物联网设备漏洞挖掘技术研究综述

郑尧文[1,2], 文 辉[2], 程 凯[1,2], 宋站威[2], 朱红松[1,2], 孙利民[1,2]

[1] 中国科学院大学网络空间安全学院 北京 中国 100049
[2] 中国科学院信息工程研究所物联网信息安全技术北京市重点实验室 北京 中国 100093

摘要 随着物联网设备的迅速发展和广泛应用, 物联网设备的安全也受到了严峻的考验。安全漏洞大量存在于物联网设备中, 而通用漏洞挖掘技术不再完全适用于物联网设备。近几年, 针对物联网设备漏洞的挖掘技术逐渐成为热点。本文将分析物联网设备漏洞挖掘技术面临的挑战与机遇, 然后从静态分析, 动态模糊测试, 以及同源性分析三个方面来介绍物联网设备漏洞挖掘技术的研究进展。最后本文将对今后该领域的研究重点和方向进行讨论和展望。

关键词 物联网设备; 漏洞挖掘; 静态分析; 模糊测试; 同源性分析
中图法分类号 TP309.1 DOI 号 10.19363/J.cnki.cn10-1380/tn.2019.09.06

A Survey of IoT Device Vulnerability Mining Techniques

ZHENG Yaowen[1,2], WEN Hui[2], CHENG Kai[1,2], SONG Zhanwei[2], ZHU Hongsong[1,2], SUN Limin[1,2]

[1] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
[2] Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Abstract With the development of Internet of Things(IoT), its security faces a huge challenge. IoT devices are prone to lots of vulnerabilities while current software vulnerability mining techniques could not be directly applied to them. Vulnerability mining techniques on IoT devices has attracted researchers' attention in these years. In this paper, we will introduce challenges and opportunities of IoT vulnerability mining techniques, and then summarize the techniques from aspects of static analysis, dynamic fuzz testing and homology analysis techniques. Finally, we will discuss the research direction in the future.

Key words Internet of Things devices; vulnerability mining; static analysis; fuzzing testing; homology analysis

1 引言

随着物联网时代的来临, 物联网设备如网络摄像头、可穿戴设备、活动追踪器、智能汽车、智能家居等终端设备得以迅速发展和广泛应用。根据 Gartner 的报告[1], 物联网设备数量在 2020 年将超过 200 亿。同时, 针对物联网设备的安全攻击事件不断攀升。主要攻击方式是利用设备漏洞获取设备控制权限, 进而传播大规模恶意代码以控制网络空间, 或利用漏洞窃取用户信息数据、劫持网络流量进行其它黑客地下产业交易。国家信息安全漏洞共享平台对 2016 年收录的物联网设备漏洞(含通用软硬件漏洞以及针对具体目标系统的事件型漏洞)进行了统计, 物联网设备漏洞呈逐年增长趋势。根据阿里移动安全团队 2015 年统计, 90%的物联网设备存在弱密钥、缓冲区溢出漏洞[2]。近几年物联网设备大量漏洞被相继披露。如: 2013 年黑帽大会上, Heffner[3]展示多款网络摄像头的溢出类、口令硬编码、命令注入类漏洞, 涉及到 DLink、TPLink、Linksys、Trendnet 设备厂商。攻击者利用这些漏洞可以进行非授权登入, 劫持摄像头实时画面。此后各种类型的物联网设备(小到智能家居, 如智能灯泡、恒温器、路由器等, 大到电动车、含有无线网络和娱乐系统的飞机)相继在黑客大会上被披露存在严重漏洞。此外, 因安全漏洞引发的真实安全事件也层出不穷。2012 年伊朗采用 GPS 干扰的方式俘获美国哨兵无人机。2016 年 10

通讯作者: 文辉, 博士, 助理研究员, Email: wenhui@iie.ac.cn。
本课题得到广东省重点研发计划(No.2019B010137004), 国家自然科学基金面上项目(No.U1636120), 国家自然科学基金青年项目(No.61702504), 国家重点研发计划(No.2018YFC1201102)资助。
收稿日期: 2019-06-01; 修改日期: 2019-08-16; 定稿日期: 2019-08-20

月21日, 黑客利用大量被Mirai病毒感染的物联网设备, 发动针对Dyn管理的DNS服务器的DDoS攻击, 影响范围涵盖美国东海岸、西海岸和欧洲部分地区, 导致Twitter、GitHub、亚马逊、PayPal、BBC、华尔街日报等很多知名网站无法访问。总体来说, 针对物联网设备安全漏洞的攻击, 不仅会造成个人隐私的泄露, 更会造成人身财产的损失, 严重的甚至会威胁到整个网络空间安全性。因此, 针对物联网设备的漏洞挖掘迫在眉睫。

针对物联网设备漏洞能够成功攻击的原因, 主要分为: (1)物联网设备在设计和开发的过程中, 主要考虑其功能性的实现, 而在设计上忽略了安全性的考虑, 在开发的过程中因疏忽引入漏洞, 并且缺乏后期安全性的检查; (2)由于设备硬件资源受限, 难以在设备上部署安全防护策略, 导致漏洞更容易被攻击者利用。由于物联网设备本身有对可移动、体积小的需求, 导致第二个原因暂时无法克服。因此, 最直接有效的方法是在设计和开发过程中, 引起对安全性的重视, 或是产品上线后及时发现物联网设备中潜在漏洞。

物联网设备存在大量未知漏洞, 为设备自身以及网络空间带来了很多潜在的威胁。尽管政府部门和安全研究团队认识到物联网设备自身漏洞带来的网络安全风险, 以及加强对物联网设备漏洞挖掘的迫切性, 但目前仍缺少行之有效的物联网设备漏洞挖掘技术手段。市面上虽有较为丰富的关漏洞挖掘的技术、产品及研究团队, 但大多是针对通用系统(Windows、Linux、Mac、Android)及其软件。在物联网设备漏洞挖掘方面, 由于不同厂商的物联网设备在软硬件方面差异巨大、物联网设备源码、文档不公开等原因, 难以构建物联网漏洞分析模型以及建立统一的动态仿真环境, 难形成高效、自动化、批量化的物联网设备漏洞发现方法。因此, 目前的物联网设备漏洞大多通过安全人员手工分析发现。

虽然通用平台的漏洞挖掘技术和工具不能完全适用于物联网设备, 但大部分程序分析技术、安全测试思路和方案仍可用于物联网设备。因此, 物联网设备漏洞挖掘技术仍然以通用程序分析和安全测试技术作为基础, 结合物联网设备特点, 开展了相关漏洞挖掘技术的研究。同时, 通用漏洞挖掘技术仍然在蓬勃发展, 给物联网设备漏洞挖掘技术不断注入新的思路。

近几年, 物联网设备漏洞挖掘技术与通用软件漏洞挖掘技术相辅相成, 发展现状主要概括为以下几个方面: 静态分析、动态模糊测试、同源性分析。静态分析主要以物联网设备固件作为分析对象, 对固件进行解析, 并建立特定漏洞类型的分析规则, 进一步使用静态程序分析技术挖掘漏洞。动态分析通过对真实运行设备或模拟仿真系统进行测试从而发现漏洞。基于同源性分析的漏洞发现是因物联网设备大量复用开源项目带来的新兴研究方向。从2014年到现在, 仅仅5年时间, 即从第一篇基于文件粒度的漏洞关联, 到基于函数粒度的漏洞关联, 发展到现在基于深度学习的高效漏洞关联。总体来说, 物联网设备漏洞挖掘技术仍处于起步阶段, 有很多开放性的研究问题。

因此, 我们在第二章论述当前通用漏洞挖掘技术的现状, 并理清物联网设备系统与通用软件存在差异性, 在此基础上分析出物联网设备漏洞挖掘技术存在的机遇和挑战; 之后, 在三、四、五章分别介绍静态分析、动态模糊测试、同源性分析三种物联网设备漏洞挖掘技术; 第六章将总结全文并展望。

2 物联网设备漏洞挖掘的挑战和机遇

2.1 通用漏洞挖掘技术

通用漏洞挖掘技术根据分析对象主要分为两大类: 基于源代码的漏洞挖掘和基于二进制的漏洞挖掘[18]。基于源代码的分析通常采用静态分析方案, 先建立特定漏洞检测规则, 并采用数据流分析、污点分析、符号执行等技术完成相应规则检测, 从而实现漏洞的挖掘。由于二进制代码通常是可执行的, 因此基于二进制的方案分为静态、动态、动静结合的。静态二进制分析方案需要首先将二进制代码转换成汇编代码, 或是进一步转换成统一表示的中间语言。之后可以通过基于模式的漏洞分析或是二进制代码比对技术实现静态的漏洞挖掘。动态二进制分析方案主要是采用模糊测技术。根据输入制导的方式可分为白盒、灰盒以及黑盒测试。动静结合的方案主要将静态分析的结果用于辅助动态测试。

