计 算 机 学 报
2020 CHINESE JOURNAL OF COMPUTERS Online

嵌入式设备固件安全分析技术研究综述

于颖超[1)] 陈左宁[2)] 甘水滔[1)] 秦晓军[1)]

[1)](数学工程与先进计算国家重点实验室，江苏无锡 214083)

[2)](中国工程院, 北京 100088)

摘 要 近年来，随着嵌入式设备的种类和数量的增加，设备之间日益增长的互联互通、制造商对安全的忽视、设备固件更新不及时或难以更新等，使其安全性成为了一个突出的问题，越来越多的设备漏洞被披露。虽然当前不断呈现出国内外安全专家和学者对嵌入式设备固件的安全分析和评测技术相关工作，但缺乏详细和全面介绍最新安全研究成果的论文，使研究人员难以系统了解嵌入式设备固件安全分析技术的研究进展。为解决此问题，本文围绕着当前嵌入式设备固件面临的安全风险，分析和总结了国内外最新的研究成果，并对相关安全技术进行了综合分析和评估。首先对嵌入式设备及其固件的表现形式、分类及获取方法、面临的安全攻击层面以及自动化解析情况进行了深入研究。然后，对嵌入式设备固件安全分析技术进行了细化分析，从静态分析、符号执行、二进制漏洞关联、动态分析平台和模糊测试等五个方面进行了详细分析和横向评估。最后对未来的研究方向进行了展望。

中图法分类号 TP309.1

Survey on the Technologies of Security Analysis on Embedded Device Firmware

Yu Yingchao [1)] Chen Zuoning [2)] Gan ShuiTao[1)] Qin XiaoJun[1)]

[1)](State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi Jiangsu, 214083)

[2)](Chinese Academy of Engineering, Beijing, 100088)

Abstract In recent years, as the increasement of the types and numbers of embedded devices as the increasing interconnection between devices and neglection of security and late or difficult equipment update, the security of embedded device has become a prominent issue, more and more device vulnerabilities have beed disclosed. Although there have been proposed some solutions towards security analysis and evaluation, the lack of detailed and comprehensive survey paper makes it difficult for researchers to systematically understand the progress of embedded firmware security analysis technologies. Focusing on the security risks faced by the current embedded firmwares, this paper gives a survey of the latest research fruits at home and abroad, and makes a comprehensive analysis and evaluation of the related security technologies. Firstly, it deeply details the manifestation, the classification and the acquistition methods of the embedded firmwares, as well as the attack surfaces and the automated extraction methods. Then, the technologies of the security analysis on the embedded firmwares have been refined and analysised, ranging from static analysis, symbol execution, the binary vulnerability association, dynamic analysis platform to fuzzing test. Finally, some future research directions have beed prospected.

Key words embedded device; firmware; static analysis; symbolic execution; firmware rehost; fuzzing test

本课题得到国家高技术研究发展计划（863 计划）[2018YFB1003600]资助.**于颖超**，女，1983 年生，博士研究生，工程师，主要研究领域为安全操作系统、嵌入式设备安全等. E-mail：yuych830305@163.com. 陈左宁，女，1957 年生，硕士，院士，研究员，主要研究方向为信息安全、计算机体系结构等. 甘水滔，男，1986 年生，博士，助理研究员，主要研究方向为系统安全、程序分析、脆弱性检测. 秦晓军，男，1975 年生，博士，高级工程师，主要研究方向为软件脆弱性分析、人工智能安全等。

1 引言

在当今“万物互联”的时代，嵌入式设备逐渐成为接入云端的重要组件。与计算、移动设备不同，相当数量的嵌入式设备出于专用性考虑，在产品投放之前，几乎不会做代码审计、安全测试等流程，加之嵌入式设备的系统架构与通用设备不同，往往缺乏相应的安全机制，一旦被攻破，危害极大。2016 年 10 月，Mirai 对美国域名服务商 Dyn 发起的 DDoS(Distributed Deny of Service)攻击，一度致使全球超百万台的 IoT(Internet of Things)设备被感染[1]。究其机理，Mirai 通过利用某款智能摄像头的漏洞，成功入侵大量的连接在互联网上的 IoT 嵌入式设备，实现了对目标网络的 DDoS 攻击。

近年来，业界尝试将各类应用比较良好的通用计算系统漏洞自动化挖掘方法应用到嵌入式设备，但应用效果一直受限于嵌入式设备存在的以下问题：

（1）底层硬件平台的复杂异构

种类繁多、厂商众多、软硬件的实现差异大是当前“万物互联”的 IoT 时代嵌入式设备的一大主要特色，底层硬件平台的复杂异构性是嵌入式设备安全分析面临的巨大障碍，针对通用的 x86 或 x86_64 平台的静/动态分析方案无法直接适用。

（2）专用性强、源码/文档不公开

大多数嵌入式系统均是厂商根据其产品特色定制的程序，不同程序员的设计风格和编码方式不同，即使是同一种网络协议和硬件接口，不同厂商不同开发人员的实现方式也不尽相同，给测试方法的通用化带来了极大阻碍。

（3）运行环境受限

出于成本和安全考虑，嵌入式设备往往会裁剪掉不必要组件，且会在出厂时封死其调试接口，避免与外界直接交互，使动态测试难以直接作用于物理设备上。即使可以通过二进制重写或静态插桩等手段[2]，将动态分析方法部署到实际物理设备中，但其分析性能极大受限于硬件 CPU(Central Processing Unit)计算能力不足，进而会影响到其分析结果。

（4）嵌入式设备程序与通用 PC 程序不同

嵌入式设备程序往往通过外围设备(比如说 wifi 模块)与外部进行交互。最直观的方式是通过移动 app 或者 web 管理接口呈现给用户，用户借助这些接口发送请求给实际的设备，设备接收到处理请求之后，再分发给具体的应用程序。后台处理程序涉及到哪些应用程序、这些程序之间如何交互对于前端用户/分析人员来说完全是黑盒，因此加剧了分析难度[3]。

尽管嵌入式设备之间硬件和应用领域差异很大，但它们在一定程度上都需要运行软件。专门为嵌入式设备定制的软件通常称为*固件*，不同于传统软件，固件通常通过一组外围设备与外界进而交互。但固件具备类似于传统软件的重要特征，其重要部分通常采用不安全的低级语言(比如 C)构建，这些语言携带的脆弱特征给固件埋下了各种安全隐患。

近年来，国内外安全专家和学者针对嵌入式设备及其固件的安全分析和测评技术提出了很多切实可行的解决方案，但目前还没有详细和全面介绍最新安全研究成果的论文。本文围绕当前嵌入式设备面临的安全风险，分析和总结了国内外最新研究成果，并对相关安全技术进行了综合分析和评估。论文架构如下：第 2 章描述嵌入式设备固件的背景知识；第 3 章详细介绍嵌入式设备面临的攻击层面；第 4 章简要描述了现有分析工作的一个总结。第 5 章描述了设备固件获取和自动化解析情况；第 6 章详细阐述了嵌入式设备固件安全分析技术，并对其进行了横向分析和评估；第 7 章对未来的研究方向进行了展望。

2 背景知识

2.1 嵌入式设备及其固件

如第 1 章所述，与通用 PC 系统一样，嵌入式设备通过驱动软件来执行指定的任务，这个软件又称之为“固件”，与传统的桌面/服务器软件相比，嵌入式设备固件的区别如下：

1、嵌入式系统固件通常直接与底层硬件交互以执行其任务。

2、嵌入式系统固件大多数情况下会被集成到嵌入式设备中，存储在 ROM(Read Only Memory)或非易失内存芯片，比如 flash 内存或 EEPROM(electrically erasable programmable read-only memory)。

3、嵌入式系统固件的格式复杂多样，没有统一的标准，并且通常以单个“blob”或“image”的形式提供，且包含了确保设备运行所需的所有内容。因此，数据、代码和元数据是交织在一起的，并且执行的入口点可能硬编码在了固件中，以方便处理器直接使用。

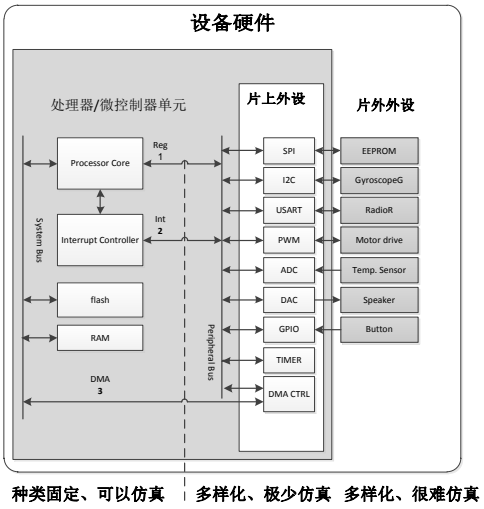

图 1 CPUs-外设接口

如图 1 所示，嵌入式设备至少由一个执行其固件的 CPU 和一组外设组成。制造商通常会在 SoC 上将其处理器单元与一组预先定义好的外设组合在一起。外设分片上外设和片外外设。固件不能直接访问片外外设，只能通过特殊的片上外设与片外外设进行交互，比如通过通用同步/异步接收/发送(USART)、总线控制器或网络接口等。CPU 与片上外设之间的交互取决于 CPU 本身，但通常可以划分为以下几种：

- **MMIO (Memory-Mapped Input/Output)**要求外设的硬件寄存器映射到 CPU 可访问的内存空间，因此，外设的状态、外设的配置以及进入和进出的数据，都可以通过读取和写入特定的内存位置来查询。
- **PMIO (Port-Mapped Input/Output)**，一些 ISAs（Industrial Standard Architecture）会引入一些特殊的指令(比如 in 和 out)与外设进行通信，此时，外设寄存器不会映射到主存，而是通过使用这些指令查询端口进行访问。
- **IRQs (Interrupt Requests)**由外设发起，通知 CPU 某种事情即将发生，比如计时器完成、新数据到达等。当一个 IRQ 到达时，CPU 尽可能快地保存它的当前状态，将执行转移到相应的 ISR(Interrupt Service Routine)。ISR 在中断上下文中执行，一旦完成，处理器将恢复到先前执行的点继续执行。
- **DMA (direct memory access)**允许外设独立于 CPU 在其它外设和主存之间传输数据，允许在处理器执行其它任务时交换大量数据。DMA 的 I/O 行为严重依赖于外设的内部设计。通常，DMA 控制器通过中断来通知 CPU 进行完整的数据传输。

固件与片外外设之间的交互进一步增加了固件分析的复杂度。由于每一个产品通常都包含了定制的电路板，所以每个固件的完整执行环境在很大程度上都是唯一的。现有的仿真工具(如 QEMU[4]和 SIMICS[5])支持的 CPU 数目相对有限，支持的片上和片外设备更少。要使用这些工具，必须实现片上和片外外设以符合固件使用的 MMIO 寄存器接口，这就需要理解并实现每一个设备的状态机和逻辑，这是一项耗时且具有挑战性的任务。

2.2 嵌入式设备固件类型分类

根据 Muench[6]，依据嵌入式设备使用的操作系统类型，将其划分为以下三种类型：

- I 型：基于通用目的操作系统的设备。改进通用的 Linux 核心，用在嵌入式系统中，通常使用轻量级的用户空间环境，比如 busybox，uClibc 等。这类设备中，与定制硬件的交互大部分是通过特定的设备驱动进行的。
- II 型：基于嵌入式操作系统的设备。通常适用于一些计算能力较低的设备，虽然可能不存在内存管理单元等高级处理器特性，但是操作系统内核和应用程序代码之间的逻辑隔离仍然存在。比如 VxWorks、ZephyrOS 这样的操作系统，常用于单用途的用户电子设备，比如 LTE 调制解调器或 DVD 播放器等。
- III 型：不具备操作系统抽象的设备。这类设备采用所谓的“monolithic firmware” [6]，其操作通常基于一个控制循环和从外设触发的中断以处理外部事件。在这些设备上运行的代码可以完全定制，也可以基于操作系统库，但最终形成的固件是一个独立的软件。

表 1 给出了这三种类型的设备固件的一个简要分析，I 型设备通常具有较强大的处理器性能和高容量的内存，其固件格式和文件系统相对健全，一般会选择标准的 Squashfs、YAFFS2、JFFS2 等文件系统，易于装载/解包，可动态仿真运行。II 型和 III 型设备，功能相对简单，代码量小，存储空间也小，其固件格式和文件系统相对不统一，多数情况下，只能按照固件的装载规则直接映射到内存空间中执行，因此仿真运行的难度较大。

表 1 不同类型设备固件的比较

设备类型	功能	代码量级	存储空间	内存管理	安全机制	固件格式	文件系统	易装载	可动态模拟
I 型	丰富	大	大	健全	有	标准	有	是	是
II 型	简单	小	小	不健全	无	自定义	有	否	否
III 型	简单	小	小	不存在	无	自定义	无	否	否

