引用格式:潘祖烈,王泰彦,周航,等.一种基于导向式模糊测试的 IoT 设备固件漏洞分析方法[J]. 信息对抗技术,2023, 2(1):38-54. [PAN Zulie, WANG Taiyan, ZHOU Hang, et al. Vulnerability analysis method for Internet of Things device firmware based on guided fuzzing[J]. Information Countermeasure Technology, 2023, 2(1):38-54. (in Chinese)]

一种基于导向式模糊测试的 IoT 设备固件漏洞分析方法

潘祖烈[1,2*],王泰彦[1,2],周 航[1,2],郭 徽[1,2]

(1. 国防科技大学电子对抗学院,安徽合肥 230037;
2. 网络空间安全态势感知与评估安徽省重点实验室,安徽合肥 230037)

摘 要 为提高物联网(Internet of Things, IoT)设备漏洞分析的准确度,在深入分析了 50 余个 MIPS 架构的 IoT 设备固件漏洞的基础上,提出了一种基于导向式模糊测试的动静结合 IoT 设备固件漏洞分析方法。获取固件程序中所有函数信息,依据数据引入函数与漏洞触发函数的函数调用关系图,定位危险代码区域。基于危险代码区域详细控制流图,计算执行路径中基本块到达漏洞触发函数的距离,动态调控种子能量,实现面向漏洞触发函数的导向性模糊测试。设计实现了面向 MIPS 架构的 IoT 设备固件漏洞分析系统 DirFirmFuzz。实验结果表明,相较于已有工具,系统漏洞分析的误报率平均缩减了 73.31%,到达漏洞触发函数的平均速度加快了 1.1~7 倍。同时,在实际环境测试过程中,发现了 D-Link、Cisco 等多个厂商的 12 个 0-day 漏洞,均已报送相关厂商进行修补。

关键词 MIPS 架构;物联网设备;固件漏洞分析;模糊测试;轻量级仿真
中图分类号 TP 311 **文章编号** 2097-163X(2023)01-0038-17
文献标志码 A **DOI** 10.12399/j.issn.2097-163x.2023.01.004

Vulnerability analysis method for Internet of Things device firmware based on guided fuzzing

PAN Zulie[1,2*], WANG Taiyan[1,2], ZHOU Hang[1,2], GUO Hui[1,2]

(1. College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China;
2. Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China)

Abstract To increase the accuracy of vulnerability analysis of Internet of Things (IoT) device firmware, an in-depth analysis of more than 50 IoT device firmware vulnerabilities of the MIPS architecture was performed, and a firmware vulnerability analysis combining dynamic and static method based on guided fuzzing was proposed. All the function information in the firmware program was obtained, and the dangerous code area was located according to function call graph of data introducing function and dangerous function. The detailed control flow diagram of dangerous code area was used to calculate the distance from the basic block to vulnerability trigger function, and the seed energy was dynamically adjused to achieve the guided fuzzing of the vulnerability trigger function. The DirFirmFuzz, a firmware vulnerability analysis system for IoT devices based on MIPS architecture was designed and implemented. The experimental results showed that comparing with the existing methods, the false alarm rate

收稿日期:2022-05-09 **修回日期**:2022-06-04
通信作者:潘祖烈,E-mail: panzulie17@nudt.edu.cn
基金项目:国家重点研发计划资助项目(2017YFB0802900)

of system vulnerability analysis could be reduced by 73.31% on average, and the average speed of DirFirmFuzz to reach the vulnerability trigger function was 1.1～7 times faster than that of the existing tools. At the same time, during the real world testing, 12 0-day vulnerabilities from multiple vendors such as D-Link and Cisco were discovered, and all of them have been reported to relevant vendors for patching.

Keywords　MIPS architecture; Internet of Things(IoT) devices; firmware vulnerability analysis; fuzzing; lightweight simulation

0　引言

随着物联网(Internet of Things, IoT)技术的飞速发展，智能路由器、视频监控、智慧医疗、智能家居等终端已经深入到人们生活的方方面面，预计到 2025 年，活跃的 IoT 设备数量将超过 300 亿[1]。然而，在为人们生活带来极大便利的同时，IoT 设备安全性问题也日益凸显[2-4]。根据国家信息安全漏洞共享平台 CNVD[5] 的统计，IoT 设备漏洞数量近年来呈快速增长态势，IoT 设备面临的网络安全威胁日益增加。固件作为 IoT 设备的核心，是一种嵌入在硬件设备中的软件，一般包含操作系统内核及文件系统。固件中存在的漏洞可能会对个人隐私、财产安全和社会安全造成极大的威胁[6]。

与桌面端软件漏洞分析方法[7]相似，随着国内外研究人员关于 IoT 设备固件漏洞的研究逐渐深入，当前主要包含有静态分析[9]、模糊测试[10]和符号执行[11]3 类 IoT 设备固件漏洞分析技术。

静态分析是一种无需在真实设备或仿真器上执行程序即可检测程序中漏洞的方法。IoT 设备相关的服务功能实现程序一般存在于文件系统中。静态分析过程中通常对获取的固件进行解压分析，提取其中的目标程序作为静态分析对象，常用的静态分析工具包括 Angr[12]、Stringer[13]、Firmalice[14]、DTaint[15]等。但是静态分析缺少程序运行时信息，从而导致误报率过高，并且普遍存在人工参与程度过高，自动化程度不足的问题，严重阻碍了大规模自动化测试。

模糊测试(Fuzzing)是通过向目标程序提供非预期的输入并执行程序，监控程序的运行状态，同时记录并进一步分析目标程序发生的异常来发现潜在漏洞的漏洞分析技术，代表性工具有 AFL[17]、LibFuzzer[18]和 honggfuzz[19]。当前各类研究的 Fuzzing 技术代码覆盖率还有待提升，而且通过随机变异生成的测试用例无法保证有效性，导致效率低下。导向式模糊测试工具如 AFLGo，通过使用智能型算法，能够在一定程度上提高覆盖率，但面临着同传统的模糊测试工具 AFL 一样的问题，即无法完全真实模拟 IoT 固件程序，因此 Fuzzing 技术在固件漏洞分析领域的应用效果还有待进一步提高。

符号执行技术是一种采用抽象符号替代具体值执行程序的技术，能够将外界输入或是需要动态加载的变量用符号值表示[20]，然后基于控制流图，对程序的执行空间进行遍历，并结合条件约束求解进行漏洞分析。但是由于通常情况下源代码不被厂商公开，导致针对源代码的符号执行技术无法应用，而且 IoT 设备程序结构通常较为复杂，在符号执行过程中经常面临路径爆炸的问题。

针对以上提到的诸多问题，本文深入分析了 50 余个 MIPS 架构的 IoT 设备固件漏洞，固件均来自于提供 Web 管理网络服务的路由器设备，发现大多数存在漏洞的固件程序中，漏洞代码中数据引入函数和漏洞触发函数之间距离相对较近，且通常同属一个直接调用函数。基于以上观察，结合距离导向的概念，将测试执行过程中覆盖的不同基本块对种子变异的影响作为种子能量分配的重要因素，提出了一种基于函数距离与导向式模糊测试的动静结合固件漏洞分析技术，并设计实现了面向 MIPS 架构的 IoT 设备固件漏洞分析系统 DirFirmFuzz。该系统针对 D-Link 厂商的 3 个设备的 4 个固件程序进行了漏洞分析实验，从静态定位技术误报率、漏洞检测率以及运行时间等方面进行了评估。实验结果表明，本文提出的方法能够提高固件模糊测试的效率与发现漏洞的概率，其中，静态定位技术的漏洞误报缩减率最低为 59.14%，最高达到 86.93%，平均

缩减了 73.31%；同时，DirFirmFuzz 针对 3 个 D-Link 厂商设备中近年来公布的 18 个 CVE 漏洞进行模糊测试时，到达漏洞触发函数的平均速度相较于现有工具加快了 1.1～7 倍，最快仅用 61 s 就检测出 CVE-2019-8313 漏洞的 crash，也成功分析出了 Cisco 厂商设备中的 0-day 漏洞（CVE-2020-3223）。

1 相关研究

IoT 的概念最初是由 ASHTON 于 1999 年提出的[21]：把所有物品通过射频识别等信息传感设备与互联网连接起来，从而实现对物品的智能化识别和管理。IoT 设备架构繁多，并且与传统平台具有一定的区别。由于 IoT 设备指令的区别，造成难以直接使用传统基于二进制分析的漏洞定位方法。

1.1 IoT 设备漏洞静态分析技术

IoT 设备固件漏洞静态分析工作主要分为 2 个阶段：目标程序提取阶段和程序分析阶段。目标程序提取阶段的主要工作是获取并解压固件，并从文件系统中提取目标；程序分析阶段的主要工作是基于获取到的目标程序，通过程序分析来分析漏洞。

