面向嵌入式 CGI 的内存破坏漏洞挖掘研究

王 东[1]，张小松[1,2*]，陈 厅[1]

(1. 电子科技大学网络空间安全研究院 成都 611731；2. 鹏城实验室网络安全研究中心 广东 深圳 518040)

【摘要】AFL-CGI-wrapper (ACW) 是桌面 CGI 程序的漏洞自动挖掘方法，其利用 QEMU 仿真器执行二进制 CGI 来实施模糊测试。但在嵌入式设备中直接应用 ACW 进行 CGI 漏洞挖掘会面临两个难题：1) 固定输入模型难以应对嵌入式设备的多样性；2) 主模块跟踪难以覆盖依赖外部调用的分支路径，导致漏洞挖掘效率低下。针对这两个问题，一种基于反馈的惰性输入模型和选择性外部函数跟踪被提出，并在原型系统 BCFuzzer 中实现。最后使用实际的嵌入式 CGI 程序集进行了实验，结果表明相比现有方法，其能自动探索更多的代码路径，也能更快地发现更多未知漏洞。

关 键 词 覆盖率驱动； 嵌入式设备； 模糊测试； 漏洞挖掘

中图分类号 TP393.08 文献标志码 A doi:10.12178/1001-0548.2019233

Research on Discovering Memory Corruption Vulnerabilities for Embedded CGIs

WANG Dong[1], ZHANG Xiao-song[1,2*], and CHEN Ting[1]

(1. Institute for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 611731;
2. Cyberspace Security Research Center, Peng Cheng Laboratory Shenzhen Guangdong 518040)

Abstract AFL-CGI-wrapper (ACW) is designed for detecting vulnerabilities in desktop CGI programs, the core idea is fuzzing CGI via executing the targeted program in the QEMU environment. However, there are two challenges when applying ACW on discovering vulnerabilities of embedded CGIs: 1) the diversity of devices makes fuzzing via fixed input-model is inefficient; 2) only tracing the main module of CGI cannot utilize coverage information to guide fuzzing to explore those code paths that are dependent on functions hosted in extern module. To overcome these two challenges, a lazy input-model based on feedback and a selective extern function tracing are presented and implemented in the prototyping system named BCFuzzer. Finally, several experiments had been conducted based on a set of embedded CGIs from some real-word devices. The result shows that the techniques of BCFuzzer can help to explore more code paths and detect more vulnerabilities as soon as possible.

Key words coverage guided; embedded device; fuzzing; vulnerability discovering

Gartner 公司预测 2020 年全世界部署的嵌入式设备在数量上将达到 200 亿，相关研究发现这类设备在开发设计上缺乏最佳安全实践，导致存在大量安全漏洞[1-4]，针对这类设备的漏洞挖掘已成为研究热点。文献 [5] 基于已知特征使用静态方法对大规模固件进行漏洞挖掘，但无法检测未知漏洞。文献 [6] 基于程序级仿真环境实现了通过动态测试来挖掘设备的 Web 接口漏洞；文献 [7] 基于系统级仿真环境实现了通过已知特征来检测漏洞。仿真环境通常对系统配置的依赖较多，而嵌入式设备的多样性使得仿真难以具备通用的配置。文献 [8] 基于设备控制应用程序进行协议报文模糊测试，但该方法只对具有控制应用程序的设备有效。文献 [9] 提出了一种设备和仿真相结合的方法，但需要较多人工参与。文献 [10] 基于符号执行实现了 MSP430 系列微控制器的漏洞挖掘，但无法用于其它控制器。文献 [11] 发现不同平台设备的内存破坏漏洞具有显著区别，基于动态测试的挖掘方法需要处理其差异。

针对上述问题，本文提出了一种基于模糊测试 CGI[12] 的嵌入式设备漏洞挖掘方法，其主要特点有：1) 受 ACW 的启发，利用覆盖率来驱动嵌入式 CGI 的模糊测试；2) 提出了一种惰性输入模

收稿日期：2019－10－24；修回日期：2020－03－06
基金项目：国家自然科学基金面上项目 (61572115，61872057)；国家重点研发计划项目 (2017YFB0802903)
作者简介：王东 (1985−)，男，博士生，主要从事网络安全、程序分析方面的研究.
通信作者：张小松，E-mail：johnsonzxs@uestc.edu.cn

