文章编号:1009-3087(2016)02-0125-07 DOI:10.15961/j.jsuese.2016.02.018

基于污点分析的嵌入式设备固件模糊测试方法

戴忠华[1,2],赵 波[1*],王 婷[2],邹 力[1]
(1.武汉大学 计算机学院,湖北 武汉 430072;2.中国信息安全测评中心,北京 100085)

摘 要:针对现嵌入式设备固件的特点,提出一种基于污点分析的改进模糊测试方法。该方法首先从漏洞利用的角度分析固件的攻击面,然后根据攻击面导出相应的安全规则,并在污点分析结果中引入了测试用例危险权重,最后设计了与危险权重相对应的模糊测试用例集合。通过利用该方法对主流设备进行的漏洞挖掘实验,成功发现隐藏于设备固件中的若干零日漏洞。实验结果证明,该方法具备一定的有效性和实用性。

关键词:嵌入式;固件;漏洞挖掘;污点;危险权重
中图分类号:TP309 **文献标志码**:A

A Fuzzing Test Method for Embedded Device Firmware Based on Taint Analysis

DAI Zhonghua[1,2], *ZHAO Bo*[1*], *WANG Ting*[2], *ZOU Li*[1]
(1. Computer School, Wuhan Univ., Wuhan 430072, China; 2. China Info. Technol. Security Evaluation Center, Beijing 100085, China)

Abstract: Combing with the characteristics of embedded device firmwares, an improved fuzzy test method was proposed. After analyzing the attack surface of the firmwares from the standpoint of exploits utilization, several security rules were derived. By introducing the crisis weights of test cases in taint analytical results, a set of fuzzy test cases that are corresponding to crisis weights was designed. The method was used to dig vulnerabilities in popular equipments, and many zero-day exploits were found. Experimental results showed that this method is effective and practical.

Key words: embedded device; firmware; vulnerability detecting; taint; risk weight

在嵌入式系统的众多安全问题中,固件所带来的安全问题十分显著,嵌入式固件一般指的是存放在设备ROM中的只读代码和数据[1],固件中隐藏的安全漏洞会带来严重的安全威胁,这具体表现在以下几个方面:首先,设备使用数量规模逐年攀升;其次固件漏洞修复周期较长;再者,固件漏洞利用代码隐蔽性高;除此之外,固件代码具备高权限。这些特点决定了固件漏洞的高危特征。根据对CNNVD和CNVD等漏洞库的统计,近年来的嵌入式设备方面的漏洞数量呈现出激增态势,2010年后,相继出现了Stuxnet、Duqu和火焰病毒;2012年,Samsung被曝光其品牌下的打印机的固件存在管理账号后门;除此之外,利用智能汽车、家庭安防设备等嵌入式设备中存在的隐患进行攻击的行为也被大量披露[2-3]。如何保护嵌入式设备固件的安全已经成为保护嵌入式系统安全的重要内容之一,而漏洞挖掘技术能够在早期就发现潜伏在系统中的各种安全隐患,对于保护系统安全具有重要意义。

现有的漏洞挖掘方法根据使用场合的不同通常可以分为黑盒方法、白盒方法以及灰盒方法[4]。模糊测试(fuzzing)[5]是一种常用且极为有效的漏洞挖掘方法[6-8],在发现安全漏洞方面具备自动化程度

收稿日期:2015-10-15
基金项目:国家重点基础研究发展计划资助项目(2014CB340600);国家高技术研究发展计划资助项目(2015AA016002);国家自然科学基金重点项目资助(61332019);国家自然科学基金资助项目(61173138;61272452);湖北省重点新产品新工艺研究开发项目资助(2012BAA03004);华为创新研究计划资助项目(YB2013110084)
作者简介:戴忠华(1979—),男,博士生,副研究员.研究方向:嵌入式系统安全和云安全.
* 通信联系人 E-mail:zhaobo@whu.edu.cn
网络出版时间:2016-3-15 16:54:27 **网络出版地址**:http://www.cnki.net/kcms/detail/51.1596.T.20160315.1654.005.html

