

嵌入式智能电表中的反汇编关键问题分析

刘金硕¹, 王谢兵^{1*}, 郑 稳², 邓 娟³, 陈 鑫¹

(1. 武汉大学 计算机学院, 武汉 430072; 2. 武汉大学 城市设计学院, 武汉 430072;

3. 武汉大学 国际软件学院, 武汉 430072)

(* 通信作者电子邮箱 jasonfor@gmail.com)

摘 要:电力企业在将反汇编技术应用于嵌入式智能电表软件一致性检测模型时出现了因不同硬件而产生差异的字节序问题和程序运行时的嵌入式设备内存受限问题,从而影响了模型整体的性能。针对这两个关键问题进行分析,结合嵌入式智能电表内部特征和硬件体系架构理论的深层探讨,依次提出了目标代码双逆置预处理算法(CDIPA)和分段反汇编算法(SDA)。对原代码使用目标代码双逆置预处理算法得到调整字节顺序的代码,分别将原代码和调整代码反汇编获得两种待测结果,结合硬件信息分析结果并以匹配度高的指令作为最终结果从而解决了字节序问题;分段反汇编算法通过调整反汇编输入规模并分次运行的方法解决设备内存受限问题。实验结果表明将这两种算法用于给定的电表设备,可以有效解决上述关键问题,同时算法也表现出良好的健壮性和可移植性。

关键词:嵌入式系统;字节序;内存受限;固件代码预处理;分段反汇编

中图分类号:TP311.5 **文献标志码:**A

Analysis of key disassembly problems based on embedded smart meter

LIU Jinshuo¹, WANG Xiebing^{1*}, ZHENG Wen², DENG Juan³, CHEN Xin¹

(1. School of Computer, Wuhan University, Wuhan Hubei 430072, China;

2. School of Urban Design, Wuhan University, Wuhan Hubei 430072, China;

3. International School of Software, Wuhan University, Wuhan Hubei 430072, China)

Abstract: Two key problems, endianness and memory capacity limit appear to be obstacles when electric enterprises implement a function consistency model for embedded smart meter software via disassembly technique, thus affecting the overall performance of the model. To solve these problems, a in-depth analysis was conducted combined with internal features of embedded smart meter and hardware architecture theory. Two algorithms named Code Double Inverse Preprocessing Algorithm (CDIPA) and Segmented Disassembling Algorithm (SDA) were proposed. CDIPA was used to generate adjusted binary code, together with raw binary as two inputs of disassembly. Thus endianness problem was solved by choosing the result more adaptable to hardware environment. SDA was adopted to decrease size of input binary so as to disassemble more times in limited memory. The experimental results show that CDIPA and SDA can effectively resolve the problems mentioned above and show up favorable robustness and portability.

Key words: embedded system; endianness; memory capacity limit; firmware code pre-process; segmented disassembly

0 引言

一般地,嵌入式设备具有如下特点:1)小巧;2)实时性;3)可装卸;4)固化代码;5)弱交互性;6)强稳定性;7)统一的接口^[1]。嵌入式开发人员需要了解嵌入式设备硬件资源不足的缺点,合理使用硬件资源开发高效的程序,这使得开发人员必须具备较高的技术水平。

反汇编即把程序的原始机器码翻译成便于阅读理解汇编代码的过程^[2]。作为软件逆向工程技术的一种实现方法,反汇编技术被广泛地运用于软件破解、病毒分析以及软件汉化等领域。目前反汇编技术相应的理论成果已经成熟,体系结构相对完整。

在智能电表的生产过程中经常出现制造商用于招标展示的样品表和竞标成功后大量投产的批量表之间存在显著差异的情况。由于检测力度不足,许多投入实际使用的批量表出现工作状态异常,对这些电表的维护造成了不必要的花费。针对此问题,电力企业提出了一种智能电表软件一致性检测方案并据此设计了智能电表软件一致性检测模型。模型以分析智能电表核心程序从而获取系统运行特征为思路,以反汇编算法分析电表固件代码功能为手段,对嵌入式智能电表进行软件功能差异测试。模型包括固件代码提取、固件代码反汇编和软件功能比较三大模块。在其核心的固件代码反汇编模块中需要将反汇编技术应用于嵌入式设备,此模块的实现需要综合嵌入式和逆向分析领域,并解决两者结合时产生的