型，克服了 ACW 的固定输入模型的低效问题；3) 设计了一种选择性外部函数跟踪，有效提升了路径探索能力；4) 继承了动态测试的优势，即不存在静态分析方法的误报。为验证上述方法的有效性，本文在原型系统 BCFuzzer 进行了相关实现，并使用嵌入式设备 CGI 程序集进行实验，结果表明：同 ACM 相比，BCFuzzer 能自动发现更多的代码路径，也能更快的发现更多未知漏洞。

1 模糊测试

模糊测试 (fuzzing) 是一种自动化或者半自动化的测试技术，它通过构造随机的非预期畸形数据作为程序输入，并监控执行过程中的异常行为来发现软件漏洞[13]。模糊测试技术分为黑盒、灰盒和白盒 3 种[14]。黑盒模糊测试实用面最广，因为它不关心被测试程序内部的实现细节，但这同时也导致了它的有效性较低[15]。白盒模糊测试需要跟踪被测试程序内部的详细信息，因此有效性是最高的，但存在诸多符号执行的限制[16]。灰盒模糊测试介于黑盒和白盒之间，跟踪被测试程序的部分内部信息，提升了模糊测试的有效性；同时，它的跟踪方法是轻量级的，能用于实际大型程序。其中，基于覆盖率驱动的灰盒模糊测试是已被实践证明有效的方法[17]，如算法 1 所示。

算法 1：ACW 模糊测试调度算法

```
Input: Seeds, CGI P
Result: MaliciousCases
fixedModel ← [“HTTP_COOKIE”, “REQUEST_METHOD”, “PATH”]
for Seed ∈ Seeds do
  for nuIterations ← 0 to Maxcnt do
    inputData ← Seed
    inputLen ← len(inputData)
    nuMutations ← RandInt(inputLen)
    for index ← 0 to nuMutations do
      byteIndex ← RandInt(inputLen)
      mutate(inputData, byteIndex)
    end
    envList,stdio ← ParseModel (inputData, fixedModel)
    status, cov ← ExecWithQemu (P, envList, stdio)
    if status is crash then
      Append inputData to MaliciousCases
    end
    if HasNewCoverage(cov) then
      Append inputData to Seeds
    end
  end
end
```

ACW 是面向桌面平台 CGI 的漏洞挖掘系统，其核心是基于覆盖率驱动的模糊测试[18]。算法 1 中的代码是 ACW 在经典模糊测试器 AFL 基础上实现的核心部分。其中，fixedModel 是预定义的 CGI 固定输入模型；ACW 采用经典数据变异方法对测试用例进行变异；ParseModel 依据 fixedModel 对变异后数据进行解析以还原成 CGI 需要的环境变量 envList 和标准输入 stdio；最后利用仿真器 QEMU 插桩执行 CGI。为提升执行效率，ACW 仅对 CGI 的主模块进行覆盖率跟踪。

2 BCFuzzer 系统

2.1 嵌入式 CGI 模糊测试的难点和解决方法

嵌入式设备多采用精简的 Web 容器和 CGI 程序，这使得 ACW 应用在嵌入式 CGI 漏洞挖掘中会面临两大难点，通过图 1 的示例 CGI 程序源码来说明。代码行 04 和 07 是位于不同的代码路径的两个漏洞。

```
01. if(strmcp(getenv('REQUEST_METHOD'), "POST")){
02.   if(strmcp(getenv('SelfHead'), "X")==0){
03.     char buf[1024];
04.     read(stdio,buf,getenv('CONTENT_LENGTH')); //crash1
05.     ...
06.   }else if(strmcp(getenv('HTTP_COOKIE'), "debugger")==0){
07.     abort(); //crash2
08.   }
09.   printf("bad headers");
10.   return 1;
11. }else{
12.   printf("invalid method");
13.   return 0;
14. }
```