2.2 物联网设备漏洞挖掘的挑战

由于物联网设备存在硬件资源受限、硬件复杂异构, 代码、文档未公开的问题, 物联网设备的漏洞挖掘存在较大的挑战。

硬件资源受限性: 通用动态二进分析技术需要在运行程序外围实施监控分析。由于物联网设备存储资源(存储)的受限性, 无法部署相关的分析模块, 导致动态分析技术无法适用。同时硬件CPU的计算能力有限, 会造成动态分析性能下降。

硬件的复杂异构: 一方面, 物联网设备 CPU 架构与通用平台不同, 导致程序指令集上存在差异性。通用软件的指令架构通常是 X86 或 X86_64。而物联网程序通常采用 ARM、MIPS、PowerPC 等嵌入式架构。因此直接基于通用 CPU 指令汇编的静态分析方案不再适用。另一方面, 物联网设备外围 I/O 硬件存在多样化, 增加动态分析技术的适配难度。

代码、文档未公开: 对于通用软件, 可以对源代码或者二进制程序进行分析以挖掘漏洞。而对于物联网设备而言, 大多数程序均为定制的商业化程序, 通常只能将设备固件作为分析对象, 只有少量设备有相应的源代码以用于分析。因此, 大量源代码分析技术不再适用于物联网设备漏洞挖掘。另外固件的组织形式和数据内容也区别于通用程序, 因此基于二进制的静态分析技术也不能直接使用。

2.3 物联网设备漏洞挖掘的机遇

物联网设备自身的特点不仅为漏洞挖掘带来挑战, 同时也带来新的机遇。

系统交互的丰富性: 虽然是对物联网设备进行漏洞挖掘, 但物联网设备通常会与终端、云等系统进行交互, 因此设备本身存在更多的攻击面。对于动态二进制分析方案, 可以充分利用外部交互系统的信息, 对新的攻击面的进行测试分析。

组件代码的大量复用: 物联网设备程序在开发的过程中, 为了节省开发成本, 大量使用开源的第三方库, 致使大量第三方组件的漏洞也存在于物联网设备中。之前基于二进制比对的静态分析技术主要是通过不同层次信息(控制流、程序块、指令级)的差异性来发现安全漏洞, 而现在可以通过不同层次信息的相似性来挖掘同源漏洞。

漏洞类型的趋同性: 通用软件漏洞类型包含内存破坏类(栈溢出、堆溢出、空指针应用、二次释放等)、输入验证类(命令注入等)、配置错误类等。漏洞存在的位置可以是在内核、驱动、用户态服务程序中。而对于含有操作系统的物联网设备固件来说, 同样存在这些类型的漏洞。因此无论是静态二进制还是动态二进制分析技术, 通用的漏洞检测规则仍然适用于物联网设备固件和程序。

2.4 物联网设备漏洞挖掘技术的发展

面对物联网设备漏洞因代码、文档未公开的分析挑战, 基于二进制的漏洞挖掘技术相比于基于源码的技术发展更加迅速。其中, 基于二进制的漏洞挖掘技术分别在静态和动态技术上都有所突破, 并在一定程度上有效地解决因硬件资源受限性(动态), 硬件的复杂异构(静态, 动态)带来的分析挑战。同时, 系统交互的丰富性也为静态、动态分析技术注入新的思想, 有效提升漏洞挖掘的精确性和效率。此外, 物联网设备中组件代码的大量复用为漏洞挖掘提出新的思路, 即通过同源性分析实现漏洞挖掘。相关内容我们将在第五章进行介绍。

除此之外, 对于物联网设备的硬件漏洞, 如近几年发现的幽灵和熔断问题, 其漏洞挖掘方法不在本文的讨论范围之内。对于通过收集设备与服务程序指纹、操作系统版本等信息, 并与漏洞库中的信息进行比较从而检测出漏洞的方法, 我们认为是漏洞检测技术, 不属于漏洞挖掘技术包含的范围。

3 静态分析技术

通用静态程序分析技术的分析对象是源代码或二进制代码。然而, 由于物联网设备程序是商业程序, 很少有厂商公开源代码和文档, 通常只能获取固件以用于静态分析。固件是物联网设备中的软件系统, 实现对设备特定硬件底层的控制。固件通常包含操作系统、文件系统、用户程序, 或者本身就是一个可执行程序。因此, 静态程序分析技术从物联网设备固件开始, 通过逆向设备固件, 建立漏洞模型, 并利用程序分析技术完成漏洞挖掘。在物联网设备数量和种类呈爆炸式增长的趋势下, 静态漏洞挖掘方法对设备固件代码采用统一的中间表达, 能够有效规避硬件复杂性、架构多样性带来的挑战。

根据当前的静态分析工作, 我们总结分析流程如图 1 所示。整个分析流程可以分为两个步骤: 目标程序提取与信息恢复和基于程序分析的漏洞发现。步骤一是从固件中提取出待分析的目标程序, 并恢复出目标程序的语法、结构信息。步骤二是建立漏洞分析规则, 基于已获得的程序信息, 通过程序分析技术, 实现漏洞的发现。此外, 符号执行技术能够有效辅助静态分析。

3.1 目标程序提取与信息恢复技术

从固件获取开始, 根据固件的不同类型, 需要选择性的完成程序集提取、目标程序提取、程序表示、执行信息恢复等分析步骤。

固件获取: 该步骤通过多种方式获得固件以用于后续分析。获取方式主要分为两大类: 直接从物联网设备中提取, 或是从厂商的官网进行下载。对于第一大类, Vasile 等[4]总结了两种方式。第一种是通过 UART 或 JTAG 硬件调试接口登入系统并读取内存, 从而恢复出固件内容; 第二种是直接利用读写器完成 flash 芯片内固件的直接提取。对于第二大类, 由于物联网设备需要进行功能升级和漏洞补丁, 厂商

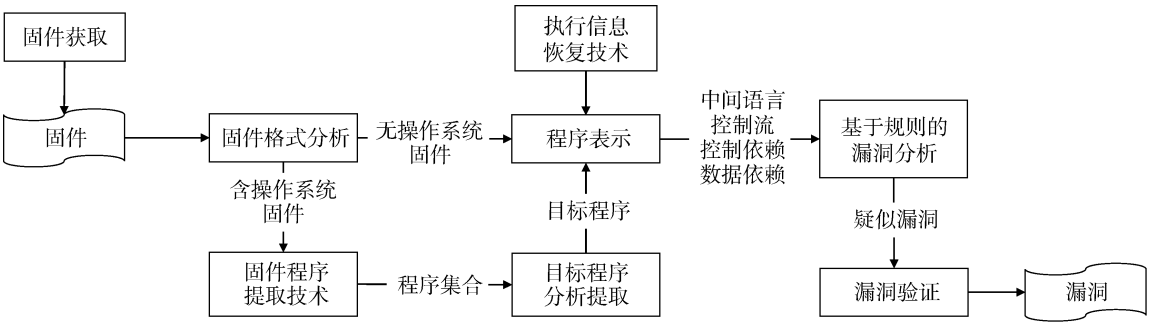

图 1　静态分析流程

Figure 1　Workflow of static analysis

会在网上提供固件信息。因此, 我们可以通过撰写爬虫程序对各个厂商官网的下载中心进行固件爬取。其中, 固件分析工具 Firmadyne 的数据库[5]已提供超过 11000 固件的信息和下载地址。

固件格式分析: 该步骤分析固件的组织形式和数据内容。固件按照组织形式通常分为两类: 包含操作系统的固件和不包含操作系统的固件。在包含操作系统的固件中, 大部分设备功能由用户程序完成, 系统调用操作、程序加载过程、库函数调用均有规范的定义。其中, 操作系统可以是通用操作系统(如定制化的 Linux), 也可以是特定嵌入式操作系统(如实时操作系统 VxWorks)。在不包含特定操作系统的固件中, 固件直接工作在设备上, 包含了简单的任务调度等类操作系统功能。无论是哪一类型的固件, 首先需要判断固件是否被压缩。若固件压缩, 需先解压, 然后再进一步分析。

固件程序提取技术: 该步骤是从包含操作系统的固件中提取所有的文件和程序。对包含操作系统的固件, 由于操作系统和文件系统通常都在头部含有特征码或特征字符串, 因此可以通过基于签名匹配的方式识别出操作系统镜像和文件系统, 并进一步对文件系统进行解压, 提取出所有文件和程序。目前 Binwalk 工具[6]可以很好地支持固件格式识别, 操作系统、文件系统的识别与分离, 文件系统的解析和提取, 并且允许特定格式分析插件的加载与使用。Firmware Mod Kit 工具[7]可以支持 Sasquash 和 Jefferson 等嵌入式文件系统的解析提取。此外, Binary Analysis Toolkit[8]也能很好地支持通用操作系统固件的解析。当前工具可以支持大部分包含操作系统固件的程序提取, 但对于自定义的文件系统, 以及厂商在标准实现上做了修改的文件系统, 当前工具将无能为力。文献[9]对因 SquashFS 头部特征修改导致 Firmware-Mod-Kit 不可用的问题做了深入分析, 并实现对修改后 SquashFS 文件系统的解析。

