Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2021-CJ-003

2021-CJ-003

物联网固件安全缺陷检测研究进展

张弛，司徒凌云，王林章

Technical Report 2021

物联网固件安全缺陷检测研究进展

张 弛[1,2], 司徒凌云[1,2], 王林章[1,2]

[1]计算机软件新技术国家重点实验室(南京大学) 南京 中国 210023
[2]南京大学 计算机科学与技术系 南京 中国 210023

摘要 固件是物联网设备的基础使能软件, 其中存在的安全缺陷是物联网设备遭受攻击的根本原因之一。由于物联网设备资源受限, 难以部署完善的安全防护机制, 身处不安全的网络环境中, 其固件缺陷一旦被恶意利用, 轻则使设备宕机, 重则威胁安全攸关领域基础设施, 造成巨大的生命财产损失。因此, 有效的固件安全缺陷检测已然成为保障物联网设备安全的关键, 也成为学术界和工业界研究的热点。面对物联网设备数量的高速增长、固件自身规模和复杂性的不断攀升、固件类型的日益多样化、固件缺陷的持续增多, 现有的物联网固件安全缺陷检测研究面临挑战。本文归纳了典型物联网固件实现缺陷类型, 分析了典型缺陷产生机理, 从静态分析、符号执行、模糊测试、程序验证、基于机器学习的方法等角度综述了现有固件缺陷检测方法。通过对不同方法优势与不足的分析, 为进一步提升固件安全缺陷检测方法的智能化、精准化、自动化、有效性、可扩展性提供指导。在此基础上, 本文展望了未来可以开展的研究工作。

关键词 物联网设备; 实时操作系统; 固件安全; 缺陷检测
中图法分类号 TP311 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2021.05.09

Research Progress on Security Defect Detection of IoT Firmware

ZHANG Chi[1,2], SITU Lingyun[1,2], WANG Linzhang[1,2]

[1]State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China
[2]School of Computer Science and Technology, Nanjing University, Nanjing 210023, China

Abstract Firmware is the basic enabling software for IoT devices, and the security defects that exist are one of the root causes of IoT devices being attacked. Due to the limited resources of IoT devices, it is difficult to deploy a complete security protection mechanism. In an insecure network environment, once the firmware defects are maliciously exploited, the device will be down, and the security infrastructure will be threatened. Caused huge loss of life and property. Therefore, effective firmware security defect detection has become the key to the security of IoT devices, and has become a hot topic in academic and industrial research. Faced with the rapid growth of the number of IoT devices, the increasing size and complexity of firmware itself, the increasing variety of firmware types, and the continued increase in firmware defects, existing IoT firmware security defect detection research faces challenges. This paper summarizes the typical types of defect implementation of IoT firmware, analyzes the mechanism of typical defect generation, and summarizes the existing firmware defect detection methods from the perspectives of static analysis, symbolic execution, fuzzing, program verification, and machine learning-based methods. Through the analysis of the advantages and disadvantages of different methods, it provides guidance for further improving the intelligence, accuracy, automation, effectiveness and scalability of the firmware security defect detection method. Further, this article also look forward to the research work that can be carried out in the future.

Key words Internet of Things devices; real-time operating system; firmware security; defect detection

1 引言

物联网(Internet of Things, IoT)是由各种拥有唯一标识的计算设备、机械或数字对象、人与物通过通信技术建立连接, 实现自主的人、机、物之间的数据传输与信息交换的系统[1]。随着设备硬件的发展与以5G[2]、NB-IoT[3]为代表的通信技术的进步, 众多物联网设备如移动终端、路由器、交换机、网络监控摄像头、智能家电、智能汽车、智能门锁、智能电表等被接入到网络空间, 广泛部署、应用在智能交

通讯作者: 王林章, 博士, 教授, Email: lzwang@nju.edu.cn。
本课题得到国家自然科学基金(No. 62032010)资助。
收稿日期: 2020-06-23; 修改日期: 2020-11-26; 定稿日期: 2021-03-05

通、智能医疗、智能电网等安全攸关的领域。根据GSMA预计, 到2025年, 全球物联网设备数目将高达25.3亿[4]。

固件(firmware)是运行在物联网设备上的核心软件之一。IEEE 标准 12207-2008 将固件定义为“硬件设备和以只读软件形式存储于硬件设备中的计算机指令和数据的结合”[5]。大部分嵌入式设备中的软件都是以二进制形式存储在只读存储器、可编程只读存储器、可擦可编程只读存储器、带电可擦可编程只读存储器、闪存等永久存储设备中, 因此该类软件一般称为固件。

固件按照其是否内置操作系统、以及内置操作系统的类型可分如表 1 所示的三类: (1)单片固件, 通常采取单个二进制镜像的形式, 无需底层操作系统, 直接基于底层硬件驱动完成所有功能, 或者只包含部分系统的库; (2)基于 Linux 的固件, 以 Linux 作为底层的系统, 基于 Linux 进行开发; (3)基于 RTOS 的固件。RTOS(real-time operating system)是指实时处理数据、没有缓冲延迟的操作系统[6]。在嵌入式应用中使用 RTOS 可以使程序最大化利用有限的计算资源、简化应用程序设计、提高开发人员效率。RTOS 受到了越来越多编程人员的重视, 涌现出一批优秀的嵌入式实时操作系统如 VxWorks[7]、QNX[8]、FreeRTOS[9]、RTEMS[10]等, 其中 FreeRTOS 因其开源免费、小巧简单等特点受到了学术界和产业界的广泛采用。

固件是设备上电后最先执行的代码, 主要负责系统硬件的初始化、加载操作系统、获取最终控制权、并为上层软件有效使用硬件设备提供调用接口。早期的计算机上的固件叫作基本输入输出系统(Basic Input Output System, BIOS), 由于其开发效率低, 功能扩展性差以及更新机制不完善等缺点, 已经逐渐被支持图形界面和鼠标操作的统一可扩展固件接口 (Unified Extensible Firmware Interface, UEFI)所取代。嵌入式设备由于存储空间小, 实现功能与计算机相比较更为单一, 因此嵌入式设备上的固件通常是指嵌入式设备中的整个软件系统, 即包含操作系统、第三方库、应用程序等。上电之后固件负责硬件平台的初始化和之后的嵌入式设备的功能实现。

固件中存在缺陷是造成物联网设备遭受安全攻击的根本原因之一。一方面, 固件主要由C语言实现, 尽管 C 语言为程序员提供了多种机制和 API(应用程序接口)确保安全, 但也需要程序员做出关键决定, 如输入合法性检查、越界检查等等。程序员对代码安全性认识不足导致编程实践中存在疏忽、失误以致程序的关键部分存在缺陷, 在连接互联网运行的场景下, 这样的缺陷成为安全漏洞, 容易被黑客利用并攻击; 另一方面, 固件的执行权限高于操作系统, 能实现对所有硬件设备的直接控制, 同时也是操作系统安全机制的盲区所在, 始于固件的攻击能够直接或间接地影响上层操作系统或应用软件的安全机制。另外, 越来越多的由早期版本固件驱动的嵌入式设备也接入物联网, 早期固件缺乏对联网环境下的安全考虑, 带来了安全威胁。

基于固件缺陷的安全攻击事故频发。典型的, 2018 年, 思科 Talos 安全研究团队发现攻击者利用恶意程序 VPNFilter 感染了全球 54 个国家的超过 50 万台路由器[11]; 2016 年, 攻击者利用物联网设备对美国域名服务器管理服务供应商 Dyn 发起分布式拒绝服务(DDoS)攻击, 制造了有史以来规模最大的 DDoS 攻击[12]。

有效的固件缺陷检测是保障物联网设备安全的关键。在联网环境下, 日益增长的嵌入式设备数目, 日益复杂的固件系统规模, 以及嵌入式设备低功耗环境下保护机制的缺失, 使得固件安全问题日益凸显, 检测固件中的缺陷成为了近年来的研究热点。

表 1 固件分类及其特点

Table 1 Firmware Classification and its Characteristics

固件类型	优点	缺点
单片固件	小巧	实现复杂, 功能简单, 可移植性较差
基于 Linux 的固件	可直接使用 Linux 中的通信、I/O、内存和进程管理等功能, 开发简单; 内存和 CPU 使用方面具有较高的效率, 且非常稳定; 可移植性高	固件较大, 需要存储空间较多
基于 RTOS 的固件	可以使用 RTOS 中的调度系统, 相关通信、I/O、内存管理等库较多, 可以选择使用; 可移植性高; 固件小巧	出现时间较短, 可能存在潜在缺陷

固件处于计算系统的核心地位, 自身的安全缺陷往往隐藏较深, 需要满足特定的条件才能被触发, 加之, 物联网设备具有多样、异构等特性, 这使得检测固件缺陷变得极其困难。现有的缺陷检测技术主要包括静态分析[40-44]、符号执行[49-53]、模糊测试[28,56-58,63-71]、程序验证[29,75-76]以及机器学习[79-88], 然而现阶段基于上述技术的缺陷检测方法与工具在

应用到物联网固件时依旧面临挑战:

(1) 无法获取源码且代码类型复杂。厂商为了保证设备安全通常不会公开固件源码; 且为了完成设备的各项功能, 固件通常混杂有汇编、Java、JavaScript 等不同类型代码, 阻碍了反编译的进行;

(2) 不同类型的固件差别较大。现有的固件根据所使用的系统可以分为单片固件、基于 Linux 的固件和基于 RTOS 的固件, 不同类型固件代码架构差异较大; 且运行在不同硬件上的固件处理器架构和内存架构差异较大, 极大阻碍了固件测试工具的可扩展性;

(3) 测试用例难以构建。固件由于其功能严重依赖外部输入和中断, 外设类型极其丰富及内部状态复杂, 因此难以构造统一的输入来满足众多外设, 也难以构造极端的测试场景去探索更深的路径。

(4) 依赖资源及技术不足。固件运行的硬件资源有限, 因此较难在物联网设备中直接运行测试软件; 且物联网设备通常很少向第三方提供完备的测试接口, 无法获取足够的固件运行时信息; 固件的反汇编技术仍然存在不足, 当固件中存在混淆机制时难以获取源码; 模拟器能力有限导致大量固件无法进行模糊测试。

本文总结了典型的固件安全缺陷类型; 分析了各类固件缺陷的机理; 综述了现有固件安全缺陷检测技术与工具, 分析了其优势与不足, 为进一步的固件缺陷检测技术研究提供了指导。