Chen 等[7]利用网络爬虫收集了 23,035 个固件镜像，对其体系结构、操作系统进行了统计，结果如图 2 所示，MIPS 架构(含大、小端，32 位)的设备排第一位，约 79.4%，第二位是 32 位的小端 ARM，占 8.9%，这两种体系总共占所有固件镜像的 90.8%。在 OS 分类中，Linux 和类 Linux 的设备(I 型)占约 50%，III 型占 42%左右，约 8%的设备使用了专用的操作系统，需要使用专有的固件提取工具才能正确提取其内核和文件系统。

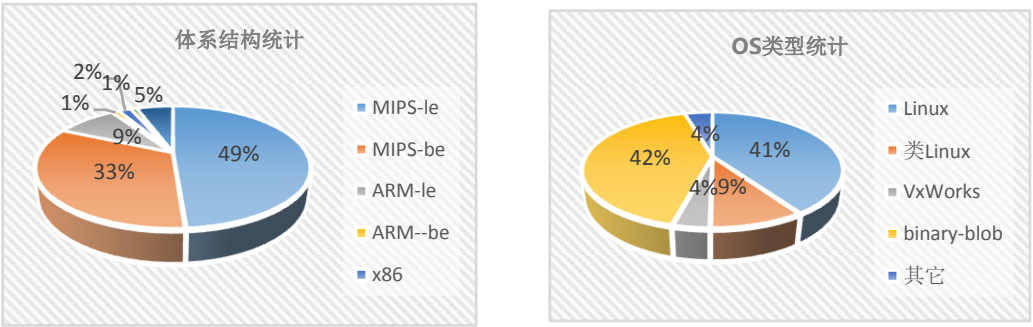

图 2 嵌入式设备固件体系结构和 OS 类型占比情况

3 嵌入式设备面临的安全风险分析

本节将从攻击层面和漏洞类型两个方面阐述当前嵌入式设备面临的安全风险，并给出现有的设备固件分析工作的一个简单总结。

3.1 攻击层面

嵌入式设备通常由硬件、固件/系统、应用软件和网络服务组件组成，本节对各个层面面临的攻击进行了一个简单的分析总结，如图 3 所示。

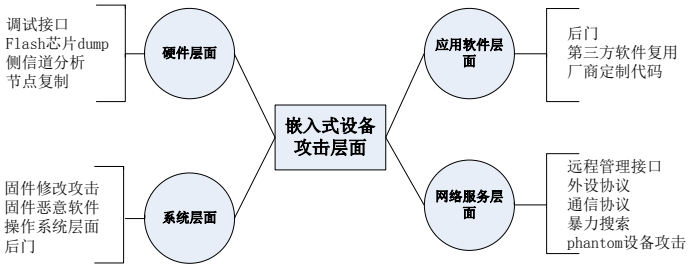

图 3 嵌入式设备面临的攻击层面

3.1.1 硬件

在整个供应链的攻击中，硬件层面最具破坏性。一旦能够探到物理硬件底层，通过相应的技术手段，可以获取对整个设备的控制权。

通过调试接口攻击 不安全的调试接口是物理设备安全检查列表中的第一项，攻击者可以通过这个接口获取高权限的 shell，进而修改或替换固件。

flash 芯片 物理设备上的 flash 芯片常用于存储固件，如果这个芯片没有设置读-写保护权限的话，攻击者可以直接读取(通过 dump 或者调试接口)固件分析并修改固件以绕过接口访问的认证。

侧信道分析攻击 侧信道分析攻击利用被攻击设备的硬件特征泄漏来提取被处理数据的信息，并利用这些信息推断出设备上的敏感信息，比如认证密钥等。攻击者不会以任何方式篡改被攻击的设备，只需要进行适当的观察(通过适当的工具远程或物理观察)就可以成功地发起攻击。根据观察到的泄漏，使用最广泛的侧信道攻击分别是微体系结构/缓存，时间、功耗和电磁发射攻击[8]。

节点复制攻击 Wei Zhou 等人[9]指出攻击者可以使用一个受其控制的虚拟设备替换掉受害者的真实设备，所有来自受害者用户的控制命令都会暴露给这个虚拟设备，进而暴露给攻击者。攻击者还可以远程接管设备，获取传感器读取到的内容来监控受害者的家，甚至操控智能家居设备，造成数据泄漏。这种新的攻击方式极大地扩大了嵌入式设备的攻击层面。

3.1.2 固件/操作系统

固件/操作系统中的漏洞影响巨大，它们可以为攻击者提供特权和控制。隐藏在固件中的恶意代码可以为攻击者提供最高级别的特权，破坏系统的启动过程，给操作系统打补丁，并获取对设备的几乎无所不能的控制。

固件恶意软件 QSnatch[1]是以 QNAP 提供的 NAS 设备为目标的恶意软件，通过修改受害者设备的固件可以窃取用户名和口令、阻止固件更新，并且控制定期调度的作业或脚本等。JungleSec 勒索软件[2]通过攻击 IPMI(Intelligent Platform Management Interface)来攻击网络上的设备。Citrix 漏洞[10]是一个企业设备固件漏洞，攻击者可以在没有认证证书的情况下，接管目标设备，访问公司的内部网络。VPNFilter 攻击[11] 是设备无法更新招致的 APT 攻击示例。

[1] https://www.zdnet.com/article/thousands-of-qnap-nas-devices-have-been-infected-with-the-qsnatch-malware/
[2] https://www.bleepingcomputer.com/news/security/junglesec-ransomware-infects-victims-through-ipmi-remote-consoles/

固件修改攻击 Basnight[12]阐述了如何恶意修改固件并且将其上传到可编程控制逻辑器上。通过逆向，攻击者可以初步了解固件更新机制，进而修改配置文件，以便在固件更新中注入恶意代码。Cui[13]分析了大量的 LaserJet 打印机固件，通过对大量的硬件组件进行逆向执行了固件修改攻击。Konstantinou [14]将固件修改定义为针对 IoT 范式的一类新的网络物理攻击，并指出固件修改攻击之所以能够成功是因为固件更新机制缺乏完整性验证或者完整性算法的鲁棒性不强造成的。

操作系统层面 嵌入式设备一般是轻量级的，由于空间受限，通常会对系统做裁剪，内核版本一般比较低，也不会及时更新，低版本内核中存在的内存溢出问题很可能会同固件一起长期存在，攻击者一旦获取内核版本，索引出低级别的溢出漏洞，就可以利用它来提升自己的特权。此外，操作系统中存在很多驱动程序[15]，远程攻击者可以利用驱动中的漏洞导致拒绝服务或执行任意代码，比如 Marvell Wifi 芯片驱动程序中存在诸如 CVE-2019-14901，CVE-2019-14897 和 CVE-2019-14896 等多个漏洞，它们会导致内核中基于堆或栈的缓冲区溢出。操作系统的特权级较高，必然也是嵌入式设备攻击者的一个重要的攻击面。

3.1.3 应用软件

后门 最简单的后门之一就是设备中存在缺省用户名和密码，虽然简单，但却经常在网络设备中出现。2013 年，D-Link 路由器中被爆出攻击者通过将浏览器的用户代理字符串设置为一个特定的字符串(“xmlset_roodkcableoj28840ybtide”)便可以在不经认证的情况下远程访问设备[17]。2016 年的 Mirai DDoS 攻击[16]就是将拥有后门账户的数十万台设备构建了一个僵尸网络。同样，攻击者通过 Telestar Digital GmbH IoT 音频设备劫持了 Telestar[1]。

第三方代码库复用 出于成本考虑，嵌入式设备的开发通常会复用大量的第三方组件，为了加速产品的研发，设计人员往往会直接使用相关功能组件，而缺乏对代码的安全性审计，因此，很容易引入未知漏洞，而且原有代码中的已知漏洞也可能不会被关注到，比如 openssl 库中的 heartbleed 漏洞就存在于很多 IoT 设备[18]中。据科恩实验室的报告[2]，第三方库导致的 N-day 安全风险占比超过发现总量的 90%。

厂商定制代码 嵌入式设备由于功能的广泛性及底层架构的异构性，每一个厂商的每一款设备的固件都需要大量的开发工作，一方面需要适配第三方库，另一方面仍然需要厂商自主开发新的代码模块，实现其特定的功能。开发人员在开发过程中安全意识不足，极易引起代码设计上的安全缺陷[19]。

3.1.4 网络服务

嵌入式网络设备中不仅存在像 Bluetooth[3]、WiFi[4]、Zigbee[5]等这样的低层通信协议，而且还存在像 HTTP、BLE[20]等这样的上层通信协议，用于与外界/用户进行交互。设备之间互联互通会使用相关的开放协议，标准不统一，且容易被破解，于是便给嵌入式设备带来了新的攻击层面。

远程管理接口不安全 为了方便管理，嵌入式设备往往会向用户提供远程管理接口，其通信协议往往会采用弱口令甚至不采用加密算法，比如在[21]中，IoT 设备连接到网络时，WiFi 口令就是以明文传输的。攻击者可以通过平台漏洞提取私人信息或者构建僵尸网络来控制安装了管理 APP 的移动手机，进而控制它所管控的智能设备。

外设协议方面 最近的一些攻击表明[22][23][24][25]，外设通常会成为攻击者执行远程利用的入口，通过欺骗一个被破坏的设备生成特定的输出，然后在设备驱动器中作为一个输入处理这个设备生成的特定输出时，会触发一个全系统破坏漏洞[22][26]。Vidgren 等人[27]阐述了敌手如何通过网络破坏使用了 ZigBee 协议的设备。

1 https://securityaffairs.co/wordpress/91069/hacking/telestar-iot-radio-devices-hack.html
2 《腾讯安全科恩实验室 IoT 产品白皮书》
3 https://www.wi-fi.org/
4 https://zigbee.org/
5 https://www.bluetooth.com/

通信协议方面 通信协议往往存在明文传输、链路劫持、重放攻击等安全风险。AR-DDoS 攻击就是通过 IoT 通信协议约束应用协议来实现的[28][29]。近年来，工控领域应用最广泛的 MQTT 协议也屡屡爆出安全风险[30][31]。需要说明的是，嵌入式设备上频繁爆出的协议漏洞未必存在于协议本身，更多的可能存在于协议的具体实现方法上。

暴力搜索攻击 攻击者可以通过扫描和分析在网设备的开放端口，在开放端口上发起暴力破解攻击，破解设备的弱加密/弱认证算法以达到其目的。Mirai 及其变种 Miori[32][33]即是通过 Telnet 使用出厂默认凭证来暴力破解并传播到其它设备的。一旦有设备被破解成功，感染了 Mirai/Miori，它将成为 botnet 的一部分，促进 DDoS 攻击。

Phantom 设备攻击 Phantom 设备攻击[34][35]关注 IoT 设备、移动 app 和云端三者之间交互过程中存在的设计缺陷，通过一个受攻击者控制的 phantom 设备远程替换/劫持受害者设备，进而控制设备或者监控/操控设备收集/生成的数据，以实现远程设备拒绝服务攻击、非法设备占用、固件窃取、敏感数据泄露等攻击。

3.2 公开漏洞分析

为了分析设备固件漏洞，本文收集并统计了 NVD 漏洞数据库[1]中的固件漏洞，并对其漏洞类型分布情况做了一个统计，需要说明的是，本文仅分析了近三年(2017.1.1-2019.12.31)的数据，且数据样本是通过匹配 firmware 以及典型的设备厂商[2]检索出来的数据去重得到的。

从互联网公开的漏洞来看，近三年的固件漏洞数量没有明显的变化趋势，在 10%左右波动，如图 4 所示。web 型(XSS、命令注入、认证绕过等)和内存型漏洞(DoS、溢出、内存破坏等)占比不分伯仲。分析其原因，一方面，大部分的嵌入式设备都会配置 web 接口方便用户管理使用，因此，嵌入式设备中的 web 服务器同样易于遭受 web 型漏洞的破坏。另一方面，这些 web 管理接口通常是使用二进制 CGIs(Common Gateway Interface)而非脚本实现的，因此，同样可能存在内存型漏洞。

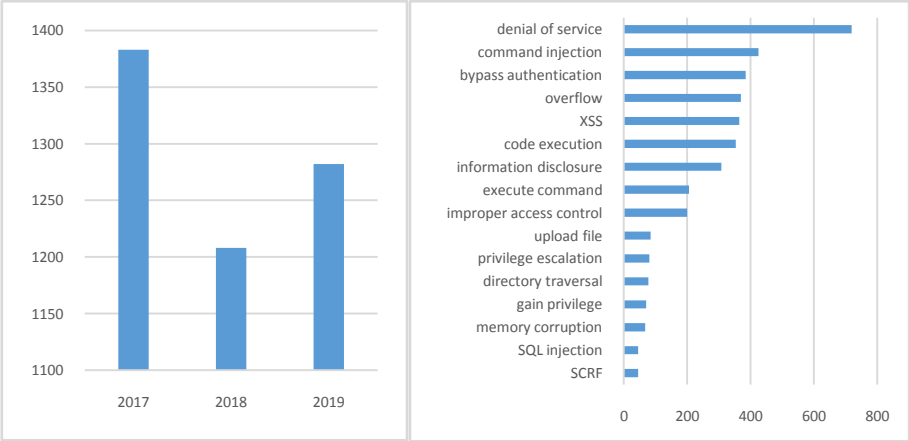

图 4 2017 年-2019 年 NVD 漏洞统计

事实上，近年来出现了大量的固件漏洞被利用的示例[36][37][38]，这表明伴随着连接设备的快速增长，攻击面也在不断增加，但分析发现，目前在嵌入式设备上发现的许多漏洞仍然可以被认为是“low-hanging fruits”[39]，比如弱身份认证、不安全的缺省配置、硬编码的凭证或未经身份验证的管理接口等[39]。这一类型的漏洞通过静态扫描可以很容易发现，而且通过增强设备厂商和消费者用户的安全意识可以很容易缓解。但是由编程错误导致的内存破坏仍然困扰着嵌入式设备开发人员和安全分析人员，传统的一些表现良好的针对内存型破坏的软件测试技术，比如模糊测试或符号执行等，在应用到嵌入式系统的适应性方面明显落后。