目标程序分析提取: 对含有操作系统的固件, 该步骤对提取出的所有程序和文件集合, 进一步提取出待分析的目标程序。当前, 目标程序的提取主要是通过搜寻特定文件名实现。若目标程序是 Web 服务程序, 只需搜索文件名包含 http 的文件, 即可找到目标程序。这种方法通常直接且有效, 但对于大规模固件的目标程序提取, 具有较低的召回率。Thomas 等[10]提出基于半监督学习的固件中二进制程序功能的分类器, 能够对 cron、dhcp、ftp、ntp-client、nvram-get-set、ping、tcp、telnet、upnp、web-server 十类服务程序在零误报率的条件下实现 96.45%的识别精确率。该工作的数据集仅包含 100 个固件, 无法证明在大规模数据集上的有效性。因此, 大规模固件的特定目标服务程序提取技术仍需深入研究。

程序表示技术: 该步骤是将二进制代码以中间语言或汇编等形式表示。无论是对于无操作系统的固件或是待分析的目标服务程序, 均需将二进制转换成汇编代码。有时为了屏蔽汇编指令格式的差异性, 需进一步转化为统一的中间语言。此外, 还需提取出程序的控制流、函数调用、字串常量信息等信息。目前 IDA pro[11]可以支持 X86、ARM、MIPS、PowerPC 等多种指令格式二进制汇编代码的转换, 涵盖大量物联网设备的指令集。同时可以恢复出物联网设备服务程序和无操作系统固件的控制流, 提取函数调用关系、字符串引用、地址访问等信息。Shoshitaishvili 等提出的 Angr 工具[12], 可以将目标程序转化为统一的 VEX 中间语言用于进一步分析。

执行信息恢复技术: 该步骤将恢复程序的加载地址、入口地址等执行信息。对于包含操作系统的固件, 其目标服务程序的文件格式(如广泛采用 Linux 下的 ELF 可执行文件格式)通常是标准、公开的, 逆向技术已经相当成熟。而对于无操作系统的固件程序, 由于缺少固件执行的描述信息, 导致程序信息恢复不完整, 无法进一步实现精准的程序分析。

缺失的信息包括: (1)固件运行时的加载地址。这将导致很多跳转无法被解析; 即使已恢复出指令信息, 控制流信息仍不完整。(2)程序入口地址。这将导致无法深入分析。Shoshitaishvili 等[13]针对该问题提出了基本的解决方案。在加载地址分析部分, 由于间接跳转表具有数值连续特征, 通过识别间接跳转表, 再根据间接跳转指令寻址与表之间的关系, 推断出较为准确的程序加载地址。对于程序入口地址, 首先根据特定指令架构的函数入口、返回指令特征, 识别出所有的函数, 并构造函数调用关系。对于没有被调用的函数(可能是中断处理函数), 将其入口地址识别为整个程序可能的入口地址。

目前, 目标程序提取与信息恢复技术能够有效解决包含操作系统的物联网设备固件的分析。从固件解压, 文件系统解析, 可执行程序解析到可执行程序的逆向, 技术基本上已经成熟。而对于无操作系统额设备固件分析, 整个固件就是特殊的可执行程序, 对该类固件的分析仍处于起步阶段。

3.2 基于程序分析的漏洞发现技术

在目标程序提取与信息恢复之后, 通过建立漏洞分析规则, 并结合静态程序分析技术, 实现漏洞的挖掘。除了通用漏洞类型(如污点类漏洞), 物联网设备漏洞挖掘更关注特定漏洞类型和模块, 如硬编码、认证绕过等后门漏洞。该漏洞类型属于物联网设备特定漏洞, 通常为了方便设备调试人员进入系统或进行特权操作。其在物联网设备上普遍存在, 而在通用程序上较为少见。后续我们将介绍针对物联网设备特定漏洞类型和模块的研究工作进展。

针对硬编码漏洞类型, Thomas[14]等提出基于静态数据分析的漏洞挖掘方法。该工作首先识别出程序中的静态数据比对函数, 通过提取函数特征并对静态数据比对函数进行建模来识别。接着通过程序控制流分析技术来判定静态数据比对的重要性(根据它影响代码块的独特性), 之后进一步评估出函数的重要性。通过进一步对重要性函数的静态分析, 发现了硬编码的认证后门漏洞, 并恢复出 FTP、SOAP 协议的重要指令集。该工作具有很强的实用性, 然而在后门漏洞挖掘与基于文本的协议指令恢复的这两个应用中, 均不能保证准确性和召回率。因此, 该方案在大规模测试中的适用性无法得到保证。为了进一步识别物联网设备固件程序中的后门, Thomas[10]利用半监督学习的方法, 构造了识别固件中二进制功能的分类器。并通过自定义二进制函数功能描述语言, 识别二进制中的非预期功能(后门)。实验表明有96.45%的二进制功能的识别准确率, 并在 Tenda 设备中挖掘出后门漏洞。

针对认证绕过漏洞类型, Shoshitaishvili 等[13]提出基于程序分析的物联网设备认证绕过漏洞的挖掘方法。首先, 该工作人为定义程序特权点(包括未认证的情况下不能输出的数据、未认证情况下不可执行的系统操作、对内存绝对地址的访问、人工分析出的相关特权函数)。在此基础上, 构建固件代码的控制流、数据依赖、控制依赖图, 然后基于切片技术生成到特权点的路径, 并采用符号执行技术求解路径条件, 从而发现物联网设备认证绕过漏洞。该工作采用切片技术对程序路径进行简化, 使符号执行分析有较高的分析性能。然而, 该工作无法解决程序被混淆之后的分析难题。

针对通用污点类漏洞类型, Cheng 等[15]提出物联网设备固件污点类漏洞的检测方法, 主要通过函数分析提取变量描述、定义对、数据类型等信息, 并解决函数数据流分析、指针别名分析、数据结构恢复等难题。该工作发现了 8 个已知漏洞和 13 个 0day 漏洞, 且在时间开销上优于 Angr。

针对协议解析模块, Cojocar 等[16]首次提出基于协议解析模块识别的物联网设备漏洞挖掘方法。该工作通过协议解析模块离散特征的提取以及分类器的构造, 实现协议解析模块的精确识别, 并聚焦于该类模块的漏洞挖掘, 发现 GPS 接收器、电表、硬盘驱动、程序逻辑控制器(PLC)多类型物联网设备的漏洞。在此基础上, Zheng 等[17]提出基于多维度特征的应用层协议解析模块的识别技术, 并与基于二进制属性图的可疑脆弱点推断技术相结合, 实现污点类漏洞的快速发现。

目前基于程序分析的漏洞发现技术能够有效的针对特定漏洞类型(硬编码、认证绕过、污点类)和模块(协议解析模块)。然而这些研究工作大多需要人工辅助, 自动化程度不足。此外, 研究工作的分析对象仍然是用户程序。在针对物联网设备特定实时操作系统方面, 仍然缺少相应的漏洞挖掘工作。

3.3 符号执行技术

符号执行技术是一种用符号值替代具体值执行程序的技术, 能够有效辅助静态分析技术。符号执行技术首先将程序输入或是关注且无法确定的变量用符号值表示, 然后根据程序控制流进行传播, 将之后的变量表示为符号值和常量构成的表达式[18]。当程序执行不同路径的时候, 会生成对符号值的约束, 可通过约束求解来分析出路径执行的条件。由于程序漏洞通常可以建模为相关变量不满足特定的约束, 因此可以利用符号执行技术辅助漏洞挖掘。

Chipounov 等[19]提出针对系统行为进行分析的平台 S2E, 通过选择性的符号执行技术和可变执行一致性模型, 实现对私有系统所有执行路径(包括内核空间、用户空间)的分析。因此, 该工具具备对物联网设备固件分析的能力。

Davidson 等[20]提出基于 KLEE 符号执行引擎[21]构建的开源工具 FIE, 可用于 MSP430 微处理固件的自动化漏洞检测。该工具的核心思想是通过符号化的方式, 建立硬件行为(中断等)的统一描述模式, 并进行符号执行分析。在此基础上, 通过状态剪枝和内存污染的方式, 提升分析覆盖率并保证能够分析简单固件程序的所有路径。该工具支持内存破坏类和外围 I/O 误用两类漏洞。但仍存在以下限制: (1)由于使用 KLEE 符号执行引擎, 该工具需要固件程序的源代码; (2)由于符号执行和设备真实执行存在差异性或是程序的误配置, 导致分析出来的所有漏洞需进一步人工确认; (3)循环带来的路径和状态爆炸问题在该工具中依然没有得到有效解决。

Shoshitaishvili 等[13]提出针对固件二进制的符号执行分析工具。该工具基于常用的符号执行引擎 Angr,构建的模型可用于检测认证绕过漏洞, 包括硬编码、认证接口后门、特权访问无认证等漏洞。该工具采用 Z3 约束求解器[22]完成符号约束求解。然而, 该工作存在以下缺陷: (1)需要人工提供安全策略以用于检测, 因此无法运用于大规模自动化分析; (2)固件可通过混淆技术来对抗程序切片技术, 或是使用复杂操作使符号执行技术失效。