文章其余部分结构如下: 第二节介绍典型的固件缺陷类型以及产生机理, 包括内存损坏、命令注入、程序逻辑缺陷、并发问题等实现缺陷, 配置缺陷以及定制缺陷; 第三节从静态分析、符号执行、模糊测试、程序验证和基于机器学习五个技术角度对现有固件缺陷检测方法进行了分析与比较; 然后, 讨论了未来的研究方向; 最后, 总结全文。

2 固件缺陷及其机理分析

Zhao 等人[13]和 Jing 等人[14]在研究中将物联网安全分为了图 1 所示的应用层、网络层和感知层三层架构, 对每一层可能出现的问题和受到的攻击进行了综述。由于不同物联网系统的设计不同, 这三个部分所在的位置也不同, 在功能简单的物联网设备中, 感知层、应用层和网络层可能都位于物联网设备的固件中; 在成体系的大型物联网系统中, 物联网设备的固件中可能仅保留感知层和网络层的部分实现。

本文针对物联网设备的固件中存在的缺陷进行了研究。通常, 错误指代码中对编程语言规范的违反; 缺陷指固件设计与代码实现上的不足; 漏洞指可以被攻击人员利用的缺陷。本文将使用缺陷来指代固件代码实现和设计上的不足, 以及可被攻击者利用的漏洞。固件中的缺陷可以分为三个大类, 分别为实现缺陷、配置缺陷和定制缺陷。

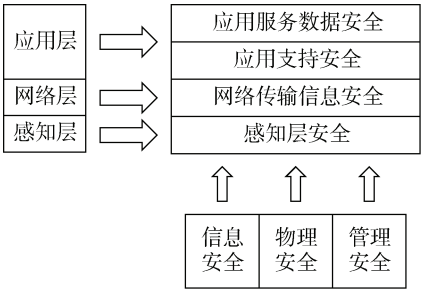

图 1 物联网安全架构

Figure 1 IoT Security architecture

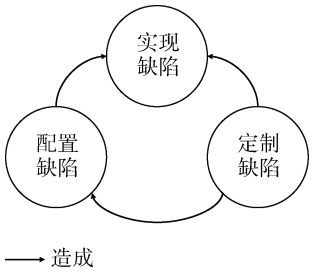

图 2 三种缺陷之间的关系

Figure 2 Relationship between three types of defects

实现缺陷指开发人员在开发过程中引入的源码上的不足, 实现缺陷由程序开发人员造成; 配置缺陷指部署人员在部署过程中编写配置文件时引入的缺陷, 配置缺陷由程序部署人员造成; 定制缺陷指用户为使固件满足特定需求或者环境而对固件进行了某些修改, 这一过程中引入的缺陷, 定制缺陷由程序使用人员造成。其中, 实现缺陷又包含内存损坏缺陷、命令注入缺陷、程序逻辑缺陷、并发问题缺陷, 配置缺陷包含功能配置参数缺陷、性能配置参数缺陷、权限配置参数缺陷。本章将对这几类缺陷的机理进行分析。

2.1 实现缺陷

2.1.1 内存损坏类缺陷

内存损坏类缺陷指不正确的内存访问导致堆、栈内存发生错误[15]。大部分固件程序本质上来说就是用 C 语言写成的程序, 只不过底层部分会需要一些汇编指令和硬件直接交互, 因此固件中的内存损坏类缺陷同普通桌面软件中的内存损坏类缺陷没有区别, 而这些缺陷也同样有可能导致被攻击者利用

的漏洞。具体来说可以有以下几类: 1)堆溢出漏洞; 2)栈溢出漏洞; 3)内存泄漏漏洞; 4)数组越界读写漏洞; 5)use-after-free 漏洞; 6)double-free 漏洞; 7)空指针解引用; 8)格式字符串漏洞。

内存损坏缺陷在缺陷检测中较为常见, 固件中的内存损坏缺陷可能会引起固件程序挂起、崩溃、重启、敏感信息泄露、拒绝服务攻击、缓冲区溢出攻击等行为, 而且内存相关缺陷大多由于外部输入引起, 攻击者可以通过构造特定的输入来触发漏洞以达到特定目的。

例如, 在 ASUS 路由器中的栈溢出漏洞会造成远程代码执行(CVE-2017-12754)[16]; FreeRTOS 的函数栈中由于不正确的错误处理逻辑导致了 double free(CVE-2018-16528) 和 uninitialized pointer free (CVE-2018-16522)[17]; Netgear 的无线驱动中包含有一个堆溢出漏洞(CVE-2006-6125), 攻击者可以通过构造一个特定的 802.11 管理片段来触发这个漏洞[18]。

2.1.2 命令注入缺陷

命令注入缺陷指由于缺少对用户输入进行完备的检查导致恶意用户可以通过构造输入来运行非预期的命令[19], 与普通桌面软件相比, 固件直接操作硬件, 拥有更高的权限, 因此命令注入缺陷的危害也更大。包含系统的固件大都支持运行命令行, 但这也给固件留下了安全隐患, 当固件没有对用户输入的命令行指令进行检查时, 攻击者将可以构造特定的指令来完成某些动作, 达到自己的目的, 如攻击者可以使用查询来查看嵌入式设备上的数据库或者获取嵌入式设备中的文件, 以造成敏感信息泄露; 可以使用删除指令来删除设备上的重要信息, 使系统无法继续运行。

例如, 在D-Link设备DIR-823G中存在一个命令注入缺陷(CVE-2019-7298), 当攻击者发送一个定制的HNAP1请求时可以触发这一缺陷[20], 缺陷的触发形成漏洞, 使攻击者可以运行任意的系统命令; D-Link DCS 网络摄像机中没有对 IP 地址进行检查, 可以通过<Camera-IP>/common/info.cgi 获取网络摄像机的配置文件, 无需任何验证就可读取其中的敏感信息(CVE-2018-18441)[21]。

2.1.3 程序逻辑缺陷

程序逻辑缺陷是指程序不严谨的逻辑所留下的缺陷, 使软件无法正常运行或给了不法分子可乘之机[22]。如系统权限等级设计不严谨, 就会使系统的底层置于危险之中; 系统或者传输过程中没有加密或者加密算法过于简单, 就会使系统有被破解的风险; 在代码中硬编码账号、密码等信息, 使系统容易被不法分子侵入, 且大部分厂商为了方便用户使用, 通常会为设备设置相同的出厂用户名和密码, 如果用户未对其进行修改, 则会有遭遇攻击的风险; 防御措施不完善就容易遭受各种攻击如拒绝服务攻击等。

例如, Hikvision 的大量设备在配置文件中硬编码了某些密码, 这使得不法分子可以通过这些密码来获取一定的权限[23]; Netgear FVG318 在 TCP 通信中使用了错误的校验和, 使设备容易受到拒绝服务攻击(CVE-2006-4143)[24]; NETGEAR WGT624 无线路由器设备中在配置文件中使用明文记录了敏感信息, 使攻击者可以获取密码并取得权限[25]。

2.1.4 并发问题缺陷

并发问题缺陷指对多线程运行的固件设计不合理导致固件运行时产生数据竞争、死锁等行为[26]。随着硬件的发展, 嵌入式设备不仅仅局限于单个任务的运行, 通常会被设计多个任务同时运行, 便会产生并发问题缺陷, 而这种缺陷无论是工程人员自己编写的调度系统还是使用现成的 RTOS 进行调度都有可能发生。为嵌入式软件提供多任务调度是RTOS 最基本的功能, 通过 RTOS 提供的 API 可以将函数程序分成独立的任务, 并为他们提供了多种调度方式, 此外还提供了任务中断、任务间通信等功能。RTOS 提供的多任务调度简单易懂, 但使用不当也会造成许多问题, 例如死锁、数据竞争等[27]。许多学者对 RTOS 上的任务调度可能产生的问题进行了研究, 提出了检测和预防的方法[28-29]。

2.2 配置缺陷

在大型软件领域常见的配置错误在固件中也经常发生, 随着开发人员对代码的可移植性和可重用性的追求, 产生了大量的配置文件来适配不同的硬件、架构和系统, 使得厂商在不同平台上的代码无需进行太多修改。广泛使用的 FreeRTOS 便是可移植的一个案例, FreeRTOS 作为一个可移植、可配置的轻量级操作系统, 在提供了大量便利的同时也带来了许多问题, 尤其是其配置问题。

具体的, FreeRTOS 支持 35 种微处理架构, 但针对于每一种架构都需要对 FreeRTOS 进行一系列的配置。用户在使用 FreeRTOS 时也需要根据自己的应用来进行相应的配置以符合自己的要求。AWS 为 FreeRTOS 编写了许多例如 TCP/IP、UDP、IO 等函数栈, 这些函数栈的使用需要用户根据需要进行配置; 开发人员在进行开发时通常会复制他人的配置文件并自己根据需要做一些修改, 但相当一部分开发人员在进行配置时并没有很好的理解这些配置的真正含义, 往往一个不起眼的不当配置就可能引起

致命的缺陷, 这些缺陷有可能会造成系统崩溃、挂起或者其他无法预知的故障。

典型的, 安全公司辛培力安(Zimperium)对FreeRTOS系统及其TCP/IP相关的函数栈进行分析, 发现了大量同配置相关的缺陷, 并获得了CVE编号如CVE-2018-16528等[17], 这些缺陷可能会造成信息泄露、远程代码执行。

配置缺陷可以分为功能配置参数缺陷、性能配置参数缺陷、权限配置参数缺陷。固件上的配置缺陷目前学术界研究缺失, 但大型软件系统和云系统中都有较多研究, 这些研究对固件配置缺陷检测或许会有启发。

2.2.1 功能配置参数缺陷

功能配置参数缺陷指的是由于部署人员的粗心或对配置项的不了解而产生的诸如配置值不当、类型不匹配、配置项缺失等问题[30]。

配置值不当包括不正确的文件路径、配置值错误的拼写等。D-Link DIR-822设备中的配置文件中有多个错误的配置且缺乏安全检查(CVE-2018-19990)[31], 这些错误可能会影响设备的内存管理; 类型不匹配指错误的配置值类型, 如给int型的配置项设置了一个字符串类型的值; 配置项的缺失是指没有对某一项配置赋予值, 如CVE-2019-10132[32]。这类错误通常只有当程序运行到需要这一配置项时才会发现, 因此潜藏较深。

2.2.2 性能配置参数缺陷

性能配置参数缺陷指性能相关配置项的值发生错误导致无法提供系统预期的性能, 或者使固件各部分无法很好兼容[30], 性能配置参数同硬件有较大联系。这类缺陷通常不会直接引起系统挂起、崩溃等异常行为, 但是无法提供系统预期的性能, 无法满足用户的需求。如在配置文件中为固件分配了较小的内存空间, 这样就无法提供预期的性能。