1 https://nvd.nist.gov/

2 收集的设备厂商包括：cisco, Huawei,TP-Link,D-Link,LinkSys, MikroTik, Netgear, OpenWrt, QNAP, SuperMicro, sinology, Tenda, Trendnet。其它的设备厂商的样本，没有纳入到分析范围。

4 现有分析工作总结

近年来涌现了很多针对设备固件的研究工作，表 2 列出了有代表性的文献。已知漏洞检测主要基于第三方库/已知固件中存在的漏洞，采用相似度检测或二进制关联算法查找新的设备固件中是否存在该漏洞。后门漏洞在于查找设备固件中存在的弱口令、硬编码以及认证绕过等不安全的配置选项。0-day 漏洞是关注最多的，目的在于发现设备固件中的未知漏洞，包括 web 型和内存型。分析发现，静态分析技术仍然是业界进行固件安全性分析的主流技术，也是目前适用性最强的一类方法，但由于静态分析误报率较高，效率较低，因此需要动态分析技术进行弥补，而针对嵌入式设备固件的动态分析，特别是传统 PC 上应用较好的模糊测试技术存在着诸多技术挑战(参见第 1 章)，需要学术界和业界持续共同的努力。

表 2 现有设备固件分析工作总结

漏洞类型	相关文献	安全分析技术
已知漏洞	Costin[17],Costin2[37],LiDeng[40],Multi_MH[41],discovRE[42],IHB[43],VDNS[44],BINARM[45],FirmUP[46],Genius[47],Gemini[48], Vulseeker[49],Vulseeker-pro[50], IoTSeeker[51]	静态分析、二进制漏洞关联
后门检测	Costin[17],Costin2[37], Stringer [52], HumIDIFy [53], Firmalice[54], Thomas[55], Thomas2[56],SMS[57][58], Laelaps[1]	静态分析、符号执行
0-day	Muench[6],Periscope[59],RPFuzzer[60],Prospect[61],MGS11[62],VDBHT1[63],IoTFuzzer[64],FirmAFL[65],SRFuzzer[66],FirmCorn[67] , P2IM[68] , HAL-fuzz[69] ,Partemu [70] , Charm[71], Prospect2[72],FirmFuzz[73]	模糊测试、动态分析、静态分析

图 5 描述了本文涉及到的嵌入式设备固件安全分析技术，包括：设备固件获取和自动化解析(第 5 章)、静态分析(6.1 节)、符号执行(6.2 节)、二进制固件漏洞关联(6.3 节)、设备固件动态分析平台(6.4 节)以及模糊测试(6.5 节)，下面将依次对其进行详细阐述和分析。

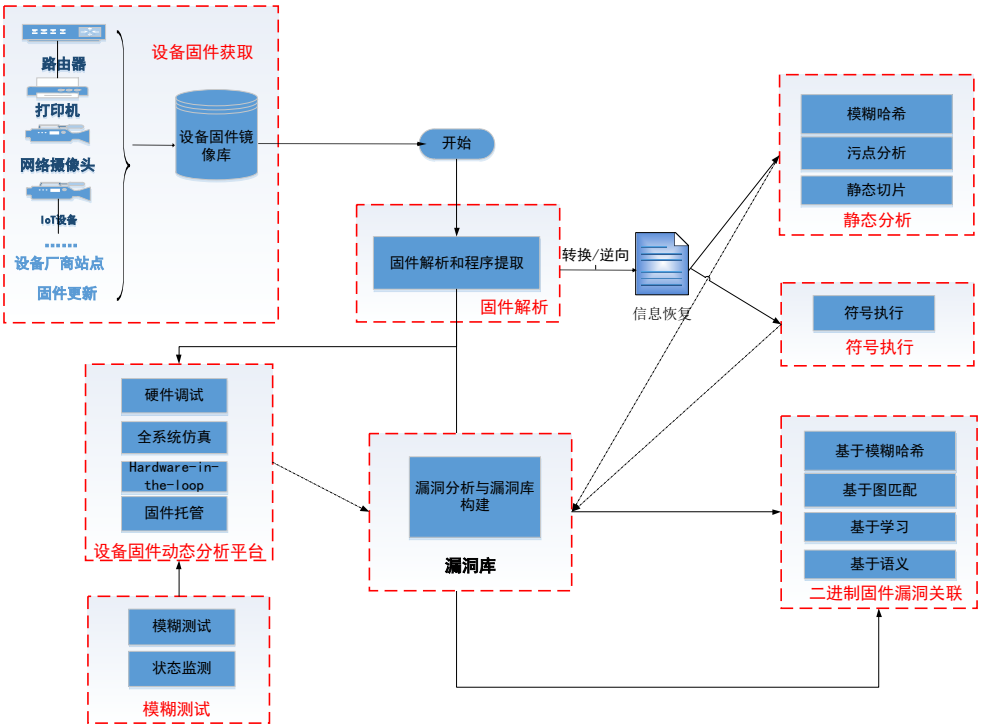

图 5 嵌入式设备固件安全分析技术一览

[1] 截止到本文发布，Laelaps 尚且是预审版：Device-agnostic Firmware Execution is Possible-- A Concolic Execution Approach for Peripheral Emulation

5 设备固件获取和自动化解析

如图 5 所示，固件获取和自动化解析是嵌入式设备固件安全分析的第一步。

5.1 设备固件获取

获取设备固件的一种典型且直接的方法就是从物理设备直接提取。跟传统 PC 不同，嵌入式设备固件通常是存储在 flash memory 芯片上的，而 flash memory 芯片通常是焊接在设备主板上的，因此，需要专门的工具和技术才能从设备上提取出固件[74][75]。

（1）通过调试接口获取

UART(Universal Asynchronous Receiver/Transmitter)是一种直观的获取设备固件访问的方式。通过直接连接到 UART，可以获取不受限制的 root shell 权限。在基于 Android 的设备上，有时可以通过 Android 调试接口 adb 获取 root shell。另一种方法是利用 bootloader 的 shell 获取 root 访问或者获取固件镜像。

JTAG(Joint Test Action Group)端口常常用于读取芯片的整个内存，通过 JTAG 端口读取设备的内存需要相应的代码接收内存 dump，并将其传输到计算机。出于安全考虑，厂商往往会在设备出厂之后，锁定设备的 JTAG 接口，避免读取设备内存或重新编程，解锁并连接 JTAG 接口可能会使固件遭受固件注入攻击。

（2） 通过 Raw Flash Dump 获取

第二种基于硬件的固件提取方法是直接读取闪存。读取旧的具有并行接口的芯片需要多个连接以及专门的程序。新技术(如 eMMC(Embedded Multi Media Card))需要的连接较少，可以使用标准的 SD 卡读卡器或专门的工具(如 easyJTAG Plus1 或 RiffBox2)读取。

（3） 通过软件方法获取

软件方法不需要物理访问实际的设备，一般可以通过访问设备厂商的 web 站点/ftp 网站，获取公开可用的固件，或者可以遵循直接下载固件更新的链接[76]，通过分析设备的网络流量获取。Costin[19]和 Chen[7]均是利用软件方法，使用一个 web 爬虫固件从各大设备厂商的 web 站点/ftp 网站，获取了数万个网络设备固件。

（4）通过设备行为模拟诱骗下载

这是一种新型的软件形式获取固件的方法，通过模拟真实设备与云通信的数据格式和交互协议，通过修改设备型号和版本号等参数欺骗云端，发送 OTA(Over The Air)更新请求，获取固件下载链接，下载相应的固件。phantom[8]详细介绍了这种行为。

5.2 设备固件的自动化解析

目前设备固件的自动化解析主要依赖于 BAT[77](Binary Analysis Tools)或 binwalk[78]。BAT 最初的设计目的是通过字符串匹配和计算二进制文件和压缩数据之间的相似性来检测 GPL 冲突。其后继工具 BANG(Binary Analysis Next Generation)[1]增加了对更多种类的解包文件格式的支持，并且为解包文件添加了新的上下文信息特性[79]。Binwalk 方便使用，易于扩展，是目前应用最广泛的固件自动化解析工具。FRAK[80]也是一个嵌入式设备固件镜像解包、分析和重新打包的框架，但是没有公开发布，并且仅支持一些固件格式(比如 Cisco IOS 和 IP phones，HP LaserJet 打印机等)。除此之外，还存在特定设备固件的定制解包器，通常应用在成功逆向了特定的固件更新格式之后，比如 Cui[15]和 Zaddach[81]。Firmware-Mod-Kit[2]收集了一组这样的工具集合，和 binwalk 一起用于提取并重构基于 Linux 的固件镜像。该工具允许解析并重构不同设备的固件镜像，且支持 squashfs 和 Jeffersonfs 等文件系统。

表 3 列出了本文中涉及到大规模固件解包工作的一些文献，给出了其使用的解包工具和

[1] https://github.com/armijnhemel/binaryanalysis-ng
[2] http://firmware-mod-kit.googlecode.com

最终的解包成功率，可以发现，目前针对设备固件的自动解包主要关注于 I 型固件，而且通常是针对路由器或摄像头的固件，因为这些设备固件的镜像通常是由简单的归档文档或标准的文件系统组成的，使用现有的工具可以很容易解压。对于 II 型和 III 型的固件，已有工具往往需要获取二进制固件的装载函数才能将二进制数据文件转换为可分析的程序镜像。然而，由于运行在不同芯片厂商的轻量级物联网设备固件的文件格式不同，每个芯片厂商的固件都有自己的装载规则。目前针对 II 型和 III 型的固件，装载基址识别、架构识别并没有一个高效的解决方案，已有的基址识别方案的准确率较低[85]，因此，在解析文件格式时，会产生误判或者直接将其分析为一个二进制数据文件。

表 3 固件解包相关文献

文献	目的	方法	解包成功率	目标设备
costin[17]	大规模漏洞检测	BAT(修改版)	33,356/172,751	网络爬虫收集的不同种类的固件
chen[7]	大规模漏洞检测	binwalk(修改版)	9,486/23,035	网络爬虫收集的不同种类的固件
genius[47]	大规模漏洞检测	未明确说明	8,126/33,045	路由器、IP Cams、AP 以及第三方开源固件
gemini[48]	大规模漏洞检测	未明确说明	8,126/33,045	路由器、IP Cams、AP 以及第三方开源固件
Stringer[52]	后门检测	binwalk,sasquash,jefferson	7,590/15,438	网络爬虫收集的不同种类的固件
HumIDIFy[53]	后门检测	binwalk,FMK,BAT	7,590/15,438	收集的 COTS 嵌入式设备固件
LiQ[82]	设备识别	binwalk	5,296/9,716	路由器、网关
FirmUp[46]	漏洞发现	binwalk	~2000/~5000	路由器、IP Cams
Dtaint[83]	漏洞发现	binwalk	6/?	路由器、IP Cams
cryptorex[84]	密码误用发现	binwalk(修改版)	521/1327	网络爬虫收集的不同种类的固件

6 设备固件安全分析技术研究

6.1 静态分析

静态分析可以在不执行程序代码的情况下，通过分析程序特征发现漏洞，是通用计算平台最常用的自动化分析工具。由于设备固件程序往往是商业程序，很少公开源代码或文档，通常只能通过对固件进行逆向，再结合一些传统的程序静态分析技术进行分析。如图 5 所示，嵌入式设备固件的静态分析过程如下：

1） 提取出固件及其中要分析的应用程序/代码段，参考第 5 章

2） 将其转换为汇编语言或中间语言描述，使用 IDA Pro[86]/Ghidra[1]这样的逆向工具，或者借助于 angr[87]这样的分析平台，将目标程序转换为统一的 VEX 中间语言做进一步的分析。IDA Pro 和 Ghidra 都可以支持 X86，ARM，MIPs 等多种指令格式的二进制汇编代码转换，而且可以恢复出函数调用关系、字符串引用等信息。

3） 结合逆向工具或者二进制分析平台，恢复程序变量、函数、结构以及 CFG(Control Flow Graph)等信息；

4） 结合静态程序分析技术，比如，模糊哈希[88]/污点分析等，实施漏洞挖掘。

6.1.1 已有工作总结

近年来，在嵌入式设备漏洞静态检测方面，除了与传统 PC 一样的通用型漏洞之外，更关注于嵌入式设备特有的一些后门类和污点类的漏洞检测，如表 4。后门类一般是为了方便设备调试人员或管理方便而特意留的一些接口，或者是开发人员开发过程中无意留下的一些