Corteggiani 等[23]提出针对现实世界物联网设备的系统级安全分析框架。目前, 物联网设备包含有源代码和二进制代码, 而源代码通常是混合了人工手写的汇编代码。在二进制分析中, 由于程序语义缺失, 直接进行程序分析存在精度不足的问题。在源代码分析中, 由于与汇编代码的混合, 造成源代码分析工具不能直接适用。为了解决这类问题, 该框架首先将高级语言源代码、人工编写汇编、二进制库以及部分硬件行为合并成 LLVM 中间语言字节码, 不仅保留了源代码的语义信息, 也保证了安全检查的准确性。然后基于 KLEE 符号执行引擎, 提出支持内存抽象、硬件交互、面向中断的嵌入式设备符号执行引擎。最后, 该框架提出高性能 JTAG 调试器, 支持内存访问向真实硬件的重定向。实验结果显示, 该系统能够有效支持 Arm Cortex-m3 芯片。

目前, 针对物联网设备的符号执行技术在源代码、二进制乃至混合代码的分析上有一定的进展。然而当前技术均需要针对特定硬件类型构建相应的符号执行引擎, 仍受限于特定的外围 I/O 硬件。此外符号执行技术在物联网设备固件分析上还存在以下局限性: (1)部分符号执行技术的分析对象是源代码, 而物联网设备的源代码通常不被厂商公开, 因此无法应用这些技术; (2)符号技术能运用于物联网设备漏洞的挖掘, 但仍需要人工定义安全策略或漏洞模型, 因此无法大规模运用于物联网设备的特定漏洞挖掘; (3)由于符号执行技术在分析复杂程序过程中, 可能面临路径和状态爆炸的情况, 因此, 程序本身也可以通过添加复杂操作来对抗符号执行技术对后门的挖掘; (4)符号执行技术在某些场景仍然难以完成分析(多个输入解, 或是密码 hash 之后进行的硬编码)。

4 模糊测试技术

模糊测试[24]是针对软件和系统非常有效的漏洞挖掘方法, 也是物联网设备动态分析最广泛使用的技术。通过向被测试对象发送随机输入, 并通过观察其行为(通常是程序崩溃), 从而发现潜在漏洞。

模糊测试根据程序执行信息的获取情况, 通常分为白盒、灰盒、黑盒测试。黑盒测试将测试对象当作黑盒子, 不获取任何程序执行信息来制导测试样例生成, 仅按照指定的规约随机生成输入样例。代表工具有 boofuzz[25](前身是 Sully[26])和 Peach[27]。白盒测试[28]深度获取程序执行信息来制导测试样例的生成, 通常会对程序源代码进行动态污点分析或符号执行以获取精确的程序执行和状态信息。代表的工具有 QSYM[29]。灰盒测试通过轻量级的监控技术, 以获得程序的特定执行信息来制导输入的生成。相比于白盒测试, 轻量级的程序监控方法使灰盒测试的吞吐率更高(通用程序甚至能达到上千次/秒), 因此能更高效的发现漏洞。代表工具有 AFL[30]、LibFuzzer[31]、honggfuzz[32]。

基于覆盖率制导的模糊测试技术是灰盒测试的一种, 主要是在监控端对程序进行插桩来收集程序执行覆盖度信息, 从而制导输入的生成。核心思想是将能发现新路径的输入当作种子去生成新的输入, 而对于不能发现的输入则丢弃。AFL 是常用的基于覆盖率制导的灰盒测试工具。由于物联网设备程序通常没有相应的源代码, 只有二进制代码用于分析。因此灰盒测试只能对 QEMU[33] 用户态模式下的代码块翻译过程进行插桩来获取覆盖率信息。

通过对已有物联网设备模糊测试工作的调研和分析, 总结系统架构如图 2 所示。其中虚线框内是模糊测试的必要组件, 而非虚线框中是可选组件, 用于进一步提高模糊测试性能。在必要组件中, 目标程

序是测试和漏洞挖掘的对象, 是可执行二进制代码, 类别包括服务程序、浏览器、操作系统、编译器、第三方库等。其接收输入数据的形式是命令行、文件读入、网络协议等。测试样例生成器通过对原始样例数据(种子)进行随机变换, 生成相应的输入递送给目标程序。异常状态检测技术(4.3)发现目标程序异常, 通常是检测到程序崩溃, 并对触发崩溃的输入进一步分析, 从而发现潜在的漏洞。

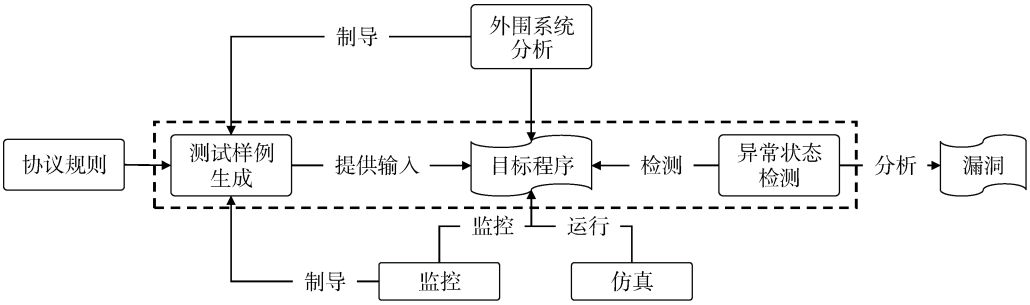

图 2　物联网设备模糊测试框架

Figure 2　Overview of IoT fuzzing

非必要组件主要用于提高模糊测试的性能。其中, 新型攻击面测试技术(4.1)针对物联网设备特有的协议通信方式, 系统内核特有机制以及内核与外围的新型交互方式, 提出新型攻击面的模糊测试方法。监控端是灰盒测试需要的组件, 实时记录目标程序的执行状态, 将信息反馈给测试样例生成器, 制导其输入的生成。设备仿真技术(4.2)为物联网设备的灰盒测试提供监控的支持。由于物联网设备程序对系统和硬件资源都有较强的依赖性, 多种设备仿真技术被提出。外围系统分析技术(4.4)通过分析外部交互系统(终端、云等), 提取协议交互的语法和语义信息, 制导测试样例生成器产生针对目标程序语法结构合法的输入, 防止测试样例生成器随机产生的输入在程序解析初期被丢弃掉, 以至于无法实现深度测试目标程序的目的。后续我们将介绍物联网设备模糊测试各组件的研究进展。

4.1　新型攻击面测试技术

相比通用系统与软件, 物联网设备拥有特有的协议交互方式, 以及内核与其他组件的新型交互方式。物联网设备丰富的交互方式引入了新型攻击面, 同时也促进对这些攻击面模糊测试方法的研究。

特殊协议攻击面: 物联网设备除了通用协议之外, 存在其它特有的协议交互接口, 而这些接口也注定成为新型攻击面。因此, 针对这些协议交互接口的模糊测试是值得研究的问题。Barreaud 等[34]提出基于模糊测试的智能卡嵌入式 Web 服务漏洞挖掘的方法。该工具基于 Peach 模糊测试引擎实现了针对服务程序和 SIM 卡之间通信协议 BIP(Bearer Independent Protocal)的安全测试。具体实现包含了将 Physcard 库结合以实现与 SIM 卡的通信, 使用 XML 文件对 BIP 协议进行建模, 在文件中添加<Expected>和<Response>标记来实现监控等。实验结果显示很多 SIM 卡并没有严格遵循欧盟通信标准协会的标准, 存在很多拒绝服务漏洞。

Lancia 等[35]提出针对 EMV 智能卡的漏洞挖掘方案。该方案基于 Sully 开源模糊测试框架, 结合 Triton 库实现与智能卡的通信及模糊测试。在实验部分, 该方案实现 EMV 的对照系统, 并让模糊测试系统将同样的命令发送给智能卡和对照系统, 最终根据两者返回结果的不一致性来检测出智能卡的异常。此外, 文献[36]和[37]也提出通过生成智能卡的正常行为白名单来检测可能存在的漏洞。由于上述工作在模糊测试输入生成的部分仍然采用盲目随机生成的机制, 因此难以快速产生触发漏洞的输入。因此, Alimi 等[38]采用基因算法生成输入用例来测试智能卡, 保证漏洞挖掘的高效性。

面对车辆控制系统, Lee 等[39]提出了针对 CAN 上 UDS 协议的模糊测试方案。通过模型驱动, 消息块粒度的输入生成以及细粒度的异常监控策略, 成功挖掘车辆 ECU 中的漏洞。