例如, 在CVE-2019-2041中, 由于没有对NFC进行合理配置而是使用了默认值, 导致了NFC无法区分个人设备[33]。

2.2.3 权限配置参数缺陷

权限配置参数缺陷允许用户做出超越自身权限的行为, 如果该用户被攻击, 则系统会遭受更大的损失。

例如, ABB VSN300 WiFi Logger Card中没有正确限制不同客户的权限, 致使攻击者可以访问到关键信息(CVE-2017-7916)[34]; Dasan H660RM设备上的BOA服务器配置将敏感数据放在了错误的位置, 低权限的用户可以随意对这些文件进行查看(CVE-2019-9976)[35]。

陈伟等人[36]、Xu等人[37]、李福亮等人[38]已对普通软件领域和互联网领域的配置缺陷产生原因、影响、测试手段、修复手段等做了大量的调研工作。Yin等人[30]的工作对现实生活中的软件进行了研究, 对配置缺陷进行了分类, 总结了配置缺陷产生的原因及影响, 且说明了配置缺陷在软件缺陷中占了非常大的比例。

2.3 定制缺陷

定制缺陷是指用户根据自己和环境需要对固件进行一定的修改, 但是这种修改使固件产生了不可预期的错误。这一错误的发生可能是由于用户对固件源码的不熟悉、固件对不同环境的适配性不好等, 定制缺陷可造成实现缺陷和配置缺陷, 这一缺陷的检测无需对整个固件进行测试, 可以将测试重点放在被修改及相关部分。

虽然固件中的缺陷分为不同的种类, 但是这些缺陷之间通常会有一定的依赖关系, 一种缺陷的发生有可能会导致其他缺陷的发生。实现缺陷可以视为最基本的缺陷, 其来源有多种, 但主要来源还是开发人员在编写过程中的粗心大意; 配置缺陷主要由部署人员引入, 配置值的本质就是程序中的变量, 变量的输入来自于配置编写人员, 如果配置值编写不当, 而代码中没有对配置值进行检查, 也会引出实现缺陷; 定制缺陷由用户自己引入, 用户在定制过程中可能会对配置或者代码进行一定的修改, 因此定制缺陷会造成实现缺陷和配置缺陷。

3 固件安全缺陷检测技术

有效的检测上述固件安全缺陷是保障物联网设备安全的关键。自动化的固件安全缺陷检测一方面可以实现在固件发布之前发现缺陷, 有效避免大量损失; 另一方面可以实现短时间内的大量固件分析, 节省了开发人员时间以及相关财力物力。

目前主流的自动化缺陷检测技术可以分为五类, 分别是静态分析、符号执行、模糊测试、程序验证、机器学习, 不同的技术有不同的应用场景和优势, 几种检测方法之间还可以进行结合, 相辅相成。下面将对这五类检测方法进行介绍。

3.1 静态分析技术

静态分析是指在不运行固件系统的情况下, 基于固件代码制品(如源码、抽象语法树、中间表示等)对其进行分析, 通过与典型固件缺陷模式的匹配, 达到检测潜在安全缺陷的目的[39]。

PIE[40]将固件通信协议中用于解析外部输入生成内部数据结构、根据输入选择执行路径的代码称为 $PARC_3$(PArser-like Routines and Complex Control-flow Code), 并总结了这部分代码的两个特征: (1)拥有大量循环来处理输入; (2)拥有许多依赖输入的条件分支。PIE 基于这两个特征, 借助了简单的机器学习方法可以高效的发现未知固件中的 $PARC_3$。固件中的通信协议是固件与外界交流的接口, 这部分代码通常复杂但缺少文档, 而且为了提高性能、减少固件规模, 协议中许多安全措施都被省略, 通信协议中对外部输入的检查是固件安全的第一道屏障。

由于固件代码量日益增多, 基于静态分析技术对固件进行缺陷分析面临可扩展性问题。为了缓解这一问题, DTaint[41]提出了对固件进行污点分析, 可以减少分析的代码量, 提高分析效率。DTaint 不需要固件源码, 使用固件可执行文件作为输入, 将其转化为中间表示后进行分析。支持指针分析和间接调用分析, 加入了对不同架构的支持, 提高了可扩展性。但由于不同系统类型的固件差别较大, DTaint 目前只支持基于 Linux 的固件。

单纯的静态分析无法获取准确的污点传播情况, 容易造成大量的误报, 可扩展性较差。Saluki[42]通过“运行”代码片段来提高污点分析的精度, Saluki 实现了定制的解释器 μflux 来“运行”从二进制文件中提取的 IR, 而不需要运行时环境支撑, 可以获取准确的路径敏感和上下文敏感数据传播信息。Saluki 支持 ARM、x86 和 x86-64 架构的二进制可执行文件, 可以兼顾普通桌面软件和固件。Saluki 还实现了可定制化的安全准则, 使污点分析支持更多的漏洞类型。

基于 Linux 的固件往往由多个组件构成, 每个组件都是一个可执行文件, 许多分析对每个可执行文件单独进行污点分析, 容易造成大量误报。KARONTE[43]在对固件进行分析时考虑了固件不同组件之间的交互和数据流动, 大大降低了污点分析的误报。

BootStomp[44]针对手机中引导程序提出了静态分析方法, 结合基于符号执行的多标签污点分析方法, 可以发现引导程序中使用被攻击者控制的存储导致的内存损坏缺陷和引导程序解锁, 大大降低了纯静态污点分析工作的误报。

目前, 基于静态分析技术对固件配置进行分析的研究较少, 大多工作都集中于对 PC 桌面软件、移动应用软件配置的分析。如 Rabkin 等人[45]的研究工作提出了一种从 Java 代码中静态提取配置项并发现其可能选项的方法。通过考察配置读取函数的返回值、配置值传递路径来推断配置的类型、约束和取值范围, 工作的重点在于发现配置项在代码中的使用位置和别名, 没有进行数据流分析, 因此误差会较大。

Xu 等人[46]设计了一个针对 MySQL、Aphace 等商业软件的配置错误进行检测的工具 SPEX。SPEX 以软件源代码和简单注释为输入, 通过检查配置值是否符合配置约束来判断是否正确。该工作存在一定的局限。首先, SPEX 只能检测单个程序上的配置是否正确, 但现实生活中许多系统不局限于单个系统; 其次, 可以推断的约束有限。

Rabkin 等人[47]通过静态分析将程序的每一行源代码映射到可能相关的配置项, 以此帮助用户快速发现源代码错误的原因。

由于固件的代码量逐渐增多、日益复杂等特征, 只对固件进行纯静态分析往往无法取得很好的效果, 而且由于不同厂商、架构、系统的固件差别较大, 较为精确的静态分析的可扩展性也较弱。通过静态分析对配置缺陷进行检测通常是通过数据流分析来发现代码中的约束, 检查配置值是否违反约束, 这一方法对功能配置参数缺陷具有较好的检测效果, 如果可以构建较为准确的数据流分析, 则通过静态分析来发现固件的功能配置缺陷则具有一定的可行性, 但是难点在于固件的许多配置同硬件相关, 这些配置往往无法通过上下文代码去推断。

3.2 符号执行技术

符号执行技术最早源于 20 世纪 70 年代中期, 基本思想是用符号化变量代替实际输入, 驱动程序模拟执行, 收集执行路径约束集合, 调用求解器求解约束产生测试输入, 进而探索程序空间, 用于发现程序缺陷[48]。符号执行使用符号值表示具体的变量, 相比模糊测试可以更加容易通过复杂的判断, 在普通软件测试中有广泛的应用。

3.2.1 特定架构的分析

Davidson 等人[49]提出了基于符号执行引擎 KLEE 的符号执行工具 FIE, 实现了对基于 MSP430 系列微控制器的固件程序有效缺陷检测。面向小型固件程序(代码少于 100 行), FIE 使用状态修剪技术来避免路径爆炸, 使用记忆模糊技术提高符号执行的效率, 同时加入了对中断和外部输入的支持, 可以分析程序所有的路径。FIE 是符号执行方法在固件测试中较早的研究, 但其只能针对特定的架构, 且可处理代码行数较少, 误报较多。Hernandez 等人[50]基于 FIE, 通过加入对 Intel 8051 MCU 的支持, 实现了针对 USB 固件的符号执行分析框架 FirmUSB。

FirmUSB 结合静态分析和符号执行, 通过充当语义查询引擎, 检查设备固件以确定其生成潜在恶意行为的能力。此外, FirmUSB 通过将二进制固件转化为中间表示进行分析, 不需要固件源码。

3.2.2 特定缺陷的分析

和 FirmUSB 一样, 许多分析工具并不是针对违反编程规范的错误, 而是面向某一类恶意行为, 使用符号执行进行语义分析来查找符合某种模式的缺陷。典型的, Oleksandr 等人[51]提出一种基于 KLEE 的符号执行工具, 用来分析 BIOS 中 SMM 中断处理程序的危险内存引用。方法是使用 S2E 通过 SMM 中断处理程序探索执行路径, 并发现中断处理程序试图访问 SMM 代码保护区域 SMRAM 之外的内存的路径。

表 2 静态分析工具比较

Table 2 Comparison of Static Analysis Tools

检测缺陷类型	工具	优点	缺点
固件实现缺陷	PIE[40]	提出了自动发现通信协议中处理外部输入代码的方法;	发现的代码特征比较固定, 可扩展性较差。
	DTaint[41]	分析不需要源码; 只对污点数据分析提高了可扩展性; 加入了对不同内存架构的支持; 支持指针分析和非直接函数调用分析。	检测缺陷类型有限, 非污点数据缺陷无法检测; 支持固件类型有限, 只支持基于 Linux 的固件。
	Saluki[42]	提高了污点分析的效率; 提供了可扩展的安全准则; 支持多个硬件架构的代码, 可扩展性强。	发现的漏洞可能无法被触发; 对函数指针分析能力有限; 存在污染过度造成误报的问题。
	KARONTE[43]	提出了检测固件不同组件之间不安全交互的方法; 降低了污点分析的误报。	遇到路径爆炸问题, 导致产生部分误报。
	BootStomp[44]	提出了多标签污点分析方法来提高精度; 静态分析和符号执行相结合降低误报。	无法很好处理路径爆炸问题和指针问题, 导致存在误报。
软件配置缺陷	配置推断工具[45]	可推断 Java 程序中配置项可能取值; 可排除配置项中的简单错误。	对不同的配置错误类型支持有限。
	SPEX[46]	通过数据流中配置项约束发现错误; 正确率较高。	只能推断单个系统的约束。
	配置项-源代码映射工具[47]	将配置项同影响的源码进行匹配; 不具备缺陷检测功能。	只适应于 Java; 特定情况下错误率高; 对配置项有一定要求。