[1] https://github.com/NationalSecurityAgency/ghidra

开发痕迹(如硬编码的证书、认证绕过等)。污点类漏洞是由于缺少数据清洗，程序以一种非预期的方式使用攻击者的恶意数据而引起的(如命令注入、缓冲区溢出等)。

针对后门类的漏洞，Costin[17]首次提出大规模自动化地分析嵌入式设备的固件，自动解压并处理固件，使用模糊哈希的方式匹配固件中存在的弱密钥，通过关联分析的方式在四种不同维度查找不同固件镜像之间的相似性。Thomas [52]提出基于静态数据比对分析的方法检测 COTS(Commercial Off-The-Shelf)设备固件中未建档的功能和硬编码的认证后门漏洞。HumIDIFy[53]提出了一种半自动化的方法来检测 COTS 嵌入式设备固件二进制内的隐藏功能(也即后门[55][56])，它使用了一个半监督学习的分类器识别出固件中的二进制以及二进制具有的功能，并将其与预期功能 profile 比较，以识别出非预期的功能。Firmalice[54]使用静态程序分析生成固件的程序依赖图，获取一个从入口点到特权程序点的认证切片，再通过符号执行判断路径的约束中是否具有确定性的约束，如果存在，就可以认定为后门。忽朝俭等人[58]提出一种基于嵌入式固件的库函数识别方法，对无文件系统固件后门进行检测。

针对污点类的漏洞，DTaint[83]首先将二进制固件转换为中间描述，对于每一个函数，通过识别出指针化名、间接调用等方式，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，追踪 sinks，执行后向深度优化遍历，生成从 sinks 到 sources 的路径，通过检查路径上的污点数据约束条件判断是否存在污点类型漏洞。SainT[89]通过跟踪敏感 sources，比如设备状态(锁/未锁)和用户信息(在家/外出)到外部 sinks(如 Internet 连接)之间的信息流动查找 IoT apps 中的敏感数据流。Zheng 等人[90]提出结合协议解析器识别技术和基于二进制程序依赖图的可疑脆弱点推断技术来加速发现嵌入式固件中的污点类的漏洞。IoTFuzzer[64]虽然是一个动态模糊测试工具，但也应用了污点分析，通过对和被测设备相关联的 APP 进行污点分析，识别出 IoT 应用程序中的命令消息中常用的硬编码的字符串、用户输入或系统 API，并记录将这些协议字段作为参数的函数，以对其进行变异。Dai 等人[91]提出了一种使用仿真器以及静态污点追踪来定位并利用固件漏洞的方法，结果表明，静态污点追踪可以有效地辅助动态分析工具定位出漏洞。

6.1.2 分析评估

表 4 从目标对象、利用的技术、针对的漏洞类型以及误报率等方面对现有的静态分析工具做了一个横向比较。可以发现，静态分析过程不需要依赖目标系统，不需要过多的计算资源，可以有效地检测到特定类型的漏洞。但其缺点是误报率高，这将导致其实用性较差。

从表中可以看出，现有的分析工具多是处理 I 型固件或固件应用程序，而对于 II 和 III 型的固件的研究较少，因为要考虑目标程序加载的偏移和地址以及如何处理与外界的交互问题，这是固件分析的基础，但是目前对 II 和 III 型固件尚未有一个成熟的解决方案。另一方面，现有方案对固件组件进行分析时，仅限于单个组件单独分析，未考虑组件之间的交互。如第 1 章所述，嵌入式设备固件不同组件之间(比如 web 服务器和后台服务程序)可能存在必要的交互和数据传输，针对单个组件的独立分析并不充分。在本文成文时，作者关注到 Karonte[92]针对此问题提出了一种多二进制静态分析方法，通过模拟并追踪多个二进制之间的交互来分析嵌入式设备的固件，可以成功检测不安全的交互进而确定漏洞。

表 4 静态分析技术分析

参考文献	目标对象	利用技术	漏洞范畴	漏洞类型	大规模	误报率
Costin[17]	基于 Linux 的嵌入式设备	模糊哈希	后门类	弱密钥、缺省证书	✓	高
Stringer[52]	COTS 嵌入式设备固件	静态数据比对	后门类	硬编码证书	✓	高
HumIDIFy [53]	COTS 嵌入式设备固件	半监督学习	后门类	非预期功能	✓	低
Firmalice [54]	二进制固件	切片+符号执行	后门类	认证绕过	×	低
忽朝俭[58]	无文件系统固件	库函数识别	后门类	未授权侦听者、非预期功能、隐藏功能	✓	高
DTaint[83]	二进制固件	污点分析	污点类	缓冲区溢出、命令注入	×	高

SainT[89]	IoT app	污点追踪	污点类	敏感数据流	✓	高
Zheng[90]	基于 Linux 的 IoT 程序	静态污点分析	污点类	提取出候选关键字	×	低
IoTFuzzer[64]	与 IoT 设备关联的 app	静态污点分析	污点类	提取出协议字段和函数	×	低
Dai[91]	固件程序	污点分析	混合	缓冲区溢出、命令注入、后门、逻辑错误	×	低

6.2 符号执行

6.2.1 已有工作总结

Avatar[93]最早将符号执行应用于嵌入式设备固件分析领域，它提供了一个仿真器后端组件，用于控制 S2E(Symbolic Sombolic Exection)，选择性地执行二进制代码。S2E 是一个选择性的符号执行平台，构建于 QEMU[4]之上。在 Avatar 框架中，S2E 提供了一个功能强大的插件接口，插件可以通过这些接口拦截仿真事件（比如，基本块翻译、指令释放或执行、内存访问、处理器异常等），还可以修改执行状态，通过符号化执行特定代码段可以检测固件中存在的漏洞。

FIE[94]基于 KLEE[95]构建了针对 MSP430 系列的嵌入式设备的固件漏洞检测工具，它根据 MSP430 微处理器设计文档对内存描述、中断描述和芯片描述进行了统一的公式化描述，提供给符号执行引擎进行符号执行分析。但是这种模拟方式不够精确，且无法应对实际执行情况下环境的一些修改，加上符号执行本身存在的路径约束求解困难和状态爆炸等问题并没有得到有效解决，因此漏报和识报率很高，且只针对 MSP430 微处理器系列产品。FirmUSB[96]使用 USB 协议的领域知识，为 8051/52 体系结构上的 USB 控制器固件专门设计一个符号执行机制，FirmUSB 应用了两个符号执行框架：FIE 和 angr，通过静态分析和符号执行，提取出固件镜像中的语义以构建出固件的功能模型，与预期的功能进行比较，以确定是否存在漏洞。相比较于不受约束的符号执行，FirmUSB 的性能可以提升七倍。Firmalice[54]的主要思想是利用符号执行去分析设备固件中登录认证相关的代码，得到可以进入特权状态的路径之后，判断这些路径的约束中是否有确定性的约束，如果存在，就可以认定为后门。

Inception[97]基于 KLEE 构建了一个 Inception 符号虚拟机，它将源码、汇编和库二进制文件一起提升到 LLVM-IR，由 Inception 符号执行这个 LLVM-IR，以探测固件镜像中存在的漏洞。结果显示，该系统能够有效支持 ARM Cortex-M3 系统产品。Chen 等人[98]等人使用 VxWorks 提供的专用远程调试特征收集而来的运行时信息来指导 SMAFE[99]，然而，该工具除了与 VxWorks 绑定之外，还需要在符号执行引擎和被分析设备之间建立一对一的对应关系，严重限制了这种方法的可伸缩性。

Gerbil[100]是针对大规模 IoT 固件中存在的特权分离漏洞检测设计的，由于符号执行整个 IoT 固件几乎不可行，所以它首先切片出最有可能存在特权分离漏洞的代码部分，做符号执行，而且在符号执行之上实现了一个路径探测框架，通过库函数识别机制，跳过复杂的库函数以缓解路径爆炸并且可以恢复间接调用以探测更深的路径。与之相似，为了克服固件仿真的难题，FIoT[101]也使用了后向代码切片，遍历 IoT 固件的 CFG，构建包含从输入数据到敏感函数调用路径的代码片段，交给符号执行，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，以检测是否会触发内存破坏。

6.2.2 分析评估

表 5 给出了现有的针对嵌入式设备固件分析的工作总结。分析发现，符号执行技术因为依赖于很多重量级的分析组件(如反汇编引擎、插桩引擎、求解器)，很难直接在智能嵌入式设备中直接使用，但可以结合一些领域知识、系统仿真、静态分析等间接方式来实现，比如 Gerbil 关注于可能发生特权分离漏洞的固件部分，FIoT 提取固件的控制流图，将一些敏感操作函数作为漏洞触发点，对程序进行切片，Firmalice 对特权认证函数进行反向切片等，约束了执行符号执行的代码规模，更有利于发挥符号执行的优势。

表 5 基于符号执行的固件分析

参考文献	符号执行引擎	目标对象	漏洞类型	结合技术	源码	真实设备	固件类型		
							I 型	II 型	III 型
Firmalice[54]	angr	固件	认证绕过	静态分析	×	×	×	✓	✓
Avatar[92]	S2E	嵌入式设备	内存型	仿真	×	✓	×	✓	✓
FIE[94]	KLEE	MSP430	内存型	领域知识	✓	×	×	✓	✓
FirmUSB[96]	KLEE,angr	8051/52	BadUSB	领域知识	×	×	×	✓	✓
Inception[97]	KLEE	嵌入式程序(含汇编)	内存型	lift-and-merge	✓	✓	×	✓	✓
Chen[98]	SMAFE	固件二进制文件	内存型	动态调试	×	✓	×	✓	×
Gerbil[100]	angr	IoT 设备	特权分离	切片规范	×	×	×	✓	✓
FIoT[101]	angr	IoT 设备	内存型	静态分析	×	×	×	✓	✓

总的来说，符号执行在嵌入式设备固件安全性分析方面有了初步应用，取得了一定的进展，但由于其环境搭建复杂，对设备也有严格的要求，所以在应用的广泛性上有限。当前的技术大多是针对特定类型的硬件体系构建的，而且未充分考虑外设的输入影响。符号执行本身不会输出漏洞，而只是生成一个约束条件集，需要检测器（比如 Z3）分析这些约束条件，某些场景下是很难求出合适的解(如，固件中常用到的密码 hash 等)，加之符号执行本身存在的路径和状态爆炸问题没有得到实质性的解决，因此，符号执行只能当作一种辅助手段。

6.3 二进制固件漏洞关联

近年来，研究人员开始探究基于同源二进制代码相似度比较的固件漏洞检测，并取得了较大的进展，如表 6 所示。二进制固件同源漏洞关联的基本思路是从二进制固件代码/漏洞代码中提取出其特征，进行某种形式的编码（方便进行相似度计算），然后对编码后的特征进行相似度计算，以确定二进制固件代码中是否存在漏洞，如图 6 所示。根据所使用的相似度算法的种类不同，可以将其划分为以下几种类型：基于模糊哈希算法的二进制关联、基于图匹配的漏洞搜索、基于学习的漏洞搜索和基于语义的漏洞搜索。

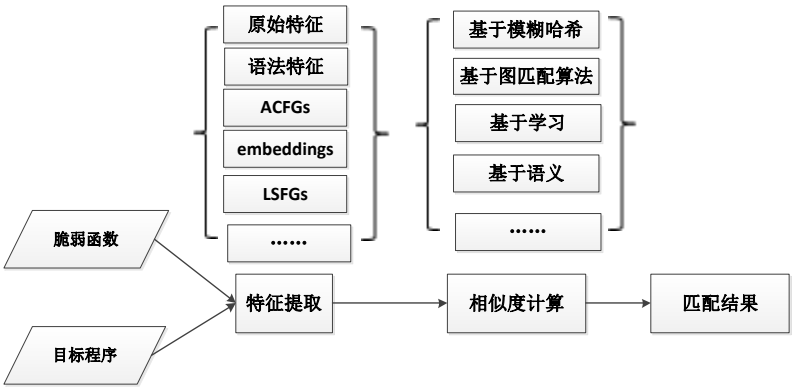

图 6 二进制固件漏洞关联流图

6.3.1 基于模糊哈希算法的二进制关联

Costin 等人[17]首次提出基于模糊哈希的大规模嵌入式固件安全分析方法，通过固件模块关联的方法，采用模糊哈希算法扫描不同固件镜像之间存在的类似漏洞，最终在 693 个固件中关联出了 38 个漏洞文件，但是这种方案需要对每个文件进行一对一的模糊哈希比较，会产生大量的时间开销，且粒度是文件级别的，精确度也不高。李登等人[40]提出的基于第三方库的同源性分析结果对同类固件的漏洞检测也是基于模糊哈希算法实现的，能够准确地对不同类型、架构、版本的固件进行漏洞检测，但由于模糊哈希本身的问题，这种检测方法仍然存在一定的误差。BinARM[45]提出了一种多阶段的检测引擎，可以有效地识别出

