中的 418 个,识别率为约 98%.

1 相关工作

1.1 在线组件版本识别

著名的网络设备在线探测工具 Nmap[5]可根据 TCP/IP 协议在不同代码实现间的差异来识别远程主机的操作系统的版本.Anderson[9]基于 TCP/IP 和 HTTP 协议多会话模型特征,在网络协议的应用层中使用标语信息来识别目标组件的版本号,该方法经常被用来做渗透测试前期的组件扫描准备工作[10].上述方法均需要与待测组件有在线网络数据包的交互,然而固件组件的动态仿真执行的研究目前仍处于初级阶段[8,9],因此它们不能应用于本文所涉及的固件组件版本的离线识别场景中.

1.2 离线固件组件版本识别

我们在之前的工作[4]中首次提出了离线组件版本识别问题,设计和实现了一种基于版本字符串的组件版本识别框架 PANDORA,该框架共分为可读字符串提取、版本字符串定位、版本变量定位和版本字符串恢复等 4 个主要步骤.PANDORA 在对 2 683 个常用 IoT 组件的版本识别实验中准确识别出了其中 2 267 个组件的版本号.经人工分析后,发现剩下 416 个组件的版本号未被识别出来的原因在于组件中并不存在版本字符串.本文的研究动机在于解决此类版本字符串缺失组件的版本识别问题.

1.3 跨平台二进制函数关联

近年来,越来越多的物联网设备厂家将第三方代码库编译并部署在不同指令集 CPU 平台上,跨平台的函数关联的需求越来越大.Pewny[10]首次提出了跨平台函数关联的应用场景,并采用中间语言表示、数值采样和最小哈希等方法实现了跨平台(X86,ARM,MIPS)的基本块语义信息提取.Feng[11]提出了使用 ACFG 表示函数,并使用无监督学习模型将 ACFG 转换为数值向量,最后使用 LSH 技术完成对漏洞函数的快速检索.Xu[12]在 Feng 的基础上使用图嵌入网络对 ACFG 进行编码从而提升了检索性能.为了降低关联结果的虚警率,Feng 又提出了一种基于条件表达式的函数关联方法[13],所采用的条件表达式具有很强的鲁棒性,能够在跨平台编译时仍然保持一致.本文借鉴了前人工作[13]的方法用于提取高精度的函数特征用于构建版本签名.

本文设计和实现了一种不依赖于版本字符串的针对开源组件的版本识别方法.该方法的核心思想是通过开源组件相邻版本源码间的差异构造版本差异链,从而将版本识别问题转化为待查组件在版本差异链上的滑动查询问题. 进一步地,为了提高识别准确率,本文采用条件判断表达式来表征版本差异链上的节点.

2 预备知识

2.1 条件表达式

条件表达式(conditional formula,简称 CF)[12]包含一个执行操作(执行操作描述了由输入生成输出的过程)和一个与之绑定的执行条件(执行条件是一个布尔表达式,当执行条件为真时则执行与之绑定的执行操作,反之为假时则不执行).

用 Backus-Naur 格式描述的条件表达式如图 1 所示.条件表达式⟨*CondFormula*⟩包含执行条件⟨*Condition*⟩和执行操作⟨*Action*⟩构成,⟨*Condition*⟩与⟨*Action*⟩之间用箭头连接,说明当⟨*Condition*⟩成立时则执行⟨*Action*⟩,否则不执行.当⟨*Condition*⟩恒成立时,则⟨*CondFormula*⟩简化为⟨Action⟩.⟨*Condition*⟩为布尔表达式⟨*Expression*⟩;⟨*Action*⟩既可以是赋值式⟨*Assignment*⟩也可以是函数调用⟨*Function*⟩;函数调用⟨*Function*⟩由函数名⟨*FnName*⟩和参数列表⟨*ParamList*⟩构成;参数列表⟨*ParamList*⟩又由一个或多个表达式⟨*Expression*⟩构成;⟨*Expression*⟩可以是字符串名称⟨*Name*⟩、数值⟨*Number*⟩、函数⟨*Function*⟩、表达式间的二元运算⟨*Expression*⟩⟨*BinOp*⟩⟨*Expression*⟩、另一个表达式⟨*Expression*⟩或表达式的内存解引用[⟨*Expression*⟩];赋值式⟨*Assignment*⟩是将等号右边的⟨*Expression*⟩代表的值赋给等号右边⟨*Expression*⟩代表的对象.