内核的新型攻击面: 物联网设备的攻击面除了用户空间与外界丰富的协议接口之外, 还存在内核与系统组件的交互接口。其中, 交互接口不仅包含了系统调用边界, 还存在内核与其他新型组件的接口。Out Of Memory Killer(OOM killer)作为内核组件, 通过杀死进程的方式解决系统内存不足的问题。通常系统允许程序申请比系统可用更多的内存, 然而会以一定概率导致系统内存资源耗尽。在这种情况下, 系统会启动 OOM killer 将特定进程杀死来解决内存不足的问题。由于物联网设备的物理内存空间有限

且无额外的交换空间, 因此当大量内存请求发生的时候, 内核会启动 OOM killer。若 OOM killer 进程杀死失败, 系统会一直处在 OOM 状态, 使得内核出现无响应状态。Sim 等[40]研究针对该组件的模糊测试方案, 通过不断请求特定内存大小, 来发现 OOM killer 失败致使系统无响应的漏洞。实验结果显示, 若测试用例申请比系统可用更多的内存, 会导致漏洞触发的概率更高。此外, 模糊测试使用适应性随机算法比纯随机算法能更快地触发漏洞。

由于半虚拟化技术在物联网设备仿真领域的发展, OKL4 微内核也得到了广泛应用。Gauthier 等[41]提出对微内核 OKL4 的模糊测试方案。该方案通过对 OKL4 系统调用建模(包括对输入的限制), 从而发现虚拟管理程序的真实漏洞。

此外, 物联网设备的内核还存在与外围 I/O 交互的边界。而这为攻击者带来新的攻击方式, 如 17 年爆出的逾 9 成用户受到 iOS 设备 Wifi 芯片漏洞。攻击者不仅可以在同一 Wifi 网络下远程控制受害设备的 Wifi 芯片, 还可以进一步攻破内核。Song 等[42]提出针对该边界的模糊测试方案。该方案通过对 Wifi 驱动-内核边界相关的内存区域进行模糊测试, 从而发现漏洞。具体来说, 由于硬件数据传输主要通过内存映射 I/O(MMIO)和直接内存存取(DMA)两种方式。因此, 该方案首先在内核中加入钩子函数来获取两种数据传输方式的空间初始化信息, 从而判断驱动代码地址访问是否属于系统-硬件边界的数据。然后, 该方案对边界内存内区域的读写指令进行监控, 通过将内存页设置为无效, 跳转到页错误处理函数, 从而获取区域读写相关指令的信息(指令地址、读写类型、读写长度、内存区域类型、源寄存器中地址、目的寄存器中地址等), 并通过修改读写数据, 实现对驱动-内核边界模糊测试的功能。

4.2 设备仿真技术

对物联网设备进行灰盒测试, 设备仿真技术是不可缺少的。若不采用设备仿真技术, 而对硬件设备直接进行灰盒测试, 则需要采用硬件监控的机制来获取执行信息。由于在硬件监控机制上, 仅有通用平台的芯片(如 INTEL)实现了程序执行跟踪功能, 而物联网设备芯片(ARM、MIPS 等)不支持, 因此无法对真实设备直接进行灰盒测试。另外, 厂商很少提供物联网设备程序的源代码, 直接在源代码中插桩监控代码也不现实。因此, 只能基于设备仿真技术, 并对仿真器进行插桩以实现灰盒测试。

在用户态仿真的工作中, AFL 模糊测试工具[30]采用 QEMU 动态翻译模拟器[33], 按照代码块粒度对目标程序指令进行翻译和执行, 保证了程序执行的高效性。其中, 指令翻译是将目标程序的指令(通常是 ARM、MIPS 等)翻译成宿主机的指令(通常是 X86)。但对于物联网设备程序, 其依赖的系统环境和硬件设备通常与宿主机不一致。当目标程序进行系统调用时, QEMU 将其转化为宿主机对应的系统调用并进行执行。由于目标程序依赖的系统环境和宿主机存在不一致性, 系统调用结果也会不同, 导致后续代码执行发生错误。

在混合仿真的研究工作中, 为了解决物联网设备程序的正确执行问题, Zaddach 等[43]提出混合仿真系统 Avatar, 通过更好的硬件支持实现物联网设备固件的动态仿真。其核心思想是连接模拟器(QEMU)和真实硬件进行混合执行, 即控制模拟器端完成固件指令的翻译和执行, 并将 I/O 操作引导到真实设备端完成。在此基础上, Avatar 使用符号执行技术发现嵌入式设备的硬编码漏洞。然而, 该工作存在以下局限性: 由于符号执行技术的引入以及软硬件之间的频繁交互, 性能相比于真实设备执行下降很多; 且难以对设备的外围 I/O 进行自动化的连接使用。为了克服 Avatar 仿真的时效性差, Koscher 等[44]提出基于 FPGA 桥接器串行连接的物联网设备调试方法, 实现物联网设备的近实时动态仿真和分析。

在系统态仿真的研究工作中, 为了克服模糊测试等动态分析技术对硬件的依赖性, Chen 等[45]提出针对 Linux 内核物联网设备固件的大规模全系统仿真平台 Firmadyne, 通过修改内核和驱动来添加硬件支持以实现物联网设备的全系统仿真。该工作主要解决了定制化外围 I/O、非易失性内存、动态文件等仿真技术难题。该系统支持 ARM 和 MIPS 两种 CPU 架构, 并完成 9486 固件的解压和 74 个漏洞利用脚本的测试。在此基础上, Costin 等[46]提出物联网设备 Web 接口的测试方法, 发现 45 个固件 Web 接口的未知漏洞。在系统仿真的基础上, TEMU[47]通过采用客户机驱动来获取客户机的运行状态信息。DECAF[48]与 TEMU 不同, 通过对 QEMU 进行插桩来访问客户机的内核数据结构, 从而实时获取客户机的进程、线程、代码块、符号等信息。该系统支持 ARM、MIPS、X86 CPU 架构以及 windows、Linux 操作系统。

在仿真工作的基础上, 黑盒测试相关工作被提出。Muench 等[49]比较了各种不同配置下的黑盒测试效率, 包括本地执行(直接测试真实设备)、半仿真、全系统仿真。其中, PANDA[50]提供其仿真技术。通过实验发现, 由于桌面系统处理器比物联网设备处理器快, 全系统仿真比本地执行有更高的执行性能。然

而, 测试吞吐率最快也没有超过 15 个/秒。

另一方面, 基于系统仿真的灰盒测试研究工作也有一定的进展。TriforceAFL[51]将 AFL 与 QEMU 系统态仿真结合, 实现全系统仿真下的模糊测试。其中完整实现了对系统仿真下内核的灰盒测试。Zheng 等[52]在 TriforceAFL 的基础上, 利用 DECAF 的客户机进程实时监控技术, 实现对物联网设备用户程序的模糊测试技术。同时, 为了解决全系统仿真性能较差的问题, 有效结合 QEMU 用户态程序仿真的高性能和系统态仿真的高兼容性, 提出基于内存映射共享技术和系统调用重定向技术的增强进程仿真机制, 并与 AFL 测试引擎结合, 在保证测试准确性的前提下, 大幅提高灰盒测试的性能。

4.3 异常状态检测技术

通用程序的异常状态检测通常是基于监测程序是否发生崩溃来判定。而对于物联网设备程序, 异常状态的检测方法通常是由程序的工作形式决定。若模糊测试是对真实设备进行, 则异常状态检测通过存活性探测来进行判定。其中包含协议栈(ICMP、TCP、UPD 等)可用性探测、硬件信号检测等。若模糊测试是对仿真平台进行, 则异常状态检测除了使用存活性探测方法之外, 还可以通过对仿真系统内存的实时监测来判断。

Chen 等[53]提出针对物联网设备的黑盒测试方案。在异常检测方面, 通过分析物联网设备的响应报文来判断是否触发了崩溃。根据传输协议类型分两个策略。基于 TCP 的通信, 通过 APP 与物联网设备指定服务的连接状态来判断; 基于 UDP 的通信, 事先从 APP 中提取出用于检测物联网设备端服务程序存活性的心跳消息, 为了不影响模糊测试性能, 每十次测试插入一个心跳消息来判断物联网设备端程序是否崩溃。

Muench 等[54]首先分析了传统异常状态检测方法(程序显式崩溃)对于物联网设备程序的普适性。通过在四类设备(通用桌面系统、基于通用操作系统的设备、基于特定嵌入式操作系统的设备、无操作系统设备)上人工构造多类型内存破坏类漏洞(格式化字符串、栈溢出、堆溢出、二次释放、空指针引用)并进行测试, 发现除通用桌面系统漏洞触发时产生显示的崩溃之外, 其他三类系统在漏洞被触发后会产生其他现象。实验结果如表 1, 会出现重启、挂起、一段时间后崩溃、继续执行(数据发生错误)以及继续正常执行(无任何异常)现象。因此传统基于程序显式崩溃的异常状态检测方法不再适用于物联网设备程序内存破坏类漏洞的发现。

表 1 文献[54]中不同系统平台的内存破坏效果

Table 1 memory corruption of different platform in [54]