好等显著优点,但是存在效率低下、覆盖率低等问题,针对这些问题,不少优化方法被提出,例如,自动化测试空间覆盖方法[9]显著提高了测试覆盖率。除了上述优化手段外,采用污点分析[10]和模糊测试相结合的方法也能带来效率和覆盖率上的提升,文献[11-12]也都是将污点分析和黑盒测试相结合,作为增强测试穿透力和覆盖率的有效方法。

尽管模糊测试方法思想成熟,但是现有的测试方法和工具几乎均是针对通用平台或者网络协议,难以适用于嵌入式设备固件漏洞挖掘的研究场景。此外,由于嵌入式设备存在诸如固件规模较大、资源有限、平台封闭、特异性强等特点,现有的模糊测试方法通常无法直接移植使用,带来了漏洞挖掘效率低下、依赖研究者自身经验等问题,严重影响了嵌入式设备安全研究进程。目前针对嵌入式设备漏洞方面的研究[2-3,13-14]也都受困于其特殊性,难以形成系统性的分析方法,漏洞挖掘结果极大地依赖研究者本身的知识和经验。可见目前还比较缺乏具有普适效果的针对嵌入式设备固件的漏洞挖掘方法,因此需要研究如何将现有的漏洞分析方法进行改进,使其适用于嵌入式固件环境。

本文将污点分析技术引入嵌入式设备固件测试中,为测试用例引入了危险权重的量化指标,并形成针对固件的漏洞挖掘方法。实验发现通过判断输入数据字段的不同危险权重,筛选出改进的模糊测试用例集合,实现了测试用例空间的优化,提升了测试用例的穿透性、命中率,有效提高了嵌入式设备固件漏洞挖掘的效率。

1 嵌入式设备固件的攻击面简述

为了有效地进行污点分析并生成优化的测试用例,需要针对从漏洞利用的角度,总结出其攻击面,为后续生成安全规则基于污点分析判断输入字段的危险权重提供基础。

所谓攻击面(attack surface),就是软件环境中所有可供攻击的"点"(也称为攻击向量入口)的集合[15]。作者认为,如果嵌入式设备固件当中存在具备安全威胁的安全漏洞,那么必然存在与恶意攻击者相沟通的远程交互渠道,即该漏洞一定隐藏在设备与外界交互机制的某个环节之中。

因此基于该规律,同时考虑到攻击者的目的,针对嵌入式设备固件进行细化,提出固件中需要关注的攻击面包括以下方面(图1):

1)操作所有设备和资源的命令的进出路径;

2)保护这上述些管理操作的相应代码段;

3)所有具有价值的敏感数据,比如高权限用户的用户名、密码、口令、配置文件数据以及数据库等;

4)保护以上数据的相应代码段。

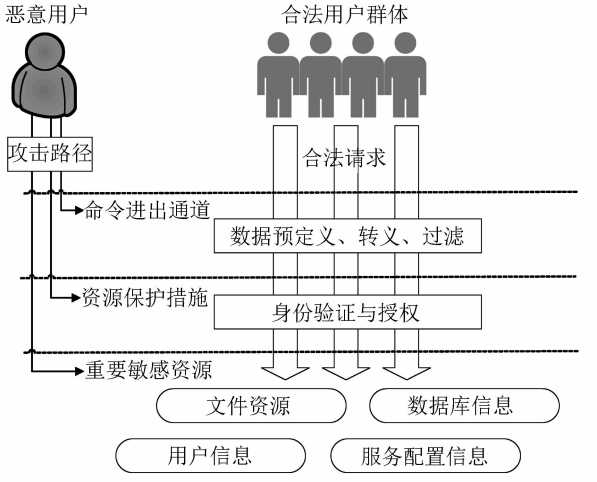

图1 攻击途径示意图

Fig. 1 Attack path diagram

2 污点分析过程

攻击面是从攻击者的角度对固件漏洞位置的大致判断,也是生成安全规则的基础。但是对漏洞挖掘来说远远不够,还需要进一步通过污点分析和跟踪确定恶意用户的输入数据是否能到达关键分支,为最终形成测试用例提供基础。