Yan 等人[52]提出利用符号执行技术来查找二进制固件中身份验证绕过缺陷, 并实现了分析框架 Firmalice。Firmalice 利用一种新型的身份验证旁路功能, 基于攻击者确定执行特权操作所需输入的能力, 如果攻击者只需通过分析固件就可以获得必要的输入来驱动固件执行特权操作, 则说明认证机制可以被忽略或绕过。另外, 模型允许推断复杂的后门程序。Firmalice 的符号执行部分参考了 KLEE, 并使用程序切片技术增强其可扩展性。

3.2.3 通用分析框架

为了支持更多固件代码的处理, Corteggiani 等人[53]基于 KLEE 开发了针对固件代码的符号执行框架 Inception, 此工作一定程度上消除了平台的差异。Inception 的主要目标是利用高级源代码的语义信息在符号执行期间检测缺陷, 同时支持低级汇编代码和与硬件外围设备的频繁交互。常见的符号执行环境通常运行与体系结构无关的代码表示, 这些代码可以从源代码派生而不丢失语义信息; 或者依赖于体系结构的二进制代码可以提升为中间表示, 该表示至少可以部分执行到符号虚拟机中, 但已丢失源代码语义信息。这两种情况差别很大, 不容易共存。Inception 通过创建一致的统一表示来解决共存问题。

基于符号执行的分析是现在进行固件安全缺陷检测的主要方法之一, 大部分基于 KLEE, 侧重点各有不同。现有的工具如 FIE、FirmUSB 等通常针对自己的研究目标引入了对特定硬件架构的支持, Inception 的工作则一定程度上消除了平台的差异, 极大的提高了工具的可扩展性, 且他们都加入了对外设输入和中断的支持, 采用了一定的方法来避免路径爆炸和提高符号执行效率。但现有符号执行方法大多需要源码, 只有 FirmUSB、Firmalice 等个别工具可以

直接以二进制固件作为输入, 因此未来的工作还需要研究如何以二进制固件直接作为输入进行分析并支持更多架构。具体如表 3 所示。

3.3 模糊测试技术

模糊测试技术最早出现于 1990 年, Miller 使用模糊测试来发现 UNIX 系统中的漏洞[54]。模糊测试技术通过在真实环境或者虚拟环境中运行程序, 向运行程序发送大量有效或无效的输入, 并观察程序在运行过程中的行为特征, 与典型缺陷行为特征相匹配以达到检测的目的[55]。按照固件运行的环境, 现有模糊测试技术可以分为基于真实设备以及基于模拟执行环境的模糊测试。

3.3.1 基于真实设备环境的模糊测试

由于真实硬件设备提供的调试接口有限, 因此基于真实设备的模糊测试通过观察系统运行日志或者系统输出来判断是否存在缺陷。Sara 等人[28]提出了一种运行时验证工具来检测 FreeRTOS 中的并发缺陷。他们的工具基于 Tracealyzers 所提取的 FreeRTOS 运行日志来进行分析, 查看任务的运行时间来检测其是否符合并发缺陷的特征。此工具可检测死锁(Deadlock Bug)、饥饿漏洞(Starvation Bug)和挂起漏洞(Suspension Bug)。

表 3 符号执行工具比较

Table 3 Comparison of Symbolic Execution Tools

检测缺陷类型	工具	优点	缺点
固件实现缺陷	FIE[49]	加入了对外设和中断的处理; 对固件程序进行完整的分析; 避免其中的路径爆炸问题; 提高了符号执行效率。	分析仅限于特定架构; 分析支持的代码行数较少。
	FirmUSB[50]	检测不需要源码; 可以检测 USB 中的 BadUSB 攻击。	没有对不同厂商进行定制; 仍存在绕过测试的方法; 支持的架构有限。
	BIOS SMM 中断分析工具[51]	可以分析 BIOS 中关键位置的内存错误。	应用范围较小, 可扩展性较差。
	Firmalice[52]	推断可被忽略或绕过的认证机制; 可分析固件中的后门程序; 使用程序切片来提高可扩展性。	程序的运行需要分析人员提供一定信息; 无法做到完全自动化。
	Inception[53]	可以分析高级语言源码和汇编代码的混编, 可扩展性较高; 可以处理不同内存架构; 支持外设交互和中断。	对低级别的 IR 生成的 bitcode 的检测水平还有待提升; 还需支持更多的二进制码和 C/C++; 还需提高自动化。

RPFuzzer[56]是一个用于检测路由协议缺陷的框架, 通过向真实设备发送大量数据包, 监控 CPU 使用与查看系统日志, 进而检测设备重启与拒绝服务漏洞。RPFuzzer 的测试用例生成分为两个阶段, 第一阶段基于手动分析和自动生成; 第二阶段基于第一阶段的测试用例和历史漏洞数据进行变异。RPFuzzer 可以针对路由器支持的多种协议进行测试。

IoTFuzzer[57]发现 IoT 终端设备往往伴随着一个与之交互的移动应用程序, 于是基于移动应用程序与真实终端设备开发了一个检测固件程序中内存损坏缺陷的黑盒模糊测试工具。可以通过对移动应用程序进行分析来推导未知协议中需要变异的字段, 生成有效的模糊测试用例, 并且克服了网络通信中的加密问题, IoTFuzzer 使用心跳机制检测设备是否发生异常。

Li 等人[58]针对 RTSP 协议的缺陷挖掘提出了利用 RTSP 协议状态间的约束关系和转移关系构造协议状态图, 基于状态图来消除测试用例中的冗余。

Yuan 等人[59]提出一个不需要源代码的在线黑盒测试工具 CODE 来对使用最为频繁的配置进行缺陷检测。CODE 通过对运行时的应用程序进行观测, 提取重复的、可预测的事件序列形成配置的访问规则, 当访问配置时违反了规则, 则表明当前配置可能发生错误。

Su 和 Attariyan 等人[60]通过在 Linux 内核级对与配置相关的动作的输入和输出进行跟踪, 推断出配置错误发生的因果关系, 并实现了工具 AutoBash。AutoBash 将配置活动分割为可以改变系统状态的动作, 通过断言来测试系统正确性。

Attariyan 等人[61]通过对二进制文件进行插桩, 在程序运行时通过控制流和数据流获取依赖关系, 通过依赖关系将异常行为定位到特定的配置项中,

实现了名为 ConfAid 的动态配置缺陷检测工具。这一工作是 AutoBash 的延申。

Zhang 等人[62]提出了一个静态分析和动态分析相结合来诊断单个配置缺陷的工具 ConfDiagnoser。ConfDiagnoser 以 Java 程序和配置项作为输入, 收集程序运行时特征, 并在数据库中进行匹配以发现缺陷。ConfDiagnoser 存在四个局限性: 工具只关注 key-value 形式的配置; 对配置缺陷的诊断仅限于一个配置项的缺陷; 不支持非确定性的缺陷; 有效性取决于数据库中存在相似且正确的程序执行。

3.3.2 基于模拟执行环境的模糊测试

Zaddach 等人[63]提出了一个框架 Avatar, 通过将仿真器的执行与实际硬件协调在一起, 实现了对嵌入式设备的复杂动态分析。Avatar 充当物理设备和模拟器之间的协调引擎, 在模拟器内部执行固件指令, 同时将输入输出操作引导到物理硬件, 利用真正的硬件来处理输入输出操作。Avatar 弥补了模拟器对外设模拟能力不足的局限, 但是仍离不开具体的硬件。Avatar 的后继版本 Avatar2[64]则是第一个面向多目标编程的通用框架, 可以用来协调连接不同的二进制分析框架、调试器、仿真器和实际物理硬件, 使不同工具之间实现互操作性。

Chen 等人[65]实现了第一个对基于 Linux 的商用物联网设备固件在模拟器中进行仿真、自动化分析的系统 FIRMADYNE, 此工作可以仿真 Linux 系统, 并完全脱离硬件进行仿真。FIRMADYNE 使用 binwalk 进行解包, 并在 QEMU 中仿真运行。在 FIRMADYNE 上可以方便地扩展各种动态分析方法。FIRMADYNE 在很大程度上实现了脱离硬件的仿真, 但是受限于 QEMU 的能力有限, 可以成功运行的固件很少, 且只能针对基于 Linux 的固件。Firm-AFL[66]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, 这是第一个针对固件的灰盒模糊测试系统, 而且提高了固件模拟执行的效率。

Avatar 和 FIRMADYNE 是当前研究较多的两个框架, 但如果脱离硬件, 都无法很好处理外设输入, 这成为在模拟环境中测试的一个难点, P2IM[67]基于外设接口抽象建模设计了一个摆脱硬件依赖的 MCU 固件模糊测试框架, 解决了模糊测试对真实硬件的依赖和模拟器对外设模拟能力有限的问题。但是当 P2IM 遇到固件通过直接内存访问(Direct Memory Access, DMA)来获取外设输入时, 通常无法识别并返回无效的值, 导致固件运行崩溃。后续的工作 DICE[68]通过识别固件中的 DMA 输入输出通道并动态创建输入缓冲区来解决这一问题, 与 P2IM 相比大大提高了测试的代码覆盖率。Laelaps[69]使用符号执行来生成外设输入, 摆脱了模拟器对外设处理能力有限的问题。Laelaps 针对 ARM Cortex-M 设备, 将 Microcontroller 固件运行在 QEMU 中, 使用符号执行来生成满足要求的外设输入。

P2IM、DICE 和 Lealaps 基于对固件的分析来生成外设输入, 一定程度上解决了模拟器对外设模拟能力有限的问题, 但支持的外设类型和输入输出类型有限。HALucinator[70]通过替换固件硬件抽象层(Hardware Abstraction Layers, HAL)使其不依赖于具体硬件, 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。

戴忠华等人[71]提出一种基于静态分析和动态调试的固件缺陷分析利用框架。使用 QEMU 搭建环境, 首先利用静态污点分析绘制污点数据传播图, 从而辅助动态调试, 快速定位缺陷。最后使用 sulley 工具进行模糊测试。这一工作对人工的依赖较大, 仍无法完全自动化。传统的模糊测试通常会生成大量的测试用例, 但是测试用例之间经常会发生冗余, 降低测试效率。

Xu 等人[72]提出了一种在软件发布之前通过模拟运行来检测潜在配置缺陷的方法, 并实现了配置缺陷检测软件 PCHECK。PCHECK 以 IR 作为输入, 并以配置接口规范和系统初始化阶段的注释作为辅助输入用来识别配置项。