收稿日期:2014-07-04;修回日期:2014-08-01。

作者简介:刘金硕(1974-),女,吉林辽源人,副教授,博士,主要研究方向:模式识别、图像处理、高性能计算;王谢兵(1992-),男,安徽无为,硕士研究生,主要研究方向:信息安全、嵌入式系统设计、大数据;郑稳(1989-),男,安徽六安人,硕士研究生,主要研究方向:计算机图形设计、虚拟现实;邓娟(1976-),女,湖北武汉人,副教授,博士,主要研究方向:数字图像处理、高性能计算;陈鑫(1989-),男,河南信阳人,硕士研究生,主要研究方向:图像处理、大数据。

关键问题。

与个人计算机相比,将反汇编算法应用于嵌入式设备中时存在硬件兼容性问题,如因不同硬件环境而产生差异的字节序问题。与此同时,考虑程序运行时嵌入式设备对内存分配的快速、高效和可靠性要求,设备内存受限成为制约反汇编性能的关键。在一致性检测模型中,字节序问题直接导致固件代码反汇编模块得不到正确结果,而设备内存受限问题造成模块执行效率较低。因此,解决这两个关键问题成为实现固件代码反汇编模块的前提。

针对字节序问题的研究大多基于软件层面,如考虑大小端存储模式对编程的影响^[3],编程时大小端的转换问题^[4]以及网络编程中的字节序处理^[5]等。而代码逆向技术中的字节序问题涉及硬件体系架构,上述方法力有不足。管海兵^[6]曾提出一种二进制翻译中的字节序调整方法,通过修改访存指令的有效地址完成不同字节序平台之间的程序翻译。虽然这种技术可应用于硬件层,但是其目的是完成不同平台之间代码的转换,与拟解决的问题目标并不契合,同时该方法需要首先将源程序转为中间指令,这一步仍旧进行了字节序调整的工作。

针对嵌入式设备内存受限问题:一方面学者们提出基于特定介质的存储方案,如郭建兴等^[7]提出的采用 NOR Flash 和 NAND Flash 混合设计的大容量、低成本存储系统的解决方案,但这类方案并不通用;另一方面有从内存动态管理的角度出发探究内存压缩管理的方法,如徐蓉^[8]提出了一种基于硬件的内存压缩系统模型;付湘等^[9]提出了一种适用于嵌入式设备的内存压缩机制,利用 Linux 的页面交换机制创建基于内存的交换分区。这类方法通常过于复杂,需要设计碎片管理算法,从而加大了系统的负荷。卢春鹏^[10]介绍了一种“一次分配,多次使用”的动态内存分配方法,虽然也涉及对内存的复杂管理,但是对于解决嵌入式设备内存受限问题有借鉴意义。

本文针对字节序和设备内存受限两个关键问题进行分析,结合对嵌入式智能电表内部特征和硬件体系架构理论的深层探讨,分别提出了不同的算法加以解决,最终的实验数据证明了算法的优越性。

1 问题描述

将固件代码反汇编技术应用于嵌入式设备中,需要考虑到硬件和软件两方面问题。硬件方面,从智能电表内部微控制单元(Micro Control Unit,MCU)芯片中提取的主控程序代码,其存储格式依赖于 MCU 内存单元的设置,如一个内存单元的字节数和高低字节顺序等。反汇编时需要预取硬件平台信息并加以分析,对其分析需要解决字节序问题。软件方面,嵌入式设备便携小巧的缺点导致必须设计低空间复杂度的反汇编算法,考虑程序运行时设备对内存分配的快速、高效和可靠性要求,对目标代码的一次性反汇编无法实现,因此需要解决设备内存受限问题。

1.1 字节序问题

对智能电表固件代码本身的研究关系到生成这段代码的硬件环境——智能电表 MCU 芯片,简单来说反汇编就是对照

指令集中的操作码将目标代码加以转换的过程。因此必须识别固件代码在 MCU 中的正确位置,并将其调整至目标处理器下的合理顺序,即固件代码在原 MCU 中的存储字节序。