由于固件源代码不可获得,所以所有的污点分析过程均是基于逆向分析得到的汇编代码,因此采用动静态结合的污点分析方法。

2.1 扩展的污点分析数学模型

在描述污点分析如何在漏洞挖掘过程当中起到作用之前,有必要先详细描述在研究中涉及到的污点分析模型。该数学模型是对漏洞挖掘研究中整个污点分析过程的数学抽象。

目前较为复杂的污点分析模型常常结合了控制流、符号执行或约束求解算法等思想[10]。但是由于研究对象只有由固件反汇编而来的汇编代码,且这些代码冗余信息丢失严重。基于以上困难,本研究中基于牛津大学的 Heelan 提出的经典污点分析模型进行了扩展[16]。

定义1 污点源集合 I 为输入数据记录 L 的集合,在每个数据记录中,L 可以有更小粒度的组成单位。

定义2 集合 M 代表待考察的内存区域的地址集合,R 代表待考察的寄存器的集合。

定义 3　M_T 代表被污染的内存区域地址集合,R_T 代表被污染的寄存器地址集合。定义 V_T 是 M_T 和 R_T 的并集,即 $V_T = M_T \cup R_T$。

定义 4　定义变量 v,v 属于 M 和 R 的并集 V,记为 $V = \{v \mid v \in M \cup R\}$。

定义 5　如果变量 v 被写入了 I 中的记录或被写入了 M_T 或 R_T 中的数据,那么则称 v 被污染。被污染的变量记为 v_T,即属于被污染变量集合,记作 $v_T \in V_T$。

根据上述抽象化定义,污点传播模型整体是一个简单的状态机。状态机中最后包含了 5 个状态,分别是初始态、未污染、被污染、危险态和保留态,其中被污染和危险状态均是非安全状态。5 种状态的演变关系如图 2 所示。

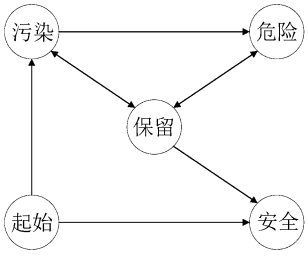

图 2　污点变量的状态变化

Fig. 2　Stain on the state of the variables change

所谓初始态,即程序还未运行时,所有相关变量均未经历读写指令操作,为初始态;污染状态即变量被写相关的指令写入了污点源集合 I 中的元素或污染变量集合 V_T 中的元素数据;所谓危险态指污点数据的传播动作,即当前的写相关指令处于攻击面对象范围之中,那么变量的状态则为危险态;保留态为状态覆盖而准备;安全状态则是为变量写入非污染数据而准备。

为了方便后续描述,根据状态的变化过程顺序,可以将 5 种状态赋予高低顺序,如图 3 所示,称处于下方的状态“小于”上方的状态。

图 3　状态大小关系定义

Fig. 3　State size relation definition

基于以上描述,经过抽象,定义以下运算:

定义 6　定义函数 $T(v)$ 为变量经过模拟执行后过程后最终确定的状态。为了方便阐述,定义 $T_i(v)$ 为变量 v 在指令 i 执行后返回的状态。

定义 7　定义函数 $S(v)$ 返回污染变量 v 的污染源。

由定义 7 可知,该函数返回的结果是一个集合,且该集合是污点源集合 I 的子集,而该函数的值空间是 I 的所有子集组成的集合。对安全变量使用该函数将会返回空集,对保留态的变量使用该函数将会返回传播历史的污点源。

根据上述函数,为了使模型更为完备,定义在变量集合 V 和函数 T 上的一类运算。

定义 8　如果变量 $v = x + y$,定义运算 $T(v) = \min\{T(x), T(y)\}$,即当某 2 个变量组合,成为组合变量,其函数返回值为 2 个变量分别返回的函数值的下界。

定义 8 表示若某个变量由污染变量和未被污染变量组成,那么,整个变量在执行后应该是被污染状态。原因在于,对于污点分析的研究实际上是污点的传播过程,状态的变化只能从被污染态进入污染态,而不能反方向,否则,对于漏洞挖掘的过程将失去意义。需要注意的是,初始状态不参与运算。