⟨*CondFormula*⟩ ::= ⟨*Action*⟩
| ⟨*Condition*⟩ '→' ⟨*Action*⟩
⟨*Condition*⟩ ::= ⟨*Expression*⟩
⟨*Action*⟩ ::= ⟨*Assignment*⟩
| ⟨*Function*⟩
⟨*Function*⟩ ::= ⟨*FnName*⟩ '(' ⟨*ParamList*⟩ ')'
⟨*ParamList*⟩ ::= ⟨*Expression*⟩
| ⟨*Expression*⟩ ',' ⟨*ParamList*⟩
⟨*Assignment*⟩ ::= ⟨*Expression*⟩ '=' ⟨*Expression*⟩
⟨*Expression*⟩ ::= ⟨*Name*⟩
| ⟨*Number*⟩
| ⟨*Function*⟩
| ⟨*Expression*⟩ ⟨*BinOp*⟩ ⟨*Expression*⟩
| '(' ⟨*Expression*⟩ ')'
| '[' ⟨*Expression*⟩ ']'

Fig.1 Backus-Naur form for conditional formula

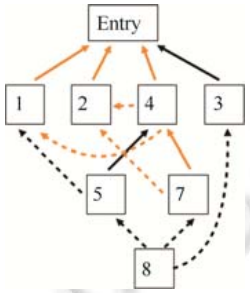
图 1 Backus-Naur 格式描述的条件表达式

2.2 程序切片

程序切片[14]是符号执行[15]、污点分析[16]、程序结构体恢复[17]等程序分析方法的支撑技术.程序 p 的关于切片准则$\langle s,v\rangle$的切片 $Slice(\langle s,v\rangle)$由 p 中所有影响变量 v 的语句或者受变量 v 影响的语句组成.前者称为后向切片,后者称为前向切片.本文方法所使用的是后向切片.按照程序的形式不同程序切片又分为基于源码的程序切片和基于二进制程序的程序切片.本文方法所使用基于二进制程序的切片.二进制切片是指面向二进制代码的程序切片技术,其中二进制代码可以用汇编代码或者中间语言代码表示.传统的程序切片分析工具[21]通常基于程序依赖图(program dependence graph,简称 PDG)进行二进制程序切片.PDG 由程序的控制依赖图和数据依赖图组成.若 $G_c=(V,C)$,$G_d=(V,D)$分别为程序 p 的控制依赖图和数据依赖图,则 PDG 为 $G_p=(V,E)$,其中 $E=C\cup D$.

```
1: int x = getch();
2: int y = getch();
3: int tmp = 0;
4: If ( x>y )
5:   tmp = x;
6: else
7:   tmp = y;
8: printf("%u\n",tmp);
```

(a) 源代码示例



(b) PDG 切片示例

Fig.2 Source code, PDG and slice example

图 2 程序源码、PDG 和切片示例

以图 2(a)中的源码为例,源码对应的 PDG 如图 2(b)所示,图 2(b)中的实线和虚线分别表示控制依赖和数据依赖. 源码部分第 7 行的后向切片如图 2(b)中黄线所示.

3 方法概述

对于一个具有 n 个版本的物联网设备开源组件,定义版本序列 $V:=\{v_1,v_2,\ldots,v_i,v_{i+1},\ldots v_n\}$,其中版本 v_{i+1} 代表版本 v_i 的下一个版本.令 s_i 表示版本为 v_i 的组件源码,B_{i+1} 表示由 s_{i+1} 编译得到的二进制文件.定义源码求差操 $diff(s_i,s_j):=\{\langle \Delta f,\{\Delta s\}\rangle\}$,其中 Δf 表示源码 s_i 相对于源码 s_j 所修改的函数,$\{\Delta s\}$表示该函数中所修改的语句.定义版本条件表达式字符串求解操作 $\phi(B_i,B_j,diff(s_i,s_j)):=\{cond\_str\}$,其中 $cond\_str$ 为根据 $d\in diff(s_i,s_j)$ 中标注的函