图 1 示例 CGI 程序的源代码

难点 1：私有环境变量的生成问题。图 1 所示 02 行代码的 SelfHead 不是典型环境变量，他在 ACW 的固定输入模型中是不存在的。因此，图 1 的 ACW 模糊测试算法是无法生成该环境变量的，覆盖率驱动也始终无法探索到 04 代码路径的漏洞。这个难点的核心是，针对不同的 CGI 程序，应采用与之对应的输入模型。BCFuzzer 设计了一种基于反馈的惰性输入模型来克服该问题。该模型不再主动假定 CGI 需要的环境变量，而是先监视运行 CGI 程序并反馈其使用的环境变量，接着根

据反馈生成输入模型。这是一种惰性的处理方式，只有实际使用到的环境变量才会被添加到输入模型。BCFuzzer 通过该方法生成自适应的输入模型，提升测试效率。

难点 2：依赖外部函数的分支路径的探索问题。为减少硬件资源的消耗，图 1 的 06 行代码中 strcmp 这类基础函数在嵌入式平台上通常被动态链接到外部系统库，在主模块中只有调用描述信息。只跟踪主模板的 ACW 无法对调用的外部函数实施覆盖率驱动，而一次性变异就生成特征字符串 debugger 来触发 07 代码路径上的漏洞是困难的。这个难点的核心是，不能只跟踪主模块的覆盖率信息。BCFuzzer 利用静态分析技术来识别 CGI 程序中可能受输入影响的外部函数，然后针对他们也进行额外跟踪，从而利用覆盖率驱动来进入 ACW 难以到达的分支路径。这种有选择性的外部函数跟踪平衡了执行效率和代码路径的深入，提升了嵌入式 CGI 漏洞挖掘的效率。

BCFuzzer 的系统架构如图 2 所示，静态分析模块 (StaticAnalyzer) 负责自动分析 CGI 程序中外部调用的信息，调度模块 (Scheduler) 通过解析输入模型来生成测试数据，并调用执行模块 (QEMU) 运行 CGI；QEMU 在运行中通过 EnvMonitor 组件来监视 CGI 实际需要的环境变量，并通过选择性外部函数跟踪组件 (ExternCallTracer) 跟踪特定外部函数。

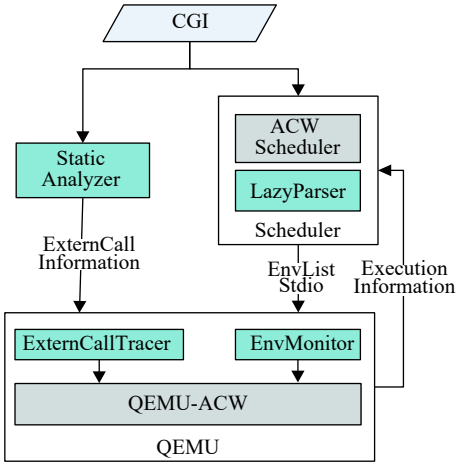


图 2　BCFuzzer 的系统架构

2.2　基于反馈的惰性输入模型

惰性输入模型包含两部分，一部分是图 2 中运行在 QEMU 的 EnvMonitor，负责监控 CGI 的环境变量；另一部分是图 2 中运行在 Scheduler 的 LazyParser，负责把测试数据解析为环境变量和标准输入。这两部分的调度如算法 2 所示。

算法 2：BCFuzzer 模糊测试调度算法

```
Input: Seeds, CGI P, ExternCallInfo ECI
Result: MaliciousCases
for Seed ∈ Seeds do
  for nuIterations ← 0 to Maxcnt do
    inputData ← Seed
    inputLen ← len(inputData)
    nuMutations ← RandInt(inputLen)
    for index ← 0 to nuMutations do
      byteIndex ← RandInt(inputLen)
      mutate(inputData, byteIndex)
    end
    envList,stdio ← ParseModel (inputData, Seed.model)
    status, cov ← ExecWithQemu (P, ECI, envList, stdio)
    if status is crash then
      Append inputData to MaliciousCases
    end
    if HasNewCoverage(cov) then
      model ← ExecWithQemu (P, ECI, envList, stdio, GENV)
      seedN ← (inputData, model)
      Append seedN to Seeds
    end
  end
end
```