模糊测试方法比较如表 4 所示, 可以发现现有的模糊测试工具大多是通过观测系统崩溃来检测是否存在缺陷。然而, Muench 等人[73]通过实验发现, 不同的固件针对内存相关缺陷的表现并不相同。许多固件在遇到内存相关缺陷时系统没有任何明显表现, 可以继续运行, 这对缺陷检测提出了新的挑战。作者提出对内存的使用进行监测来判断是否存在缺陷, 但是这一方法目前只能在模拟器中进行。此外, 现实中大量其他实现缺陷的表现也不相同。基于实际执行环境的模糊测试由于硬件资源有限、调试接口不完备, 可以获取的固件运行信息较少, 可以检测到的缺陷有限。基于模拟执行环境的方法可以方便地观察固件运行时的状态、构造极端运行环境、提高测试的代码覆盖率, 但模拟执行仍受限于模拟器的能力有限, 现有的模拟器只能运行部分固件, 可扩展性较差。尤其是对于中断和外部输入的处理能力有限, 未来的工作可以用其他方法来弥补模拟器的不足, 如使用符号执行根据路径约束来推断外设输入、通过模糊测试来随机生成外设输入、通过一定

的反馈机制来引导生成外设输入等。现有的针对配置缺陷的检测方法则是传统的动态测试, 这些方法在固件上实施具有理论可行性, 难点仍在于固件的有效执行和执行时信息的获取。

表 4　模糊测试工具比较

Table 4　Comparison of Dynamic Testing Tools

检测缺陷类型	工具	优点	缺点
固件实现缺陷	并发缺陷运行时验证工具[28]	可以检测并发缺陷; 检测来自于实际运行, 真实有效。	在真实环境中运行时一些极端情况很难构造。
	RPFuzzer[56]	测试可以针对多种协议, 可扩展性高; 测试用例有效性高。	测试需要一定的手动分析, 自动化程度受限。
	IOTFuzzer[57]	可以准确定位协议需要变异的字段; 可以克服通信协议的加密问题。	获取的固件运行时信息有限; 无法定位到缺陷的具体位置。
	基于协议状态图消除测试用例冗余[58]	提高了模糊测试效率。	针对复杂协议的测试效率低下, 还需对协议状态图优化。
	Avatar[63]、Avatar2[64]	协调不同测试工具, 取长补短; 可扩展性强。	本身不具有测试功能。
	Firmadyne[65]	实现了对基于 Linux 的固件的完全仿真运行; 提供了检测方法的扩展接口。	自身测试功能较弱; 只支持基于 Linux 的固件。
	Firm-AFL[66]	提高了全系统仿真的效率; 第一个固件灰盒测试工具。	受限于模拟器; 可运行固件有限。
	P2IM[67]	对外设进行了抽象, 摆脱了硬件依赖。	没有对 DMA 进行建模; 支持架构有限。
	DICE[68]	对 DMA 进行了建模; 可扩展性强, 可以方便地移植到其他分析工具上。	对 DMA 通道的识别存在一定局限; 对 DMA 缓冲区大小的识别不是非常准确。
	Laelaps[69]	使用符号执行生成外设输入, 摆脱了硬件依赖。	仅支持 ARM Cortex-M 的固件; 仅支持非 Linux 固件。
	HALucinator[70]	使固件同底层硬件解耦, 摆脱了模拟环境对硬件模拟能力有限的问题。	仅限于使用了 HAL 的固件; HAL 函数匹配方法仍存在局限性。
	动态分析与静态分析相结合方法[71]	结合静态分析提高了分析的效率。	对人工的依赖较大, 自动化程度低。
软件配置缺陷	CODE[59]	对比真实执行状态判断缺陷真实有效。	只能在 windows 中使用, 无法扩展到其他系统中。
	AutoBash[60]	记录用户对配置的修改, 可回滚到正确状态, 并提供正确的修改方法。	程序运行的前提是必须存在正确的配置。
	ConfAid[61]	可以对发生异常的配置项进行定位。	只能诊断一个配置项引起的异常, 但通常配置项之间有相互关系。
	ConfDiagnoser[62]	缺陷检测误报率低。	程序运行需要构建错误数据库。
	PCHECK[72]	运行不会对系统产生破坏; 检测精度较高。	无法检测需要外部上下文的程序。

3.4　程序验证技术

程序验证技术是指以数学和逻辑为基础, 对系统进行说明、设计和验证, 通过形式规约来描述系统的行为或者系统应该满足的性质, 采用形式化验证来验证系统是否满足需求和具备这些性质, 即是否满足规约[74]。

Prakash Chandrasekaran 等人[29]提出了一种在多核处理器上运行 FreeRTOS 的调度方案, 并通过 Spin 建模语言进行建模验证了他们的方案可以避免数据竞争和死锁。这一工作为固件程序的并发问题缺陷检测提供了解决方法。

Xie[75]以反编译技术为基础, 通过 Kripke 结构描述模型进行建模来对固件进行模型检验。Zhang 等人[76]提出了基于行为时序逻辑(temporal logical of actions, TLA)的软硬件协同验证固件缺陷分析技术, 将硬件工作流程和软件对硬件的调用相结合分析来发现计算机开机过程中存在的缺陷。

Huang 等人[77]提出了一种对大型云系统进行配

置验证的框架 ConfValley。其核心是一种简单的声明性的配置验证语言 CPL, 该语言将核心验证逻辑与实现细节分离, 允许紧凑地描述规范并独立于基础配置表示, 好处是验证代码变得可维护、模块化、可并行化, 并且适用于各种配置源。

现有对固件程序进行形式化验证的工作较少, 但是已有的工作已经证明了形式化验证在保证固件安全方面的有效性。FreeRTOS 作为轻量级的嵌入式操作系统, 为固件提供调度是其基本功能之一, 使用 Spin 建模语言对其进行程序验证为使用建模语言对固件程序的多线程调度进行程序验证提供了可能; 大型云系统中的配置主要用来设置云系统运行环境和协调各个组件, 这点和固件具有一定的相似之处, 固件中的配置也主要用来设置底层硬件相关信息、协调外设的使用, 因此 ConfValley 对固件配置的验证有一定的借鉴作用。

3.5 基于机器学习的方法

现有的基于机器学习的方法大多通过静态分析或者动态执行来提取程序特征, 使用一些机器学习算法来学习已有的缺陷特征, 并在程序中查找已知缺陷[78]。现有的基于机器学习的固件缺陷检测方法以目标匹配对象划分可以分为三类, 分别为上下文无关的函数匹配、上下文敏感的函数匹配和二进制固件文件匹配。

3.5.1 上下文无关的函数匹配

Feng 等人[79]提出一个基于控制流程图(CFG)的缺陷搜索引擎 Genius。作者提出将 CFG 转换为高级数字特征向量, 降低了 CFG 匹配的开销, 解决了现有跨平台缺陷搜索技术的可扩展性问题, 进一步提高了搜索的准确性。与现有的缺陷搜索方法相比, Genius 具有以下两个好处: 首先, 所学习的特征对于跨体系结构的变化趋向于比原始 CFG 特征变化更小; 其次, Genius 显著提高了缺陷搜索效率。

在之后的研究中 Feng 等人[80]还提出了一种基于从原始二进制码中提取的条件公式作为高级语义特征进行代码搜索的方法, 并根据此实现了工具 XMATCH。使用条件公式可以明确地捕获两种缺陷: 1)错误的数据依赖; 2)缺失或无效的条件检查。使用条件公式进行代码搜索具有两个优点: 1)显著提高了搜索的准确性, 在二进制代码级别上消除了特定平台的差异; 2)为分析人员提供可以解释的证据, 以审查搜索结果并确定易受攻击的功能。除了根据二进制代码中不同的上下文语义来搜索安全缺陷外, 还进一步使分析人员能够通过生成的可解释诊断报告来检查搜索结果, 以便确定目标程序中显示的缺陷代码。

Xu 等人[81]对神经网络进行了修改, 使其可以将从固件中提取的属性控制流图(attributed control flow graph, ACFG)转化为数字特征向量-嵌入向量(embedding vectors), 这一方法大大缩减了嵌入向量的生成时间和模型的训练时间。作者基于这一方法构建了工具 Gemini。

Gao 等人[82]提出了基于语义学习的代码相似性计算工具 VulSeeker, 作者从固件中提取标记了的语义流图(Labeled Semantic Flow Graph, LSFG), 该图同时包含数据流图和控制流图, 将图中的边标记为 0 和 1 分别来表示控制流和数据流, 作者使用从 LSFG 中提取的基本块特征作为数字向量, 以此数字向量为输入通过语义感知的 DNN 模型计算, 得出函数的嵌入向量, 通过计算两个函数的嵌入向量的余弦距离来计算相似性。与 Gemini 相比, VulSeeker 在 top-10 和 top-50 的相似结果中分别多发现了 50%和 13.89%的缺陷。之后作者扩展形成新的工作 VulSeeker-Pro[83], 通过仿真执行候选函数来对获得语义签名表示的动态执行轨迹, 计算候选函数同缺陷函数的 Jaccard 相似度来对候选函数重新排序, 提高识别的精度。

常青等人[84]针对现有跨平台缺陷检测方法准确率低的问题, 提出基于神经网络和局部调用结构匹配的 2 阶段跨平台固件缺陷关联方法。以函数为最小关联单元, 对函数调用图、函数内控制流图、函数基本信息进行特征选择和数值化处理, 并采用神经网络计算待匹配函数对的相似程度, 在此基础上采用结构化匹配方法进一步提高匹配精度。

3.5.2 上下文敏感的函数匹配

前面的工作大多针对固件中的单个函数来进行匹配, 但往往缺陷会跨多个函数出现, 这种情况下上述工作将难以应对。David 等人[85]提出了基于 Angr 的静态分析工具 FirmUP, 用于程序在过程间层面进行缺陷相似性匹配。作者的方法是将其他的过程作为程序执行的环境, 将过程分解为比基本块更小的 Strands-based representation, 将其中的寄存器名和地址偏移一致化, 以函数为单位制成一张表, 以表中相同的代码片段的数量作为相似度来进行匹配。匹配使用了 back-and-forth games 算法。

3.5.3 二进制固件文件匹配

Andrei 等人[86]的工作收集了大量的固件程序, 设计了分布式的架构对其进行解包和简单的静态分析, 并实现了一个引擎来比较和确定数据集中所有对象之间的相似性, 可以快速地将已知易受攻击的设备的缺陷“传播”到以前不知道受到相同缺陷影响的其他系统。