IED 固件中的脆弱函数，同时实现高精度，实验结果表明这个检测引擎比现有的模糊匹配方法快 3 个数量级。陈昱等人[43]通过深度学习二进制文件中的可读字符串，然后对编码向量生成局部敏感哈希从而实现快速检索，解决了大规模设备固件同源二进制文件检索方法的时间开销问题，只需不到 1s 的时间即可完成一次针对 22,594 个固件的同源二进制文件检索任务。

6.3.2 基于图匹配算法的漏洞搜索

Multi-MH[41]是首个跨体系结构的二进制代码相似度检测工作，该方法对心脏滴血漏洞进行关联时取得了较好的效果，但其对不同平台的函数控制流图较为敏感。FirmUp[46]提出使用规一化的片段表示程序以解决跨编译器、跨优化选项、跨体系结构的问题，其主要方法是将函数进行基本块级别的拆分，然后将基本块进一步切片成更小的片段，再将这种片段中的寄存器名和地址偏移归一化，以函数为单位制成一张表，以两个表中相同的代码片段的数量作为相似度比较的依据。为了提升效率，discovRE[42]提取更轻量级的语法级别特征来加速特征提取，并且在图匹配之前通过简单的函数级特征进行预过滤，以提升搜索的效率。然而，根据 Feng[47]，这种预过滤方法并不可靠，可能会产生很多的漏报，导致搜索精确度的严重下降。理论上来说，这种依赖于 pairwise 图匹配来检测相似度的算法，必然是低效的。

6.3.3 基于学习的二进制漏洞搜索

相比较于直接匹配两个控制流图，Genius[47]从控制流图中学习更高级别的数值特征描述，然后再基于学习到的特征构建搜索。与原始的 CFG 特征相比，学习到的特征对于跨体系结构的更改更能保持不变性，而且通过局部敏感哈希的方式索引转换后的更高级别的特征，可以使搜索 bug 的速度比现有的基于图的搜索快出好几个数量级。针对 Genius 在训练图嵌入所需的代码库时需要的时间开销大的问题，Xu[48]提出基于神经网络构建图嵌入模型，以执行跨平台的相似度检测，实验结果表明，相比于 Genius，训练时间从 1 周下降到了 30 分钟~10 个小时，而且能识别出更多额外的漏洞代码。针对传统的“一对一”关联匹配检索方法的时间复杂度不能满足大规模同源检索的需求，IHB[43]设计并实现了一种时间复杂度为 O(lgN)同源二进制文件检索方法，其核心思想是通过深度学习网络编码(双向双层 Bi-GRU)二进制文件中的可读字符串，然后对编码向量生成局部敏感哈希从而实现快速检索，其不足之处在于对包含字符串较少或者做过字符串混淆处理的二进制文件的检索能力较差。

6.3.4 基于语义的漏洞搜索

虽然上述方法在漏洞搜索领域获取了一定的效果，但在复杂场景中应用时，存在两个局限性。第一，获取的函数语义信息不充分、不准确，导致了很高的假阳率。第二，大部分的方法需要很高的计算开销，这就使得它们很难应用于复杂的大型二进制文件。Gao 提出了 VulSeeker[49] 和 VulSeeker-Pro[50]，结合 CFG 和 DFG(Data Flow Graph)以形成 LSFG(Labeled Semantic Flow Graph)作为特征向量并且应用了深度学习执行跨平台二进制漏洞搜索的工具。与现有的方法(如 Gemini [48])相比，精确度要高很多。IoTSeeker[51]无缝结点了 Vulseeker 和 Vulseeker-pro，并且支持跨体系结构的函数仿真以获取更好的性能。通过将动态方法应用到快速获取的与漏洞相似的小范围候选函数，提高了搜索的精度，同时又减少了时间的需求。

6.3.5 分析评估

本文从漏洞关联比较的粒度、特征、编译方案、相似度算法、使用的数据集对二进制固件漏洞关联算法进行了总结和描述，如表 6 所示，并且从精确度、效率、可扩展性以及跨平台支持等四个维度对其进行了一个简单的评估。

表 6 二进制设备固件漏洞关联分析方法比较

项目名称	粒度	特征	编码方案	相似度算法	精度	效率	跨平台	可扩展	数据集
Costin[17]	文件	文件	无	模糊哈希	低	低	是	高	数据集 V
LiDeng[40]	文件	文件&字符	无	二进制差量分析、字符	低	低	是	高	第三方库

		串		串常量匹配、模糊哈希					
Multi_MH[41]	基本块	I/O 对	抽样&MinHashing	基于语义哈希的比较	低	低	是	低	数据集 I
discovRE[42]	函数	多维	图匹配	过滤器	低	低	是	低	数据集 I、II
IHB[43]	文件	字符串	Bi-GRU	LSH 相似度哈希索引	高	高	是	高	数据集 I
VDNS[44]	函数	多维	ReliefF 特征选择	函数数据相似度&局部调用结构整体相似度	高	高	是	低	数据集 I
BINARM[45]	函数	多维	无	多阶段检测&模糊哈希	高	高	是	低	数据集 VI
FirmUP[46]	函数	strand	归一化	strand 的数量	高	高	是	高	数据集 III、IV
Genius[47]	基本块	ACFGs	基于码本的图嵌入	局部敏感哈希	高	高	是	高	数据集 I、II、III、IV
Gemini[48]	基本块	ACFGs	基于 NN 的图嵌入	Siamense	高	高	是	高	数据集 I、III、IV
IoTSeeker[51]	基本块	LSFG	语义感知的 NN	嵌入向量的余弦距离	高	高	是	高	数据集 I、III

表中所使用的数据集说明：数据集 I：开源库 busybox, openssl, coreutils，不同编译器（gcc，clang），不同优化级别（O0-O3），不同体系架构（x86,mips,arm）编译而来。数据集 II：公开的固件集（DD-WRT Firmware Image r21676、ReadyNAS Firmware Image v6.1.6）。数据集 III：固件镜像数据集（genius，33045 个）。数据集 IV：漏洞数据集（CVE）。数据集 V：共享证书、自我签名证书、口令。数据集 VI：IED 固件中的脆弱开源库，针对不同平台编译出来的漏洞数据库。

如表 6 所示，二进制固件漏洞关联分析从最初的文件相似度比较到细粒度的函数比较，再到更细粒度的基本块比较，从文件特征，到字符串、控制流图、语义特征提取，甚至是多维特征选择（比如 discovRE 使用结构特征和数值特征，BINARM 使用函数 shap、分支和路径，VDNS 采用函数数值特征和局部调用特征），编码方案从最初的基于特征选择，到基于图匹配，再到基于神经网络的图嵌入，漏洞关联速度和准确度都有了很大的提升。粒度越细、编码特征越丰富、编码方案越智能，精确度就越高，目前的技术通过引入敏感哈希索引等方式，同时保证了漏洞关联分析的性能，在大规模二进制固件漏洞搜索方面体现了绝对的优势。

6.4 动态分析平台

动态分析是根据实际的执行环境来执行的，无论是物理平台，还是仿真平台，还是两者的结合，都取决于硬件的可用性或仿真器的完整性。

6.4.1 基于硬件的调试

在嵌入式设备领域，传统的动态分析技术，往往通过硬件调试接口(如 UART 或 JTAG)，结合软件插桩技术来实现，并且取得了一定的成功，比如在路由器[102]、plc[103]和打印机[15]上，展示了动态二进制固件分析的可行性，而且这种方法准确可靠。但是，它依赖于物理硬件，且只能调试存在并保留调试接口的设备，而且很大程度上依赖于手工执行，不利于大规模自动化的分析。另一方面，嵌入式设备，特别是低端系统，受尺寸限制往往没有足够的空间来存放插桩的二进制，而且，二进制插桩工具也极易受到相同的空间限制的阻碍。因此，流行的插桩技术和工具可能对类型 I 和类型 II 是可行的，但对于类型 III 仍然极具挑战。

6.4.2 全系统仿真

嵌入式设备和 OS 供应商有时会开发专用的仿真器用于内部测试或方便第三方的开发，然而，即便有现成的设备专用仿真器(如 SIMICS[5]和 Bochs[104]) 通常也不适用于安全性研究，因为它们通常是闭源的，并且缺乏与表现较好的动态分析工具的集成。

QEMU 支持多种 ISAs、免费且开源，因此被广泛应用于固件安全性分析的仿真平台设计中。Chen[7]首次提出了基于 QEMU 对固件镜像进行大规模的动态仿真分析—Firmadyne，成功仿真与成功提出并识别出来体系结构的固件镜像的百分比高达 96.6%，论证了其仿真能力的有效性。Firmadyne 通过访问 web 页、收集 SNMP 信息并且测试固件是否易于遭受一组特定的已知和手写利用的攻击，执行了对仿真固件的自动化的动态分析，论证了基于全系统仿真的动态分析的可行性。FirmAFL[65]在 Firmadyne 提供的全系统仿真能力的基础上，混合

了 QEMU 提供的全系统和用户模式仿真，使用 AFL[109]提供了对嵌入式 Linux 应用程序的高吞吐量的模糊测试。Costin 等人[37]描述了一种与 Firmadyne 类似的方法，但其关注点在于分析 web 接口。同样地，Qemu STM32[105]项目，扩展了 QEMU 仿真了 STM32 芯片，证明当有完整的硬件文档时，通过一定的能力仿真硬件，完全仿真 III 型固件镜像也是可能的。LuaQemu[1]项目在 QEMU 中嵌入了一个 Lua 解释器，提供了定制的钩子，以访问 QEMU 没有实现的硬件外设 ，而且为二进制代码提供了灵活的 API 交互支持，并且成功应用到了 Samsung Galaxy S6 上的 Broadcom WiFi SoC 进行动态分析，但其分析方式目前仍然需要人工参与。PARTEMU[70]提出通过仿真被测目标所需要的硬件和软件依赖来实施对被测目标的动态测试是可行的，并且成功应用到了 Android 智能手机上的可信应用程序的测试中。

全系统仿真解决方案可以在不存在物理设备的情况下进行动态测试，从而实现更大的并行化，并且可以收集大量的有关正在运行的固件的信息。缺点在于它只适用于目标固件所访问的所有外设都是已知的且可以成功仿真的情况，然而，这种情况很少发生。

6.4.3 Hardware-in-the-loop

Hardware-in-the-loop 最初是由 Avatar[92]提出的，其核心思想是使用一个修改后的仿真器（固件代码在仿真器中执行）将与外设之间的交互转发给实际的物理设备处理，以缓解对全系统仿真需要对每个外设进行仿真带来的负担。

Avatar 在嵌入式物理设备上注入了一个特殊的软件代理，它在仿真器中仿真运行固件指令，而将仿真器无法处理的 I/O 访问全部转发给实际的物理设备上的这个特殊的软件代理进行处理。但由于 Avatar 使用像 UART/JTAG 这样的低带宽通道进行物理设备和仿真器之间的数据传输，带来了严重的性能开销(5 个内存操作/秒)，一些中断未来得及处理的中断只能丢弃。SURROGATES[106]在主机的 PCIE 总线和被测机的 JTAG 接口之间建立了一个定制的、低时延的 FPGA 桥，以一种近乎实时的方式仿真并插桩嵌入式系统，进而可以使用多种动态分析系统对被测设备进行测试。与之相似，Inception[97]基于 Xilinx ZedBoard FPGA 定制了一个 Inception Debugger 组件将基于 KLEE 的符号虚拟机发出的高级别的读/写命令有效地转换为低级别的 JTAG 信号。PROSPECT[61]在 QEMU 中构建了一个虚拟的字符设备，拦截内核中与字符设备进行通信的系统调用，并将其转发到目标嵌入式系统上的正确的字符设备，在目标设备上执行并将执行结果反馈给分析系统。在其后续的延展工作中，Kammerstetter[72]通过缓存预期的外设行为，并且当外设的行为偏离预期时重置缓存，对固件的状态做了近似，改进了系统的可伸缩性，但是这种解决方案的限制条件很多：仅适用于类型 I 型的固件、仅适用于使用字符设备驱动实现的外设、仿真固件和目标系统之间必须能够建立基于 TCP 的网络连接等等，实用性不高。Charm[71]将 Android 设备驱动移植到工作台虚拟机上，由监控程序截获虚拟机中设备驱动发出的与其 I/O 设备进行交互的操作请求，通过一个定制的低时延的 USB3.0 通道发送给实际的移动系统，同时来自移动系统 I/O 设备的中断也会发送给虚拟机中的设备驱动去处理。虽然实际的移动系统仍然需要用来执行非频繁的低级别的 I/O 操作，但是设备驱动完全是在虚拟机内运行，因此可以被动态分析。

Hardware-in-the-loop 既提供了全仿真的优势，同时也缓解了了解并仿真 I/O 操作带来的负担。然而，这种灵活性是牺牲性能（需要与真实设备交互）和可扩展性（需要配对每一个仿真实例和物理设备）换取的，而且仍然要依赖于实际的物理设备，因此限制了其应用范围。

6.4.4 固件托管

固件托管的基本思想是分析物理设备及其固件，理解固件希望从周边硬件（外围设备）中获取什么，然后尝试对其进行建模，一起替代硬件，使得可以仅使用软件组件对固件进行分析。本质上来说，就是考虑如何将固件执行从固件执行所依赖的硬件中分离出来，形成一个近似于实际执行环境的全仿真执行环境。