2.2　污点跟踪算法

污点跟踪算法是一个循环,分析每一条目标指令,对其是否为污点源进行判断,详细步骤解释如下:

Step 1:首先针对当前运行位置的指令进行分析,取该指令,判断其类型、指令功能、指令源操作数、指令目的操作数,为下一步的判断做准备。然后进入 Step 2 过程。

Step 2:根据 Step 1 得到的指令功能和类型判断该指令是否存在数据写功能。如果都不是,则单步前进,并回到 Step 2;否则,进入 Step 3。

Step 3:当该指令存在数据写入功能时,而且不是跳转指令,取操作数,判断其目的操作数是否存在污点数据的传播,即根据前述状态模型的定义,判断被操作数是否与污点源相关。若是,则将变量置为污染态,然后进入 Step 5;否则,则单步前进回到 Step 2。

Step 4:若该指令是跳转指令,则判断其跳转参数是否与污染数据有关。若有参数来源于污点变量,则单步前进,然后进入 Step 1;否则,等待返回,然后进入 Step 1。

Step 5:对于 Step 3 中的指令,根据前述攻击面

定义的安全规则对指令进行判定,若是,则将被操作数关联的变量置入危险状态变量集合,同时求污点源字段,该字段关联的危险权重自增,最后单步前进,转入 Step 1。

算法的伪代码描述如下:

```
Input:读取运行位置指令
Output:污点危险权重
Begin
For ValidIns(Ins) = True
    InsOperand = GetOperand(Ins)
    If InsOperand = 0 then
        Continue
    End if
    Var = GetVar(InsOperand)
    If IsTaint(Ins) = True then
        SetTaintState(Var)
        If IsDanger(Ins) = True then
            SetDangerState(Var)
            If IsInsDangerArea(Ins) = True
                Source = GetSource(Var)
                AddWeight(Source)
            End if
        End if
    Else if IsTaint(Ins) = False and IsCover(Var, Ins) = True then
        SetTempState(Var)
        End if
    End if
End
```

3 基于污点分析的测试用例生成

模糊测试是整个漏洞挖掘研究的落脚点,首先确定危险权重的累加判断规则,然后基于危险权重实现测试用例的生成,形成覆盖面广、穿透力强的测试用例。

3.1 安全判断规则

危险权重的值依赖于污染状态到危险状态的转换,该状态转换过程是有依据的,即根据当前分析的污点指令和变量是否符合安全判断规则。安全判断规则分为2个方面,一般以预先定义好的表的形式实现。

1)基于危险操作的判断规则

①C 语言中的危险库函数。在 C 语言的库函数中,存在着较多字符串处理、命令执行等类型的函数,例如,字符串拷贝 strcpy 等函数。很多不安全的代码由这些函数所引起。因此,在污点分析过程中,如果传播过程涉及到了这些危险库函数,那么则可以认为被分析的变量符合危险判断规则。

②代码中的可疑函数。嵌入式设备固件中存在的一大类高危漏洞是后门漏洞。考虑恶意用户利用后门漏洞的一般规律,通常为隐蔽信道或者权限绕过,如果传播过程涉及到了可疑的系统调用,那么则可以认为被分析的变量符合危险判断规则。

③危险指令序列。考虑到编译器的因素,部分危险操作并非直接经过库函数,而是会由编译器内联地展开成指令序列。因此也有可能存在危险。

2)基于敏感区域的判断规则

①命令进出入路径。无论合法还是非法,请求均会通过该路径进入不同的处理分支,如果污点传播发生在该路径上,则符合规则。

②身份认证和授权。该部分的代码是设备安全机制的核心,若其存在问题,威胁毋庸置疑。

③操作重要资源代码段。重要资源包括数据库、配置文件等。这些资源通常与设备管理息息相关,如果存在污点传播动作,则认为符合规则。

因此根据上述解释,如果污点传播发生满足2类条件中的1类,则可以认为污点变量处于危险状态,污点源危险权重自增。

3.2 测试用例生成

一般地,测试用例的形式为 $X = \{x_1, x_2, x_3, \cdots, x_n\}, n \in N$,其中,$x_n$ 表示测试用例中的1个字段,根据污点分析后每个字段有对应的危险权重,测试用例中不同字段的危险权重的形式为 $D = \{\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_n\}, n \in N$,其中,$x_n$ 与 α_n 相对应。