字节序,又称端序或尾序(Endianness)^[11]。在计算机科学领域中,字节序是指存放多字节数据的字节的顺序,不同的处理器体系可能采用不同的字节序。字节序指示了一个字符的哪个字节存储在低地址。如果最低有效位(Least Significant Bit,LSB)在最高有效位(Most Significant Bit,MSB)的前面,即 LSB 为低地址,则该字节序是小端序(Little-Endian);反之则是大端序(Big-Endian)。不同的微处理器的体系架构不同,采用不同的字节序生成的目标代码也不相同,如何正确识别目标代码的字节序是一个亟待解决的问题。

1.2 设备内存受限问题

嵌入式系统的特点决定了其运行方式,和通用计算机不同,嵌入式系统的硬件和软件都必须高效率地设计,量体裁衣,去除冗余^[12]。

嵌入式设备的便携式特性决定了设备必须小巧轻便,导致设备无法承载更大的容量。由于嵌入式系统的内核比较小,而传统的大型工程类项目产品的核心代码规模比较大(至少达到几百 KB),因此核心代码在嵌入式设备内存中的一次性反汇编就无法实现,如何实现规模庞大的核心代码的完整性反汇编是一个难题。

2 解决方案

2.1 字节序问题

反汇编的流程是由目标代码得到汇编指令,目标代码的顺序直接影响最终结果。而字节序的差异体现在目标代码的顺序差异上,因此在运用反汇编算法处理之前调整目标代码为合理的顺序,使得对目标代码的语义理解和源机器上达到一致。

以智能电表 M16C 微处理器代码反汇编为例。从提取目标代码的角度出发,假设默认获取到的目标代码所在内存的实际地址是从小到大递增的方向,那么 Big-Endian 模式下,由于内存单元地址的方向性保持完整,因此可直接反汇编;而 Little-Endian 模式下,目标代码的地址递增的方向与真实指令的字节方向相反,所以需要进行处理。

Little-Endian 模式下,智能电表 M16C 嵌入式微处理器内存单元为 8 位,目标代码处理的过程如图 1 所示。

例如,假设提取的目标代码的十六进制表示形式为“0123456789ABCDEF02468ACE”,那么进行反汇编的代码应当为“CE8A4602EFCDA8967452301”。分析发现,这一步的过程等价如表 1 所示。

表 1 的过程用代码描述,即抽象为目标代码字节序问题的预处理方法——代码双逆置预处理算法(Code Double Inverse Preprocessing Algorithm,CDIPA),CDIPA 的具体流程如下所示:

- 1)开始,目标代码存放在数组中,数组元素用十六进制表示。
- 2)将目标代码长度设置为 4 的整数倍,不足添零。
- 3)部分逆置:将存放目标代码的数组以 4 个元素为一个

单元进行分割,并将每个单元内部对半逆置,即高低字节的逆置。

4) 前后逆置:将上面处理完的每个单元编号为 $0, 1, \dots, n-1$, 分别将编号和为 $n-1$ 的两个单元的内容进行逆置,即 0 单元和 $(n-1)$ 单元交换内容; 1 单元和 $(n-2)$ 单元交换内容; 2 单元和 $(n-3)$ 单元交换内容……

5) 结束。

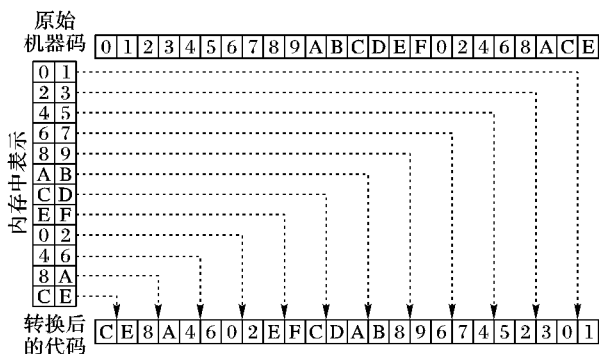


图1 目标代码处理流程示意图

表1 M16C 嵌入式微处理器机器码处理示表

输入	输出结果
目标代码	0123456789ABCDEF02468ACE
部分逆置	23016745AB89EFCDA602CE8A
前后逆置	CE8A4602EFCDA8967452301