由于不同指令架构的汇编指令存在较大差异，SHOSHITAISHVILI 等[12]提出的 Angr 框架能够将不同指令架构的目标程序的汇编指令进一步抽象统一转化为 VEX 中间语言，之后研究人员就可以进行基于程序分析的漏洞分析。THOMAS 等[13]于 2015 年提出 IoT 设备固件认证绕过漏洞分析方法，在程序分析工具 Angr 的基础上设计并实现了二进制分析框架 Stringer，通过代码切片与路径求解检测出认证绕过漏洞，然而该方法仍然存在符号执行中路径爆炸、复杂约束求解等问题。SHOSHITAISHVILI 等[14]于 2017 年提出基于静态数据分析的漏洞分析方法 Firmalice 来检测 IoT 设备特有的后门漏洞（硬编码漏洞）。然而该方法的准确性无法保证，难以适用于大规模测试。CHENG 等[15]于 2018 年提出基于静态分析的 IoT 设备固件污点类漏洞分析方法，针对污点类型漏洞的特点，用户可控的数据输入点经过不安全路径到达敏感漏洞函数触发相应漏洞。

目前静态分析技术应用于 IoT 设备固件漏洞分析时，缺少程序运行时信息，从而导致误报率过高，并且普遍存在人工参与程度过高、自动化程度不足的问题，严重阻碍了大规模自动化测试的进行。

1.2 动态挖掘技术的发展与存在的问题

模糊测试是通过向目标程序提供非预期的输入并执行程序，记录并分析目标程序发生的异常来发现潜在漏洞的漏洞分析技术。CHEN 等[22]提出了一种黑盒模糊测试框架 IoTFuzzer，基于官方移动应用程序控制 IoT 设备时的通信数据，对 IoT 设备进行有效分析。但由于大部分 IoT 设备没有对应的移动应用程序，并且需要对实体设备进行分析，这些因素限制了框架的大规模应用。传统模糊测试工具（如：AFL，LibFuzzer 和 honggfuzz）在对 IoT 设备测试时，缺少硬件监控机制来获取执行信息，因此设备仿真技术对于 IoT 设备灰盒测试至关重要。

用户态仿真处理过程中，AFL 模糊测试工具对目标程序指令的翻译工作主要是通过传统的模拟器程序 QEMU[23]模拟器按照代码块粒度来完成的。混合仿真开展相关研究中，Zaddach 等[24]提出了混合仿真框架 Avatar，将 QEMU 与真实硬件进行连接，以便支持其混合执行。KOSCHER 等[25]针对 Avatar 仿真时效性差的问题，用 FPGA 桥接器进行调试，以便 IoT 设备做出相应分析和完成实时动态仿真。系统态仿真的研究中，CHEN 等[26]明确指出针对 Linux 内核 IoT 设备固件的全系统仿真框架 Firmadyne。COSTIN 等[27]在基于 Firmadyne 提出的 Web 接口测试方法中，检测出 45 个固件 Web 接口存在未知漏洞。TriforceAFL[28]将 QEMU 的系统模式和 AFL 进行结合，确保以系统仿真为基础，实现相应的模糊测试。ZHENG 等[29]基于 TriforceAFL，对 IoT 设备固件程序运用二进制动态分析平台 DECAF 的客户机进程实时监控技术，完成对其模糊测试。

模糊测试分析内存漏洞能力较强，不过在固件漏洞的分析能力上尚且存在一定的局限性。其原因是 Fuzzing 技术无法较好地在代码语义上做出分析，通过随机变异的测试用例效率低下，大部分测试用例在测试阶段初期就被丢弃，无法有效提高代码覆盖率。导向式模糊测试工具如 AFLGo[30]，使用以基本块距离为导向的策略与模拟退火算法相结合，作为一种通用方法对 IoT

固件模糊测试中覆盖率提升有一定借鉴意义，但依旧面临 IoT 固件程序的模拟仿真问题。

2　基于导向式模糊测试的 IoT 设备固件漏洞分析方法

面向 IoT 设备固件漏洞，针对现有静态分析技术误报率较高，动态仿真环境普适性较差，模糊测试效率不高等问题，深入研究 IoT 固件漏洞的特点，提出一种基于函数距离与导向式模糊测试的动静结合固件漏洞分析技术，整个过程如图 1 所示。

首先，通过静态分析技术获取危险代码区域，在此基础上，计算出危险代码区域中所有基本块到达漏洞触发函数的距离，生成基本块距离信息表。然后，对危险代码区域进行轻量级仿真，为模糊测试提供稳定、准确的动态执行环境，并在模糊测试的过程中监控种子的执行轨迹，根据轨迹中基本块的信息计算出种子距离。最后，根据种子距离动态调控种子能量，进而控制种子生成测试用例的数量，使得距离目标区域近的种子能够生成更多的测试用例，提升测试的效果。

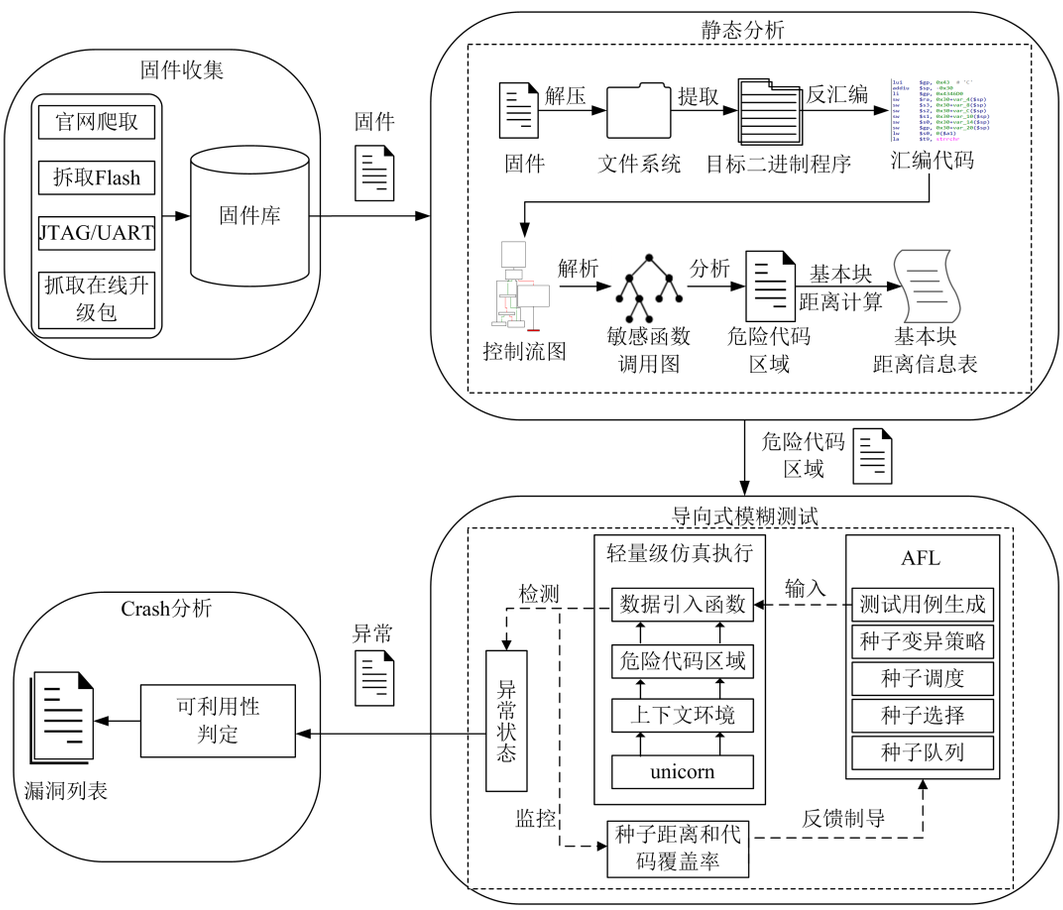

图 1　固件漏洞分析方法概览图

Fig. 1　Overview of the firmware vulnerability analysis method

2.1　基于函数距离的危险代码区域定位方法

通过分析 50 余个 IoT 固件漏洞后发现，漏洞代码中的数据引入函数和漏洞触发函数之间的距离通常较近，并且现有的大部分漏洞中数据引入函数和漏洞触发函数的直接调用函数是同一个函数。基于这一特征，本文设计了基于函数距离的危险代码区域静态定位方法，执行基本概览如图 2 所示。首先，解析固件二进制程序获得程序的反汇编指令序列，在反汇编指令序列上分析得到程序的控制流图与函数调用图，结合缓冲区溢出漏洞和命令注入漏洞的常见函数，标注出敏感函数及其对应的敏感函数调用关系；然后，判断调用图中数据引入函数与漏洞触发函数之间是否存在路径，若存在，则计算两者的距离；最后，对获得的所有路径进行处理，分析出每条路径中数据引入函数和漏洞触发函数在敏感函数调用图中的公共祖先函数，即路径中直接或者间接调用数据引入函数和漏洞触发函数的函数，按