1)字段间的依赖关系处理

在测试过程中,常见的情况是为了进入某个分支,需要某个字段满足某个条件,例如,某 Web 服务器首先匹配 IP 地址后,才会去处理接下来的其他字段。为了提高测试用例的穿透力,有必要对此情况进行处理。

假设若干个字段存在依赖关系 $R = \langle x_i, x_{i+1}, \cdots, x_j \rangle$,即当这 $j - i + 1$ 个字段满足依赖条件才能测试到某个分支,此时将这 $j - i + 1$ 个字段看成一个字段,危险权重为其中的最大值,即 $D_R = \max\{\alpha_i, \alpha_{i+1}, \cdots, \alpha_j\}$。同时为了保证测试覆盖率,先将测试用例集分为无依赖集合和依赖集合,再分别进行测试。

2)不同字段对测试用例空间的影响

危险权重越大意味着更高的漏洞发现概率,而更高的概率则意味着需要更多的测试用例进行触发,因此测试用例的分布是有一定的规律的。在本研究中采取的依据是:污点分析中得出的污点源字段危险权重与根据该字段生成的测试用例数量呈正比。

假设无依赖和依赖测试用例总量分别为 M 和 N,那么,各个字段所代表的测试用例集合数量为:

① 无依赖情况下,以字段 x_i 为目标的测试用例规模为 $C_i = M \cdot \alpha_i / \sum \alpha_j$;

② 有有依赖情况下,以字段集 x_i^R 为目标的测试用例规模为 $C_i^R = N \cdot D_i^R / \sum D_j^R$。

字段危险权重的高低理应和字段在污点分析中的结果有关。在污点分析过程中,已经对每个污点源进行了危险权值的累积,而当运行完毕,该权重的累积过程也就结束。此时的危险权重代表了对应的污点数据字段到达危险区域的可能性,如果测试用例无法到达危险区域,那么,就可以确定根据该字段生成的测试用例无法触发导致危险的异常,因此在该字段上生成大量的测试用例将会极大地降低测试效率。

4　方法实现与实验结果分析

在原型系统的构建过程中,采用了 IDA Pro 作为污点分析插件的开发平台,以 Sulley 作为模糊测试的执行框架。

污点分析过程是本文方法的核心和关键,在实现中包含 4 个模块,模块间的工作流程如图 4 所示。

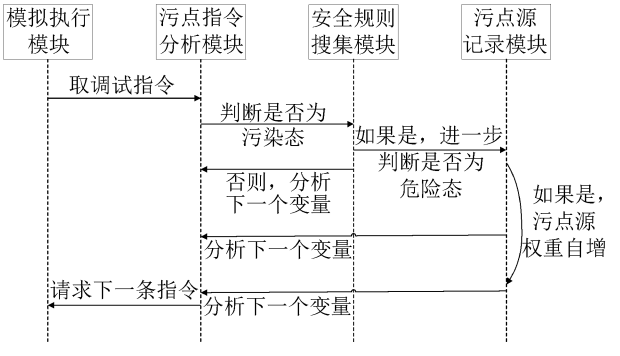

图 4　污点分析各模块工作流程

Fig. 4　Taint analysis work flow of modules

指令对污点的判断是通过预先定义好的表进行查找,以 MIPS 指令集为例,其主要污点指令如表 1 所示。

表 1　污点指令说明

Tab. 1　Command operation of taint

指令名	指令功能	说明
move	寄存器传送	主要污点传播指令
add	寄存器间加	与污点传播有关
abs/dabs	绝对值	与污点传播有关
neg/dneg	求反	与污点传播有关
and/andi	与	与污点传播有关
or/ori	或	与污点传播有关
sll/srl/sra	移位操作	与污点传播有关
slt/stli/sltu	条件设置	不存在污点传播
div/mul	除/乘	污点传播关系较小
lb/ld/ldl/ldr	加载数据	主要污点传播指令
sb/sdr/sc…	写入内存	主要污点传播指令
j/jr/jal/jalr	跳转指令	无关
b/beqz/bne	条件转移	无关

