Firmware[C]. *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2018: 430-441.
- [6] Zhang C, Situ L Y, Wang L Z. Research Progress on Security Defect Detection of Io T Firmware[J]. *Journal of Cyber Security*, 2021, 6(3): 141-158.
(张弛, 司徒凌云, 王林章. 物联网固件安全缺陷检测研究进展[J]. *信息安全学报*, 2021, 6(3): 141-158.)
- [7] Srihith I D, Donald A D, Srinivas T A S, et al. Firmware Attacks: The Silent Threat to Your IoT Connected Devices[J]. *International Journal of Advanced Research in Science, Communication and Technology*, 2023: 145-154.
- [8] Zhou C Y, Zhang H X, Li X H, et al. Analysis on Firmware Protection and Vulnerability Analysis Technology of Internet of Things Equipment[J]. *Secrecy Science and Technology*, 2022(2): 45-50.
(周诚远, 张慧翔, 李晓辉, 等. 浅析物联网设备固件保护与漏洞分析技术[J]. *保密科学技术*, 2022(2): 45-50.)
- [9] Zheng Y W, Wen H, Cheng K, et al. Io T Security Threat and Security Model[J]. *Journal of Cyber Security*, 2023, 8(5): 81-95.
(郑尧文, 文辉, 程凯, 等. 物联网安全威胁与安全模型[J]. *信息安全学报*, 2023, 8(5): 81-95.)
- [10] Muench M, Stijohann J, Kargl F, et al. What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices[C]. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.
- [11] Costin A, Zarras A, Francillon A. Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces[C]. *The 11th ACM on Asia Conference on Computer and Communications Security*, 2016: 437-448.
- [12] Redini N, Machiry A, Wang R Y, et al. Karonte: Detecting Insecure Multi-Binary Interactions in Embedded Firmware[C]. *2020 IEEE Symposium on Security and Privacy*, 2020: 1544-1561.
- [13] Feng X T, Sun R X, Zhu X G, et al. Snipuzz: Black-Box Fuzzing of IoT Firmware via Message Snippet Inference[C]. *The 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021: 337-350.
- [14] Zheng Y W, Song Z W, Sun Y Y, et al. An Efficient Greybox Fuzzing Scheme for Linux-Based IoT Programs through Binary Static Analysis[C]. *2019 IEEE 38th International Performance Computing and Communications Conference*, 2019: 1-8.
- [15] Chen L, Wang Y, Cai Q, et al. Sharing more and checking less: Leveraging common input keywords to detect bugs in embedded systems[C]. *30th USENIX Security Symposium*, 2021: 303-319.
- [16] Wang D, Zhang X S, Chen T, et al. Discovering Vulnerabilities in COTS IoT Devices through Blackbox Fuzzing Web Management Interface[J]. *Security and Communication Networks*, 2019, 2019(1): 5076324.
- [17] Yu B, Wang P F, Yue T, et al. Poster: Fuzzing IoT Firmware via Multi-Stage Message Generation[C]. *The 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 2525-2527.
- [18] Chen J Y, Diao W R, Zhao Q C, et al. IoTfuzzer: Discovering Memory Corruptions in IoT through App-Based Fuzzing[C]. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.
- [19] Sulley. <https://github.com/OpenRCE/sulley>. 2015.
- [20] Boofuzz. <https://github.com/jtpereyda/boofuzz>. 2018.
- [21] Marin E, Singelée D, Yang B H, et al. On the Feasibility of Cryptography for a Wireless Insulin Pump System[C]. *The Sixth ACM Conference on Data and Application Security and Privacy*, 2016: 113-120.
- [22] Liu K Z, Yang M, Ling Z, et al. On Manually Reverse Engineering

- Communication Protocols of Linux-Based IoT Systems[J]. *IEEE Internet of Things Journal*, 2021, 8(8): 6815-6827.
- [23] Wei X, Liu R H, Xu F K. Reverse Analysis of Industrial Control Protocol Based on Static Binary Analysis[J]. *Application of Electronic Technique*, 2018, 44(3): 126-130.