照距离大小对所有路径中的公共祖先函数进行分类，最后将结果作为危险代码区域输出。

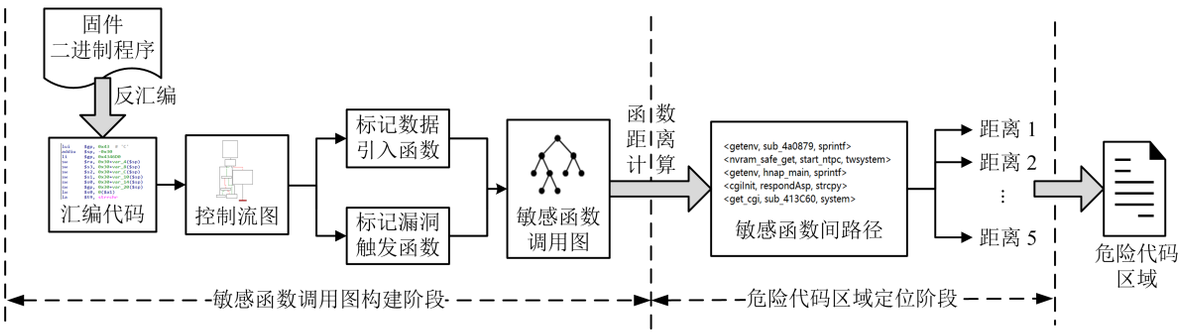

图 2 危险代码区域定位过程概览图

Fig. 2 Overview of locating vulnerable code areas

2.1.1 敏感函数调用图构建

本文的研究对象为 IoT 设备固件中最常见的缓冲区溢出漏洞和命令注入漏洞，这 2 类漏洞都由字符串或者内存操作函数引入。若字符串或内存操作函数（例如 gets 函数）不进行变量边界检查，同时使用了触发缓冲区溢出的 strcpy、sprintf 等函数或者使用了触发命令注入漏洞的 system、twsystem 等函数，就能完成漏洞的触发与利用。基于对这一过程的理解和抽象，将敏感函数定义为：漏洞触发函数和数据引入函数，并根据前期分析的 50 余个 MIPS 架构固件中的历史漏洞对其进行了总结，其中漏洞大多为命令执行漏洞与缓冲区溢出漏洞。表 1 列出了缓冲区溢出和命令注入 2 类漏洞对应的漏洞触发函数，表 2 列出了常见的数据引入函数。

表 1 漏洞触发函数

Tab. 1 Vulnerability trigger function

漏洞类型	函数名称
缓冲区溢出类	strcpy、sprintf、snprintf、strncpy、strcat、printf、fprintf、memcpy、memncpy
命令注入类	system、popen、twsystem、execve

表 2 数据引入函数

Tab. 2 Data introduction function

函数类型	函数名称
输入数据文件	read、fscanf、getc、fgetc、fgets、vfscanf
键盘输入	scanf、getchar、gets
网络数据	recv、recvfrom
环境变量	getenv、get_cgi、nvram_get、find_var、websGetVar、httpGetEnv

本文研究重点关注敏感函数，算法 1 是敏感函数调用图 G 的构建过程。首先生成固件程序的控制流图（control flow graph，CFG）（算法 1 第 2 行），判断 CFG 中的每一个基本块是否存在函数序言，函数序言通常是指函数开始的一段代码，用于创建函数的堆栈以及暂存调用函数的上下文环境。如果基本块中存在函数序言则标记为函数，并记录该函数的函数名和地址，加入程序函数集合 Funcs 中（算法 1 第 3～8 行）。将函数集合中的所有函数作为节点加入图 G 中（算法 1 第 9 行）。为了生成敏感函数调用图，将数据引入函数集合 Source_funcs 和漏洞触发函数集合 Sink_funcs 中，并将两集合合并为敏感函数集合 Danger_funcs（算法 1 第 11 行）。敏感函数集合中的每一个函数其实都是图 G 中的叶子节点，接下来需要寻找这些敏感函数的调用函数，并在图中加入对应边表示相应的调用关系（算法 1 第 11～17 行）。对敏感函数集合中的每一个函数先判断是否在程序函数集合 Funcs 中，如果是，则通过 Add_edge 函数在图 G 中加边。加边过程中，需要涉及最大递归次数问题，算法中 N 值表示函数的递归次数，由于实际分析过的历史漏洞中，大多数递归次数在 3 以内（考虑时间效率因素，迭代数越少效率越高），因此从准确性与运行效率两个角度出发进行权衡，文本中暂时选择 3 作为最大递归次数。依靠算法所示对图中所有敏感函数都进行加边操作，最终构建出敏感函数调用图 G。

算法 1 敏感函数调用图构建过程

输入：固件程序 fb

输出：敏感函数调用图 G

1. G←ϕ；Funcs←ϕ

2. CFG←CFGGenerate(fb)

```
3.  for each block in CFG. blocks do
4.      if block is a funtion then
5.          Funcs←block
6.      end if
7.  end for
8.  return Funcs
9.  G←add_nodes_from(Funcs)
10.
11.  Danger_funcs←Source_funcs⋃Sink_funcs
12.  for each func in Danger_funcs do
13.      if func in Funcs then
14.          G←Add_edge(G, func, 0)
15.      end if
16.  end for
17.  return G
18.
19.  Function Add_edge(G, func, N)
20.      if N>2 then
21.          return G
22.      end if
23.      for each caller_func of func do
24.          G. addedge(func, caller_func)
25.          G←Add_edge(G, func, N+1)
26.      end for
27.  end Function
```

敏感函数调用图样例如图 3 所示。

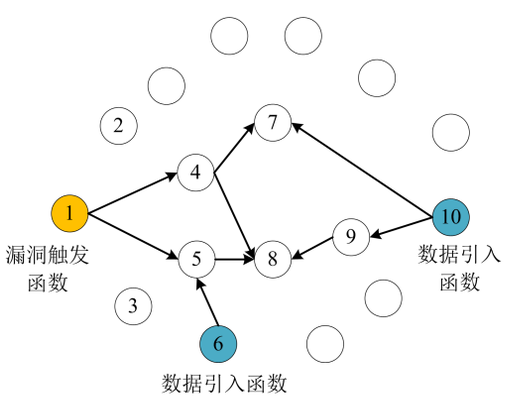

图 3　敏感函数调用图

Fig. 3　The callgraph of sensitive function

2.1.2　危险代码区域定位

危险代码区域定位是在敏感函数调用图的基础上，分析数据引入函数与漏洞触发函数之间是否存在路径，并计算函数间距离。之后，找出数据引入函数和漏洞触发函数在敏感函数调用图中的公共祖先函数，依据是否存在公共祖先函数和路径的距离长度来判断指定数据引入函数到漏洞触发函数的代码区域是否为危险代码区域。

本文中函数的距离是指在敏感函数调用图中，如果 2 个函数节点之间存在路径，将该路径中函数节点数量减 2(2 个函数自身节点)则为这 2 个函数之间的距离。算法 2 描述了函数距离的计算过程。在敏感函数调用图中以数据引入函数为起点，判断漏洞触发函数与数据引入函数之间是否存在路径，如果存在，则将两者之间所有路径都保存在集合 Paths 中(算法 2 第 6 行)，之后计算出所有路径中数据引入函数与漏洞触发函数之间的距离，并将距离存储在集合 Distances 中。

算法 2　函数距离计算过程

输入：敏感函数调用图 *G*

输出：漏洞触发函数与数据引入函数间路径集合 *Paths* 及其每条路径对应距离 *Distances*

```
1.  Paths←ϕ;Distances←ϕ
2.  for each source in Source_funcs do
3.      for each sink in Sink_funcs do
4.          if has_path(G,source,sink) then
5.              Paths←all_paths(G,source,sink)
6.          end if
7.      end for
8.  end for
9.  return Paths
10.
11.  for each path in Paths do
12.          distance←len(path)−2
13.          Distances←distance
14.  end for
15.  return Distances
```