数和表达式定位出的 B_i 相对于 B_j 的一个版本条件表达式字符串.定义正向相邻版本差异签名 $sig_{i\to i+1}:=\phi(B_i,B_{i+1},diff(s_i,s_{i+1}))$.定义逆向相邻版本差异签名 $sig_{i\to i\text{-}1}:=\phi(B_i,B_{i\text{-}1},diff(s_i,s_{i\text{-}1}))$.$B^{input}$ 表示待识别版本的二进制组件.

定义正向查询签名:

$$Q^{+}_{B_{input}}:=\phi(B^{input},B_{i+1},diff(s_i,s_{i+1})).$$

定义逆向查询签名:

$$Q^{-}_{B_{input}}:=\phi(B^{input},B_{i\text{-}1},diff(s_i,s_{i\text{-}1})).$$

定义正向版本滑动相似度:

$$sim^{+}_{B_{input}}(i):=\frac{\cap(Q^{+}_{B_{input}},sig_{i\to i+1})}{\cup(Q^{+}_{B_{input}},sig_{i\to i+1})}.$$

定义逆向版本滑动相似度:

$$sim^{-}_{B_{input}}(i):=\frac{\cap(Q^{-}_{B_{input}},sig_{i\to i\text{-}1})}{\cup(Q^{-}_{B_{input}},sig_{i\to i\text{-}1})}.$$

基于以上的定义,版本签名链的匹配过程如图 3 所示.整个版本签名匹配过程实际上是一个代表当前版本的指针在一个双向链表上的滑动过程,该双向链表中节点 v_i 的“出边”为版本差异签名 $sig_{i\to i+1}$ 和 $sig_{i\to i\text{-}1}$,“入边”为版本差异签名 $sig_{i\text{-}1\to i}$ 和 $sig_{i+1\to i}$.滑动的规则如下:

若当前版本指针位置的正向版本滑动相似度 $sim^{+}_{B_{input}}(i)$ 大于阈值 δ 时,版本指针向右滑动;

若当前版本指针位置的逆向版本滑动相似度 $sim^{-}_{B_{input}}(i)$ 大于阈值 δ 时,版本指针向左滑动;

若当前版本指针位置的正向版本滑动相似度 $sim^{+}_{B_{input}}(i)$ 小于阈值 δ 且逆向版本滑动相似度 $sim^{-}_{B_{input}}(i)$ 小于阈值 δ 时,当前版本指针所对应的版本即为系统预测的输入组件的版本.

本小节中所涉及的两个关键算子 $diff(s_i,s_j)$和 $Q^{+/-}_{B_{input}}(i)$ 的详细设计将在接下来的两个小节中分别进行介绍.

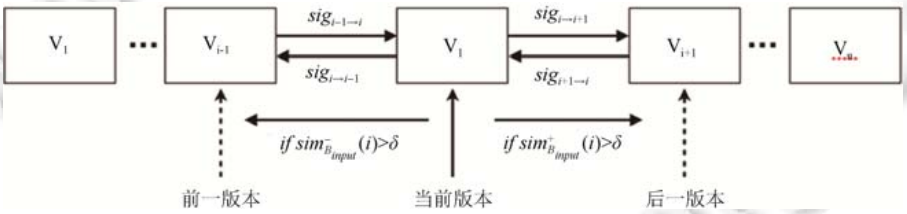


Fig.3 Version signature matching process

图 3 版本签名匹配过程

本小节详细介绍前述 $diff(s_i,s_j):=\{\langle\Delta f,\{\Delta s\}\rangle\}$算子的实现过程. $diff(s_i,s_j):=\{\langle\Delta f,\{\Delta s\}\rangle\}$算子的输入是来自同一个组件两个不同版本的源码 s_i 和 s_j,输出是标记出的版本差异函数和语句所对应二进制代码中的位置,整个位置标记过程如图 4 所示.首先从 Github、Source Forge 等开源组件发布网站上爬取物联网固件开源组件的版本仓库;然后基于源码 diff 工具比较相邻版本源码的差异,得到源代码中哪些函数被修改、该函数中哪些语句被修改了的信息;最后在从上述两份源码编译出的两份二进制代码中标记出所有被修改的函数 Δf 和语句$\{\Delta s\}$,构成算子$\{\langle\Delta f,\{\Delta s\}\rangle\}$.