Chen 等人[87]发现固件代码在不同编译环境、优化选项下编译出来的二进制码并不完全相同, 但代码中通常会包含一些具有“编译不变性”的字符串, 如调试字符串、标语字符串等, 如果两个文件中的可读字符串内容和顺序基本一致, 则很大可能这两个文件是同源的。作者利用深度学习来编码可读字符串, 对编码字符串生成局部敏感 Hash 从而实现快速检索, 将搜索的时间复杂度将为了 $O(\lg N)$, 大大缩短了搜索时间。

同一厂商生产的固件通常会包含一些相同的文件, 容易受到同一缺陷的干扰。Zou 等人[88]通过向物联网设备发送报文来获取物联网设备返回的协议标语信息, 用自然语言处理方法来进行处理, 最后获取设备分类, 以发现是否同有缺陷的设备为同一厂商。

3.5.4 基于模板的配置缺陷检测

Zhang 等人[89]根据配置项与执行环境之间的相互作用和配置项之间的相关性来检测配置缺陷, 从一组给定的配置中利用数据挖掘技术学习配置规则, 以此规则来检测其他配置的正确性, 并实现了自动化测试工具 EnCore。

基于机器学习的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。目前的机器学习方法大多是以静态分析方法的延伸出现, 只有文献[86]中的工作是直接对固件解包出来的文件进行机器学习匹配, 文献[88]利用物联网设备的输出信息对其分类, 其他的方法建立的模型必须基于静态分析得出的结果, 因此效果的好坏很大程度上依赖于程序特征的提取和相应的机器学习模型的选择。基于机器学习的方法与静态分析方法的区别在于侧重点不同, 基于机器学习的方法使用简单的静态分析方法得出程序特征, 使用的静态分析方法不足以独立发现其中的缺陷。使用的机器学习模型需要学习过程和判断过程, 如何更好的抽取程序特征和如何对固件程序选取有效的特征和选择高效的匹配方法是这类方法的主要研究问题。现有的方法使用了不同的精度, 文献[86]精度最差, 但是胜在数据集够多; 上下文无关的匹配方法使用了不同的程序特征和学习模型, 致力于提高匹配的精度, 且 VulSeeker-Pro[83]还使用了额外的方法来提高精度, 但是都无法匹配跨函数缺陷; FirmUP[85]使用了上下文敏感的方法, 进一步提上了匹配的精度。这些方法所使用的静态分析方法忽略了静态分析在固件上进行的难点, 丢失了一部分语义, 受到的影响大小根据方法所选择的程序特征的不同而不同, 所有这些方法都需要大量的人工确认。针对这类方法的总结如表 5 所示。

表 5 基于机器学习的方法比较

Table 5 Comparisons of Analysis Methods Based on Machine Learning

检测缺陷类型	工具	优点	缺点
固件单个过程内缺陷	Genius[79]	消除了平台差异; 工具具有一定鲁棒性。	受限于静态分析的精度; 无法处理用于避免相似性检测的混淆代码。
	XMATCH[80]	提高了搜索的准确性; 消除了不同平台的差异。	对循环的处理较弱, 影响精度; 无法处理跨多个函数的缺陷。
	Gemini[81]	相比之前工作极大缩短了模型训练时间和特征提取处理时间。	性能随代码复杂度的提高下降较快; 精度不够高。
	VulSeeker[82]、VulSeeker-Pro[83]	搜索速度快, 精度高; 可以消除不同平台的差异。	且无法处理跨函数缺陷。
	基于神经网络和局部调用结构匹配的 2 阶段跨平台固件缺陷关联方法[84]	不依赖指令集, 有一定跨平台能力; 具有较高精度。	无法检测跨函数缺陷。
固件跨过程缺陷	FirmUP[85]	消除了平台的差异性; 有较高精度; 考虑到过程所处上下文。	需要人工参与校验。
	大量程序比较方法[86]	搜索范围广。	工作量大, 不适合针对某一特定固件使用。
固件文件中的缺陷	编译不变性字符串匹配[87]	大大提升了搜索速度。	依赖编译不变性字符串, 但是这些字符串并不承担核心功能, 在代码中较少。
	物联网设备厂商型号识别[88]	较为准确得识别了已有物联网设备的厂商和型号。	匹配力度粗, 精度较低。
软件配置缺陷	EnCore[89]	静态提取配置项同环境之间的依赖关系和配置项之间的依赖关系, 形成模板, 并对这些模板进行学习。	无法检测跨组件的配置错误。

3.6 缺陷检测辅助工作

由于固件在代码结构、运行环境、功能等方面同普通软件存在较大差别, 固件的测试目前面临了第 1 章所述的种种困难, 国内也有大量学者针对这些挑战提出了部分解决方案。

面向无法获取源码这一问题, 解放军信息工程大学李清宝团队在固件代码反编译领域取得了大量成果。文献[90]讨论了固件代码逆向中的指令归一化、控制流恢复、中断向量表等关键问题, 设计了逆向平台 amPro; 文献[91]针对固件严重依赖中断提出了基于中断向量表重构的固件代码反汇编技术, 大大提高了反汇编的精度; 文献[92]提出了动静态两个方面的固件代码控制流恢复算法, 提高了固件控制流恢复的全面性, 为后续的分析提供了有力的支持; 文献[93]提出快速位运算方法和区间生成算法,提高了在固件反汇编中计算字节运算和位运算取值范围的效率; 文献[94]则基于反汇编技术提出了对固件代码的形式化验证方法和多路径固件恶意行为检测方法。

面向固件解码的判定问题, 中科院信息工程研究所孙利民团队提出了基于分类回归树的固件解码状态检测算法[95], 可以自动化分析大量固件解码状态; 面向固件系统、结构、支持硬件不同等问题, 提出了基于获取规则的方法来自动化发现和标注 IoT 设备的类型、供应商和型号[96], 和基于自然语言处理来分析网页内容识别设备指纹的方法[97], 方便其他学者获取固件详细信息; 还提出了基于自然语言处理分析网络上的缺陷报告来理解物联网设备被攻击的原因, 并协助抵抗攻击[98]。文献[99]对固件获取、固件格式分析、固件程序提取、目标程序分析提取、程序表示技术、执行信息恢复技术等进行了综述。

4 未来研究方向

随着嵌入式设备数量呈爆炸式增长, 学术界和工业界逐渐将更多的精力投入到嵌入式设备的安全中来。固件作为嵌入式设备的软件系统, 其安全性对嵌入式设备的重要不言而喻, 虽然目前对固件安全的研究在不断地增多, 但在许多方面仍有欠缺。其中, 静态分析技术本身具有一些缺陷, 如可扩展性差、误报较高等, 在分析中无法应对代码行数较多的项目, 但是随着边缘计算的兴起, 固件中的代码行数会成倍增长, 静态分析的应用将会有些困难, 许多固件存在混淆机制, 给反编译工作造成了一定的阻碍; 目前的符号执行技术已可以直接使用二进制可执行文件作为输入, 但是仍需对不同的架构进行定制, 可扩展性仍受到阻碍; 模糊测试可以分为在真实硬件上运行和在模拟器中运行, 在真实硬件上执行要面对的问题是真实硬件上的资源有限, 无法获取足够多固件运行时信息, 在模拟器中运行要面临的问题是目前的模拟器还无法完美的模拟硬件的真实状态, 外设的输入无法正确给出, 因此无法很好的对固件进行测试; 程序验证在软件安全中起着至关重要的作用, 重大的项目只有通过了程序验证才可发布, 但是程序验证较为繁琐, 可处理的代码量较少, 模型在面对异构的固件时可扩展性较差; 基于机器学习的检测技术近几年来在安全领域一直是一个热点, 这一方法不受软件平台、架构的限制, 难点在于如何为代码进行建模、提取代码特征、如何选择机器学习算法, 在普通软件安全领域成熟的检测手段在固件测试中应该也可以使用, 但基于机器学习的测试方法误差会较高。

5G 技术的发展将会使得联网设备数量井喷式发展, 嵌入式设备的安全问题亟待解决, 本文对未来的可以进一步探索的研究方向进行了总结。

4.1 面向固件缺陷检测评估的海量缺陷固件测试基准构建

由于固件种类丰富, 其运行的平台、架构等千变万化, 目前仍没有工具可以对多个种类的固件进行测试, 通常是针对某一类型或者某一架构的固件进行测试, 因此在测试中学者会自己构建方便自己实验使用的测试集, 到目前为止仍没有一个通用的测试基准可供大家使用。未来可以构造这样一个测试基准, 包含不同架构、不同类型的固件, 且固件中可植入不同类型的缺陷。测试基准的构建将有助于学术人员的实验, 也有助于不同工具的比较。

4.2 摆脱设备依赖的固件仿真运行环境构建

随着 QEMU 性能的不断提高、对不同固件的适配性不断增强, 在虚拟机中对固件进行模糊测试变得可行, 但是当前的模拟执行环境仍存在一定的局限性, 如对外设的模拟能力有限, 对硬件架构类型的支持有限等。虽然当前已有方法来缓解这一局限, 但仍不足以解决商用设备中外设种类多、硬件架构复杂等导致无法在仿真环境中运行的难题。未来研究可以着眼于固件仿真能力的提高, 覆盖更多的外设类型和硬件架构, 提高对商用物联网设备固件的仿真能力。

4.3 数据驱动的固件漏洞预测与目标制导模糊测试技术

随着代码规模的不断攀升, 固件类型与应用场景的不断多样化, 以及漏洞类型与特征的不断复杂

化, 传统依赖于人工机理分析确定漏洞检测规则的静态分析方法在实际的应用扩展性方面面临挑战。随着大数据技术、人工智能技术的不断进步, 如何有效利用智能化技术, 通过挖掘已有海量代码中的漏洞信息、结构特征、语义特征、代码修复、更新信息, 形成智能化的漏洞预测模型, 实现自动化生成漏洞检测规则, 进而高效地预测潜在漏洞, 并为后续目标制导的模糊测试提供制导目标是未来值得研究的方向之一。

4.4 面向配置缺陷检测的测试用例生成技术

近年来配置缺陷的不断发生引起了许多学者的注意, 固件中的配置缺陷同样不容忽视。针对普通软件领域的配置缺陷使用人工智能技术辅助识别取得了很好的效果, 但目前的研究仅停留在检测一些浅显的缺陷上, 如拼写错误、类型错误等。进行合理配置同固件的应用环境有着强烈的联系, 但现有的工作对外界环境的考虑较少, 因此还需将固件的应用环境进行建模分析; 其次使用机器学习对配置缺陷进行检测需要大量的学习数据, 但通常针对某一特定领域或者将固件部署在新的平台上时无法获取大量数据用于训练。未来的研究可以将机器学习技术应用于辅助静态分析和动态分析, 对分析过程中的一些程序特征进行特征提取, 运用机器学习技术进行进一步分析。