图 4 为敏感函数距离情况举例。如果数据引入函数与漏洞触发函数被同一函数调用，则两者间距离为 1，函数 A 调用数据引入函数的同时又调用了漏洞触发函数，其在函数调用图中的路径可表示为：[数据引入函数，函数 A，漏洞触发函数]。如果数据引入函数的调用函数调用了漏洞触发函数的调用函数，或者漏洞触发函数的调用函数调用了数据引入函数的调用函数，如图 4(a)所示，则两者距离为 2。当函数存在关系为：[数据引入函数，函数 A，函数 B，函数 C，漏洞触发函数]，即如图 4(b)左侧所示，则两者距离为 3。由于在构建敏感函数调用图时，调用图 G 加边的最大递归次数为 3，所以敏感函数间的最大距离为 5，如图 4(c)所示，在函数调用图 G 的路径可表示为：[数据引入函数，函数 C，函数 B，函数 A，函数 D，函数 E，漏洞触发函数]。

图 4 敏感函数距离情况举例

Fig. 4 The callgraph of sensitive function

获得所有路径及距离后，需要确定数据引入函数和漏洞触发函数在敏感函数调用图中的公共祖先函数。公共祖先函数的定位方法是：以敏感函数调用图中数据引入函数节点或者漏洞触发函数节点为起点，根据边的方向反向迭代搜索节点并标记为调用函数，直至不存在可以搜索的节点；若数据引入函数与漏洞触发函数的调用函数为同一节点函数，则该节点为数据引入函数和敏感函数的公共祖先函数。然后，依照距离长度对所有路径中的公共祖先函数进行分类，根据代码中预先设定所感兴趣的路径距离长度，将相应分类结果作为危险代码区域输出。以图 4 为例，经过迭代搜索后发现，图 4(a)中数据引入函数节点与漏洞触发函数节点间距离为 2；图 4(b)中二者间距离为 3；图 4(c)中二者间距离为 5。假设实验中感兴趣的路径距离长度为 2 时，则图 4(a)中所示代码区域为要输出的危险代码区域。在前期对 50 余个 MIPS 架构真实漏洞分析过程中发现，数据引入函数与漏洞触发函数之间的路径距离长度大多不超过 2，因此在实际测试中，通常将感兴趣的路径距离长度设为 2。

2.2 基于基本块距离的导向式模糊测试方法

导向式模糊测试技术是一种能够对程序目标区域进行快速导向并检测漏洞的模糊测试技术，程序目标区域一般是指程序静态分析定位出的可能存在漏洞的危险代码区域。本文提出了一种基于基本块距离的导向式模糊测试技术，其中基本块指的是程序顺序执行的语句序列，执行时只从其入口进入，从其出口退出。导向式模糊测试技术工作流程如下：

1) 在静态分析阶段，计算危险代码区域中每个基本块到漏洞触发函数的距离，并保存到危险代码区域仿真脚本。

2) 在模糊测试阶段，从种子队列中选择种子对仿真的危险代码区域进行模糊测试，记录该种子执行轨迹中的基本块信息，根据静态分析阶段得到的基本块距离计算出种子达到漏洞触发函数的距离。种子即为可输入程序的测试用例，并可以基于此生成更多的测试用例；种子执行轨迹指使用种子测试用例时，程序的指令执行路

径，以及每一执行时刻的程序内存、寄存器状态。

3）以种子距离为依据动态调控种子能量，种子能量决定了种子变异的时间，进而影响到种子变异生成测试用例的数量。

4）用种子变异生成的测试用例对仿真的危险代码区域进行测试，如果测试过程中增加了代码覆盖率，则将该测试用例加入种子队列，在测试过程中如果遇到会导致程序崩溃的异常与错误，或是观测到程序运行至漏洞可利用状态的情况，则输出 crash。

循环操作步骤 2）～4），当种子队列中所有种子都被测试过一次，则一轮测试结束，之后从种子队列重新选择第一个种子进行测试直到结束。

2.2.1　基本块距离计算

实现导向式的关键在于计算出种子到漏洞触发函数的距离。算法 3 描述了危险代码区域中基本块距离的计算方法。算法的输入是危险代码区域 DCode_area，输出是基本块距离信息表 D_TableD_Table。首先，生成危险代码区域的详细控制流图 DCFG（算法 3 第 2 行），将控制流图中的所有边都赋予权重，权重的值为所属基本块出度的倒数，也就是所属基本块后续基本块的数量（算法 3 第 3～6 行）。对于控制流图中的所有基本块，求出其距离目标基本块集的所有路径，对于每条路径都计算其距离，其中的最短路径距离作为该基本块距离 $d(\text{block},T)$，并加入基本块信息表 D_Table 中（算法 3 第 8～18 行）。算法 3 第 20～25 行叙述了每条路径距离的计算方法，将该路径中所有边的权重值相加后的结果作为路径的距离返回。

算法 3　危险代码区域中基本块距离计算方法

输入：危险代码区域 DCode_area

输出：基本块距离信息表 D_Table

```
1. D_Table←ϕ; Paths←ϕ
2. DCFG←CFGGenerate(DCode_area)
3. for each edge in DCFG. edges do
4.       block_outdegree=num_successnodes of block
5.       Weight_edge=1/block_outdegree
6. end for
7.
8. for each block in DCFG. blocks do
9.       Paths←all_paths(block, T)
10.      for each path in Paths do
11.            tem_distance=Caculate_distance(path)
12.            if tem_distance<d(i, T) then
13.                  d(block, T)=tem_distance
14.            end if
15.      D_Table←d(block, T)
16.      end for
17. end for
18. return D_Table
19. Function Caculate_distance(bolck, path, T)
20.      distance←0
21.      for each edge in path do
22.            distance=distance+Weight_edge
23.      end for
24.      return distance
25. end Function
```

图 5 为程序路程示意图。以图 5 为例，目标基本块 Target 是指包含漏洞触发函数的基本块，基本块 1 和基本块 2 到达目标基本块 Target 的最短路径相同。实际上经过基本块 2 到达目标基本块 Target 的路径有 2 条，而经过基本块 1 达到目标基本块 Target 的路径只有 1 条，也就是经过基本块 2 更加容易达到漏洞触发函数。

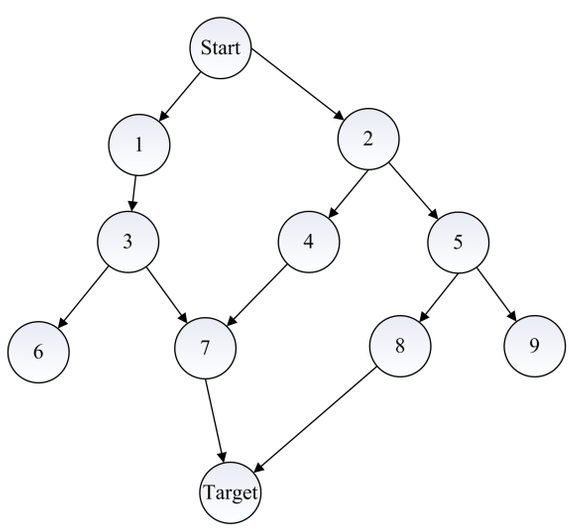

图 5　程序路径示意图

Fig. 5　Graph of program path

为了提高距离计算的准确性，根据控制流图中基本块的出度，赋予基本块之间的边相应的权重，公式为：

$$W_{\text{edge}\_ij}=\frac{1}{i_{\text{out}\_\text{degree}}} \tag{1}$$

式中，W_{edge_ij} 表示控制流图中由基本块 i 指向基本块 j 的边的权重，$i_{\text{out}\_\text{degree}}$ 表示基本块 i 的出

度，也就是后续基本块的数量。利用式(1)对图 5 中所有边进行赋值，结果如图 6 所示。

对所有边赋予权重后，求出基本块 i 和目标基本块之间所有路径距离，取其中最短距离作为基本块 i 的距离，公式为：

$$d(i,T)=\begin{cases}0, & \text{if } i\in T\\ \min(d(i,T)), & \text{if } i\notin T\end{cases} \tag{2}$$

式中，$d(i,T)$表示基本块 i 的距离，T 表示目标基本块集合，如果基本块属于目标基本块，则该基本块距离为 0，否则该基本块的距离为所有路径距离中的最短距离。例如，基本块 2 到目标基本块的路径有 2 条，距离分别为 2 和 5/2，选择距离小的 2 作为基本块 2 的距离。