漏洞类型	平台			
	桌面系统	类型 1	类型 2	类型 3
格式化字符串	√	√	×	×
栈溢出	√	√	√	! (挂起)
堆溢出	√	! (延迟崩溃)	×	×
二次释放	√	√	×	×(数据错误)
空指针引用	√	√	√(重启)	×(数据错误)

为了检测这三类物联网设备的 5 类内存破坏类漏洞, 该工作提出段访问监测、格式化标识监测、堆对象监测、函数调用栈监测、函数帧监测、栈对象监测 6 位一体的异常状态检测方案。实验结果表明组合的异常检测策略能完全检测出 3 类物联网设备的 5 类内存破坏类漏洞。相比于存活性探测, 组合的异常检测策略产生的性能开销基本可以接受。

4.4 外围系统分析技术

静态分析目标程序能够提取出程序协议的语法和语义信息, 能够有效制导模糊测试构造合法输入, 从而提高模糊测试的深度和覆盖度。然而, 物联网设备固件有时难以获取, 即使可以从官网上获取或者是设备中提取, 有时也存在加密、压缩格式未知的难题, 因此无法进一步分析设备程序。物联网设备除了自身嵌入的服务程序之外, 在外部还存在多类型的系统进行交互, 如移动终端、云等。因此, 除了通过对目标程序进行分析之外, 还可以对外围交互系统进行程序分析, 挖掘出合法的协议交互方式, 有利于模糊测试的深度漏洞挖掘。

在外围系统分析的研究中, Chen 等[53]提出 APP 分析与模糊测试相结合的物联网设备内存破坏类漏洞挖掘的工具 IoTFuzzer。由于很多物联网设备带有官方 APP, 且 APP 中含有丰富的协议信息(URL、命令、加解密信息), 通过自动化分析相关移动 APP 中的数据流信息, 更好的了解未知通信协议, 从而生成更好的测试样例来发现漏洞。整个 APP 分析分为两个步骤: UI 分析和数据流分析。UI 分析主要是找到 UI 元素到网络 API 之间的程序路径, 即找到网络 API 相关的事件; 数据流分析采用污点跟踪技术, 以用户输入、硬编码字符串、系统 API 作为污点源, 而将网络 API 和加密函数作为敏感点, 通过污点传播

分析, 识别出协议字段以及接受协议字段的函数。在模糊测试部分, 针对 APP 分析发现的字段, 对参数字段进行变异。策略包括: 针对字符串参数, 通过更改长度以尝试发现堆栈溢出漏洞; 对数值参数, 更改值以发现整数溢出漏洞; 另外改变值的类型或者提供空值。实验挖掘出 9 个设备中 15 个多类型内存破坏类漏洞(4 个空指针引用, 5 个栈溢出, 2 个堆溢出, 4 个未知漏洞)。同时, 与 Sulley, BED 两个模糊测试系统做了对比, 由于预先做了 APP 分析来制导输入生成, 挖掘效率有了很大的提高。

IoTFuzzer 存在一定的局限性, 包括: (1)物联网设备需要有相关的 APP 用于分析, 而对于很多物联网设备来说, 没有对应的 APP; (2)需要真实物理设备用于连接测试; (3) 仍然存在性能问题, 吞吐率没有超过 1 个/秒; (4)该工具通过崩溃检测来发现漏洞, 然而在文献中也提到, 很多溢出类漏洞不一定造成程序崩溃; (5)测试的代码覆盖率只包含处理与 APP 交互的那部分代码。

在物联网设备的动态分析技术中, 模糊测试是最广泛使用和最高效的技术之一。针对通用程序模糊测试工作主要集中在测试用例生成的策略上。通过收集到的程序执行信息制导模糊测试, 以实现高覆盖度、深度测试等不同的模糊测试目标。而这些技术突破可以直接迁移到物联网设备的模糊测试工作中。相反, 针对物联网设备模糊测试工作主要解决其自身特点引入的技术挑战, 从而实现有效乃至高效的模糊测试。除了模糊测试引擎, 以及用于制导模糊测试输入生成的监控模块与通用模糊测试基本一致以外, 新型攻击面测试, 异常检测, 外围系统分析, 设备仿真都存在严峻挑战。在新型攻击面测试方面, 需要研究物联网设备特殊协议和内核新型攻击面的测试技术。在物联网设备异常检测方面, 仅仅通过外部存活性探测的方式已不能准确判断异常, 需要综合程序堆栈状态和硬件输出等多层级行为来准确判断。在外围系统分析方面, 主要是针对传统的基于输入随机变异的方式不再适用物联网设备的问题。需要借助对外部交互软件的分析, 提取出协议语法信息, 从而实现基于生成的模糊测试, 提高模糊测试的测试深度。在设备仿真技术方面, 需要支持物联网设备的精准执行以及执行信息的提取。目前, 可以支持比较多的是网络摄像头、路由器等基于 Linux 内核的固件, 对于其他非 Linux 内核以及无操作系统固件, 还支持的较少。此外, 由于可仿真的物联网设备数量非常有限, 不利于模糊测试技术的评测, 因此可以通过文献[55]中的技术来插入真实的漏洞作为评测对象。

5 同源性分析技术

物联网设备固件的开发通常复用了大量第三方开源组件, 如 OpenSSL 等。这就导致在不同厂商、类型、 CPU 架构的设备固件中, 存在着大量由同一源代码编译而成的二进制代码。因此, 研究人员开始探究基于同源二进制代码相似性比较的大规模漏洞发现的方法, 并取得了较大的进展。

同源性分析技术主要基于固件中文件、程序、函数、代码块的相似性, 实现跨平台、跨架构的同源漏洞发现。目前, 主要分为基于二进制文件的粗粒度相似性比较和基于代码片段(函数)细粒度相似性比较两大类。

5.1 基于文件相似性比较

同源性分析技术最初是基于文件粒度的相似性分析, 从而实现的漏洞的快速发现。Costin 等[56]第一次进行公开、大规模的固件安全性静态分析, 在 693 个固件中发现 38 个未知漏洞; 通过相似文件关联, 发现某些漏洞感染了 123 不同产品; 证实了某些漏洞感染了网络空间 14 万台设备。该工作是第一次提出通过固件模块关联的方式发现同源漏洞, 但仍然停留在粗粒度文件相似性比较的方式上, 方案比较单一, 同源漏洞发现的精确度不够。

针对同源二进制文件搜索需要 O(N)时间复杂度的问题, Chen 等[57]提出基于字串敏感哈希的物联网设备固件同源二进制搜索方案。实验结果显示该方案能达到 92.88%的准确率和 2.83%的误报率。由于上述工作对字串信息有较强的依赖性, 造成无法适用于字串信息过少或没有的二进制。Chen 等[58]针对该问题, 提出基于神经网络的二进制非字串特征的编码技术, 并通过局部敏感哈希实现同源二进制的搜索, 相较之前的工作有 15%~20%的性能提升。

5.2 基于代码块(函数)相似性比较

为了能够更加精准地发现同源漏洞, 研究人员提出基于代码片段相似性比较的漏洞发现方法。该类方法相比于文件粒度的方式更加精准。

Lakhotia 等[59]提出基于语义模板的相似代码片段快速关联方法。在此基础上, Pewny 等[60]提出基于语义签名的漏洞关联算法, 将基本块内指令转化为表达式进行相似度匹配, 提高了相同代码架构下漏洞函数的关联精度。为了实现跨 CPU 架构的漏洞发现, Pewny 等[61]将不同指令代码转化为中间语言, 利用 hash 生成漏洞基本块的摘要, 并结合代码控制流

结构进行图匹配, 从而实现精确的同源漏洞发现。然而该工作没有对物联网设备固件代码进行分析, 且需要对每个基本块的输入输出进行特征提取, 存在较大的性能开销。

针对语义特征提取带来的较大性能开销问题, Eschweiler 等[62]提出基于算数和调用指令数量等轻量级语法特征的图匹配方法, 并在匹配前采用函数级特征预分析以提高搜索效率。该工作仍然依赖图匹配模型, 具有较大的开销, 性能瓶颈没有得到彻底解决。因此, Feng 等[63]提出基于控制流图嵌入匹配的固件同源漏洞发现方法, 通过代码库将控制流图编码成特征向量(图嵌入), 并使用局部敏感哈希进行索引, 能够快速发现跨平台物联网设备固件的漏洞。该方案针对 8,126 固件 420,558,702 函数进行一次 bug 查询的平均时长不超过 1s。同时该工作新发现 38 个漏洞, 并证实了其中 23 个漏洞的真实性。针对 D-Link 的两款最新固件, 该方案在 0.1s 内完成 154 个的漏洞查询, 并发现 103 个新漏洞, 证实了其中 16 个漏洞的真实性。该工作实现了一定规模物联网设备同源漏洞的快速发现, 但在训练图嵌入所需的代码库时仍有较大的时间开销, 且代码库质量受训练集影响。针对该问题, Xu 等[64]提出基于神经网络的跨平台相似代码检测方法, 训练时间从 1 周下降到 30 min~10 h, 相较于上述方法能够识别出更多额外的漏洞代码。