[1] https://www.securitynewspaper.com/2017/07/10/emulation-exploration-bcm-wifi-frame-parsing-using-luaqemu/

Pretenter[108]通过观测固件-物理设备之间的真实交互记录，使用机器学习/模式识别为每一个外设创建模型，再使用全系统仿真器(QEMU)或程序分析引擎(angr)结合生成的外设模型实现固件托管分析。P2IM[68]关注于连接处理器和片上外设的处理器-外设接口(参见图 1)，根据处理器设计文档，提供与之等价的接口模型，模拟外部行为以处理外部 I/O 操作。Avatar^2[110]允许分析人员选择不同的执行环境，共同在相同的执行状态上执行其分析任务，其优势在于可以在多个分析环境中共享不同分析工具得到的分析状态。目前，Avatar^2 支持的动态分析工具包括：GDB、OpenOCD[111], QEMU, angr 和 PANDA[112][113]。HALucinator[69]通过拦截 HALs(Hardware Abstract Layers)，插入其替代的功能，打破了固件和硬件之间的紧密耦合，实现了固件托管的功能。但是它仅适用于固件中使用了芯片厂商提供了 HALs 的那些设备，并且编译环境要跟固件的编译环境类似，仅这两点就大大限制了其可用范围[114]。

另一种固件托管的方法是利用符号执行来处理未知外设。一种是在符号执行引擎内执行固件，并且将硬件交互的结果视作是符号数据(参见 6.2 节)。另一种是仍然在仿真器中执行固件，在遇到未知外设时，使用符号执行来推断外设输入，如 Laelaps，在 QEMU 中执行固件，当访问未实现的外设时，QEMU 将会被卡住，此时将执行切换到符号执行，以求解出适用于当前外设访问操作的正常输入值，并将其喂给 QEMU，引导 QEMU 继续执行。

6.4.5 分析评估

表 7 从漏洞发现能力、精确度、可扩展性、稳定性和自动化程度等几个维度对嵌入式设备固件动态分析平台给出了一个综合评估。总的来说，系统仿真技术可以有效地利用当前现有的漏洞挖掘技术，避免因嵌入式设备资源有限带来的插桩困难，但是，由于底层硬件平台的复杂异构以及外设种类的类型繁多且实现方式不一，就目前的技术而言，要实现与硬件设备完全一样的全系统仿真环境是不现实的，常常会因为没有提供完整的固件运行环境而造成程序执行不下去。Hardware-in-the-loop 的方法是一种折衷方案，既提供了可以有效利用现有漏洞挖掘技术的仿真环境，又可以使用实际的物理设备处理无法仿真的外设请求，然而，它依赖于实际的物理设备，仿真环境和物理设备之间来回切换，也会影响到动态分析技术的运行速度和效果。固件托管可以以纯软件的方式替代硬件，便于插桩和运行时监控以更好地指导测试，因此更为了业界广泛应用的动态分析方法，但是与 Hardware-in-the-loop 比，与外设交互的精度要低，可能仿真返回的值只能保证仿真环境能够正常运行，不卡壳，但是这个值可能永远不会在真实系统上出现，且目前大多数固件托管技术仅关注 MMIO 和中断，并没有涉及外设 DMA 操作，其对设备的安全影响程度如何也没有公开的论文涉及，因此仍然需要进一步的分析和完善。

表 7 动态分析方法比较

分析方法	漏洞发现能力	精确度	可扩展性	稳定性	自动化程度	综合能力排名
基于硬件的调试	×	✓	×	×	×	4
全系统仿真	✓	✓	✓	✓	✓	1
Hard-in-the-loop	✓	✓	×	×	×	3
固件托管	✓	×	✓	✓	✓	2

6.5 模糊测试

在嵌入式设备固件的动态分析技术中，模糊测试是应用最广泛也是最高效的，图 7 给出了嵌入式设备固件模糊测试的一般流程。目标测试平台可以是真实设备、仿真平台、固件托管平台；测试用例生成可以是生成式、变异式。

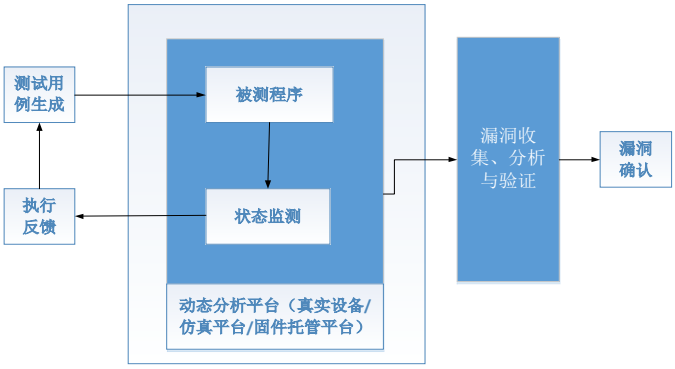

图 7 模糊测试的一般流程

6.5.1 模糊测试分类

根据程序执行反馈的获取情况，可以将模糊测试分为白盒、黑盒和灰盒，如表 8。

（一） 白盒测试

白盒测试通过分析被测程序的内部机制和执行被测程序时收集的信息来生成测试用例，白盒测试通常会对程序进行动态污点分析[115]或符号执行[116]以获取精确的程序执行和状态信息。IoTFuzzer[64]采用了一种基于污点的模糊测试方法，利用数据流分析以确定如何控制 IoT app 以生成有意义的测试用例，以对远程设备进行模糊测试。FirmCorn[67]提出了一种面向漏洞的模糊测试框架，通过脆弱代码搜索算法定位出 IoT 固件中的脆弱点，制定模糊测试策略做针对性的 fuzzing。

（二） 黑盒测试

黑盒测试是将测试对象当作黑盒子，按照指定的规范随机生成测试用例。在嵌入式设备测试领域，一些表现良好的协议测试工具(如 Peach[1]、Sulley[117]、Boofuzz[2]等)都属于黑盒测试的范围。RPFuzzer[60]提出了一种两阶段的模糊测试测试用例生成模型，结合人工分析和历史漏洞数据生成有效的测试用例，并且引入了基于修改的 Dynamips 的调试器[3]，在异常发生时记录寄存器的值，以有效定位漏洞。Muench[6]的模糊测试会话使用了 boofuzz，从外部生成输入并喂给目标测试系统，由于其输入空间探测是盲目的、随机的，因此能够查找到 bug 的机率非常低。IoTFuzzer[64]通过自动化解析 IoT 官方移动 APP 中的程序逻辑，生成有意义的测试消息，发送给被测设备，并且通过 heartbeat 包和响应信息判断模糊测试的效果。SRFuzzer[66]对采集到的网络流量进行解析并进行针对性的变异执行模糊测试。FirmFuzz[73]使用了一个 headless 的浏览器（受 fuzzer 控制）作为一个代理服务器捕获并变异输入。

（三） 灰盒测试

灰盒测试的典型特点就是可以使用目标的执行反馈来制导测试用例的生成。FirmAFL[65]是首个应用在 IoT 固件上的高吞吐量的灰盒 fuzzer，它充分利用了 QEMU 用户模式的高吞吐特征和 QEMU 系统模式的高兼容特征，对 IoT 固件中的程序执行覆盖率制导的高吞吐灰盒模糊测试，比全系统灰盒测试 TriforceAFL[4]提速了 8.2 倍。PROSPECT[61]通过模糊测试不同的网络协议实现并且监控部分仿真的系统的状态，在火灾报警系统中发现了一个先前未知的 0-day 漏洞。Charm[71]使用 Syzkaller[118]对部分仿真的 android 设备驱动进行了模糊测试。Mulliner[62]和 Van[63]根据 GSM 规范设计了一种基于生成的 fuzzer，测试功能手机和智能手机中的 GSM 实现。P2IM[68]使用 AFL 作为其 drop-in fuzzer，对固件进行模糊测试。HALucinator[69]使用 AFL-Unicorn[119]做为其模糊测试组件，AFL-Unicorn 使用一个灵活的 API 结合了 QEMU 的 ISA 仿真特征，并且提供了 AFL 的覆盖率插桩和 fork-server 能力。PARTEMU[70]使用

[1] http://www.peachfuzzer.com/
[2] https://github.com/jtpereyda/boofuzz
[3] http://dynagen.org/tutorial.html
[4] https://github.com/nccgroup/TriforceAFL

TriforceAFL 作为其模糊测试组件的代码基，不同点在于 TriforceAFL 中是将 QEMU 作为一个被测进程启动的（即 afl –QQ 模式），而 PARTEMU 则是单独启动 QEMU，再通过一个代理将其与 AFL 连接，对 AFL 来说，这个代理可以看作是被测进程。

6.5.2 模糊测试中的状态监测

如第 1 章所述，嵌入式设备程序形态多样(二进制程序、shell 脚本、web 程序等)，使用的程序设计语言也多种多样(C、C++、汇编、xml、Lua 等)，漏洞模式会随之变化，因此，异常行为检测的方法也应该随程序的工作形式随之变化。

Muench[6]首次分析了传统的异常检测方法对于嵌入式设备程序的普适性，验证了传统的基于显式崩溃的内存破坏漏洞检测技术不再适用于嵌入式设备内存破坏类的漏洞检测。传统模糊测试往往依赖于可观测的崩溃来检测内存 bugs，而对于 II 型和 III 型设备来说，破坏性的输入几乎不会触发任何崩溃。受 ASAN[120]和 Valgrind[121]的影响，Muench[6]引入了 6 种简单的启发式规则：segment tracking，format specifier tracking，heap object tracking，call stack tracking, call frame tracking，stack object tracking，分别应用在 NAT(native)，FE(Full Emulation)，PE/MF(Parital Emulation/Memory Forwarding)，PE/PM(Partial Emulation/Peripheral Modeling)四种模式的测试环境中，实验数据表明：部分仿真的测试环境以及组合式的异常检测可以提升嵌入式设备的模糊测试能力。

P2IM[68]在 Muench[6]描述的段追踪启发式检测规则的基础上，实现了一个非常基本的内存错误检测器，实施了对每一个内存段所需要的最小许可：对于 flash 需要 R+X 权限，对于 RAM、外设和系统控制时钟，需要 R+W 权限，其它的内存则没有访问权限。因此，它只能检测跨区域边界的内存破坏以及违反访问许可权限的内存破坏。HALucinator[69]则提供了一个类似于 ASAN 的堆-检查（heap-checking）实现。AFL-Unicorn[130]通过将不同的执行错误(如非法内存访问)转换为模糊测试过程中发出的等价的进程信号(如 SIGSEGV)来检测崩溃，提供正确的信号给 AFL。IoTFuzzer[64]则是通过分析 IoT 设备的响应报文来判断是否触发了崩溃，针对 TCP 通信，IoTFuzzer 依据服务的连接状态来决断；针对 UDP 的通信，则根据事先注入的 heartbeat 消息来判断。SRFuzzer[66]扩展了存活性检测，引入了基于响应的监控器(检测前端触发的 XSS 问题)、基于路由的监控(检测命令注入和 XSS)和基于信号的监控(通过 ptrace[122][123]监控 SIGSEGV 和 SIGABRT 以检测内存破坏漏洞)。

RPFuzzer[60]使用了三种方法监控被测路由，包括：监控 CPU 的利用率，以有效地检测由处理测试协议数据的进程的 CPU 使用异常所引起的 DoS 攻击；发送正常的监控数据，以监控路由器崩溃或重启；检查系统日志，以检测路由器崩溃、重启、僵尸进程等等。FirmFuzz[73]通过监控增强固件所生成的日志以检测命令注入、缓冲区溢出和空指针解引用漏洞。FirmCorn[67]使用了内存破坏检查和异常检测来监控崩溃。针对 silent 的内存破坏，FirmCorn 会在敏感函数调用之后，检查栈数据以查看是否存在溢出，同时会监控执行过程中的异常情况，并且识别异常以确定所执行的程序是否发生了崩溃。监控到崩溃之后，FirmCorn 会记录发生崩溃的测试用例，提交给用户以确定漏洞的具体位置。

6.5.3 分析评估

如表 8 所示，本节对 6.5 节所涉及到的嵌入式设备固件模糊测试工具做了一个综合分析评估。由于测试对象、所使用的监控手段和关注的漏洞类型不同，加上有些工具并没有开放源码，所以本文并没有对其漏洞发现能力和精确度做详细的评估，只是对其测试的目标、测试生成方法以及基线 fuzzer 和状态监控机制做了一个综合分析。

分析发现，现有的测试工具可以分为两种：一种是物理设备存在的情况下，从网络协议层面进行测试，另一种是基于仿真/半仿真环境的测试，其测试面可以是应用程序、操作系统和网络协议接口，这与 6.4 节的分析吻合。有意思的是，尽管模糊测试方法最初的设计目的是发现程序中的内存型设计缺陷，但越来越多的模糊测试工具也开始发现固件中的 web