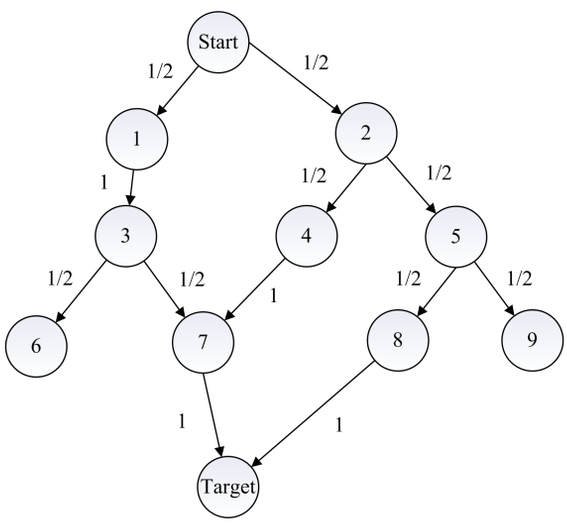

图 6 边赋予权重示意图

Fig. 6 Graph of edge weight assignment

2.2.2 种子距离计算与能量调控

在静态分析阶段能够获取到危险代码区域所有基本块到漏洞触发函数的距离信息表，在模糊测试阶段，记录每个种子的执行轨迹，计算执行轨迹中所有基本块的距离之和作为该种子的距离，公式为：

$$D(s,T)=\sum_{i\in P_{\text{path}}(s)} d(i,T) \tag{3}$$

式中，$D(s,T)$表示种子 s 到漏洞触发函数基本块集 T 的距离，$P_{\text{path}}(s)$表示种子 s 的执行路径中所有基本块，$d(i,T)$为基本块 i 的距离。对种子距离进行归一化处理，公式为：

$$\min D=\min_{s\in S}[D(s,T)] \tag{4}$$

$$\max D=\max_{s\in S}[D(s,T)] \tag{5}$$

$$\widetilde{D}(s,T)=\frac{D(s,T)-\min D}{\max D-\min D} \tag{6}$$

式中，S 表示种子集合，min D 表示最小种子距离，max D 表示最大种子距离，$\widetilde{D}(s,T)$为归一化后的种子距离 $\widetilde{D}\in[0,1]$。

在模糊测试技术中，种子能量是用来调控种子变异的重要参数，为了实现模糊测试的导向性，应该使距离漏洞触发函数越近的种子获得的种子能量越高，该种子变异生成的测试用例数量越多，到达漏洞触发函数的测试用例也就越多。

本文引入种子执行轮次 r 作为影响因子，影响种子能量计算过程中距离的占比。种子执行轮次指的是：基于该种子生成测试用例并执行的轮数/次数。引入该影响因子的目的是在模糊测试开始阶段种子距离的对于种子能量的影响程度不高，而随着执行轮次的增加而影响程度越来越大。种子能量的计算公式为：

$$P(s,T)=C\cdot\left((1-\widetilde{D}(s,T))\cdot\left(1-\frac{1}{r}\right)+\frac{1}{r}\right) \tag{7}$$

式中，$P(s,T)$为种子 s 的能量，系数 C 是常数，$\widetilde{D}(s,T)$是归一化后的种子 s 的距离，r 为执行轮次。在模糊测试开始时，执行第 1 轮测试时的 r 为 1，距离为 1 和距离为 0 的种子获得的能量相同，都为 C；但是，随着执行轮次的增加，在种子能量调控函数的影响下，种子距离对于种子能量计算的影响程度逐渐增大，不同距离的种子之间的能量差异越来越大，导向式模糊测试过程也从初期的随机搜索转变为对目标区域的集中测试。

3 实验与评估

3.1 实验环境设置

本文在开源模糊测试器 AFL-unicorn 的基础上设计实现了基于导向式模糊测试的固件漏洞分析原型系统 DirFirmFuzz；该系统主要的模块实现包括：

1) 使用 Angr 生成危险代码区域详细的控制流图；

2) 使用 Python 的 networkx 包实现对控制流程图的解析并计算基本块距离；

3) 使用 Python 为每个危险代码区域编写 unicorn 仿真脚本。

实验环境如表 3 所示。

为了验证系统的有效性，选取了 MIPS 架构的 D-Link 厂商的 3 个设备的 4 个固件程序与 Cisco 厂商的 RV110W 设备固件进行测试，测试固件程序包括 http 服务程序、cgi 程序和动态链接库文件。值得注意的是，本文介绍的为一种通用方法，因此针对除路由器以外的自控制类、传感类设备，只要其中漏洞涉及数据引入函数与漏洞触发函数且具有距离较近的特点，理论上皆可使用本文方法进行检测。具体测试目标的信息如表 4 所示。

表 3　测试环境

Tab. 3　Testing environment

环境	项目	配置及版本信息
硬件环境	处理器	Intel(R)Core(TM) i7-8750H@1.90 GHz
	内存	16 GB
	硬盘	M.2 NVME 512 G
	调试工具	USB 转 TTL RS232
软件环境	操作系统	Ubuntu 16.04 LTS 64
	Angr	8.19.10.30
	IDA Pro	Verison 7.5.200728
	Unicorn-Engine	Version 1.0.2
	Python2	2.7.12
	gdbserver	gdbserver-7.7.1-mipsel-mips32-v1

表 4　测试目标详细信息

Tab. 4　Details of testing target

厂商	设备	固件版本	架构	产品类型	目标程序
D-Link	DIR-878	v1.12	MIPS	路由器	/bin/rc
			MIPS	路由器	/lib/librcm.so
	DIR-859	v1.05	MIPS	路由器	/htdocs/cgibin
	DIR-825	v2.10	MIPS	路由器	/sbin/httpd
Cisco	RV110W	v1.2.2.5	MIPS	路由器	httpd

3.2　实验结果分析

3.2.1　危险代码区域定位效果测试

表 5 给出了系统分析结果。从表 5 的结果可以看出，本文提出的危险代码区域定位方法分析出的危险代码区域数量较 Angr 有了大幅度的减少，数量缩减率最低为 59.14%，最高可达到 86.93%，平均缩减了 73.31%。实际在固件程序中，共存在 12 个真实漏洞，且都包含在本文方法检测结果中，这表明本文提出的方法能够降低现有工具的误报率。

表 5　系统分析结果

Tab. 5　Results of system analysis

ID	Program	Size /kb	Angr	SFVLocator	Shrinkage (%)	Vul. num
1	/bin/rc	457	306	40	86.93	4
2	/lib/librcm.so	283	505	168	66.73	4
3	cgibin	156	93	38	59.14	3
4	httpd (DIR-825)	530	383	75	80.42	1

通过人工分析验证，本系统输出的危险代码区域存在 12 个真实漏洞，这表明了本文提出的危险代码区域定位方法能够有效地定位真实固件漏洞代码。12 个真实固件漏洞具体信息如表 6 所示，包括对应的 CVE 编号、漏洞类型、数据引入函数、漏洞触发函数及其之间的距离，其中，Distance 表示 CVE 漏洞中数据引入函数和漏洞触发函数之间的距离。

3.2.2　导向式分析

实验选取 3 个设备中近年公开的 CVE 漏洞作为导向目标，分别使用 DirFirmFuzz 和 AFL-Unicorn 对目标程序进行实验。实验结果如表 7～9 所示。Arrive 表示达到漏洞触发点的次数，TTE 表示程序第 1 次覆盖到漏洞触发函数的时间。Enhance 为改善因子，值为 AFL-Unicorn 的 TTE 与 DirFirmFuzz 的 TTE 的商，表示相比于 AFL-Unicorn，DirFirmFuzz 的效率提升了多少。

从表 7～9 的结果可以看出，DirFirmFuzz 达到漏洞触发函数的平均速度比 AFL-Unicorn 快 1.1～7 倍，因为 CVE 所在危险代码区域的代码规模与逻辑复杂程度不同，因此在案例之间存在速度提升比例的差异。在 CVE-2020-10216 中提升效果最不明显，DirFirmFuzz 到达漏洞触发函数的速度比 AFL-Unicorn 快了 1.09 倍。在 CVE-2020-29557 中提升效果最为显著，DirFirmFuzz 到达漏洞触发函数的时间仅用了 AFL-Unicorn 花费时间的 14.14%。将表中数据绘图能够展示得更加清楚，如图 7～9 所示。实验结果表明使用本文的方法可以快速生成覆盖固件目标程序指定位置的测试用例，能够提高在已知危险代码区域的模糊测试效率。

表 6 详细固件漏洞信息

Tab. 6 Details of firmware vulnerabilities

CVE-ID	漏洞类型	数据引入函数	漏洞触发函数	Distance
CVE-2019-8312 CVE-2019-8313 CVE-2019-8314 CVE-2019-8315 CVE-2019-8317 CVE-2019-8318 CVE-2019-8319	Command Injection	nvram_safe_get	twsystem	1
CVE-2019-8316	Command Injection	nvram_safe_get	twsystem	2
CVE-2019-20215 CVE-2019-20216 CVE-2019-20217	Command Injection	getenv	lxmldbc_system	1
CVE-2020-10213	Command Injection	get_cgi	system	1