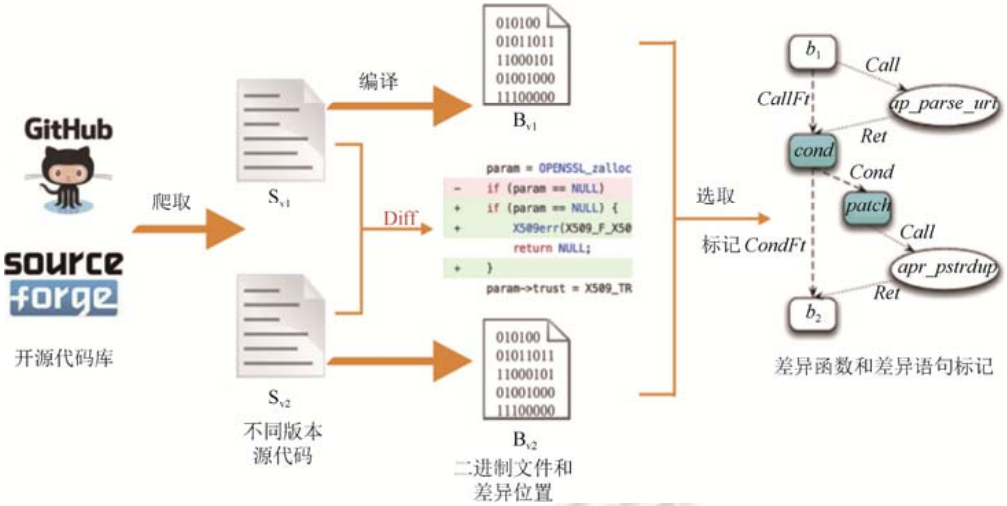


Fig.4 Difference function and difference statement automation mark

图 4 差异函数和差异语句自动化标记

4 版本签名条件字符串生成

本节详细介绍上述正向查询签名 $Q^{+}_{B_{input}}(i):=\phi(B^{input},B_{i+1},diff(s_i,s_{i+1}))$ 的生成过程,逆向查询签名 $Q^{-}_{B_{input}}(i)$ 与之类似,不再赘述. $Q^{+}_{B_{input}}(i):=\phi(B^{input},B_{i+1},diff(s_i,s_{i+1}))$ 算子的输入是待查询的二进制组件 B^{input} 和当前版本指针位置 i.输出是 B^{input} 与 B_i 比较得到的版本签名条件字符串. 整个版本签名条件字符串的生成过程如图 5 所示.