型漏洞[60][66][73]。

从表 8 中可以发现，现有的工具，大多都是能够测出来崩溃或异常，而具体的漏洞确认仍然要依赖于人工分析手段才能获得，究其原因，是缺乏有效的状态监控和错误检测机制导致的。正如 Muench[9]所示，模糊测试或导向执行通常依赖于可观测的崩溃作为程序执行过程中出错的直接影响，桌面系统通常提供了多种保护措施，这些措施会在程序发生错误时触发崩溃，然而，这些措施很少应用在嵌入式设备中(或者应用非常有限)，而且桌面系统崩溃常常会伴随着错误信息一起发生，而嵌入式系统缺乏等价的 I/O 能力。此外，嵌入式设备中的一些安全风险往往是由一些 silent 破坏引起的，目前的监测能力尚无法捕获这样的破坏[9]。据我们所知，目前尚不存在针对嵌入式设备固件 silent 和非 silent 的内存/web 型破坏进行自动化监控和分析的技术手段，这也是下一步研究的重点。

此外，现有的固件模糊测试工具只有少量的针对网络协议的测试考虑了输入结构的生成和变异，大部分的模糊测试更关注的是如何在嵌入式设备固件上应用模糊测试，而应用的效果如何提升尚未有明确的案例说明，也从另一方面证实了，嵌入式设备固件模糊测试的难度和目前尚处在初始阶段的事实。

7 总结与展望

本文围绕着嵌入式设备固件安全分析相关工作，从嵌入式设备面临的攻击、设备固件的获取和自动化解析、设备固件安全分析技术等方面对当前的研究工作进行了全面的分析总结，并对相关工作进行了综合评估。需要说明的是：本文涉及的嵌入式设备固件安全分析技术的分类方法与通用 PC 无异，方法本身的优劣性与其应用环境无关，因此，本文没有对所涉及的安全分析技术进行纵向评估，而只是对所属分析方法相关工作进行了横向分析评估。

分析表明，虽然近年来业界对嵌入式设备及其固件的安全分析技术的研究逐渐兴起，并取得了一定的进展，特别是将 AI 技术应用到二进制固件漏洞的关联，以及基于固件托管的动态分析技术，都体现了新技术与传统安全的融合。然而，相比于通用 PC，嵌入式设备固件安全性分析仍然处于起步阶段，还有许多开放性的工作需要进一步研究。

1、 自动化的固件托管由于可以以纯软件的方式替代硬件，便于插桩和运行时监控以更好地指导测试，成为了理论上最适用于嵌入式设备固件动态分析的手段。然而，目前来看，要实现全自动化的固件托管仍然有很长的道路。
2、 作为一种折衷方案，借助于内存转发、外设建模、符号执行这样的机制，构建半自动化的固件托管在未来一段时间内必将会是持续的研究热点，但目前的仿真模式大多仅关注 MMIO 和中断，并没有涉及外设的 DMA，DMA 对设备的安全影响程度如何也没有公开的说法。
3、 虽然现有的一些针对固件的模糊测试工具能够针对部分设备固件进行高效的测试，但目前的测试工具在输入生成、状态监控和异常检测等各个方面都存在局限性。特别是随着嵌入式设备的交互性越来越强，测试用例的构建要充分考虑与外设系统有关的输入，而且要考虑漏洞的多样性，丰富状态监控和异常检测机制。

致 谢 感谢所有对本文研究给予支持和帮助的人！

表 8 固件模糊测试工具比较

工具	目标对象	测试层面	模糊测试		基线 fuzzer	状态监控	漏洞类型	漏洞确认	设备类型			源码依赖	硬件依赖	外设	是否开源
			测试例	测试方法					I	II	III				
Muench[6]	设备/板子	网络接口	随机生成	黑盒	boofuzz	自定义的启发式规则	FS/BO/DF/NDP	人工	✓	✓	✓	×	✓	真实/模拟	✓
PeriScope[59]	智能手机	WiFi 驱动	基于生成	灰盒	AFL	内存访问监控	BO/NPD/AD/DF	自动	✓	✓	×	×	✓	真实	✓
RPFuzzer[60]	路由器	SNMP 协议	基于生成&基于变异	黑盒	自研	CPU 利用率、发送监控数据和检查系统日志	空 UDP 包/SNMP 请求包	人工	✓	✓	✓	×	✓	真实	×
Prospect[61]	火警系统	驱动	基于变异	黑盒	自研	debugger VM	N/A	人工	✓	×	×	×	×	真实	×
IoTFuzzer[64]	IoT 设备	网络接口	基于生成	黑盒	自研	heartbeat 机制	BO/NPD/MC	人工	✓	✓	✓	×	✓	真实	×
FirmAFL[65]	仿真固件	应用程序	基于随机	灰盒	AFL	N/A	MC	人工	✓	×	×	×	×	N/A	✓
SRFuzzer[66]	soho routers	管理接口	基于生成	灰盒	自研	基于响应、基于代理、基于信号	MC/CI/XSS/ID	人工	✓	✓	✓	×	✓	真实外设	×
FirmCorn[67]	IoT 设备	应用程序	基于生成&基于变异	灰盒	N/A	内存破坏检查和异常检测	BO/MC	人工	✓	×	×	×	✓	N/A	✓
P2IM[68]	MCU 设备	接口	随机生成	灰盒	AFL	段追踪启发式	MC/IO/ETT	人工	×	✓	✓	×	×	外设建模	✓
HAL-fuzz[69]	HALs 的设备	接口	基于交互	灰盒	AFL	类 ASAN 的清洗器	MC/BO/DoS	人工	×	×	✓	×	×	仿真	✓
Partemu[70]	TrustZone TAs	TAs	基于生成	灰盒	AFL	N/A	MC/BO/NPD	人工	✓	×	×	×	×	仿真	×
Charm[71]	智能手机	驱动	基于模板	灰盒	syzkaller	KASAN	NPD/UAF/OOB/DBZ	人工	✓	×	×	✓	✓	真实	✓
FirmFuzz[73]	IoT 设备	网络接口	基于生成	灰盒	自研	监控器	CI/BO/XSS/NPD	人工	✓	×	×	×	×	N/A	✓

CI：命令注入，MC：内存破坏，BO：缓冲区溢出，XSS：跨站脚本，NPD：空指针解引用，ID：信息泄漏，DoS：拒绝服务，DF：double free，FS：格式化字符串，IO：整数溢出，NP：网络包，ETT：错误的类型转换，OOB：越界，DBZ：divide-by-zero，UA：unaligned access，AD：地址泄漏，DF：double fetch，N/A：表示论文中未提及

参考文献

[1] Margolis J , Oh T T , Jadhav S , et al. An In-Depth Analysis of the Mirai Botnet[C]// International Conference on Software Security & Assurance. 2017.

[2] V. Alimi, S. Vernois, and C. Rosenberger, “Analysis of embedded applications by evolutionary fuzzing,” in International Conference on High Performance Computing & Simulation (HPCS). IEEE, 2014.

[3] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. Robert- son, F. Ulrich, and R. Whelan, “LAVA: Large-scale automated vulnerability addition,” in 37th IEEE Symposium on Security and Privacy (SP), 2016.

[4] Bellard F. QEMU, a fast and portable dynamic translator[C]//USENIX Annual Technical Conference, FREENIX Track. 2005, 41: 46.

[5] Magnusson P S, Christensson M, Eskilson J, et al. Simics: A full system simulation platform[J]. Computer, 2002, 35(2): 50-58.

[6] Marius Muench, Jan Stijohann, Frank Kargl, Aurélien Francillon,and Davide Balzarotti. What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices. In Network and Distributed System Security Symposium (NDSS), 2018

[7] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In Network and Distributed System Security Symposium (NDSS), 2016.

[8] Mengjia Yan, Read Sprabery, Bhargava Gopireddy,et al. Attack Directories, Not Caches: Side Channel Attacks in a Non-Inclusive World[C]// Attack Directories, Not Caches: Side Channel Attacks in A Non-inclusive World. 0.

[9] Zhou W, Jia Y, Yao Y, et al. Phantom Device Attack: Uncovering the Security Implications of the Interactions among Devices IoT Cloud and Mobile Apps[J]. arXiv. org, 2019.

[10] Li H C, Liang P H, Yang J M, et al. Analysis on cloud-based security vulnerability assessment[C]//2010 IEEE 7th International Conference on E-Business Engineering. IEEE, 2010: 490-494.

[11] Sicato S, Costa J, Sharma P K, et al. VPNFilter malware analysis on cyber threat in smart home Network[J]. Applied Sciences, 2019, 9(13): 2763.

[12] Basnight Z, Butts J, Lopez Jr J, et al. Firmware modification attacks on programmable logic controllers[J]. International Journal of Critical Infrastructure Protection, 2013, 6(2): 76-84.

[13] Cui A, Costello M, Stolfo S. When firmware modifications attack: A case study of embedded exploitation[J]. 2013.

[14] Bencsáth B, Buttyán L, Paulik T. XCS based hidden firmware modification on embedded devices[C]//SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks. IEEE, 2011: 1-5.

[15] I. D. Foster, A. Prudhomme, K. Koscher, and S. Savage, “Fast and Vulnerable: A Story of Telematic Failures,” in 9th USENIX Workshop on Offensive Technologies (WOOT), 2015.

[16] Kolias C, Kambourakis G, Stavrou A, et al. DDoS in the IoT: Mirai and other botnets[J]. Computer, 2017, 50(7): 80-84.

[17] Costin A, Zaddach J, Francillon A, et al. A large-scale analysis of the security of embedded firmwares[C]//23rd {USENIX} Security Symposium ({USENIX} Security 14). 2014: 95-110.

[18] Ghafoor I, Jattala I, Durrani S, et al. Analysis of OpenSSL heartbleed vulnerability for embedded systems[C]//17th IEEE International Multi Topic Conference 2014. IEEE, 2014: 314-319.

[19] Liu M, Zhang Y, Li J, et al. Security analysis of vendor customized code in firmware of embedded device[C]//International Conference on Security and Privacy in Communication Systems. Springer, Cham, 2016: 722-739.

[20] Fawaz K, Kim K H, Shin K G. Protecting privacy of {BLE} device users[C]//25th {USENIX} Security Symposium ({USENIX} Security 16). 2016: 1205-1221

[21] Li C, Cai Q, Li J, et al. Passwords in the air: Harvesting wi-fi credentials from smartcfg provisioning[C]//Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks. 2018: 1-11.

[22] Artenstein N. Broadpwn: Remotely compromising Android and iOS via a bug in Broadcom’s Wi-Fi chipsets[J]. Black Hat USA, 2017.

[23] Beniamini G. Over the air–vol. 2, pt. 3: Exploiting the Wi-Fi stack on Apple devices[J]. 2017.

[24] Duflot L, Perez Y A, Valadon G, et al. Can you still trust your network card[J]. CanSecWest/core10, 2010: 24-26.

[25] Duflot L, Perez Y A, Morin B. Run-time firmware integrity verification: what if you can’t trust your network card[J]. CanSecWest/-core11, Vancouver (Canada), 2011: 9-11.

[26] Kupfer G, Tsafrir D, Amit N. IOMMU-resistant DMA attacks[D]. Computer Science Department, Technion, 2018.

[27] Vaccari I, Cambiaso E, Aiello M. Remotely Exploiting AT Command Attacks on ZigBee Networks[J]. Security and Communication Networks, 2017, 2017.

[28] Naik N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP[C]//2017 IEEE international systems engineering symposium (ISSE). IEEE, 2017: 1-7.

[29] Rahman R A, Shah B. Security analysis of IoT protocols: A focus in CoAP[C]//2016 3rd MEC international conference on big data and smart city (ICBDSC). IEEE, 2016: 1-7.

[30] Andy S, Rahardjo B, Hanindhito B. Attack scenarios and security analysis of MQTT communication protocol in IoT system[C]//2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI). IEEE, 2017: 1-6.

[31] Yassein M B, Shatnawi M Q, Aljwarneh S, et al. Internet of Things: Survey and open issues of MQTT protocol[C]//2017 International Conference on Engineering & MIS (ICEMIS). IEEE, 2017: 1-6.

[32] Liu Y, Wang H. Tracking mirai variants[J]. Virus Bulletin, 2018.

[33] Hwang R H, Peng M C, Nguyen V L, et al. An LSTM-Based Deep Learning Approach for Classifying Malicious Traffic at the Packet Level[J]. Applied Sciences, 2019, 9(16): 3414.

[34] 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 ({USENIX} Security 19). 2019: 1133-1150.

[35] Zhou W, Cao C, Huo D, et al. Logic Bugs in IoT Platforms and Systems: A Review[J]. arXiv preprint arXiv:1912.13410, 2019.

[36] Beniamini G. Over The Air: Exploiting Broadcom's Wi-Fi Stack.(2017)[J]. Retrieved May, 2017, 19: 2017.

[37] Costin A, Zarras A, Francillon A. Automated dynamic firmware analysis at scale: a case study on embedded web interfaces[C]//Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. 2016: 437-448.

[38] Périgaud F, Gazet A, Czarny J. Subverting your server through its BMC: the HPE iLO4 case[J]. Recon Brussels, 2018.

[39] Alrawi O. Security Evaluation of Home-Based IoT Deployments[J]. 2019.