整个算法中,将目标代码长度拓展为 4 的整数倍是为了便于之后的循环处理。CDIPA 涉及到两个阶段的逆置:部分逆置过程是为了调整每个字的高低字节,此过程采用一维数组即可实现;前后逆置过程是为了得到正确的指令组合,此过程采用二维数组结构实现。

由上述分析可知,在 Little-Endian 模式下获取的固件代码需要使用 CDIPA 加以处理,在未知大小端的情况下只需要比较目标代码在使用该算法处理后反汇编和不使用该算法处理直接反汇编的结果,通过分析两种结果与相应 MCU 指令集语法特征的相似程度即可得到代码在 MCU 中的真实端序和正确的反汇编结果。

2.2 设备内存受限问题

通用情况下,鉴于嵌入式设备内存容量的限制问题无法通过扩展更大容量的存储介质解决,因此考虑将相对庞大的目标代码分段进行反汇编。以智能电表 M16C 微处理器为例,其中目标代码大小为 185 KB,嵌入式设备内存为 32 KB。本文提出一种具体的分段反汇编算法 (Segmented Disassembling Algorithm, SDA), SDA 的具体流程如下所示:

1) 开始。 $i = 0$ 。

2) 将目标代码转换为十六进制表示并存入缓冲区 buffer。

3) 以 4 位为一个单元将目标代码编号。

4) 目标代码文件流是否为空,是则转 9); 否则转 5)。

5) 读取 $i \sim (i + 799)$ 号的代码单元进入内存。

6) 运用 CDIPA 依次反汇编 $i \sim (i + 798)$ 个单元。

7) 考察第 $(i + 799)$ 个单元是否为前一条指令的部分,识别下一个单元的成分:若是新指令的起始单元,则 $i = i + 800$,

转 4); 若是当前指令的部分,则 $i = i + 800$, 转 8)。

8) 寻找当前指令的起始单元编号 $start_id$, 得到当前指令的长度 len , 从 buffer 中读取 $[len - (799 - start_id)]$ 个单元进入内存, 将本条指令反汇编, 转 4)。

9) 结束。

对 SDA 的相关说明:

1) 步骤 2) 中的缓冲区 buffer 是目标代码存放区域的虚拟表示, 并不实际存储在内存中。

2) 步骤 7) 中识别下一个单元成分时, 下一个单元尚未读入内存, 只是对其进行分析。

3) 算法中的参数 800 是依据实例环境设置的, 在其他环境下可作测试可以相应修改。

4) SDA 可以实现分段读取目标代码进行反汇编, 考虑到内存处于多任务运行状态, SDA 设置每次读取的代码大小约占系统内存大小的 80%。

考虑不同嵌入式系统中的内存大小不同, 通过调整 SDA 中的相应参数, 即可实现不同硬件环境下的反汇编, 因此能够表现出良好的健壮性和可移植性。

3 实验分析

实验时使用了 RENESAS 的 M16C 系列芯片, MCU 型号为 R5F364AEDFA, 硬件环境: 内存为 4 GB, 处理器为 Intel Core i3-2120 CPU 3.30 GHz, 操作系统版本为 Windows 7 64 位。

3.1 CDIPA 实验

针对字节序问题, 在未知字节序是 Big-Endian 模式还是 Little-Endian 模式时, 使用 CDIPA 将目标代码分别进行处理, 从固件中提取的目标代码如下:

```
F27C0340ECFD02B37205070AC274318B7630FD556E8176301A6
E837630046EEAF400807630076EC27432DDF400C27433D7F400
8376300C6E807630076EC27434C7F4008376300C6E807631076E
C27435B7F4008376310C0201D950FE0BEBDBEBDF1473547D00
0404A18172018A3BEBDF1073507D000430A1047360729ADBF5F
DB47D4BEBDB14A1067211C9CE7DADF27D
```