相比算法 1，BCFuzzer 使用了惰性输入模型来生成测试数据。为支持惰性输入模型，扩展了 ACW 的用例数据结构 Seed，增加一个 model 成员来存储输入模型。模型解析时，在 Seed.model 中每遍历到一个环境变量名，就从变异数据中读取字符串作为环境变量值，从而形成环境变量键值对；然后将未对应到环境变量名的剩余数据作为标准输入 stdio，并形成数据长度的环境变量键值对；最后把这些传递给 QEMU 去执行。每当有变异数据探索了新路径，BCFuzzer 再次运行 CGI 并加载 EnvMonitor 来获取 CGI 需要的环境变量以生成新路径的输入模型。

2.3　选择性外部函数调用跟踪

选择外部调用跟踪包括两部分：外部函数调用的定位和覆盖率收集。针对外部函数调用的定位，

BCFuzzer 使用算法 3 所述的静态切片来识别 CGI 中特定外部函数的调用指令。首先通过 GetBinarayCFG 获取控制流图 cfg，其次通过 GetInputSource 获取全部的数据输入指令集合 sourceList，然后通过前向切片来分析输入数据传播到的目标指令，最后通过 IsExternCallIns 识别这些指令中的外部调用指令即得到了可能受输入影响的外部调用指令。

算法 3：特定外部函数的调用指令识别算法

```
Input: FuncNames, Target CGI P
Result: eCallList
cfg ← GetBinarayCFG(P)
sourceList ← GetInputSource(P)
for source ∈ sourceList do
  insList ← Forwad Binary Slice (source.addr, cfg, P)
  for ins ∈ insList do
    if IsExternCallIns(ins) then
      Append (ins.addr) to eCallList
    end
  end
end
```

BCFuzzer 使用算法 4 所示的基于外部调用能量的方法对 CGI 进行选择性的外部函数调用跟踪。

算法 4：基于基本块的选择性外部调用跟踪算法

```
Input: block,lastBlockInMain externCallEnergy
Result: TracedBlock
TracedBlock ← NULL
if IsMain(block) then
  if NOT lastBlockInMain then
    ClearEnergy(externCallEnergy)
  end
  if HasExternCall(block) then
    AssignEnergy(externCallEnergy)
  end
  TracedBlock ← block
  lastBlockInMain ← TRUE
else
  lastBlockInMain ← FALSE
  if HasEnergy(externCallEnergy) then
    TracedBlock ← block
  end
end
```

具体来说，该算法的核心机制是对外部函数进行覆盖率跟踪，必须要消耗能量 externCallEnergy。具体来说，QEMU 动态执行从 CGI 主模板切换到外部函数时，根据 CGI 的静态分析结果来判定这个外部函数是否需要跟踪。如果需要，就初始化能量 externCallEnergy，这样在外部函数代码执行期间就因有能量而被持续跟踪以收集覆盖率信息。最后，QEMU 执行返回到主模板时清空能量。

3 实验评估

为验证所提方法的有效性，本文在 ACW 的基础上实现了原型系统 BCFuzzer，其采用 LD_PRELOAD 机制和扩展 ACW 的用例数据结构来实现惰性输入模型[19-20]，使用 IDA Pro 脚本技术[21] 和 QEMU 动态插装实现选择性外部调用跟踪。最后通过固件下载和硬件读取方式收集多个厂商设备的 CGI 程序，并以此作为测试集进行实验验证。

3.1 实验环境

表 1 是收集的 77 个嵌入式设备 CGI 程序集的详情，覆盖了主流的嵌入式处理器架构 MIPS 和 ARM，以及数据端大小顺序。仿真器不支持 64 位指令，最后可测试的程序数量为 69 个。

表 1 嵌入式 CGI 程序测试集

设备 型号	CGI数量	MIPS 32	MIPS 64	ARM 32	小端 顺序	仿真运行
DG834	3	√	×	×	√	√
DGN1000	5	√	×	×	×	√
JNDR3000	6	√	×	×	×	√
R8500	3	×	×	√	√	√
K2	4	√	×	×	√	√
K1	4	√	×	×	√	√
E1700	2	√	×	×	√	√
LRT	6	×	√	×	×	×
DIR505	3	√	×	×	×	√
DIR645	16	√	×	×	√	√
DIR815	9	√	×	×	√	√
COCR3902	14	×	×	√	√	√
DAP3690	2	×	√	×	×	×