(魏晓, 刘仁辉, 许凤凯. 基于静态二进制分析的工控协议逆向解析[J]. *电子技术应用*, 2018, 44(3): 126-130.)
- [24] Zhou W, Jia Y, Yao Y, et al. Discovering and understanding the security hazards in the interactions between {IoT} devices, mobile apps, and clouds on smart home platforms[C]. *28th USENIX security symposium*, 2019: 1133-1150.
- [25] Fu L R, Ji S L, Lu K J, et al. CPscan: Detecting Bugs Caused by Code Pruning in IoT Kernels[C]. *The 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021: 794-810.
- [26] Angelakopoulos I, Stringhini G, Egele M. FirmSolo: Enabling dynamic analysis of binary Linux-based IoT kernel modules[C]. *32nd USENIX Security Symposium*, 2023: 5021-5028.
- [27] Song D, Hetzelt F, Das D, et al. PeriScope: An Effective Probing and Fuzzing Framework for the Hardware-OS Boundary[C]. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019: 1-15.
- [28] Thomas S L, Francillon A. Backdoors: Definition, Deniability and Detection[C]. *Research in Attacks, Intrusions, and Defenses*, 2018: 92-113.
- [29] Thomas S L. Backdoor Detection Systems for Embedded Devices[D]. Birmingham: University of Birmingham, 2018.
- [30] Thomas S L, Garcia F D, Chothia T. HumIDIFy: A Tool for Hidden Functionality Detection in Firmware[C]. *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2017: 279-300.
- [31] Costin A, Zaddach J, Francillon A, et al. A Large-Scale Analysis of the Security of Embedded Firmwares[J]. *Proceedings of the 23rd USENIX Security Symposium*, 2014: 95-110.
- [32] Shoshitaishvili Y, Wang R Y, Hauser C, et al. Fimalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware[C]. *Proceedings 2015 Network and Distributed System Security Symposium*, 2015: 1: 1.1-8.1.
- [33] Hu C J, Xue Y B, Zhao L, et al. Backdoor Detection in Embedded System Firmware without File System[J]. *Journal on Communications*, 2013, 34(8): 140-145.
(忽朝俭, 薛一波, 赵粮, 等. 无文件系统嵌入式固件后门检测[J]. *通信学报*, 2013, 34(8): 140-145.)
- [34] Yao Y, Zhou W, Jia Y, et al. Identifying Privilege Separation Vulnerabilities in IoT Firmware with Symbolic Execution[C]. *Computer Security*, 2019: 638-657.
- [35] Thomas S L, Chothia T, Garcia F D. Stringer: Measuring the Importance of Static Data Comparisons to Detect Backdoors and Undocumented Functionality[C]. *Computer Security*, 2017: 513-531.
- [36] Chen D D, Egele M, Woo M, et al. Towards Automated Dynamic Analysis for Linux-Based Embedded Firmware[C]. *Proceedings 2016 Network and Distributed System Security Symposium*, 2016: 1: 1.1-8.1.
- [37] Zhu L P, Fu X T, Yao Y, et al. FloT: Detecting the Memory Corruption in Lightweight IoT Device Firmware[C]. *2019 18th IEEE International Conference on Trust, Security and Privacy In Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering*, 2019: 248-255.
- [38] Qasem A, Shirani P, Debbabi M, et al. Automatic Vulnerability Detection in Embedded Devices and Firmware[J]. *ACM Computing Surveys*, 2022, 54(2): 1-42.
- [39] Binwalk. <https://github.com/ReFirmLabs/binwalk>. 2015.
- [40] Binary Analysis Tool. <https://github.com/armijnhemel/binaryanalysis>. 2016.
- [41] firmware-mod-kit. <https://code.google.com/archive/p/firmware-mod-kit>. 2012.
- [42] Tien C W, Tsai T T, Chen I Y, et al. UFO - Hidden Backdoor Discovery and Security Verification in IoT Device Firmware[C]. *2018 IEEE International Symposium on Software Reliability Engineering Workshops*, 2018: 18-23.