4.5 面向定制化缺陷的分析与检测框架

定制化缺陷来自于用户, 是用户在使用过程中对固件的不当定制造成的, 用户在使用过程中无法获取对固件代码的完全理解, 这一缺陷的检测与修复缺乏固件开发人员的支持, 因此如果可以有工具来辅助用户对定制化的固件代码进行缺陷检测, 则可以大大提高固件的可扩展性。定制化缺陷的检测无需对整个固件程序进行重新测试, 只需对定制化部分测试即可, 但目前尚未出现这一方向的研究工作, 未来可在这一方向做出拓展。

5 总结

本文针对物联网设备固件中潜在的安全缺陷, 首先概述了典型的固件缺陷类型以及产生机理, 包括内存损坏、命令注入、程序逻辑、并发等实现缺陷和配置缺陷以及定制缺陷。接着, 从静态分析、符号执行、模糊测试、程序验证、基于机器学习五个技术角度对现有固件缺陷检测方法进行了分析与比较。然后, 针对研究现状存在的不足, 提出了未来可以探索的研究方向, 主要包括面向固件缺陷检测评估的海量缺陷固件测试基准构建、摆脱设备依赖的固件仿真运行环境构建、数据驱动的固件漏洞预测与目标制导模糊测试技术、面向配置缺陷检测的测试用例生成技术和面向定制化缺陷的分析与检测框架。

参考文献

[1] Yaqoob I, Hashem I A T, Ahmed A, et al. Internet of Things Forensics: Recent Advances, Taxonomy, Requirements, and Open Challenges[J]. *Future Generation Computer Systems*, 2019, 92: 265-275.

[2] 5G. https://en.wikipedia.org/wiki/5G. Jun. 2020.

[3] NB-IoT. https://en.wikipedia.org/wiki/Narrowband_IoT. Jun. 2020.

[4] GSMA. https://www.gsma.com/. Jun. 2020.

[5] 12207-2008 - ISO/IEC/IEEE International Standard - Systems and software engineering-Software life cycle processes. https:// ieeexplore. zieee.org/document/4475826. Jun. 2020.

[6] Real-time operating system. https://en.wikipedia.org/wiki/Real-time_operating_system. Jun. 2020.

[7] VxWorks. https://www.windriver.com/products/vxworks/. Jun. 2020.

[8] QNX. https://blackberry.qnx.com/en/. Jun. 2020.

[9] FreeRTOS. https://www.freertos.org/. Jun. 2020.

[10] RTEMS. https://www.rtems.org/. Jun. 2020.

[11] VPNFilter. https://en.wikipedia.org/wiki/VPNFilter. Jun. 2020.

[12] Read Dyn's Statement on the 10/21/2016 DNS DDoS Attack | Dyn Blog. https://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/. Jun. 2020.

[13] Zhao K, Ge L N. A Survey on the Internet of Things Security[C]. *2013 Ninth International Conference on Computational Intelligence and Security*, 2013: 663-667.

[14] Jing Q, Vasilakos A V, Wan J F, et al. Security of the Internet of Things: Perspectives and Challenges[J]. *Wireless Networks*, 2014, 20(8): 2481-2501.

[15] Xu J, Ning P, Kil C, et al. Automatic Diagnosis and Response to Memory Corruption Vulnerabilities[C]. *The 12th ACM conference on Computer and communications security - CCS '05*, 2005: 223–234.

[16] CVE-2017-12754. https://github.com/coincoin7/Wireless-Router-Vulnerability/blob/master/Asus_DeleteOfflineClientOverflow.txt. Jun. 2020.

[17] FreeRTOS TCP/IP Stack Vulnerabilities – The Details. https://blog.zimperium.com/freertos-tcpip-stack-vulnerabilities-details/. Jun. 2020.

[18] CVE-2006-6125. https://www.kb.cert.org/vuls/id/403152/. Jun. 2020.

[19] Su Z D, Wassermann G. The Essence of Command Injection Attacks in Web Applications[J]. *ACM SIGPLAN Notices*, 2006, 41(1): 372-382.

[20] CVE-2019-7298. https://github.com/leonW7/D-Link/blob/master/Vul_2.md. Jun. 2020.

[21] CVE-2018-18441. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-18441. Jun. 2020.

[22] Felmetsger V, Cavedon L, Kruegel C, et al. Toward Automated Detection of Logic Vulnerabilities in Web Applications[C]*USENIX Security Symposium*. 2010: 143-160.

[23] CVE-2017-7923. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7923. Jun. 2020.

[24] CVE-2006-4143. https://cxsecurity.com/issue/WLB-2006080097. Jun. 2020.

[25] CVE-2006-1003. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1003. Jun. 2020.

[26] Liu C, Zou D, Luo P, et al. A Heuristic Framework to Detect Concurrency Vulnerabilities[C]. *ACSAC*. 2018: 529-541.

[27] CVE-2018-4027. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-4027. Jun. 2020.

[28] Asadollah S A, Sundmark D, Eldh S, et al. A Runtime Verification Tool for Detecting Concurrency Bugs in FreeRTOS Embedded Software[C]. *The 17th International Symposium on Parallel and Distributed Computing*, 2018: 172-179.

[29] Chandrasekaran P, Shibu Kumar K B, Minz R L, et al. A Multi-Core Version of FreeRTOS Verified for Datarace and Deadlock Freedom[C]. *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign*, 2014: 62-71.

[30] Yin Z N, Ma X, Zheng J, et al. An Empirical Study on Configuration Errors in Commercial and Open Source Systems[C]. *The Twenty-Third ACM Symposium on Operating Systems Principles - SOSP '11*, 2011: 159-172.

[31] CVE-2018-19990. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-19990. Jun. 2020.

[32] CVE-2019-10132. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-10132. Jun. 2020.

[33] CVE-2019-2041. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-2041. Jun. 2020.

[34] CVE-2017-7916. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7916. Jun. 2020.

[35] CVE-2019-9976. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9976. Jun. 2020.

[36] Chen W, Huang X, Qiao X Q, et al. Research on Software Misconfiguration Troubleshooting[J]. *Journal of Software*, 2015, 26(6): 1285-1305.
(陈伟, 黄翔, 乔晓强, 等. 软件配置错误诊断与修复技术研究[J]. *软件学报*, 2015, 26(6): 1285-1305.)

[37] Xu T Y, Zhou Y Y. Systems Approaches to Tackling Configuration Errors: A Survey[J]. *ACM Computing Surveys*, 2015, 47(4): 70.

[38] Li F L, Yang J H, Wu J P, et al. Research on Internet Automatic Configuration[J]. *Journal of Software*, 2014, 25(1): 118-134.
(李福亮, 杨家海, 吴建平, 等. 互联网自动配置研究[J]. *软件学报*, 2014, 25(1): 118-134.)

[39] Chess B, McGraw G. Static Analysis for Security[J]. *IEEE Security & Privacy*, 2004, 2(6): 76-79.

[40] Cojocar L, Zaddach J, Verdult R, et al. PIE: Parser Identification in Embedded Systems[C]. *The 31st Annual Computer Security Applications Conference on - ACSAC 2015*, 2015: 251-260.

[41] Cheng K, Li Q, Wang L, et al. DTaint: Detecting the Taint-Style Vulnerability in Embedded Device Firmware[C]. *The 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2018: 430-441.

[42] Gotovchits I, Van Tonder R, Brumley D. Saluki: finding taint-style vulnerabilities with static property checking[C]. *The NDSS Workshop on Binary Analysis Research*. 2018.

[43] 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.

[44] Redini N, Machiry A, Das D, et al. Bootstomp: on the security of bootloaders in mobile devices[C]. *26th {USENIX} Security Symposium*. 2017: 781-798.

[45] Rabkin A, Katz R. Static Extraction of Program Configuration Options[C]. *The 33rd international conference on Software engineering - ICSE '11*, 2011: 131-140.

[46] Xu T Y, Zhang J Q, Huang P, et al. Do not Blame Users for Misconfigurations[C]. *The Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013: 244-259.

[47] Rabkin A, Katz R. Precomputing Possible Configuration Error Diagnoses[C]. *The 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011: 193-202.

[48] Baldoni R, Coppa E, D'elia D C, et al. A Survey of Symbolic Execution Techniques[J]. *ACM Computing Surveys*, 2018, 51(3): 50.

[49] Davidson D, Moench B, Ristenpart T, et al. {FIE} on firmware: Finding vulnerabilities in embedded systems using symbolic execution[C]. *22nd {USENIX} Security Symposium*. 2013: 463-478.

[50] Hernandez G, Fowze F, Tian D J, et al. FirmUSB: Vetting USB Device Firmware Using Domain Informed Symbolic Execution[C]. *The 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 2245-2262.

[51] Bazhaniuk O, Loucaides J, Rosenbaum L, et al. Symbolic Execution for {BIOS} Security[C]. *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*. 2015: 8.

[52] Shoshitaishvili Y, Wang R, Hauser C, et al. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware[C]. *NDSS*. 2015.

[53] 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.

[54] 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.

[55] Li J, Zhao B D, Zhang C. Fuzzing: A Survey[J]. *Cybersecurity*, 2018, 1(1): 1-13.

[56] Wang Z, Zhang Y, Liu Q. RPFuzzer: A Framework for Discovering Router Protocols Vulnerabilities Based on Fuzzing[J]. *KSII Transactions on Internet & Information Systems*, 2013, 7(8).

[57] Chen J, Diao W, Zhao Q, et al. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing[C]. *NDSS*. 2018.

[58] Li J L, Chen Y L, Li Z, et al. Mining RTSP Protocol Vulnerabilities Based on Traversal of Protocol State Graph[J]. *Computer Science*, 2018, 45(9): 171-176.
(李佳莉, 陈永乐, 李志, 等. 基于协议状态图遍历的 RTSP 协议漏洞挖掘[J]. *计算机科学*, 2018, 45(9): 171-176.)

[59] Yuan D, Xie Y, Panigrahy R, et al. Context-based online configuration-error detection[C]. *The 2011 USENIX conference on USENIX annual technical conference*. USENIX Association, 2011: 28.

[60] Su Y Y, Attariyan M, Flinn J. AutoBash[J]. *ACM SIGOPS Operating Systems Review*, 2007, 41(6): 237-250.