实验采用了 3 种基于覆盖率驱动的模糊测试工具：ACW、ACW-Full 和 BCFuzzer。ACW 是桌面平台 CGI 模糊测试器，这里整合了运行环境封装使其支持嵌入式 CGI；ACW-Full 是 ACW 的扩展，加入了对全部执行代码的强制插桩以获得最大化的代码跟踪；BCFuzzer 是在 ACW 的基础上整合了惰性输入模型和选择性外部调用跟踪。

实验采用 Linux 计算机，配置为 Intel I7 CPU，6 核 12 线程，内存 64 GB，磁盘 512 GB SSD，操作系统 Ubuntu1604，采用 docker 技术实施 CGI 程

序的并行测试。

3.2　路径发现实验

针对基于覆盖率驱动的模糊测试方法，路径探索能力直接影响其漏洞挖掘能力。为验证 BCFuzzer 对路径探索的提升，使用表 1 的程序集进行实验。因测试资源有限，实验对表 1 的 67 个 CGI 程序进行了精简，即同一厂商的不同设备的相同名字的 CGI 程序仅取一个作为代表，形成最终的 34 个程序集。每个 CGI 程序仅被测试运行 1 h，用时合计 102 h。

图 3 是 3 种模糊测试工具在每个 CGI 程序的测试过程中发现的路径数量。针对该实验结果分析可知：ACW 的路径发现能力最弱，这符合实验的预期，因为它采用固定输入模型且不跟踪外部调用函数，限制了路径探索；ACW-Full 的发现能力优于 ACW，这也符合实验的预期，因为 ACW-Full 通过跟踪全部执行代码收集了更多覆盖率信息，从而驱动发现更多的代码路径；BCFuzzer 针对绝大部分 CGI 程序 (30 个) 的发现能力都显著优于 ACW 和 ACW-Full，路径发现数量是他们的两倍及以上；而针对其他 4 个程序，其发现能力没有提升。由此可见，针对绝大部分嵌入式 CGI 程序，BCFuzzer 在路径发现能力上都优于现有的模糊测试工具。

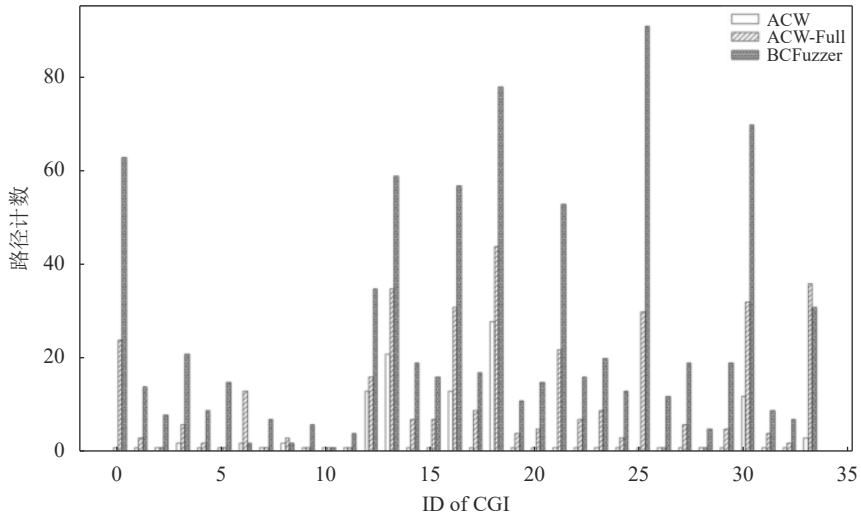
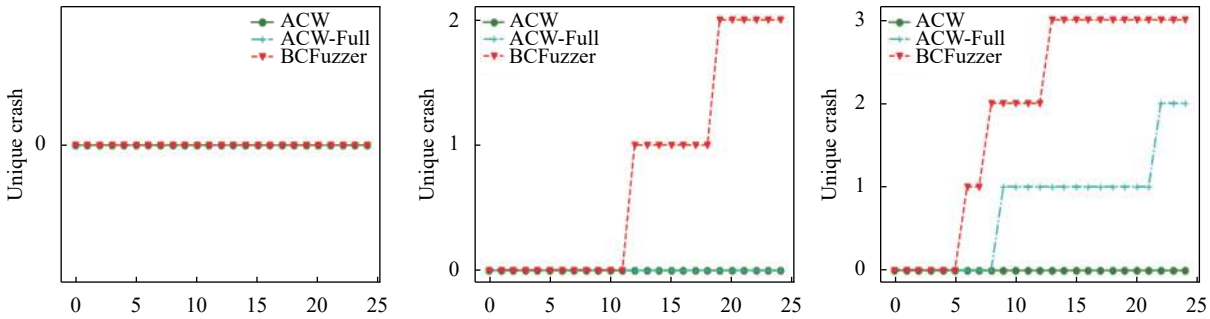