- [43] Zaddach J, Costin A. Embedded devices security and firmware reverse engineering[C]. *Black-Hat USA*, 2013.
- [44] pkcrack. <https://github.com/keyunluo/pkcrack>. 2017.
- [45] ARCHPR. <https://cn.elcomsoft.com/archpr.html>.
- [46] signsrch. https://github.com/nihilus/IDA_Signsrch. 2015.
- [47] file. <https://github.com/file/file>. 2011.
- [48] Li T, Tian Y J, Ge Y C, et al. A Survey of IoT Firmware Vulnerability Security Detection[J]. *Journal of Information Security Research*, 2022, 8(12): 1146-1155.
(李涛, 田迎军, 葛阳晨, 等. 物联网固件漏洞安全检测综述[J]. *信息安全研究*, 2022, 8(12): 1146-1155.)
- [49] Zhu R J, Zhang B F, Mao J J, et al. Determining Image Base of ARM Firmware Based on Matching String Addresses[J]. *Acta Electronica Sinica*, 2017, 45(6): 1475-1482.
(朱瑞瑾, 张宝峰, 毛军捷, 等. 一种基于匹配字符串地址判定 ARM 固件装载基址的方法[J]. *电子学报*, 2017, 45(6): 1475-1482.)
- [50] vxhunter. <https://github.com/PAGalaxyLab/vxhunter>. 2019.
- [51] Wen H H, Lin Z Q, Zhang Y Q. FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities from Bare-Metal Firmware[C]. *The 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020: 167-180.
- [52] Chen L B, Cai Q P, Ma Z B, et al. SFuzz: Slice-Based Fuzzing for Real-Time Operating Systems[C]. *The 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022: 485-498.
- [53] Rbasefind. <https://github.com/sgayou/rbasefind>. 2017.
- [54] Pewny J, Garmany B, Gawlik R, et al. Cross-Architecture Bug Search in Binary Executables[C]. *2015 IEEE Symposium on Security and Privacy*, 2015: 709-724.
- [55] Jin X, Pei K X, Won J Y, et al. SymLM: Predicting Function Names in Stripped Binaries via Context-Sensitive Execution-Aware Code Embeddings[C]. *The 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022: 1631-1645.
- [56] Interactive Disassembler Professional. <https://hex-rays.com/ida-pro/>.
- [57] Ghidra. <https://ghidra-sre.org/>.

- [58] radare2. <https://www.radare.org/n/radare2.html>.
- [59] capstone. <http://www.capstone-engine.org/>.
- [60] keystone. <https://github.com/keystone-engine/keystone>.
- [61] Chen J Y, Zuo C S, Diao W R, et al. Your IoTs Are (Not) Mine: On the Remote Binding between IoT Devices and Users[C]. *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2019: 222-233.
- [62] Yu L, Liu Y, Jing P, et al. Towards automatically reverse engineering vehicle diagnostic protocols[C]. *31st USENIX Security Symposium*, 2022: 1939-1956.
- [63] Bai X L, Xing L Y, Zheng M, et al. IDEA: Static Analysis on the Security of Apple Kernel Drivers[C]. *The 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020: 1185-1202.
- [64] Tsang R, Joseph D, Asmita, et al. FANDEMIC: Firmware Attack Construction and Deployment on Power Management Integrated Circuit and Impacts on IoT Applications[C]. *Proceedings 2022 Network and Distributed System Security Symposium*, 2022.
- [65] Ren Y Z, Zhang Y W, Ai C W. Survey on Taint Analysis Technology[J]. *Journal of Computer Applications*, 2019, 39(8): 2302-2309.
(任玉柱, 张有为, 艾成炜. 污点分析技术研究综述[J]. *计算机应用*, 2019, 39(8): 2302-2309.)
- [66] Celik Z B, Babun L, Sikder A K, et al. Sensitive information tracking in commodity IoT[C]. *27th {USENIX} Security Symposium*, 2018: 1687-1704.