表 2 是得到的汇编结果。

得到反汇编结果后可以根据 M16C 指令集的程序特征和语法规义检查对其进行判断。一般对于较复杂的指令集, 可以采用代码结构分析技术辅助决策。对照表 2 中的指令发现, Big-Endian 模式的程序结果偏离 M16C 指令集特征的幅度较大, 而 Little-Endian 模式的程序结果符合所属指令语法特征。因此得出结论: 本段目标代码在 MCU 中是按 Little-Endian 模式存储的。

3.2 SDA 实验

在 WinCE 系统下, 系统内存为 32 KB, 每次反汇编的代码段约为 25 KB ($32 \text{ KB} * 80\%$)。针对智能电表内存受限问题的 SDA 结果如表 3 所示。

由表 3 可知, 在源文件大小相同时, 最终形成的汇编指令文件 WinCE 比 Windows 7 上的略大, 这是因为不同段之间存在少许冗余。在 WinCE 上的运行时间远多于 Windows 7, 一方面是分段次数较多, 另一方面, WinCE 内存空间限制导致每次反汇编任务耗时较多。

表 2 汇编结果

Big-Endian 模式汇编结果		Little-Endian 模式汇编结果	
0000 MOV. B: S 0xF27C	0000 DEC. W A0	0051 ADD. B: S #0x00, R0H	004C JNE 0x0010F
0002 PUSHM SB	0001 MOVLL R1H, R0L	0054 MOV. B: S 118:8[SB], A0	004E MOV. B: G A1, -201:8[SP]
0004 MOV. B: S R0L, -131:8[FB]	0003 BCLR: S #0, -148:11[SB]	0057 MOV. B: S R0L, 12:8[FB]	0051 JMP. W 0x08352
0007 MOV. B: Z #0 R0H	0005 JSR. A &(2B302)	005B MOV. W: Q #0, R1	0054 OR. B: G #0x0C, R0H
0009 MOV. B: S R0H, 114:8[SB]	0009 MOV. B: S R0H, 7:8[SB]	005E JMP. B 0x00085	0057 MOV. B: S R0L, 1:8[FB]
000B MOV. B: S 7:8[FB], R0L	000B MOV. B: S -190:8[FB], R0L	0061 STZX #0xEB, #0x0B, 0xEBDB	0059 MOV. W: Q #5, R0
000E PUSH. W: S A0	000D MOV. B: G R0H, -245:8[SP]	0064 MUL. W #0x1473, A0	005B JMP. B 0x00067
0010 CMP. B: G #0x74, 49:8[FB]	0010 OR. B: G #0xFD, R0L	0067 BRK	005D INT #001B
0014 MOV. B: S R0L/R0H, A0	0013 BNOT: S #5, 110:11[SB]	006A NOP	005F INT #001F
0016 BNOT: S #5, -131:11[SB]	0015 TST. W [A1], [A0]	006D NOP	0061 AND. B: S R0L/R0H, R0H
0018 CMP. B: G #0x6E, R0H	0017 MOV. B: S R0L/R0H, A0	006F MOV. B: G -15:8[A0], R0H	0062 MOV. W: G A1, A0
001A OR. B: S 48:8[FB], R0L	0018 OR. B: S 110:8[FB], R0L	0072 STZX #0xEB, #0x01, 0x8A3B	0064 JMPL. A R2R0
001B CMP. B: G #0x6E, R1H	001A ADD. B: S #0x76, R0H	0073 MUL. W #0x1073, R0	0066 NOP
001C MOV. B: S R0L/R0H, A0	001C MOV. B: S R0L/R0H, A0	0074 BRK	0067 NOP
0020 NOP	001D NOP	0077 NOP	0068 ADD. W: G 2:8[A0], R1
0025 JMP. W 0x01011	001E JNE 0x00109	0079 ADD. W: G R3, R0	006B MOV. B: S R0L, -246:8[SB]
0028 TST. B R0L, R0L	0020 JMP. W 0x08021	007C MOV. W: G R0, A0	006D CMP. B: S 0xDFEB, R0L
0029 MOV. B: S 0x7630	0023 OR. B: G #0x07, R0L	007E MOV. B: G [A0], R0L	0070 AND. B: S R0L/R0H, R0L
002A MOV. B: G #0x6E, R1L	0026 JNE 0x000E9	007F JSR. A &(ADBF5)	0071 MOV. W: G A1, R0
002B JMP. W 0x01111	0028 MOV. B: G R1L, -163:8[SP]	0083 ADD. B/W: G #4, SP	0073 JMPL. A R2R0
002F MOV. B: G #0x00, R1L	002B JMP. W 0x0C22C	0084 STZX #0xEB, #0x4B, R0H	0075 NOP
0034 JMP. W 0x01111	002E MOV. B: G R1H, -169:8[SP]	0086 ADD. W: G R1, A0	0076 MOV. B: S R0L/R0H, A0
0036 CMP. B: G #0x00, R1H	0031 JMP. W 0x08332	0088 MOV. B: G R0L, [A0]	0077 ADD. W: G R0, A0
0038 TST. B [A0], 0C30:16[SB]	0034 OR. B: G #0x0C, R0L	0089 STZ #0xC9, 17:8[FB]	0079 MOV. W: G [A0], R0
003A MOV. B: S 0x7630	0037 JNE 0x000B8	008B DEC. B 125:8[SB]	007B MOV. B: G -5:8[A1], -139:8[SB]
003C MOV. B: G #0x6E, R1L	0039 OR. B: G #0x07, R0L	008D EXITD	007F JSR. A &(B7DB4)
003E JMP. W 0x01111	003C JNE 0x000FF		0083 INT #001B
0041 CMP. B: G #0x00, R1H	003E MOV. B: G A0, -185:8[SP]		0085 AND. B: S R0L/R0H, R0H
0044 TST. B [A0], 0C30:16[SB]	0041 JMP. W 0x08342		0086 ADD. W: G R0, [A0]
0047 MOV. B: S 0x7631	0044 OR. B: G #0x0C, R0L		0088 MOV. B: G R0H, R0H
004B MOV. B: G #0x6E, R1L	0047 JNE 0x000C8		008A ADD. W: Q #-4, AD7D:16[SB]
004E JMP. W 0x01101	0049 OR. B: G #0x07, R0H		008E DEC. W A0