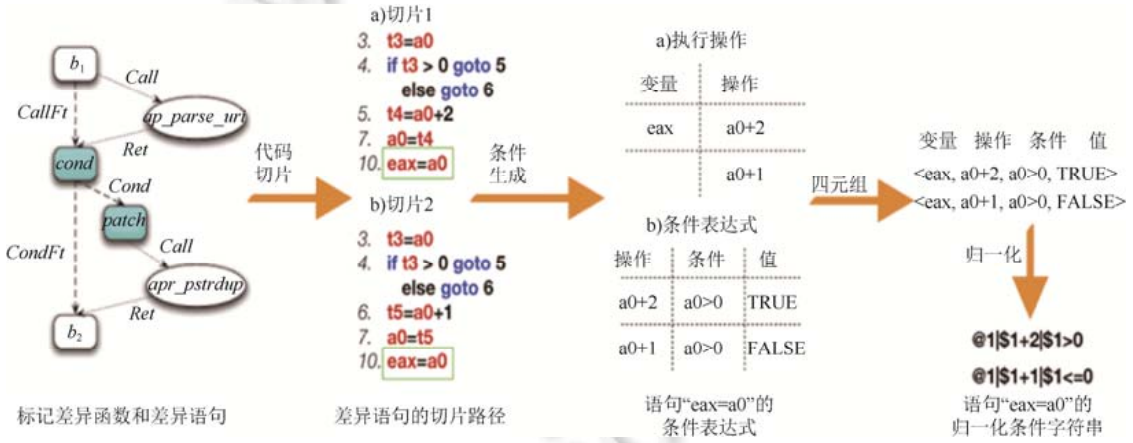


Fig.5 Version signature condition string generation

图 5 版本签名条件字符串生成

首先以 Δs 为槽点对 Δf 进行代码切片,如从图中的 *eax*=*a*0 出发得到了两段代码切片 *Slice*1 和 *Slice*2;然后利用数据流依赖分析得到图中描述执行操作和条件表达式的 *Action* 和 *ConditionalFormula* 表格;接着将条件表达式用四元组表示出来,如图 5 中的四元组⟨*eax*,*a*0+2,*a*0>0,*TRUE*⟩;最后考虑到交换律运算对操作数的不变性,对四元组进行字符串归一化处理,处理的规则为:

(1) 将输出变量按照出现的顺序依次编码为@*n*;

(2) 将输入变量按照出现的顺序依次编码为$*n*;

(3) 用“|”分割不同的域.

如图 5 所示,四元组⟨*eax*,*a*0+2,*a*0>0,*TRUE*⟩归一化之后的结果为 @1|$1+2|$1>0 .

5 实 验

5.1 数据集

本文实验采用解压自实际固件的不同版本的二进制组件来评估本方案的版本识别性能.为了尽可能真实地模拟使用场景,我们使用网络爬虫从 12 个物联网厂商官网爬取可下载的固件并对固件进行解压获取其中的二进制组件.我们对其中曾经爆出过严重安全漏洞的 Samba(CVE-2017-7494)、Msmtp(CVE-2016-6771)、Nginx(CVE-2017-7529)和 Libgcrypt(CVE-2017-0379)等 4 款二进制组件进行版本识别实验.组件所属的厂商及其版本分布情况见表 1.由表 1 可知,Samba 来源于 4 个厂商共 12 个版本,Msmtp 来源于 8 个厂商共 6 个版本,Nginx 来源于 3 个厂商共 12 个版本,Libgcrypt 来源于 5 个厂商共 5 个版本,因此该数据集具有一定的代表性.为了便于对比,按照能否被 PANDORA 正确识别版本为依据将数据集分开计数,见表 2.

Table 1 The source of data set

表 1 数据集来源

组件名称	生产厂商	版本			
Samba	Dlink, NetGear, Qnap, Zyxel	3.2.7 3.5.11 4.2.1	3.3.14 3.6.24 4.4.7	3.3.4 3.6.25 4.5.5	3.3.8 3.6.9 4.6.6
Nginx	JiRouter, Lenovo, NetGear	0.54.0 1.10.1 1.6.2	0.55.2 1.10.2 1.6.3	0.55.3 1.10.3 1.8.0	1.10.0 1.12.1 1.8.1
Msmtp	Trendnet, Foscam, Dlink, Linksys, Tenda, Zyxel, Openwrt, Ddwrt	1.4.27 1.6.1	1.4.31 1.6.2	1.4.32 1.6.5	
Libgcrypt	Dlink, JiRouter, Zyxel, Xiaomi, NetGear	1.5.3 1.7.5	1.6.1 1.8.3	1.7.2	

Table 2 The test data set of version identification experiment

表 2 版本识别实验测试数据集

二进制组件	PANDORA 可识别数量	PANDORA 未识别数量	总数量
Samba	63	7	70
Msmtp	140	8	148
Nginx	80	36	116
Libgcrypt	0	94	94