- [67] Liu L X, Pan Z L, Li Y, et al. Firmware Vulnerability Static Localization Method Based on Front-End and Back-End Correlation Analysis[J]. *Netinfo Security*, 2022, 22(8): 44-54.
(刘翎翔, 潘祖烈, 李阳, 等. 基于前后端关联性分析的固件漏洞静态定位方法[J]. *信息网络安全*, 2022, 22(8): 44-54.)
- [68] Zhang H, Chen W T, Hao Y, et al. Statically Discovering High-Order Taint Style Vulnerabilities in OS Kernels[C]. *The 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021: 811-824.
- [69] Shoshitaishvili Y, Wang R Y, Salls C, et al. SOK: (State of) the Art of War: Offensive Techniques in Binary Analysis[C]. *2016 IEEE Symposium on Security and Privacy*, 2016: 138-157.
- [70] Cadar C, Dunbar D, Engler D R. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs[C]. *OSDI*, 2008, 8: 209-224.
- [71] Davidson D, Moench B, Ristenpart T, et al. FIE on Firmware: Finding Vulnerabilities in Embedded Systems Using Symbolic Execution[C]. *USENIX Security Symposium*, 2013: 463-478.
- [72] Corteggiani N, Camurati G, Francillon A. Inception: System-wide security testing of real-world embedded systems software[C]. *27th {USENIX} security symposium*, 2018: 309-326.
- [73] Ai C W, Dong W Y, Gao Z C. A Novel Concolic Execution Approach on Embedded Device[C]. *The 2020 4th International Conference on Cryptography, Security and Privacy*, 2020: 47-52.
- [74] Nethercote N, Seward J. Valgrind[J]. *ACM SIGPLAN Notices*, 2007, 42(6): 89-100.
- [75] Zaddach J, Bruno L, Francillon A, et al. Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares[C]. *Proceedings 2014 Network and Distributed System Security Symposium*, 2014: 1-16.
- [76] Cao C, Guan L, Ming J, et al. Device-Agnostic Firmware Execution Is Possible: A Concolic Execution Approach for Peripheral Emulation[C]. *Annual Computer Security Applications Conference*, 2020: 746-759.
- [77] Zhou W, Guan L, Liu P, et al. Automatic firmware emulation through invalidity-guided knowledge inference[C]. *30th USENIX Security Symposium*, 2021: 2007-2024.
- [78] Eschweiler S, Yakdan K, Gerhards-Padilla E. DiscovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code[C]. *Proceedings 2016 Network and Distributed System Security Symposium*, 2016: 58-79.
- [79] Feng Q, Zhou R, Xu C, et al. Scalable graph-based bug search for firmware images[C]. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: 480-491.
- [80] Shirani P, Collard L, Agba B L, et al. BINARM: Scalable and Efficient Detection of Vulnerabilities in Firmware Images of Intelligent Electronic Devices[C]. *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2018: 114-138.
- [81] David Y, Partush N, Yahav E. FirmUp[J]. *ACM SIGPLAN Notices*, 2018, 53(2): 392-404.
- [82] Feng Q, Wang M, Zhang M, et al. Extracting conditional formulas for cross-platform bug search[C]. *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017: 346-359.
- [83] Dinaburg A, Ruef A. Mcsema: Static translation of x86 instructions to llvm[C]. *ReCon 2014 Conference*, 2014.
- [84] Xu X J, Liu C, Feng Q, et al. Neural Network-Based Graph Embedding for Cross-Platform Binary Code Similarity Detection[C]. *The 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 363-376.
- [85] Gao J, Yang X, Fu Y, et al. VulSeeker: A Semantic Learning Based Vulnerability Seeker for Cross-Platform Binary[C]. *2018 33rd IEEE/ACM International Conference on Automated Software Engineering*, 2018: 896-899.
- [86] Liu B C, Huo W, Zhang C, et al. \$a\$ Diff: Cross-Version Binary Code Similarity Detection with DNN[C]. *2018 33rd IEEE/ACM International Conference on Automated Software Engineering*, 2018: 667-678.