图 3　嵌入式 CGI 程序路径发现实验结果图

3.3　漏洞挖掘实验

实验对 CGI 程序分别采用 3 种模糊测试进行长时间的自动化测试，以此来验证他们的漏洞挖掘能力。使用路径发现实验的 34 个被测试 CGI 程序作为候选测试对象，因测试资源有限对 34 个程序进行了二次筛选，根据他们的体积最终选择了 6 个不同设备的 CGI 程序进行独立的 24 h 测试。

图 4 是挖掘实验的结果图，分析可知：ACW 的漏洞挖掘能力最差，针对 6 个嵌入式 CGI 没有发现一个崩溃；ACW-Full 的漏洞挖掘能力优于 ACW，发现了 2 个嵌入式 CGI 的崩溃；BCFuzzer 的漏洞挖掘能力最强，发现了 5 个嵌入式 CGI 的崩溃，数量是 ACW-Full 的 2.5 倍；针对 ACW-Full 发现崩溃的 2 个 CGI 程序，BCFuzzer 发现的唯一崩溃数量是 ACW-Full 的 1.5 倍。由此可见，BCFuzzer 比 ACW 和 ACW-Full 能发现更多的漏洞，并能更快地发现漏洞。



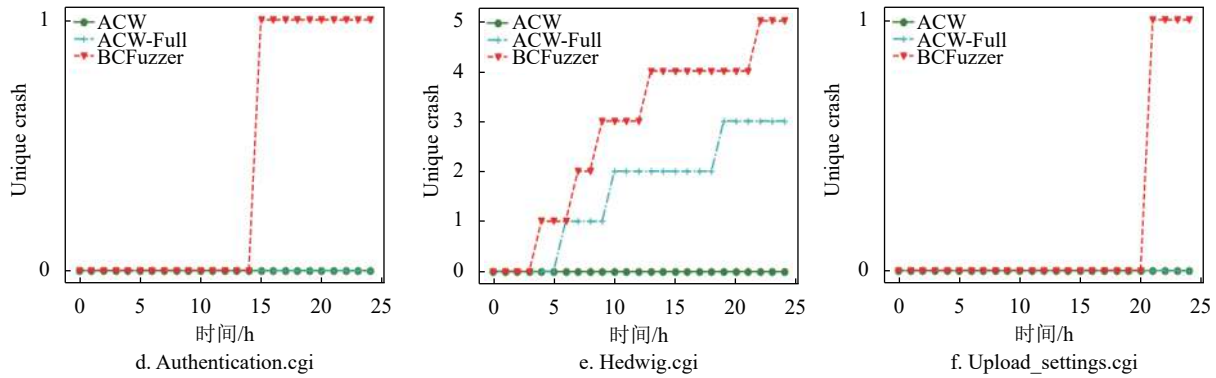


图 4 漏洞挖掘实验结果图

4 结 束 语

BCFuzzer 的惰性输入模型实现了自动生成 CGI 必需的环境变量，选择外部函数调用跟踪实现了依赖外部函数的条件分支的覆盖率驱动，提升了对嵌入式设备 CGI 漏洞的挖掘能力。最后通过实验验证了 BCFuzzer 的有效性。

参 考 文 献

[1] HO G, LEUNG D, MISHRA P, et al. Smart locks: Lessons for securing commodity internet of things devices[C]// Proceedings of the 11th ACM Asia Conference on Computer and Communications Security. New York: ACM, 2016: 461-472.