表 3 WinCE 设备中反汇编结果数据

操作系统	系统内存/KB	源文件大小/KB	汇编指令文件大小/KB	运行时间/s
WinCE	32	185	658	452.738
Windows 7	4 194 304	185	537	10.258

由此可见,SDA 具有如下特点:

- 1)算法采用分段反汇编的策略,每次固定读取约占系统内存大小 80% 的目标代码进行反汇编,初步解决了核心代码的完整性反汇编。
- 2)对于较大的目标代码,算法将进行几百甚至几千次分段汇编,效率不高。算法耗时的主要原因是硬件的限制,最大限度地优化算法并不能有效缩短运行时间。
- 3)算法的思想对于具体实现嵌入式设备中代码完全反汇编,如目标代码片段的自动生成以及反汇编有一定的参考价值。

4 结语

本文针对固件代码反汇编技术在应用于嵌入式设备时出现的两个瓶颈问题——字节序问题和嵌入式设备内存受限问

题提出了对应的解决方法。

字节序问题 只需要在 Little-Endian 模式下使用 CDIPA 对目标代码进行处理,Big-Endian 模式下不作处理。在未知大小端的情况下需要比较目标代码在使用该算法处理后反汇编和不使用该算法处理直接反汇编的结果,通过分析两种结果与相应 MCU 指令集语法特征的相似程度得到正确结果。

嵌入式设备内存受限 采用 SDA 只需根据不同环境调整参数(每次读取的代码块大小)即可。

本文以 WinCE 系统中实现智能电表代码反汇编为实例测试,实验数据证明所提算法可以有效解决上述问题,同时算法也表现出良好的健壮性和可移植性。然而针对两个问题的方案仍存在不足,如 CDIPA 的优化,实现自动设定 SDA 的参数等,相信对这些问题的深入研究有助于解决方案的进一步完善。

参考文献:

[1] TU G, YANG F, HU G. Embedded operating system review[J]. Application Research of Computers, 2000, 17(11): 4-5. (涂刚, 阳富民, 胡贯荣. 嵌入式操作系统综述[J]. 计算机应用研究, 2000, 17(11): 4-5.)