插件实现过程大量继承 IDA Pro SDK 中已定义好的数据结构,极大地降低了开发过程中的重复工作。

通过污点分析插件,结合现有的测试框架生成测试用例,在实际设备上进行了漏洞挖掘测试。

实验环境的操作系统为 Windows 8 宿主机和 Linux Kali、Windows XP,其中,Kali、Windows XP 运行于虚拟机软件 VMware 10.0 中;第 3 方软件和工具包括 Metasploit、Binwalk、Firmware Mod Kit、IDA pro 6.3、Sulley 等。由于工具和虚拟机对硬件要求较高,硬件配置包括 Intel core i3 3217U,4 GB 内存。所有软件和工具均能在该硬件平台上正常运行。

测试设备的选取具备 3 个原则:

1)进行测试的目标设备应该是目前在市场上广泛使用的设备,设备的生产厂家具备一定的市场知名度和市场占有率,用户群体较大,所发现的漏洞需要具备较高的价值和影响力;

2)目标嵌入式设备的固件是可以较为容易获取,不需要付出较大代价便可以从设备内部或者设备厂商处获得固件;

3)目标嵌入式设备固件未采用混淆、加密等措施进行保护。

经过对广泛、大量的品牌设备进行测试,发现污点分析插件得到的数据整体符合预期。以某设备固件中的 Web 服务器为例,污点分析结果显示输入数据的不同字段危险权重有明显差别,如图 5 所示。

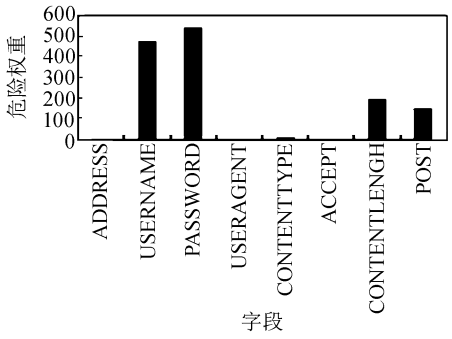

图5 某设备目标文件输入字段危险权重分布

Fig. 5 Distribution of risk weighted from the target file input field

本研究发现了多个嵌入式设备中存在多个异常,并有若干异常经过后续分析被确定为漏洞。将本文方法和传统的模糊测试的漏洞挖掘能力进行了对比,结果如表2所示。由表2可知,本文方法加入污点分析模块后比不加入污点分析模块的传统模糊测试,在漏洞挖掘能力和发现异常能力上优越性明显。

表2 漏洞挖掘能力对比

Tab. 2 Vulnerability mining ability comparison

方法	典型漏洞个数	典型异常个数
污点分析	6	4
传统模糊测试	2	0

进一步,给出本研究测试得到的漏洞成果数据,如表3所示,这几个漏洞均是以前从未发现过的漏洞,属于零日漏洞,具有极高的安全风险。

表3 典型漏洞

Tab. 3 Typical vulnerability

编号	漏洞类型	位置	漏洞详情
1	缓冲区溢出	Web	不安全的 strcpy
2	缓冲区溢出	Web	可控字符串过长
3	无身份验证	TFTP	默认 TFTP 开启
4	命令注入	Web	SHELL 命令注入
5	弱密码	TFTP	任意文件可下载
6	弱密码	ADB	ADB 默认开启

在实验中,还有部分影响较大的异常尚未发现其成因,典型的异常如表4所示。

综合分析整体上的漏洞结果和数据,本文设计的漏洞挖掘方法以及基于该方法实现的工具集,对嵌入式设备固件的分析较为有效,尤其对缓冲区溢出、命令注入、权限绕过等类型的漏洞具备较好的发现能力。