[2] GARTNER. Internet of things (IoT) market[EB/OL]. [2017-07-10]. https://www.gartner.com/newsroom/id/3598917.

[3] CONSTANTIN L. Hackers found 47 new vulnerabilities in 23IoTdevicesatDEFCON[EB/OL].[2017-05-23].http://www.csoonline.com/article/3119765/security/hackers-found-47-new-vulnerabilities-in-23-iot-devices-at-defcon.html.

[4] NAWIR M, AMIR A, YAAKOB N, et al. Internet of things (IoT): Taxonomy of security attacks[C]//2016 3rd International Conference on Electronic Design. Piscataway: IEEE, 2016: 321-326.

[5] COSTIN A, ZADDACH J, FRANCILLON A, et al. A large-scale analysis of the security of embedded firmwares[C]//Proceedings of the 23rd USENIX conference on Security Symposium. Berkeley: USENIX Association, 2014: 95-110.

[6] COSTIN A, ZADDACH J, 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. New York: ACM, 2016: 437-448.

[7] CHEN D D, WOO M, BRUMLEY D, et al. Towards automated dynamic analysis for linux-based embedded firmware[C]//Proceedings of Network and Distributed Systems Security Symposium. Reston: ISOC, 2016: 1-16.

[8] CHEN J, DIAO W, ZHAO Q, et al. IoTFuzzer: Discovering memory corruptions in IoT through app-based fuzzing[C]// Proceedings of Network and Distributed Systems Security Symposium. Reston: ISOC, 2018: 1-15.

[9] ZADDACH J, BRUNO L, FRANCILLON A, et al. AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares[C]//Proceedings of Network and Distributed Systems Security Symposium. Reston: ISOC, 2014: 1-16.

[10] DAVIDSON D, MOENCH B, RISTENPART T, et al. FIE: Finding vulnerabilities in embedded systems using symbolic execution[C]//Proceedings of the 22nd USENIX conference on Security. Berkeley: USENIX Association, 2013: 463-478.

[11] MUENCH M, STIJOHANN J, KARGL F, et al. What you corrupt is not what you crash: Challenges in fuzzing embedded devices[C]//Proceedings of Network and Distributed Systems Security Symposium. Reston: ISOC, 2018: 1-15.

[12] SHANG Guo-qiang, CHEN Yan-ming, ZUO Chao, et al. Design and implementation of a smart IoT gateway[C]// 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing. Piscataway: IEEE, 2013: 720-723.

[13] WU Zhi-yong, WANG Hong-chuan, SUN Le-chang, et al. Survey of fuzzing[J]. Application Research of Computers, 2010, 27(3): 829-832.

[14] ZOU Quan-chen, ZHANG Tao, WU Run-pu, et al. From automation to intelligence: Survey of research on vulnerability discovery techniques[J]. J Tsinghua Univ(Sci & Technol), 2018, 58(12): 1079-1094.

[15] BöHME M, PHAM V T, ROYCHOUDHURY A. Coverage-based greybox fuzzing as markov chain[J]. IEEE Transactions on Software Engineering, 2017, 45(5): 489-506.

[16] GODEFROID P, LEVIN M Y, MOLNAR D. SAGE: Whitebox fuzzing for security testing[J]. Queue, 2012, 10(1): 20-27.

[17] LCAMTUF. American fuzzy lop[EB/OL]. [2019-06-02]. http://lcamtuf.coredump.cx/afl/.

[18] FLOYD. Afl-cgi-wrapper[EB/OL]. [2019-03-09]. https://github.com/floydfuh/afl-cgi-wrapper.

[19] JIAO Long-long, LUO Sen-lin, LIU Wang-tong, et al. Fuzz testing for binary program based on genetic algorithm[J]. Journal of ZheJiang University (Engineering Science), 2018, 52(5): 1014-1019.

[20] ZOU Wei, GAO Feng, YAN Yun-qiang. Dynamic binary instrumentation based on QEMU[J]. Journal of Computer Research and Development, 2019, 52(4): 730-741.

[21] WANG Jing-feng, ZHANG Bao-wen, LIU Feng-ge. A method for detecting buffer overflow based on IDA Pro[J]. China Information Security, 2007, 12(3): 129-131.

编 辑 刘飞阳