5.2 实验结果

首先,分别对 Samba,Msmtp,Ngnix 和 Libgcrypt 使用 PANDORA 进行版本识别,得到版本识别成功的组件集合和识别失败的组件集合;然后分布针对识别成功和失败的组件集合使用 PROTUES 进行版本识别.如图 6 所示,绿色实线代表 PANDORA 识别成功,红色实线代表 PANDORA 识别失败,绿色虚线代表 PROTUES 识别成功,红色虚线代表 PROTUES 识别失败.由图 5 所示的识别结果可知,PROTUES 的版本识别率接近 90%,且对于 PANDORA 未能识别的组件依然保持很高的识别率,说明两种方法具有很好的互补性,因此可以将两者串联起来使用,即先使用 PANDORA 进行识别,然后对与 PANDORA 不能识别的那些组件使用 PROTUES 进行识别.先使用 PANDORA 而不是 PROTUES 识别的原因在于 PANDORA 识别的成本较低,不需要事先构建版本签名库.单独使用 PANDORA 和 PROTUES 两种方法以及将两者串联起来时的版本识别性能如图 7 所示.由图 7 可知,串联后对 4 个测试组件的版本识别率均达到了 97%以上(Samba:98.6%,Msmtp:99.3%,Nginx:97.4%,Libgcrypt:94.7%),高于使用单一方法所能达到的识别率(PANDORA:90%,94.6%,69.0%,0%;PROTUES:88.6%,89.2%,97.4%,94.7%).特别地,对于 Libgcrypt 组件,由于缺乏版本字符串,PANDORA 的识别率为 0%,而 PROTUES 方法仍可以达到 94.7%的识别率.

综上所述,由于PROTUES方法不依赖于版本字符串因此适合与依赖版本字符串的放PANDORA串联使用,串联之后的版本识别率有了显著提升.在对来自于4种开源组件Samba,Msmtp,Nginx和Libgcrypt的共428个二进制文件进行的版本识别实验结果表明PROTUES能够准确识别的版本数达到418个,识别准确率约为98%.此外由于采用的是版本签名匹配而不是全组件匹配的方法,版本识别时间可以忽略不计.

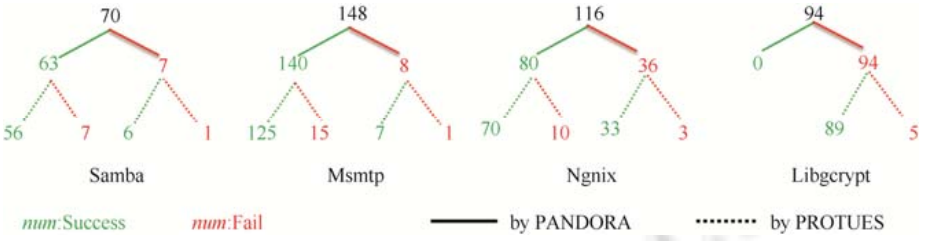


Fig.6 The tree of version identification results

图 6 版本识别结果树

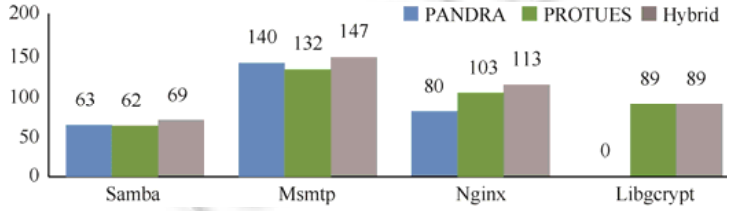


Fig.7 The improvement of two methods in series 6on version recognition performance

图 7 两种方法串联对版本识别性能的提升

6 结束语

本文设计和实现了一种基于条件表达式的开源组件版本识别方法.该方法不依赖于版本字符串,比传统方法具有更大的适用范围.

在对具有版本标签的428个组件的识别实验中,本文方法共正确识别出其中418个组件的版本号,识别准确率约为98%.实验表明本文方法能够很好地完成针对实际固件的组件版本识别任务.

本文方法的不足之处为需要事先获得待分析组件的源码才能构建版本签名,拟在下一步工作中研究基于二进制的版本补丁比较技术,并基于此直接构建物联网设备开源组件二进制版本签名.