表4 典型异常

Tab. 4 Typical anomalous

编号	异常表现
7	当测试用例发送超过 200×10^4 字符串,即可导致 httpd 服务器崩溃,设备将拒绝服务,重启后恢复
8	当测试请求包含大于等于136个“\/”,可以导致设备崩溃,设备将拒绝服务,重启后恢复
9	当认证用户名字段超过622字节,服务进程崩溃,设备重启后服务恢复
10	当测试请求包头的数据包大小字段的内容大于实际数据包的大小时,设备服务崩溃,设备重启后服务恢复

5 结 论

针对固件规模较大和测试用例覆盖面过广的问题,利用污点分析技术设计了针对固件的模糊测试用例集,提高了测试用例集的覆盖率和命中率。提出的漏洞挖掘方法在实际设备固件上表现良好,效率较高、覆盖程度良好。从发现漏洞的实验成果来看,方法整体较为有效。

下一步的工作将集中于性能优化和ARM平台的进一步实现,同时将进一步分析输入字段间依赖关系对测试用例空间的具体影响。

参考文献:

[1]中国科学院研究生院国家计算机网络入侵防范中心. 2013年2月十大重要安全漏洞分析[J]. 信息网络安全,2013(4):98.

[2]Zaddach J, Costin A. Embedded devices security and firmware reverse engineering[R]. Las Vegas: Black-Hat USA, 2013.

[3]中国科学院大学国家计算机网络入侵防范中心. 2014年8月十大重要安全漏洞分析[J]. 信息网络安全,2014(10):93-94.

[4]Sutton M, Greene A, Amini P, et al. Fuzzing: Brute force vulnerability discovery[M]. Upper Saddle River: Addison-Wesley Professional, 2007.

[5]Zhu Xueyong, Wu Zhiyong, Atwood J W. A new fuzzing technique for software vulnerability mining using multi-dimensional inputs[J]. Journal of Communication and Computer, 2011, 8(2): 88-95.

[6]Lee H R, Shin S H, Choi K H, et al. Detecting the vulnerability of software with cyclic behavior using Sulley[C]// Proceedings of 2011 7th International Conference on Advanced Information Management and Service (ICIPM). Je-

ju:IEEE,2011:83 – 88.

[7]Li HongHui,Qi Jia,Liu Feng,et al. The research progress of fuzz testing technology[J]. Scientia Sinica Informationis,2014,44(10):1305 – 1322.[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.]

[8]Xiong Qi,Peng Yong,Yi Shengwei,et al. Survey on the Fuzzing technology in industrial network protocols[J]. Journal of Chinese Computer Systems,2015,36(3):497 – 502.[熊琦,彭勇,伊胜伟,等. 工控网络协议 Fuzzing 测试技术研究综述[J]. 小型微型计算机,2015,36(3):497 – 502.]

[9]Ding Yi,Lisnianski A. Fuzzy universal generating functions for multistate system reliability[J]. Fuzzy Sets and Systems,2008,159(3):307 – 324.

[10]Clause J,Li Wanchun,Orso A. Dytan:A generic dynamic taint analysis framework[C]//Proceedings of the 2007 International Symposium on Software Testing and Analysis. New York:ACM,2007:196 – 206.

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

[12]Zhu Guanmiao,Zeng Fanping,Yuan Yuan,et al. Blackbox fuzzing testing based on taint check[J]. Journal of Chinese Computer Systems,2012,33(8):1736 – 1739.[朱贯森,曾凡平,袁园,等. 基于污点跟踪的黑盒 fuzzing 测试[J]. 小型微型计算机系统,2012,33(8):1736 – 1739.]

[13]Liu Qixu,Zhang Chongbin,Zhang Yuqing,et al. Research on key technology of vulnerability threat classification[J]. Journal on Communications,2012,33(Z1):79 – 87.[刘奇旭,张翀斌,张玉清,等. 安全漏洞等级划分关键技术研究[J]. 通信学报,2012,33(Z1):79 – 87.]

[14]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.[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.]

[15]Manadhata P K,Wing J M. An attack surface metric[J]. IEEE Transactions on Software Engineering,2011,37(3):371 – 386.

[16]Heelan S. Automatic generation of control flow hijacking exploits for software vulnerabilities[D]. Oxford:University of Oxford,2009.

(编辑　赵　婧)