此外, 针对物联网设备固件跨平台漏洞关联准确率低的问题, 常青等[65]提出基于神经网络和局部调用结构匹配的两阶段跨平台关联方法。该方法先从函数间调用图、函数内控制流图、函数基本信息三类中选择特征, 然后数值、归一化处理, 接着利用神经网络实现函数对相似性的比对, 并利用结构特征进一步提升精确度。该方法在 OpenSSL 二进制文件上的匹配性能指标 Top1 上有超过两倍的提升, 且在 DLink 路由器固件的漏洞函数关联上有很好的效果。该工作实现了一定规模物联网设备固件的同源漏洞精准发现。

由于深度学习在视频, 图像、语音等领域取得了巨大进展, 研究人员开始探究深度学习在漏洞分析领域的应用。Li 等[66]提出基于深度学习的漏洞检测方案。该方案第一次探究深度学习如何运用于漏洞检测领域。通过调研分析, 先前采用基于机器学习的方法能够构造漏洞检测的分类器, 但仍需人为主观、繁琐的特征定义。该工作将代码片段(由语义相关的代码行构成)转化为向量用于表征程序, 并进行深度学习。实验结果显示在误报率合理的情况下, 该方法能够保证较低的漏报率。虽然该工作没有在物联网设备上进行漏洞检测, 但对物联网设备漏洞同源性分析有一定的指导意义。

物联网设备漏洞的同源性分析从最初的粗粒度文件相似性比对到细粒度的代码块(函数)相似性比对, 从代码块语义特征(输入输出对)提取, 到轻量级语法、结构特征提取, 从基于图结构的匹配到基于图嵌入特征的匹配, 漏洞关联速度和精准度有了很大的提升。目前的技术能够有效提取文件或代码块的特征并进行高效编码, 通过敏感哈希索引等方式, 保证了同源漏洞搜索的性能和精确性。

6 总结与展望

当前的物联网设备的漏洞挖掘技术在静态分析、动态模糊测试、同源性分析技术上都有一定的进展。其中, 静态分析技术能够有效解决固件的解析, 以及固件中通用漏洞的分析问题。但对于物联网设备特定漏洞的高效分析, 仍然缺乏深入的思考和探究。此外, 对于无操作系统和包含特定嵌入式操作系统的固件, 仍然缺少系统性的分析和研究。

对于模糊测试技术, 目前能够实现部分设备(如路由器、网络摄像头)固件的高效仿真和灰盒测试, 从而快速发现漏洞。但对于整个物联网设备生态, 目前的模糊测试技术和工具仍然在输入生成、仿真、监控、异常检测等各个方面存在局限性。

对于同源性分析技术, 目前的技术已支持大规模固件的多层次关联, 从而实现同源漏洞的发现。之后的发展方向应该是针对特定漏洞类型, 有效提取特征并进行编码, 从而实现特定类型同源漏洞的精准, 快速发现。

总体来说, 物联网设备漏洞挖掘技术仍然处于起步阶段, 未来我们仍将从这三大类技术入手, 一方面提出通用性的方法和技术, 另一方面也将针对特定类型的设备、漏洞, 研究相应的挖掘技术。

参考文献

[1] Gartner says 8.4 billion connected “things” will be in use in 2017, up 31 percent from 2016, Gartner. http://www.gartner.com/en/newsroom/.

[2] 阿里移动安全发布《2015 物联网安全年报》, https://www.cnblogs.com/alisecurity/p/5261794.html, 2015.

[3] Exploiting Network Surveillance Cameras Like a Hollywood Hacker, https://media.blackhat.com/us-13/US-13-Heffner-Exploiting-Network-Surveillance-Cameras-Like-A-Hollywood-Hacker-WP.pdf.

[4] Sebastian Vasile, David Oswald, and Tom Chothia, "Breaking All the Things—A Systematic Survey of Firmware Extraction Techniques for IoT Devices," in *Seventeenth Smart Card Research and Advanced Application Conference (CARDIS'18)*, pp. 171-185, 2018.

[5] "Firmadyne datasheet," https://cmu.app.boxcn.net/s/hnpvf1n72uccnhyfe307rc2nb9rfxmjp/folder/6601681737.

[6] "Binwalk," ReFirm Labs, Inc. https://github.com/ReFirmLabs/binwalk.

[7] "Firmware Mod Kit," Bitsum, https://bitsum.com/firmware_mod_kit.htm.

[8] "Binary Analysis Next Generation (BANG)," https://github.com/armijnhemel/binaryanalysis-ng.

[9] "Mucking About With SquashFS," Craig, http://www.devttys0.com/2014/08/mucking-about-with-squashfs/ Aug, 2014

[10] Sam L. Thomas, Flavio D. Garcia, Tom Chothia, "HumIDIFy: A Tool for Hidden Functionality Detection in Firmware," in *International conference on detection of intrusions and malware, and vulnerability assessment(DIMVA'17)*, pp. 279-300, 2017.

[11] "Ida: About - hex-ray," https://www.hex-rays.com/products/ida/index.shtml.

[12] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna, "SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis," in *Proc. of IEEE Symposium on Security and Privacy (S&P'16)*, pp. 138-157, 2016.

[13] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna, "Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware," in *Network and Distributed System Security Symposium (NDSS'15)*, 2015.

[14] Sam L. Thomas, Tom Chothia, Flavio D. Garcia, "Stringer: Measuring the Importance of Static Data Comparisons to Detect Backdoors and Undocumented Functionality." *European Symposium on Research in Computer Security (ESORICS'17)*, pp. 513-531, 2017.

[15] Kai Cheng, Qiang Li, Lei Wang, Qian Chen, Yaowen Zheng, Limin Sun, and Zhenkai Liang, "DTaint: Detecting the Taint-Style Vulnerability in Embedded Device Firmware," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18)*, pp. 430-441, 2018.

[16] Lucian Cojocar, Jonas Zaddach, Roel Verdult, Herbert Bos, Aurélien Francillon, and Davide Balzarotti, "PIE: Parser identification in embedded systems," in *Annual Computer Security Applications Conference (ACSAC'15)*, 2015.

[17] Yaowen Zheng, Kai Cheng, Zhi Li, Shiran Pan, Hongsong Zhu and Limin Sun, "A Lightweight Method for Accelerating Discovery of Taint-Style Vulnerabilities in Embedded Systems," in *International Conference on Information and Communication Security(ICICS'16)*, pp. 27-36, 2016.

[18] 吴世忠, 郭涛, 董国伟, 张普含, "软件漏洞分析技术, " *科学出版社*, 2014.

[19] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea, "S2E: A platform for in-vivo multi-path analysis of software systems," *in International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[20] Drew Davidson, Benjamin Moench, Thomas Ristenpart and Somesh Jha, "FIE on firmware: finding vulnerabilities in embedded systems using symbolic execution," in *USENIX Security Symposium*, pp. 463-478, 2013.

[21] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler, "KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI'08)*, pp. 209-224, 2008.

[22] De Moura, Leonardo Mendonca, and Nikolaj Bjorner, "Z3: an efficient SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, pp. 337-340, 2008.

[23] Nassim Corteggiani, Giovanni Camurati, and Aurélien Francillon, "Inception: System-Wide Security Testing of Real-World Embedded Systems Software," in *USENIX Security Symposium*, pp. 309-326, 2018.

[24] Chen Chen, Baojiang Cui, Jinxin Ma, Runpu Wu, Jianchao Guo, and Wenqian Liu, "A systematic review of fuzzing techniques," *Computers & Security*, pp. 118-137, 2018.

[25] "boofuzz," Pereyda J, https://github.com/ jtpereyda/boofuzz, 2016.

[26] "Sulley: A pure-python fully automated and unattended fuzzing framework," https://github.com/OpenRCE/sulley.

[27] "Peach Fuzzer: Discover unknown vulnerabilities," https://www.peach.tech/products/.

[28] Patrice Godefroid, Michael Y. Levin, and David Molnar. "Automated whitebox fuzz testing," in *Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[29] Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang, and Taesoo Kim, "QSYM: A practical concolic execution engine tailored for hybrid fuzzing," in *USENIX Security Symposium*, 2018.

[30] "American fuzzy lop," M. Zalewski, http://lcamtuf.coredump.cx/afl/.

[31] "LibFuzzer–a library for coverage-guided fuzz testing," http://llvm.org/docs/LibFuzzer.html.

[32] "honggfuzz, a security oriented, feedback-driven, evolutionary, easy-to-use fuzzer with interesting analysis options," http://honggfuzz.com/.

[33] Fabrice Bellard, “QEMU, a fast and portable dynamic translator,” *in Proceedings of the Annual Conference on USENIX Annual Technical Conference* (ATC’05), pp. 41–46, 2005.

[34] Matthieu Barreaud, Guillaume Bouffard, Nassima Kamel, and Jean-Louis Lanet, “Fuzzing on the HTTP protocol implementation in mobile embedded web server,” in *Proceeding of C&ESAR*, 2011.