表 7 D-Link DIR-878 漏洞触发函数覆盖情况

Tab. 7 Coverage of vulnerability trigger function in D-Link DIR-878

CVE-ID	工具	TTE/s	Arrive	Enhance
CVE-2019-8312	DirFirmFuzz	25	2	—
	AFL-Unicorn	98	2	3.92
CVE-2019-8313	DirFirmFuzz	14	1	—
	AFL-Unicorn	47	1	3.35
CVE-2019-8314	DirFirmFuzz	23	2	—
	AFL-Unicorn	46	2	2.00
CVE-2019-8315	DirFirmFuzz	52	5	—
	AFL-Unicorn	215	2	4.13
CVE-2019-8316	DirFirmFuzz	31	3	—
	AFL-Unicorn	104	1	3.35
CVE-2019-8317	DirFirmFuzz	49	4	—
	AFL-Unicorn	303	3	6.18
CVE-2019-8318	DirFirmFuzz	168	8	—
	AFL-Unicorn	502	5	2.98
CVE-2019-8319	DirFirmFuzz	37	8	—
	AFL-Unicorn	41	4	1.10
CVE-2021-30072	DirFirmFuzz	76	7	—
	AFL-Unicorn	386	3	5.07

表 8 DIR-859 漏洞触发函数覆盖情况

Tab. 8 Coverage of vulnerability trigger function in D-Link DIR-859

CVE-ID	工具	TTE/s	Arrive	Enhance
CVE-2019-20215	DirFirmFuzz	45	2	—
	AFL-Unicorn	67	2	1.48
CVE-2019-20216	DirFirmFuzz	51	4	—
	AFL-Unicorn	88	4	1.72
CVE-2019-20217	DirFirmFuzz	42	2	—
	AFL-Unicorn	89	1	2.11
CVE-2019-17621	DirFirmFuzz	53	4	—
	AFL-Unicorn	301	3	5.67

表 9 D-Link DIR-825 漏洞触发函数覆盖情况

Tab. 9 Coverage of vulnerability trigger function in D-Link DIR-825

CVE-ID	工具	TTE/s	Arrive	Enhance
CVE-2020-29557	DirFirmFuzz	54	5	—
	AFL-Unicorn	382	3	7.07
CVE-2020-10216	DirFirmFuzz	385	3	—
	AFL-Unicorn	423	3	1.09
CVE-2020-10215	DirFirmFuzz	186	2	—
	AFL-Unicorn	504	2	2.70
CVE-2020-10214	DirFirmFuzz	47	4	—
	AFL-Unicorn	65	3	1.38
CVE-2020-10213	DirFirmFuzz	128	2	—
	AFL-Unicorn	206	2	1.60

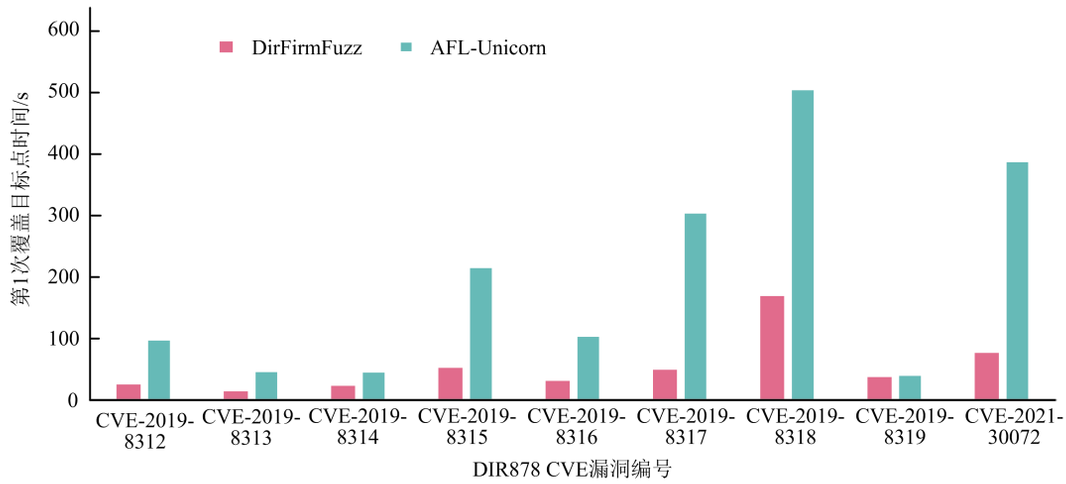

图 7　D-Link DIR-878 漏洞触发函数覆盖情况

Fig. 7　Coverage of vulnerability trigger function in D-Link DIR-878

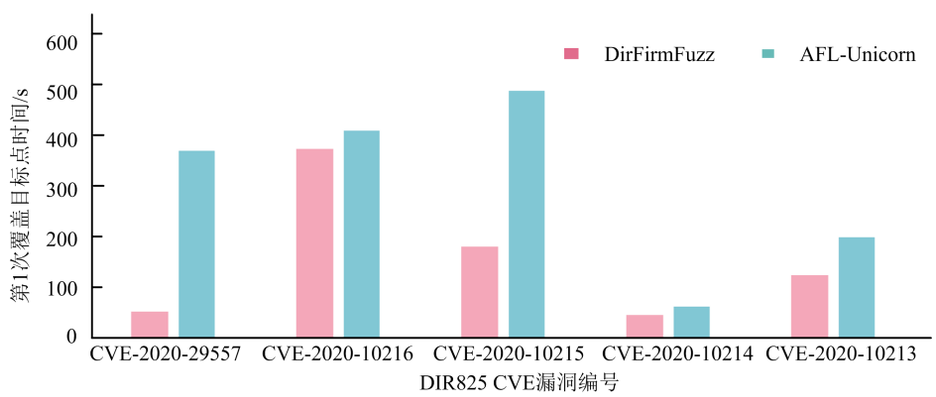

图 8　D-Link DIR-859 漏洞触发函数覆盖情况

Fig. 8　Coverage of vulnerability trigger function in D-Link DIR-859

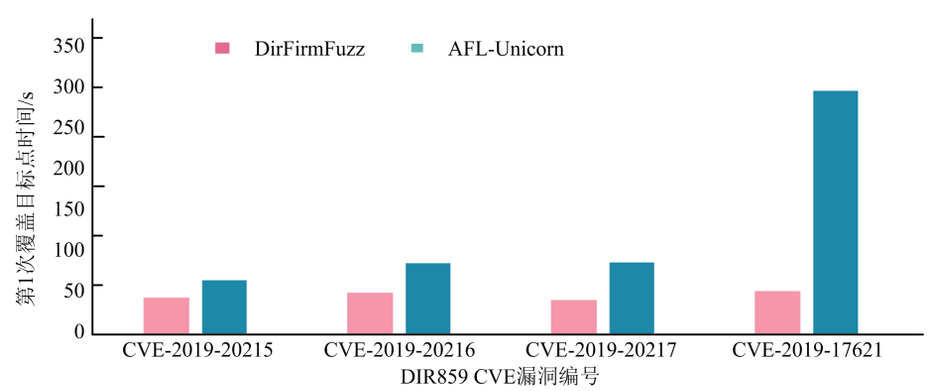

图 9　D-Link DIR-825 漏洞触发函数覆盖情况

Fig. 9　Coverage of vulnerability trigger function in D-Link DIR-825

3.2.3　触发 crash 测试

为了验证 DirFirmFuzz 能够检测出 crash，同样将 3 个设备中近年公开的 CVE 漏洞作为目标，分别使用 DirFirmFuzz 和 AFL-Unicorn 对目标程序进行实验，实验结果如表 10～12 所示，表中 TTCrash 表示第 1 次产生有效 crash 的时间。

从表 10～12 可以看出，针对 3 个设备的 18 个 CVE 漏洞，最快仅用 61 s 就检测出 CVE-2019-8313 漏洞的 crash，时间最长的 CVE-2020-10213 花了 808 s 才检测出 crash。在对 CVE-2019-8318，CVE-2019-17621 与 CVE-2020-10213

的测试中，DirFirmFuzz 未获得比 AFL-Unicorn 更短的时间，原因在于基于距离的导向式无法保证后续测试都围绕存在真实漏洞的代码段进行，对其他没有漏洞的危险代码段也会进行测试，从而影响真实漏洞代码处出现 crash 的速度。DirFirmFuzz 能够在有限时间内检测 crash，证明本文设计的面向固件模糊测试的轻量级仿真异常检测机制的有效性，能够在有限时间内检测出真实 IoT 设备固件漏洞中缓冲区溢出漏洞和命令注入漏洞的异常。