- [87] Yang J, Fu C, Liu X Y, et al. Codee: A Tensor Embedding Scheme for Binary Code Search[J]. *IEEE Transactions on Software Engineering*, 2022, 48(7): 2224-2244.
- [88] Pei K, Xuan Z, Yang J, et al. Trex: Learning execution semantics from micro-traces for binary similarity[J]. *arXiv preprint arXiv:2012.08680*, 2020.
- [89] Kim G, Hong S, Franz M, et al. Improving cross-platform binary analysis using representation learning via graph alignment[C]. *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022: 151-163.
- [90] Zhao B, Ji S, Xu J, et al. A large-scale empirical analysis of the

- vulnerabilities introduced by third-party components in IoT firmware[C]. *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022: 442-454.
- [91] Cai Zirui Research on Heuristic Simulation and Vulnerability Detection of Typical IoT Device Firmware [D]. Xi'an University of Electronic Science and Technology, 2022.
(蔡子睿. 典型物联网设备固件启发式仿真与漏洞检测研究[D]. 西安: 西安电子科技大学, 2022.)
- [92] Bellard F. QEMU, a fast and portable dynamic translator[C]. *USENIX annual technical conference, FREENIX Track*, 2005, 41: 46.
- [93] Unicorn <https://www.unicorn-engine.org/>.
- [94] Qiling. <https://github.com/qilingframework/qiling>.
- [95] Emux. <https://github.com/therealsaumil/emux>.
- [96] Panda. <https://github.com/panda-re/panda>.
- [97] Kammerstetter M, Platzer C, Kastner W. Prospect: Peripheral Proxying Supported Embedded Code Testing[C]. *The 9th ACM Symposium on Information, Computer and Communications Security*, 2014: 329-340.
- [98] Muench M, Nisi D, Francillon A, et al. Avatar 2: A multi-target orchestration platform[C]. *Proc. Workshop Binary Anal. Res*, 2018, 18: 1-11.
- [99] Koscher K, Kohno T, Molnar D. {SURROGATES}: Enabling {Near-Real-Time} dynamic analyses of embedded systems[C]. *9th USENIX Workshop on Offensive Technologies*, 2015.
- [100] Kammerstetter M, Burian D, Kastner W. Embedded security testing with peripheral device caching and runtime program state approximation[C]. *10th International Conference on Emerging Security Information, Systems and Technologies*, 2016, 118: 118.
- [101] Gustafson E, Muench M, Spensky C, et al. Toward the analysis of embedded firmware through automated re-hosting[C]. *22nd International Symposium on Research in Attacks, Intrusions and Defenses*, 2019: 135-150.
- [102] Ruge J, Classen J, Gringoli F, et al. Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets[C]. *29th USENIX Security Symposium*, 2020: 19-36.
- [103] Zheng Y, Davanian A, Yin H, et al. FIRM-AFL: High-Throughput Greybox Fuzzing of IoT Firmware via Augmented Process Emulation[C]. *USENIX Security Symposium*, 2019: 1099-1114.
- [104] Kim M, Kim D, Kim E, et al. FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis[C]. *Annual Computer Security Applications Conference*, 2020: 733-745.
- [105] Jiang M H, Ma L, Zhou Y J, et al. ECMO: Peripheral Transplantation to Rehost Embedded Linux Kernels[C]. *The 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021: 734-748.
- [106] Clements A A, Gustafson E, Scharnowski T, et al. {HALucinator}: Firmware re-hosting through abstraction layer emulation[C]. *29th USENIX Security Symposium*, 2020: 1201-1218.
- [107] Miller B P, Fredriksen L, So B. An Empirical Study of the Reliability of UNIX Utilities[J]. *Communications of the ACM*, 1990, 33(12): 32-44.
- [108] Li W, Guan L, Lin J, et al. From library portability to para-rehosting: Natively executing microcontroller software on commodity hardware[J]. arXiv preprint arXiv:2107.12867, 2021.
- [109] Seidel L, Maier D, Muench M. Forming faster firmware fuzzers[C]. *Proceedings of the 32nd USENIX Conference on Security Symposium*, 2023: 2903-2920.