[35] Lancia, J., Un framework de fuzzing pour cartes à puce: application aux protocoles EMV. In *Symposium sur la Sécurité des Technologies de l’Information et des Communications (SSTIC)*, 2011.

[36] Germain Jolly, Baptiste Hemery, and Christophe Rosenberger. “Generation of Local and Expected Behaviors of a Smart Card Application to Detect Software Anomaly,” in *International Conference on Availability, Reliability and Security*, pp. 474-480, 2015.

[37] Germain Jolly, Sylvain Vernois, and Christophe Rosenberger. “An Observe-and-Detect Methodology for the Security and Functional Testing of Smart Card Applications,” in *International Conference on Information Systems Security(ICISS’16)*, pp. 282-289, 2016.

[38] Vincent Alimi, Sylvain Vernois, and Christophe Rosenberger , “Analysis of embedded applications by evolutionary fuzzing,” in *International Conference on High Performance Computing Simulation (HPCS’14)*, pp. 551–557, 2014.

[39] Hyeryun Lee, Kyunghee Choi, Kihyun Chung, Jaein Kim, and Kangbin Yim, “Fuzzing CAN Packets into Automobiles,” in *International Conference on Advanced Information Networking and Applications*, pp. 817-821, 2015.

[40] Kwan Yong Sim, Feiching Kuo, and Robert G. Merkel, “Fuzzing the out-of-memory killer on embedded Linux: an adaptive random approach,” in *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC’11)*, pp. 387-392, 2011.

[41] Amaury Gauthier, Clement Mazin, Julien Iguchi-Cartigny, and Jean-Louis Lanet, “Enhancing Fuzzing Technique for OKL4 Syscalls Testing,” in *International Conference on Availability, Reliability and Security*, pp. 728-733, 2011.

[42] Dokyung Song, Felicitas Hetzelt, Dipanjan Das, Chad Spensky, Yeoul Na, Stijn Volckaert, “PeriScope: An Effective Probing and Fuzzing Framework for the Hardware-OS Boundary,” in *Network and Distributed Systems Security Symposium (NDSS’19)*, pp. 1-15, 2019.

[43] Jonas Zaddach, Luca Bruno, Aurelien Francillon, and Davide Balzarotti, “AVATAR: A framework to support dynamic security analysis of embedded systems’ firmwares,” in *Network and Distributed System Security Symposium (NDSS’14)*, 2014.

[44] Karl Koscher, Tadayoshi Kohno, and David Molnar, “SURROGATES: Enabling Near-Real-Time Dynamic Analyses of Embedded Systems,” in *USENIX Workshop on Offensive Technologies (WOOT’15)*, 2015.

[45] 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’16)*, 2016.

[46] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. “Automated dynamic firmware analysis at scale: A case study on embedded web interfaces,” in *ACM Asia Conference on Computer and Communications Security (ASIACCS’16)*, 2016.

[47] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena, “A high-level overview covering Vine, TEMU, and Rudder ,” in *Proceedings of the 4th International Conference on Information Systems Security*, 2008.

[48] Andrew Henderson, Aravind Prakash, Lok Kwong Yan, Xunchao Hu, Xujiewen Wang, Rundong Zhou, and Heng Yin, “Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform,” in *International Symposium on Software Testing and Analysis (ISSTA’14)*, July 2014.

[49] Marius Muench, Aurélien Francillon, and Davide Balzarotti, “Avatar2: A multi-target orchestration platform,” in *Workshop on Binary Analysis Research (BAR’18)*, 2018.

[50] Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan, “Repeatable reverse engineering with panda,” in *Proceedings of the 5th Program Protection and Reverse Engineering Workshop (PPREW-5)*, 2015.

[51] “TriforceAFL,” NCC-Group, https://github.com/nccgroup/TriforceAFL, 2017.

[52] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, Limin Sun, “Firm-AFL: High-throughput greybox fuzzing of IoT firmware via augmented process emulation,” in *USENIX Security Symposium*, 2019.

[53] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang, “IoTFuzzer: Discovering memory corruptions in iot through app-based fuzzing,” *in Networked and Distributed System Security Symposium (NDSS’18)*, 2018.

[54] 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’18)*, 2018.

[55] Brendan Dolan-Gavitt, Patrick Hulin, Engin Kirda, Tim Leek, Andrea Mambretti, Wil Robertson, Frederick Ulrich, and Ryan Whelan, “Lava: Large-scale automated vulnerability addition,” in *IEEE Symposium on Security and Privacy(S&P’16)*, pp. 110-121, 2016.

[56] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide

Balzarotti. “A large-scale analysis of the security of embedded firmwares,” in *USENIX Security Symposium*, pp. 95-110, 2014.

[57] Yu Chen, Hong Li, Weiwei Zhao, Lin Zhang, Zhongjin Liu, and Zhiqiang Shi, “IHB: A scalable and efficient scheme to identify homologous binaries in IoT firmwares,” in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, pp. 1-8, 2017.

[58] Yu Chen, Hong Li, Yuan Ma, Zhiqiang Shi, and Limin Sun, “Robust Network-Based Binary-to-Vector Encoding for Scalable IoT Binary File Retrieval,” in *International Conference on Wireless Algorithms, Systems, and Applications(WASA’18)*, pp. 53-65, 2018.

[59] Arun Lakhotia, Mila Dalla Preda, and Roberto Giacobazzi, “Fast location of similar code fragments using semantic ‘juice’,” in *Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop (PPREW’13)*, 2013.

[60] Jannik Pewny, Felix Schuster, Christian Rossow, Lukas Bernhard, and Thorsten Holz, “Leveraging semantic signatures for bug search in binary programs,” in *Proceedings of the 30th Annual Computer Security Applications Conference(ACSAC’14)*, 2014.

[61] Jannik Pewny, Behrad Garmany, Robert Gawlik, Christian Rossow, and Thorsten Holz, “Cross-architecture bug search in binary executables,” in *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[62] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla, “discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code,” in *Network and Distributed Systems Security Symposium (NDSS’16)*, 2016.

[63] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin, “Scalable graph-based bug search for firmware images,” *in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (CCS’16), pp. 480–491, 2016.

[64] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song, “Neural network-based graph embedding for cross-platform binary code similarity detection,” in *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS’17)*, 2017.

[65] Chang Qing, Liu Zhongjin, Wang Mengtao, Chen Yu, Shi Zhiqiang, Sun Limin, “VDNS: An Algorithm for Cross-Platform Vulnerability Searching in Binary Firmware,” *Journal of Computer Research and Development*, vol. 53, no. 10, pp. 2288-2298, 2016.
(常青, 刘中金, 王猛涛, 陈昱, 石志强, 孙利民, “VDNS: 一种跨平台的固件漏洞关联算法”, *计算机研究与发展*, 2016, 53(10): 2288-2298.)

[66] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong, “VulDeePecker: A Deep Learning-Based System for Vulnerability Detection,” *in Network and Distributed System Security Symposium (NDSS’18)*, 2018.

郑尧文 于 2013 年在四川大学计算机科学与技术专业获得学士学位。现在中国科学院大学网络空间安全专业攻读博士学位。研究领域为 IoT 安全，嵌入式设备安全。研究兴趣包括二进制分析，模糊测试，漏洞分析与利用。Email: zhengyaowen@iie.ac.cn

文 辉 于 2016 年在中国科学院大学信息安全专业获得博士学位。现任中国科学院信息工程研究所助理研究员。研究领域为信息安全、数据挖掘。研究兴趣包括：物联网安全、恶意代码检测与分析、数据关联与挖掘。Email: wenhui@iie.ac.cn

程 凯 于 2014 年在西安电子科技大学计算机科学与技术专业获得学士学位。现在中国科学院信息工程研究所网络空间安全专业攻读博士学位。研究领域为 IoT 安全、嵌入式设备安全。研究兴趣包括：二进制逆向、固件安全分析、漏洞挖掘。Email: chengkai@iie.ac.cn

宋站威 于 2017 年在北京工业大学计算机科学与技术专业获得硕士学位。现任中国科学院信息工程研究所研究实习员。研究领域为物联网安全，工控安全。研究兴趣包括二进制分析，模糊测试，漏洞分析与利用。Email: songzhanwei@iie.ac.cn

朱红松 于2009年在中国科学院计算技术研究所计算机体系结构专业获得博士学位。现任中国科学院信息工程研究所研究员。研究领域为网络空间安全。研究兴趣包括：物联网安全、网络对抗、智能攻防，网络空间安全测量和威胁态势感知等。Email: zhuhongsong@iie.ac.cn

孙利民 于1998年在国防科技大学计算机科学与技术专业获得博士学位。现任中国科学院信息工程研究所研究员。研究领域为物联网安全、工控安全。研究兴趣包括：工控系统漏洞挖掘与关联、在线设备发现与识别、工控系统入侵诱捕与行为分析、工控系统入侵检测与监管。Email: sunlimin@iie.ac.cn