表 10 D-Link DIR-878 crash 触发情况

Tab. 10 Trigger results of D-Link DIR-878 crash

ID	Program	CVE-ID	Vulnerability	AFL-Unicorn TTCrash/s	DirFirmFuzz TTCrash/s
1	/bin/rc	CVE-2019-8312	Command Injection	209	123
2	/lib/librcm. so	CVE-2019-8313	Command Injection	115	61
3	/lib/librcm. so	CVE-2019-8314	Command Injection	103	69
4	/lib/librcm. so	CVE-2019-8315	Command Injection	480	267
5	/lib/librcm. so	CVE-2019-8316	Command Injection	188	135
6	/bin/rc	CVE-2019-8317	Command Injection	563	352
7	/bin/rc	CVE-2019-8318	Command Injection	536	670
8	/bin/rc	CVE-2019-8319	Command Injection	148	78
9	/bin/prog. cgi	CVE-2021-30072	Buffer overflow	600	462

表 11 D-Link DIR-859 crash 触发情况

Tab. 11 Trigger results of D-Link DIR-859 crash

ID	Program	CVE-ID	Vulnerability	AFL-Unicorn TTCrash/s	DirFirmFuzz TTCrash/s
1	/htdocs/cgibin	CVE-2019-20215	Command Injection	123	112
2	/htdocs/cgibin	CVE-2019-20216	Command Injection	166	139
3	/htdocs/cgibin	CVE-2019-20217	Command Injection	144	131
4	/htdocs/cgibin	CVE-2019-17621	Command Injection	354	354

表 12 D-Link DIR-825 crash 触发情况

Tab. 12 Trigger results of D-Link DIR-825 crash

ID	Program	CVE-ID	Vulnerability	AFL-Unicorn TTCrash/s	DirFirmFuzz TTCrash/s
1	/usr/sbin/anweb	CVE-2020-29557	Buffer overflow	610	436
2	/sbin/httpd	CVE-2020-10213	Command Injection	323	808
3	/sbin/httpd	CVE-2020-10214	Command Injection	1037	690
4	/sbin/httpd	CVE-2020-10215	Command Injection	190	112
5	/sbin/httpd	CVE-2020-10216	Command Injection	367	334

3.3 真实漏洞挖掘验证

为了进一步测试 DirFirmFuzz 的功能，同时证明对真实设备的漏洞分析有效性，本文选择 MIPS 架构的 Cisco RV110W 设备固件的 httpd

程序展开测试。整体测试流程如下：

1）从官网中获取 Cisco RV110W 的 1.2.2.5 版本固件 RV110W_FW_1.2.2.5.bin。

2）使用 Binwalk 将固件解压后提取其中的 httpd 程序，通过静态分析模块得到危险代码区域，部分函数距离为 1 或 2 的危险代码区域如表 13 所示；表 13 中的危险代码区域是定位得到的包含数据引入函数和漏洞触发函数的最近公共祖先函数，Address 表示危险代码区域的起始地址，也就是对应最近公共祖先函数的首地址，Source 表示数据引入函数。

表 13　Cisco RV110W httpd 静态分析的部分结果

Tab. 13　Static analysis results of Cisco RV110W httpd

危险代码区域	Address	Source	漏洞触发函数	函数距离
sub_4242D8	0x04242D8	get_cgi	sscanf	1
validate_range	0x042B0A8	get_cgi	sscanf	2
guest_logout_cgi	0x04317f8	get_cgi	scanf	1
gozila_cgi	0x0432b68	get_cgi	strcpy	1
apply_cgi	0x0433494	get_cgi	strcpy	1
validate_port_manage	0x0437c9c	nvram_get	system	2
delete_single_leases	0x0440aa8	get_cgi	sprintf	1
validate_schtb	0x04508a4	get_cgi	sscanf	1
Check_TSSI	0x046c92c	get_cgi	strcpy	1
wl_validate_multi_name	0x0462d50	get_cgi	sprintf	1
StartContinueTx	0x046d788	get_cgi	sprintf	1
ej_dump_syslog_ng	0x0478208	nvram_get	sscanf	1

3）通过上传静态编译过的 gdbserver，远程调试获取危险代码区域起始地址的上下文环境，用来做动态执行环境的初始状态。获取起始点的上下文环境后，对危险代码区域进行导向式模糊测试，每组实验重复 5 次，每次实验持续 24 小时，记录 5 次实验的平均值。在模糊测试过程中，对第 1 次达到漏洞触发函数的时间和第 1 次产生有效 crash 的时间进行记录，将记录结果绘制成表 14。表中 time to target 表示第 1 次到达漏洞触发函数的时间，time to crash 表示第 1 次产生 crash 的时间。通过表 14 能看出，表中的 12 个危险代码区域中只有 guest_logout_cgi 产生了 crash，对该危险代码区域其中的一次模糊测试过程中第 1 次产生 crash 的时间进行记录，该次的 time to crash 是 652 s。

表 14　Cisco RV110W httpd 导向式模糊测试分析结果

Tab. 14　Results of Cisco RV110W httpd directed fuzzing

ID	危险代码区域	漏洞触发函数	time to target/s	time to crash/h
1	sub_4242D8	sscanf	430	>24
2	validate_range	sscanf	576	>24
3	guest_logout_cgi	scanf	333	0.17(612 s)
4	gozila_cgi	strcpy	640	>24
5	apply_cgi	strcpy	603	>24
6	validate_port_manage	system	343	>24
7	delete_single_leases	sprintf	227	>24
8	validate_schtb	sscanf	337	>24
9	validate_schtb	strcpy	184	>24
10	wl_validate_multi_name	sprintf	338	>24
11	StartContinueTx	sprintf	175	>24
12	ej_dump_syslog_ng	sscanf	195	>24

4）对 guest_logout_cgi 模糊测试过程中产生的 crash 进一步分析，逆向固件程序 httpd，利用 IDA Pro7.5 反编译代码区域得到简化后的伪代码如图 10 所示。

从图 10 可以看出，产生 crash 的成因是从第 10 行的数据引入函数 get_cgi 获取外部输入 submit_button，在满足第 12 行的判断条件后，达到第 15 行的漏洞触发函数 sscanf 时触发的 crash。

```
1  int guest_logout_cgi(){
2  ...
3  char buf2 [64];
4  char buf1 [68];
5  ...
6  char* cip = get_cgi("cip");
7  char* cmac = get_cgi("cmac");
8  //数据引入函数get_cgi
9  //submit_button变量是通过外部获取
10 char* submit_button = get_cgi("submit_button");
11 ...
12 int v = strstr(submit_button, "status_guestnet.asp");
13 if(v != NULL){
14     //漏洞触发函数sscanf
15     sscanf(submit_button,"%[^;];%*[^=]=%[^\n]", buf1, buf2);
16     ...
17 }
18 ...
19 }
```

图 10　guest_logout_cgi 部分伪 c 代码

Fig. 10　Part of pseudocode of guest_logout_cgi

进一步分析发现只要给设备发送的请求包中 submit_button 的格式满足 submit_button=status_guestnet.asp；seesion_id=xxxxxxxxxxxxxxx，这样执行完 sscanf 后 buf1 中的内容为 status_guestnet.asp，buf2 中的内容

为 xxxxxxxxxxxxxxx。因为没有对 submit_button 内容做限制，而 buf2 的大小有限，进而导致缓冲区溢出漏洞。该漏洞对应 CVE 编号为 CVE-2020-3223。

3.4 未知漏洞挖掘测试

为了进一步测试文中提出方法的效果，本文使用 DirFirmFuzz 对 MIPS 架构 IoT 设备进行了进一步的漏洞测试，共发现 12 个 0-day 漏洞，均已报送相关厂商并进行修补，其中，Cisco 设备的 3 个 0-day，Netgear 设备 7 个 0-day，D-Link 设备 2 个 0-day，漏洞类型包括缓冲区溢出、命令注入。目前分配编号的漏洞有 7 个，包括 2 个 CVE 编号和 5 个 CNNVD 编号，如表 15 所示。

表 15 已分配编号的漏洞

Tab. 15 Vulnerabilities with assigned number

厂商	漏洞类型	CVSS/评级	漏洞编号
Cisco	缓冲区溢出	9.8	CVE-2021-1164
	命令执行	7.2	CVE-2021-1360
NETGEAR	未公开	高危	CNNVD-202007-1469
	未公开	中危	CNNVD-2020007-1471
	未公开	高危	CNNVD-2020007-1472
	未公开	中危	CNNVD-2020007-1473
	未公开	高危	CNNVD-2020007-1474

1）Cisco 厂商设备的 2 个高危 CVE 编号，其中一个 CVSS 评分为 9.8，并得到 Cisco 公司的官方公开致谢。CVE-2021-1164（超高危，CVSS3.1 评分 9.8）、CVE-2021-1360（高危，CVSS3.1 评分 7.2）。漏洞基本信息如下：

CVE-2021-1164：Cisco 小型商用 RV110W、RV130、RV130W 和 RV215W 路由器的 upnp 服务中存在缓冲区溢出漏洞，影响上述型号设备的所有版本固件。攻击者可以利用此漏洞，在未经授权的情况下远程获取设备控制权限，达到任意代码执行（RCE）的效果。