- [110] Feng B, Mera A, Lu L. P2im: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling[C]. *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020: 1237-1254.
- [111] Mera A, Feng B, Lu L, et al. DICE: Automatic Emulation of DMA Input Channels for Dynamic Firmware Analysis[C]. *2021 IEEE Symposium on Security and Privacy*, 2021: 1938-1954.
- [112] Scharnowski T, Bars N, Schloegel M, et al. Fuzzware: Using Precise {MMIO} Modeling for Effective Firmware Fuzzing[C]. *31st USENIX Security Symposium*, 2022: 1239-1256.
- [113] Johnson E, Bland M, Zhu Y F, et al. Jetset: Targeted Firmware Rehosting for Embedded Systems[C]. *USENIX Security Symposium*, 2021: 321-338.
- [114] Ma X Y, Zeng Q, Chi H T, et al. No More Companion Apps Hacking but One Dongle: Hub-Based Blackbox Fuzzing of IoT Firmware[C]. *The 21st Annual International Conference on Mobile Systems, Applications and Services*, 2023: 205-218.
- [115] Busch M, Machiry A, Spensky C, et al. TEEzz: Fuzzing Trusted Applications on COTS Android Devices[C]. *2023 IEEE Symposium on Security and Privacy*, 2023: 1204-1219.
- [116] Li W Q, Shi J M, Li F J, et al. μ AFL $\$$: Non-Intrusive Feedback-Driven Fuzzing for Microcontroller Firmware[C]. *2022 IEEE/ACM 44th International Conference on Software Engineering*, 2022: 1-12.
- [117] Srivastava P, Peng H, Li J, et al. Firmfuzz: Automated iot firmware introspection and analysis[C]. *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, 2019: 15-21.
- [118] Hernandez G, Muench M, Maier D, et al. FirmWire: Transparent Dynamic Analysis for Cellular Baseband Firmware[C]. *Proceedings 2022 Network and Distributed System Security Symposium*, 2022.
- [119] TriforceAFL. <https://github.com/nccgroup/TriforceAFL>. 2016.
- [120] Kim T, Kim C H, Rhee J, et al. {RVFuzzer}: Finding input validation bugs in robotic vehicles through {Control-Guided} testing[C]. *28th USENIX Security Symposium*, 2019: 425-442.
- [121] Kim H, Ozmen M O, Bianchi A, et al. PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles[C]. *Proceedings 2021 Network and Distributed System Security Symposium*, 2021.
- [122] Kim H, Ozmen M O, Celik Z B, et al. PatchVerif: Discovering Faulty Patches in Robotic Vehicles[C]. *USENIX Security Symposium*, 2023:3011-3028.
- [123] Bulekov A, Das B, Hajnoczi S, et al. Morphuzz: Bending (Input) Space to Fuzz Virtual Devices[C]. *31st USENIX Security Symposium*, 2022: 1221-1238.



马晓成 于 2020 年在上海工程技术大学计算机科学与技术专业获得学士学位, 现在信息工程大学网络空间安全专业攻读硕士学位。研究领域为软件安全分析、物联网安全。研究兴趣包括: 静态分析、漏洞挖掘、固件安全。Email: wuliangttkx@foxmail.com



王允超 于 2020 年在信息工程大学网络空间安全专业获得博士学位, 现任信息工程大学讲师。研究领域为软件与系统漏洞分析。研究兴趣包括: 逆向工程、漏洞挖掘。Email: w_yunchao@sina.com



于洪洋 现于信息工程大学网络空间安全专业攻读学士学位。研究领域为嵌入式设备漏洞挖掘。研究兴趣包括: 嵌入式设备固件仿真与模拟、固件提取技术。Email: yu_yang_yang1318@163.com



魏强 于信息工程大学获得博士学位。现任信息工程大学教授, 博士生导师。研究领域为软件安全分析、工控系统安全。研究兴趣包括网络安全、工业互联网安全和漏洞发现。Email: funnywei@163.com