References:

[1] Peng Y, Jiang CQ, Xie F, *et al*. Advances in information security of industrial control systems. Journal of Tsinghua University (Science and Technology), 2012(10):1396–1408 (in Chinese with English abstract).

[2] Mitchel LR, Chen IR. A survey of intrusion detection techniques for cyber-physical systems. ACM Computing Surveys. 2014,46(4): 55–56.

[3] Kushner D. The real story of stuxnet. IEEE Spectrum, 2013,50(3):48–53.

[4] Zhang WD, Chen Y, Li H, *et al*. PANDORA: A scalable and efficient scheme to extract version of binaries in IoT firmwares. In: Proc. of the 17th Int'l Conf. on Communications (ICC2017), IEEE, 2018. 1–6.

[5] Orebaugh A, Pinkard B. Nmap OS fingerprinting. Nmap in the Enterprise. 2008. 161–183.

[6] Anderson B, McGrew D. OS fingerprinting: New techniques and a study of information gain and obfuscation. In: Proc. of the 2017 Conf. on Communications and Network Security, 2017.

[7] Cui A, Stolfo SJ. A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan. In: Proc. of the 26th Computer Security Applications Conference. ACM Press, 2010. 97−106.

[8] Chen DD, Egele M, Woo M, *et al*. Towards automated dynamic analysis for linux-based embedded firmware. In: Proc. of the 2016 Conf. on Network and Distributed System Security. 2016.

[9] Zaddach J, Bruno L, Francillon A, *et al*. Avatar: A framework to support dynamic security analysis of embedded systems firmwares. In: Proc. of the 2014 Conf. on Network and Distributed System Security. 2014.

[10] Pewny J, Garmany B, Gawlik R, *et al*. Cross-Architecture bug search in binary executables. In: Proc. of the 37th Security and Privacy. IEEE, 2015. 709−724.

[11] Feng Q, Zhou R, Xu C, *et al*. Scalable graph-based bug search for firmware images. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2016. 480−491.

[12] Xu X, Liu C, Feng Q, *et al*. Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2017.

[13] Wang M, Wang M, Zhang M, *et al*. Extracting conditional formulas for cross-platform bug search. In: Proc. of the 2017 ACM on Asia Conf. on Computer and Communications Security. ACM, 2017. 346−359.

[14] Horwitz S, Reps T, Binkley D. Interprocedural slicing using dependence graphs. ACM Trans. on Programming Languages and Systems, 1990,12(1):26−60.

[15] King JC. Symbolic execution and program testing. Communications of the ACM, 1976,19(7):385−394.

[16] Bosman E, Slowinska A, Bos H. Minemu: The world's fastest taint tracker. In: Proc. of the Int'l Conf. on Recent Advances in Intrusion Detection. Springer-Verlag, 2011. 1−20.

[17] Caselden D, Bazhanyuk A, Payer M, *et al*. HI-CFG: Construction by binary analysis and application to attack polymorphism. In: Proc. of the European Symp. on Research in Computer Securi. Springer-Verlag, 2013. 164−181.

[18] Brumley D, Jager I, Avgerinos T, *et al*. BAP: A binary analysis platform. In: Proc. of the Int'l Conf. on Computer Aided Verification. Springer-Verlag, 2011. 463−469.

附中文参考文献:

[1] 彭勇,江常青,谢丰,等.工业控制系统信息安全研究进展.清华大学学报(自然科学版),2012(10):1396−1408.

张卫东(1986－),男,山东烟台人,博士生,主要研究领域为物联网安全,工业信息安全.

尹丽波(1973－),女,高级工程师,博士生导师,主要研究领域为信息安全.

李红(1989－),男,博士,助理研究员,CCF专业会员,主要研究领域为物联网安全,隐私保护,区块链.

文辉(1986－),男,博士,助理研究员,CCF专业会员,主要研究领域为物联网安全,数据挖掘,模式识别.

孙利民(1966－),男,博士,研究员,CCF 高级会员,主要研究领域为物联网安全,工控安全,无线传感网.