CVE-2021-1360：Cisco 小型企业 RV110W 和 RV215W 路由器 Web 管理接口远程命令执行，影响上述型号设备的所有版本固件。攻击者可以利用此漏洞在授权下远程获取设备控制权限，实现任意代码执行（RCE）的效果。

2）NETGEAR 厂商设备：CNNVD-202007-1469（高危）、CNNVD-2020007-1471（中危）、CNNVD-2020007-1472（高危）、CNNVD-2020007-1473（中危）、CNNVD-2020007-1474（高危）。由于国家信息安全漏洞库相关规定，相关细节不便在此公开。

4 总结与展望

针对现有技术在测试 IoT 设备时，存在的静态分析手段误报率高、动态仿真与监控难以实现、针对固件的模糊测试效率不高等问题，本文提出了基于导向式模糊测试的固件漏洞分析技术，在静态分析获得的危险代码区域的基础上，计算出危险代码区域中所有基本块达到漏洞触发函数的距离，生成基本块距离信息表；对危险代码区域进行轻量级仿真，在模糊测试过程中对种子的执行轨迹中的基本块进行记录，进而计算出种子距离，模糊测试器以种子距离为依据对种子能量进行动态调控以实现对危险代码区域的导向性。通过 CVE 漏洞触发函数覆盖对比实验表明，本文提出基于导向式模糊测试的固件漏洞分析技术能够提高固件模糊测试的效率，提高在危险代码区域发现漏洞的概率。本文在 MIPS 架构 IoT 设备固件漏洞分析技术研究中取得了一定的成果，但是仍存在一些待解决的问题，可以对 MIPS 架构 IoT 设备固件漏洞分析技术进行更深入的研究，未来的工作可以从完整恢复程序控制流图，固件全系统仿真监控技术与分析更多类型漏洞等方面开展研究。

参 考 文 献

[1] State of the IoT 2020. 12 billion IoT connections, surpassing non-IoT for the first time[EB/OL]. (2020-11-19)[2022-05-08]. https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[2] ZHANG Z K, CHO M, WANG C W, et al. IoT security: ongoing challenges and research opportunities[C]//Proceedings of IEEE International Conference on Service-oriented Computing & Applications. [S.l.]: IEEE, 2014.

[3] WURM J, HOANG K, ARIAS O, et al. Security analysis on consumer and industrial IoT devices[C]//Proceedings of Asia & South Pacific Design Automation Conference. [S.l.]: IEEE, 2016: 519-524.

[4] JIN H K. A survey of IoT security: risks, requirements, trends, and key technologies[J]. Journal of

Industrial Integration and Management，2017，2(2)：1750008.

[5] China national vulnerability database (CNVD). 国家信息安全漏洞共享平台[EB/OL].[2022-05-08]. https：//www. cnvd. org. cn/.

[6] GIRALDO J，SARKAR E，CARDENAS A，et al. Security and privacy in cyber-physical systems：a survey of surveys[J]. IEEE Design & Test，2017，34(4)：7-17.

[7] 刘剑，苏璞睿，杨珉，等. 软件与网络安全研究综述[J]. 软件学报，2018，29(1)：42-68.
LIU Jian，SU Purui，YANG Min，et al. Software and cyber security—a survey[J]. Journal of Software，2018，29(1)：42-68. (in Chinese)

[8] CHEN D D，EGELE M，WOO M，et al. Towards automated dynamic analysis for linux-based embedded firmware[C]// Proceedings of Network and Distributed System Security Symposium. [S. l.：s. n.]，2016.

[9] 杨宇，张健. 程序静态分析技术与工具[J]. 计算机科学，2004(2)：171-174.
YANG Yu，ZHANG Jian. Program static analysis techniques and tools[J]. Computer Science，2004(2)：171-174. (in Chinese)

[10] 萨顿，格林，阿米尼段念，等. 模糊测试：强制发掘安全漏洞的利器[M]. 段念，赵勇，译. 北京：电子工业出版社，2013.

[11] KING J C. Symbolic execution and program testing[J]. Communications of the ACM，1976，19(7)：385-394.

[12] SHOSHITAISHVILI Y，WANG R，SALLS C，et al. Sok：(state of) the art of war：offensive techniques in binary analysis[C]//Proceedings of 2016 IEEE Symposium on Security and Privacy (SP). [S. l.]：IEEE，2016：138-157.

[13] THOMAS S L，CHOTHIA T，GARCIA F D. Stringer：measuring the importance of static data comparisons to detect backdoors and undocumented functionality[C]//Proceedings of European Symposium on Research in Computer Security. [S. l.]：Springer，Cham，2017：513-531.

[14] SHOSHITAISHVILI Y，WANG R，HAUSER C，et al. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware[C]//Proceedings of Network and Distributed System Security Symposium. [S. l.：s. n.]，2015：1-8.

[15] CHENG K，LI Q，WANG L，et al. D-Taint：detecting the taint-style vulnerability in embedded device firmware[C]// Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). [S. l.]：IEEE，2018：430-441.

[16] 李红辉，齐佳，刘峰，等. 模糊测试技术研究[J]. 中国科学：信息科学，2014，44(10)：1305-1322.
LI Honghui，QI Jia，LIU Feng，et al. The research progress of fuzz testing technology[J]. Science China：Information Sciences，2014，44(10)：1305-1322. (in Chinese)

[17] American Fuzzy Lop. American fuzzy lop (2.52b)[EB/OL].[2022-05-08]. https：//lcamtuf. coredump. cx/afl/.

[18] LibFuzzer. LibFuzzer：a library for coverage-guided fuzz testing [EB/OL]. [2022-05-08]. http：//llvm. org/docs/LibFuzzer. html.

[19] Honggfuzz. Honggfuzz [EB/OL]. [2022-05-08]. http：//honggfuzz. com.

[20] 吴世忠，郭涛，董国伟，等. 软件漏洞分析技术进展[J]. 清华大学学报(自然科学版)，2012，10：1309-1319.
WU Shizhong，GUO Tao，DONG Guowei，et al. Development of software vulnerability analysis technology[J]. Journal of Tsinghua University (Science and Technology)，2012，10：1309-1319. (in Chinese)

[21] ASHTON K. That "Internet of Things" thing[J]. RFID Journal，2009，22(7)：97-114.

[22] CHEN J，DIAO W，ZHAO Q，et al. IoTfuzzer：discovering memory corruptions in Iot through app-based fuzzing[C]//Proceedings of National Down Syndrome Society. [S. l.：s. n.]，2018.

[23] BELLARD F. QEMU，a fast and portable dynamic translator[C]//Proceedings of Annual Conference on USENIX Annual Technical Conference. [S. l.：s. n.]，2005.

[24] ZADDACH J，BRUNO L，FRANCILLON A，et al. Avatar：a framework to support dynamic security analysis of embedded systems' firmwares[C]//Proceedings of National Down Syndrome Society. [S. l.：s. n.]，2014：1-16.

[25] KOSCHER K，KOHNO T，MOLNAR D. SURROGATES：enabling near-real-time dynamic analyses of embedded systems[C]//Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT 15). [S. l.：s. n.]，2015.

[26] CHEN D D，WOO M，BRUMLEY D，et al. Towards automated dynamic analysis for linux-based embedded firmware[C]//Proceedings of National Down Syndrome Society. [S. l.：s. n.]，2016：1-8.

[27] 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. [S. l. ; s. n.], 2016: 437-448.
[28] TriforceAFL. TriforceAFL [EB/OL]. [2022-05-08]. https: //github. com/nccgroup/TriforceAFL.
[29] ZHENG Y, DAVANIAN A, YIN H, et al. FIRM-AFL: high-throughput greybox fuzzing of Iot firmware via augmented process emulation[C]// Proceedings of the 28th USENIX Security Symposium (USENIX Security 19). [S. l. ; s. n.], 2019: 1099-1114.
[30] BOHME M, PHAM V T, NGUYEN M D, et al. Directed greybox fuzzing[C]//Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security. [S. l. ; s. n.], 2017: 2329-2344.

作 者 简 介

潘祖烈
男,1976 年生,博士,教授,研究方向为网络空间安全
E-mail:panzulie17@nudt. edu. cn

王泰彦
男,1998 年生,博士研究生,研究方向为二进制代码相似性检测
E-mail:wangty@nudt. edu. cn

周 航
男,1997 年生,研究方向为 IoT 安全、嵌入式安全、漏洞分析与利用
E-mail:zhouhang@nudt. edu. cn

郭 徽
男,1991 年生,博士,研究方向为恶意代码检测
E-mail:vance-guo@outlook. com

责任编辑 董 莉