[40] Li Deng, Yin Qing, Lin Jian, et al. Firmware Vulnerability Detection in Embedded Device Based on Homology Analysis[J]. Computer Engineering, 2017, 43(1) : 72-78 (in Chinese)

李登, 尹青, 林键, 等. 基于同源性分析的嵌入式设备固件漏洞检测[J]. 计算机工程, 2017 (1): 72-78.

[41] Pewny J, Garmany B, Gawlik R, et al. Cross-architecture bug search in binary executables[C]//2015 IEEE Symposium on Security and Privacy. IEEE, 2015: 709-724.

[42] Eschweiler S, Yakdan K, Gerhards-Padilla E. discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code[C]//NDSS. 2016.

[43] Chen Y, Li H, Zhao W, et al. IHB: A scalable and efficient scheme to identify homologous binaries in IoT firmwares[C]//2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). IEEE, 2017: 1-8.

[44] Chang Qing, LiuZhongjin, Wang MengTao, 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 (in Chinese)

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.

[45] Shirani P, Collard L, Agba B L, et al. Binarm: Scalable and efficient detection of vulnerabilities in firmware images of intelligent electronic devices[C]//International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2018: 114-138.

[46] David Y, Partush N, Yahav E. FirmUp: Precise static detection of common vulnerabilities in firmware[J]. ACM SIGPLAN Notices, 2018, 53(2): 392-404.

[47] 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.

[48] Xu X, Liu C, Feng Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 363-376.

[49] Gao J, Yang X, Fu Y, et al. VulSeeker: a semantic learning based vulnerability seeker for cross-platform binary[C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 2018: 896-899.

[50] Gao J, Yang X, Fu Y, et al. Vulseeker-pro: Enhanced semantic learning based binary vulnerability seeker with emulation[C]//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018: 803-808.

[51] Gao J, Yang X, Jiang Y, et al. Semantic Learning Based Cross-Platform Binary Vulnerability Search For IoT Devices[J]. IEEE Transactions on Industrial

Informatics, 2019.

[52] Thomas S L, Chothia T, Garcia F D. Stringer: measuring the importance of static data comparisons to detect backdoors and undocumented functionality[C]//European Symposium on Research in Computer Security. Springer, Cham, 2017: 513-531.

[53] Thomas S L, Garcia F D, Chothia T. HumIDIFy: a tool for hidden functionality detection in firmware[C]//International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2017: 279-300.

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

[55] Thomas S L, Francillon A. Backdoors: Definition, Deniability and Detection[C]//International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, Cham, 2018: 92-113.

[56] Thomas S L. Backdoor detection systems for embedded devices[D]. University of Birmingham, 2018.

[57] Wang D, Zhang X, Ming J, et al. Resetting Your Password Is Vulnerable: A Security Study of Common SMS-Based Authentication in IoT Device[J]. Wireless Communications and Mobile Computing, 2018, 2018.

[58] Hu Chaojian, Xue Yibo, Zhao Liang, et al. Backdoor detection in embedded system firmware without file system[J]. Journal on Communications, 2013, 34(8): 140-145(in Chinese)

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.

[59] Song D, Hetzelt F, Das D, et al. PeriScope: An Effective Probing and Fuzzing Framework for the Hardware-OS Boundary[C]//NDSS. 2019.

[60] 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).

[61] Kammerstetter M, Platzer C, Kastner W. Prospect: peripheral proxying supported embedded code testing[C]//Proceedings of the 9th ACM symposium on Information, computer and communications security. 2014: 329-340.

[62] Mulliner C, Golde N, Seifert J P. SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale[C]//USENIX Security Symposium. 2011, 168.

[63] Van Den Broek F, Hond B, Torres A C. Security testing of GSM implementations[C]//International Symposium on Engineering Secure Software and Systems. Springer, Cham, 2014: 179-195.

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

[65] 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 ({USENIX} Security 19). 2019: 1099-1114.

[66] Zhang Y, Huo W, Jian K, et al. SRFuzzer: an automatic fuzzing framework for physical SOHO router devices to discover multi-type vulnerabilities[C]//Proceedings of the 35th Annual Computer Security Applications Conference. 2019: 544-556.

[67] Gui Z, Shu H, Kang F, et al. FIRMCORN: Vulnerability-Oriented Fuzzing of IoT Firmware via Optimized Virtual Execution[J]. IEEE Access, 2020, 8: 29826-29841.

[68] Feng B, Mera A, Lu L. P 2 IM: Scalable and Hardware-independent Firmware Testing via Automatic Peripheral Interface Modeling[J].

[69] Clements A A, Gustafson E, Scharnowski T, et al. HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation[J].

[70] Harrison L, Vijayakumar H, Padhye R, et al. PARTEMU: Enabling Dynamic Analysis of Real-World TrustZone Software Using Emulation[C]//Proceedings of the 29th USENIX Security Symposium (USENIX Security 2020)(To Appear). 2019.

[71] Talebi S M S, Tavakoli H, Zhang H, et al. Charm: Facilitating dynamic analysis of device drivers of mobile systems[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 291-307.

[72] 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 (SECUWARE). 2016.

[73] 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.

[74] Vasile S, Oswald D, Chothia T. Breaking All the Things—A Systematic Survey of Firmware Extraction Techniques for IoT Devices[C]//International Conference on Smart Card Research and Advanced Applications. Springer, Cham, 2018: 171-185.

[75] Shwartz O, Mathov Y, Bohadana M, et al. Opening Pandora's box: effective techniques for reverse engineering IoT devices[C]//International Conference on Smart Card Research and Advanced Applications. Springer, Cham, 2017: 1-21.

[76] FERRER M E. Towards automatic firmware extraction from update applications[J]. 2018.

[77] Hemel A, Kalleberg K T, Vermaas R, et al. Finding software license violations through binary code clone detection[C]//Proceedings of the 8th Working Conference on Mining Software Repositories. 2011: 63-72.

[78] Heffner C. Binwalk: Firmware analysis tool[J]. URL: https://code. google. com/p/binwalk/(visited on 03/03/2013), 2013.

[79] Hemel A. Better unpacking binary files using contextual information[J]. 2019.

[80] Cui A. Embedded Device Firmware Vulnerability Hunting Using FRAK[J]. Black Hat USA, 2012.

[81] Zaddach J, Kurmus A, Balzarotti D, et al. Implementation and implications of a stealth hard-drive backdoor[C]//Proceedings of the 29th annual computer security applications conference. 2013: 279-288.

[82] Li Q, Feng X, Wang R, et al. Towards fine-grained fingerprinting of firmware in online embedded devices[C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018: 2537-2545.

[83] 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 (DSN). IEEE, 2018: 430-441.

[84] Zhang L, Chen J, Diao W, et al. CryptoREX: Large-scale Analysis of Cryptographic Misuse in IoT Devices[C]//22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019). 2019: 151-164.

[85] Zhu Ruijin, Zhang Baofeng, Mao Junjie, et al. Determining Image Base of ARM Firmware Based on Matching String Addresses[J]. Acta Electronica Sinica, 2017, 45(6): 1475-1482(in Chinese)

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.

[86] Pro I D A. IDA PRO[J]. 2011.

[87] Shoshitaishvili Y, Wang R, Salls C, et al. Sok:(state of) the art of war: Offensive techniques in binary analysis[C]//2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016: 138-157.

[88] Hoglund M G. Fuzzy hash algorithm: U.S. Patent 8,484,152[P]. 2013-7-9.

[89] Celik Z B, Babun L, Sikder A K, et al. Sensitive information tracking in commodity IoT[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 1687-1704.

[90] Zheng Y, Song Z, Sun 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 (IPCCC). IEEE, 2019: 1-8.

[91] Dai Zhonghua, Fei Yongkang, Zhao Bo, et al. Research on the localization of firmware vulnerability based on stain tracking.[J]. Journal of Shangdong University(Natural Science), 2016, 51(9): 41-46(in Chineses)

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.

[92] Redini N, Machiry A, Wang R, et al. KARONTE: Detecting Insecure Multi-binary Interactions in Embedded Firmware[C]//2020 IEEE Symposium on Security and Privacy (SP). 431-448.

[93] 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.

[94] Davidson D, Moench B, Ristenpart T, et al. {FIE} on firmware: Finding vulnerabilities in embedded systems using symbolic execution[C]//Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13). 2013: 463-478.

[95] 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.

[96] Hernandez G, Fowze F, Tian D, et al. Firmusb: Vetting USB device firmware using domain informed symbolic execution[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2245-2262.

[97] Corteggiani N, Camurati G, Francillon A. Inception: System-wide security testing of real-world embedded systems software[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 309-326.

[98] Chen T, Zhang X S, Ji X L, et al. Test generation for embedded executables via concolic execution in a real environment[J]. IEEE Transactions on Reliability, 2014, 64(1): 284-296.

[99] Chen T, Zhang X, Zhu C, et al. Design and implementation of a dynamic symbolic execution tool for windows executables[J]. Journal of Software: Evolution and Process, 2013, 25(12): 1249-1272.

[100] Yao Y, Zhou W, Jia Y, et al. Identifying Privilege Separation Vulnerabilities in IoT Firmware with Symbolic Execution[C]//European Symposium on Research in Computer Security. Springer, Cham, 2019: 638-657.

[101] Zhu, Lipeng, et al. "FIoT: Detecting the Memory Corruption in Lightweight IoT Device Firmware." 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering. IEEE, 2019.

[102] Cui A, Stolfo S J. Defending embedded systems with software symbiotes[C]//International Workshop on Recent Advances in Intrusion Detection. Springer, Berlin, Heidelberg, 2011: 358-377.

[103] Garcia L, Brasser F, Cintuglu M H, et al. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit[C]//NDSS. 2017.

[104] Kevin P. Lawton. Bochs: A Portable PC Emulator for Unix/X[J]. Linux Journal, 1996, 1996(29es):7.

[105] Beckus A. Qemu with an stm32 microcontroller implementation[J]. 2012.

[106] Koscher K, Kohno T, Molnar D. {SURROGATES}: Enabling near-real-time dynamic analyses of embedded systems[C]//9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15). 2015.

[108] 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 ({RAID} 2019). 2019: 135-150.

[109] Zalewski M. American fuzzy lop[J]. 2014.

[110] Muench M, Nisi D, Francillon A, et al. Avatar2: A multi-target orchestration platform[C]//Proc. Workshop Binary Anal. Res.(Colocated NDSS Symp.). 2018, 18: 1-11.

[111] Högl H, Rath D. Open on-chip debugger–openocd–[J]. Fakultat fur Informatik, Tech. Rep, 2006.

[112] Henderson A, Prakash A, Yan L K, et al. Make it work, make it right, make it fast: building a platform-neutral whole-system dynamic binary analysis platform[C]//Proceedings of the 2014 International Symposium on Software Testing and Analysis. 2014: 248-258.

[113] Dolan-Gavitt B, Hodosh J, Hulin P, et al. Repeatable reverse engineering with PANDA[C]//Proceedings of the 5th Program Protection and Reverse Engineering Workshop. 2015: 1-11.

[114] T. Shellphish, “Cyber Grand Shellphish,” Phrack Papers, 2017.

[115] Godefroid P. Random testing for security: blackbox vs. whitebox fuzzing[C]//Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007). 2007: 1-1.

[116] Ganesh V, Leek T, Rinard M. Taint-based directed whitebox fuzzing[C]//2009 IEEE 31st International Conference on Software Engineering. IEEE, 2009: 474-484.

[117] Amini P, Portnoy A. Sulley: Pure python fully automated and unattended fuzzing framework[J]. May, 2013.

[118] Drysdale D. Coverage-guided kernel fuzzing with syzkaller[J]. Linux Weekly News, 2016, 2: 33.

[119] Voss N. afl-unicorn: Fuzzing arbitrary binary code[J]. 2018.

[120] Serebryany K, Bruening D, Potapenko A, et al. AddressSanitizer: A fast address sanity checker[C]//Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12). 2012: 309-318.

[121] Nethercote N, Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation[J]. ACM Sigplan notices, 2007, 42(6): 89-100.

[122] Padala P. Playing with ptrace, Part I[J]. Linux Journal, 2002, 2002(103): 5.

[123] Padala P. Playing with ptrace, part II[J]. Linux J, 2002, 104: 4.

Yu Ying-chao, born in 1983, Ph.D. candidate. Her research interests include operating system security and embedded device security.

Chen Zuo-Ning, born in 1957, Chinese Academy of engineering, Ph.D. Her research interests include Software theory, operating system and information security.

Gan Shui-Tao, born in 1985, Ph.D. His research interests includeSoftware security and vulnerability analysis.

Background

This research belongs to Information Security and Embedded Device Security area, focusing on the research progress of embedded device firmware security. Although security experts and scholars at home and abroad have done some researchs on the security analysis and evaluation technology of embedded device firmware, there is no paper that introduces the latest security research results in detail and comprehensively. In order to solve this problem, this paper analyzes and summarizes the latest research results at home and abroad, and makes a comprehensive analysis and evaluation of related security technologies, focusing on the security risks faced by the firmware of embedded devices.

This work is supported by the National high and new technology research and development plan under Grant No.2018YFB1003600.