[61] Attariyan M, Flinn J. Automating Configuration Troubleshooting with Dynamic Information Flow Analysis[C]. *OSDI'10: The 9th USENIX conference on Operating systems design and implementation*. 2010: 237-250.

[62] Zhang S, Ernst M D. Automated Diagnosis of Software Configuration Errors[C]. The 35th International Conference on Software Engineering, 2013: 312-321.

[63] Zaddach J, Bruno L, Francillon A, et al. AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares[C]. *NDSS*. 2014, 14: 1-16.

[64] Muench M, Nisi D, Francillon A, et al. Avatar2: A multi-target orchestration platform[C]. *Proc. Workshop Binary Anal. Res.*. 2018, 18: 1-11.

[65] Chen D D, Woo M, Brumley D, et al. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware[C]. *NDSS*. 2016, 16: 1-16.

[66] Zheng Y, Davanian A, Yin H, et al. FIRM-AFL: high-throughput greybox fuzzing of iot firmware via augmented process emulation[C]. *28th {USENIX} Security Symposium*. 2019: 1099-1114.

[67] Feng B, Mera A, Lu L. P^{2}IM: Scalable and Hardware-Independent Firmware Testing via Automatic Peripheral Interface Modeling (Extended Version)[EB/OL]. 2019

[68] Mera A, Feng B, Lu L, et al. DICE: Automatic Emulation of DMA Input Channels for Dynamic Firmware Analysis[EB/OL]. 2020

[69] CAO C, GUAN L, MING J, et al. Device-Agnostic Firmware Execution is Possible: A Concolic Execution Approach for Peripheral Emulation[J]. arXiv preprint, 2019.

[70] Clements A A, Gustafson E, Scharnowski T, et al. HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation[C]. *29th USENIX Security Symposium*, 2020: 1-18.

[71] Dai Z H, Fei Y K, Zhao B, et al. Research on the Localization of Firmware Vulnerability Based on Stain Tracking[J]. *Journal of Shandong University (Natural Science)*, 2016, 51(9): 41-46, 52.
(戴忠华, 费永康, 赵波, 等. 基于污点跟踪的固件漏洞定位研究[J]. *山东大学学报(理学版)*, 2016, 51(9): 41-46, 52.)

[72] Xu T, Jin X, Huang P, et al. Early detection of configuration errors to reduce failure damage[C]. *12th {USENIX} Symposium on Operating Systems Design and Implementation*, 2016: 619-634.

[73] Muench M, Stijohann J, Kargl F, et al. What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices[C]. *NDSS*, 2018.

[74] Zhai J. *Research on Automatic Specification Generation and Verifica- Tion for Program Verification*[D]. Nanjing: Nanjing University, 2016.
(翟娟. 程序验证中的规约生成与验证技术研究[D]. 南京: 南京大学, 2016.)

[75] Xie X D. Research on Firmware Malicious Code Detection Technology Based on Model Checking[D]. PLA Information Engineering University, 2012(in Chinese).
(谢晓东. 基于模型检验的固件恶意代码检测技术研究[D].解放军信息工程大学, 2012.)

[76] Zhang P H, Tian X, Lou K W. Firmware Vulnerability Analysis Based on Formal Verification of Software and Hardware[J]. *Chinese Journal of Network and Information Security*, 2016, 2(7): 59-68.
(张朋辉, 田曦, 楼康威. 基于软硬件协同形式验证的固件漏洞分析技术[J]. *网络与信息安全学报*, 2016, 2(7): 59-68.)

[77] Huang P, Bolosky W J, Singh A, et al. ConfValley: A Systematic Configuration Validation Framework for Cloud Services[C]. *The Tenth European Conference on Computer Systems*, 2015: 1-16.

[78] Briand L C. Novel Applications of Machine Learning in Software Testing[C]. *The Eighth International Conference on Quality Software*, 2008: 3-10.

[79] Feng Q, Zhou R D, Xu C C, et al. Scalable Graph-Based Bug Search for Firmware Images[C]. *CCS '16: The 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: 480-491.

[80] Feng Q, Wang M H, Zhang M, et al. Extracting Conditional Formulas for Cross-Platform Bug Search[C]. *ASIA CCS '17: The 2017 ACM on Asia Conference on Computer and Communications Security*, 2017: 346-359.

[81] Xu X J, Liu C, Feng Q, et al. Neural Network-Based Graph Em-

bedding for Cross-Platform Binary Code Similarity Detection[C]. *2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 363-376.

[82] Gao J, Yang X, Fu Y, et al. VulSeeker: A Semantic Learning Based Vulnerability Seeker for Cross-Platform Binary[C]. *The 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018: 896-899.

[83] Gao J, Yang X, Fu Y, et al. VulSeeker-Pro: Enhanced Semantic Learning Based Binary Vulnerability Seeker with Emulation[C]. *The 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018: 803-808.

[84] Chang Q, Liu Z J, Wang M T, et al. VDNS: An Algorithm for Cross-Platform Vulnerability Searching in Binary Firmware[J]. *Journal of Computer Research and Development*, 2016, 53(10): 2288-2298.
(常青, 刘中金, 王猛涛, 等. VDNS: 一种跨平台的固件漏洞关联算法[J]. *计算机研究与发展*, 2016, 53(10): 2288-2298.)

[85] Chang Q, Liu Z J, Wang M T, et al. VDNS: An Algorithm for Cross-Platform Vulnerability Searching in Binary Firmware[J]. *Journal of Computer Research and Development*, 2016, 53(10):2287-2297.
(常青, 刘中金, 王猛涛, 等. VDNS: 一种跨平台的固件漏洞关联算法[J]. *计算机研究与发展*, 2016, 53(10):2287-2297.)

[86] David Y, Partush N, Yahav E. FirmUp[J]. *ACM SIGPLAN Notices*, 2018, 53(2): 392-404.

[87] Andrei C , Zaddach J , Francillon A , et al. A Large-Scale Analysis of the Security of Embedded Firmwares[J]. *The Usenix Security Symposium*, 2014:95-110.

[88] Chen Y, Liu Z J, Zhao W W, et al. A Large-Scale Cross-Platform Homologous Binary Retrieval Method[J]. *Journal of Computer Research and Development*, 2018, 55(7): 1498-1507.
(陈昱, 刘中金, 赵威威, 等. 一种大规模的跨平台同源二进制文件检索方法[J]. *计算机研究与发展*, 2018, 55(7): 1498-1507.)

[89] Zou Y C, Liu S, Yu N, et al. IoT Device Recognition Framework Based on Web Search[J]. *Journal of Cyber Security*, 2018, 3(4): 25-40.
(邹宇驰, 刘松, 于楠, 等. 基于搜索的物联网设备识别框架[J]. *信息安全学报*, 2018, 3(4): 25-40.)

[90] Zhang J Q, Renganarayana L, Zhang X L, et al. EnCore[J]. *ACM SIGARCH Computer Architecture News*, 2014, 42(1): 687-700.

[91] Hu G. Research on Key Technology For Firm-Code Reverse Analysis[D]. PLA Information Engineering University, 2011(in Chinese).
(胡刚. 固件代码逆向分析关键技术研究[D].解放军信息工程大学, 2011.)

[92] Cui C, Li Q B, Hu G, et al. Firm-Code Disassembly Technology Based on IVT Reconstruction[J]. *Computer Science*, 2012, 39(7): 302-304, 316.
(崔晨, 李清宝, 胡刚, 等. 基于中断向量表重构的固件代码反汇编技术[J]. *计算机科学*, 2012, 39(7): 302-304, 316.)

[93] Cui C. Research on Techniques of Firmware Control Flow Graph Recovery[D]. PLA Information Engineering University, 2012(in Chinese).
(崔晨. 固件代码控制流图恢复技术研究[D]. 解放军信息工程大学, 2012.)

[94] Xie X D, Li Q B, Wang W, et al. Variable Intervals Analysis of Firmware Code Based on Binary-Bit Operation[J]. *Computer Science*, 2013, 40(1): 107-111.
(谢晓东, 李清宝, 王炜, 等. 基于位运算的固件代码变量区间分析法[J]. *计算机科学*, 2013, 40(1): 107-111.)

[95] Zhang C Y. Research on Analysis Technology of Security Flaw in Firmware[D]. PLA Information Engineering University, 2011.
(张翠艳. 固件代码安全缺陷分析技术研究[D]. 解放军信息工程大学, 2011.)

[96] Wang M T, Liu Z J, Chang Q, et al. An Automated Analysis Method for Large-Scale Embedded Device Firmware[J]. *Journal of Beijing University of Posts and Telecommunications*, 2017, 40(z1): 98-102.
(王猛涛, 刘中金, 常青, 等. 面向大规模嵌入式设备固件的自动化分析方法[J]. *北京邮电大学学报*, 2017, 40(z1): 98-102.)

[97] Feng X, Li Q, Wang H, et al. Acquisitional rule-based engine for discovering internet-of-things devices[C]. *27th {USENIX} Security Symposium*. 2018: 327-341.

[98] Li Q, Feng X, Wang H N, et al. Automatically Discovering Surveillance Devices in the Cyberspace[C]. *The 8th ACM on Multimedia Systems Conference*, 2017: 331-342.

[99] Feng X, Liao X, Wang X F, et al. Understanding and Securing Device Vulnerabilities through Automated Bug Report Analysis[C]. *USENIX Security Symposium*, 2019: 887-903.

[100] Zheng Y W, Wen H, Cheng K, et al. A Survey of IoT Device Vulnerability Mining Techniques[J]. *Journal of Cyber Security*, 2019, 4(5): 61-75.
(郑尧文, 文辉, 程凯, 等. 物联网设备漏洞挖掘技术研究综述[J]. *信息安全学报*, 2019, 4(5): 61-75.)

张弛 于 2018 年在四川大学计算机学院获得学士学位。现于南京大学计算机与科学技术系攻读博士学位。研究领域为软件工程, 软件安全。Email: zhangchi@seg.nju.edu.cn

司徒凌云 于 2020 年在南京大学计算机科学与技术系获得博士学位。现任南京大学信息管理学院助理研究员。研究领域为软件工程, 软件与系统安全。研究兴趣包括: 静态分析、模糊测试, 漏洞挖掘。Email: stly@nju.edu.cn

王林章 于 2005 年在南京大学取得计算机软件与理论专业博士学位。现任南京大学计算机科学与技术系教授。研究领域为软件工程、信息安全。研究兴趣包括: 软件安全性测试、模型驱动的软件测试及验证、自动化软件测试工具。Email: lzwang@nju.edu.cn