Vine IL 的污点分析时间消耗如表4所示。

表4 污点分析优化结果

软件	静态指令数目	动态指令数目	分析时间/s		
			基于 Vine IL 的污点分析	生成语义规则	基于语义规则的污点分析
fg	39 230	1 882 960	52	4	1
Feiq	25 383	1 402 349	39	3	1
Httpd	21 956	1 333 704	35	3	1

由表4可以算出,本文方法的污点分析的时间消耗只占基于 Vine IL 的污点分析时间消耗 11.5% ~ 14.2%,其原因在于不重复的静态指令只占动态指令的 2% 左右,而本文方法正是只对不重复的静态指令生成语义规则,因此大大降低了污点分析的时间开销。

4 结语

在当前离线污点传播分析技术的基础上,设计了一种指令的语义描述规则,用于描述指令的污点传播语义,利用中间语言自动生成汇编指令的语义规则,再根据语义规则进行污点传播分析,避免了现有污点分析方法中指令重复执行导致的重复语义解析,提高了污点分析的效率。

参考文献:

- [1] LU K. The vulnerabilities detection technology research and implementation based on dynamic taint analysis [D]. Chengdu: University of Electronic Science and Technology of China, 2013. (陆开奎. 基于动态污点分析的漏洞攻击检测技术研究[D]. 成都: 电子科技大学, 2013.)
- [2] LIU Y, WANG M, SU P, *et al.* Communication protocol reverse engineering of malware using dynamic taint analysis [J]. *Acta Electronica Sinica*, 2012, 40(4): 661 - 668. (刘豫, 王明华, 苏璞睿, 等. 基于动态污点分析的恶意代码通信协议逆向分析方法[J]. 电子学报, 2012, 40(4): 661 - 668.)
- [3] LIU X. A non control data attack defense model design and implementation based on the pointer taint analysis [D]. Nanjing: Nanjing University, 2013. (刘小龙. 基于指针污点分析的非控制数据攻击防御模型的设计和实现[D]. 南京: 南京大学, 2013.)
- [4] WANG Z. Dynamic taint analysis of binary code based on symbolic execution [D]. Shanghai: Shanghai Jiao Tong University, 2010. (王卓. 基于符号执行的二进制代码动态污点分析[D]. 上海: 上海交通大学, 2010.)
- [5] WANG T. The vulnerability mining for binary program [D]. Beijing: Peking University, 2011. (王铁磊. 面向二进制程序的安全漏洞挖掘[D]. 北京: 北京大学, 2011.)
- [6] HUANG Z. An improved dynamic taint analysis model [D]. Wuhan: Huazhong University of Science and Technology, 2011. (黄昭. 一种改进的动态污点分析模型[D]. 武汉: 华中科技大学, 2011.)
- [7] PATIL H, PEREIRA C, STALLCUP M, *et al.* PinPlay: a framework for deterministic replay and reproducible analysis of parallel programs [C]// *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. New York: ACM, 2010: 2 - 11.
- [8] LUK C K. Pin: building customized program analysis tools with dynamic instrumentation [EB/OL]. [2014-06-12]. <http://www.cs.virginia.edu/papers/p190-luk.pdf>.
- [9] BRUENING D L. Efficient, transparent, and comprehensive runtime code manipulation [D]. Cambridge: MIT, 2004.
- [10] NETHERCOTE N, SEWARD J. Valgrind: a framework for heavyweight dynamic binary instrumentation [EB/OL]. [2014 - 04 - 20]. <http://valgrind.org/docs/valgrind2007.pdf>.
- [11] SONG D, BRUMLEY D, YIN H, *et al.* BitBlaze: a new approach to computer security via binary analysis [M]. Berlin: Springer, 2008: 1 - 25.
- [12] BRUMLEY D, JAGER I, AVGERINOS T, *et al.* BAP: a binary analysis platform [C]// *Proceedings of the 23rd International Conference on Computer Aided Verification*. Berlin: Springer, 2011: 463 - 469.
- [1] LU K. The vulnerabilities detection technology research and implementation based on dynamic taint analysis [D]. Chengdu: University of Electronic Science and Technology of China, 2013. (陆开奎. 基于动态污点分析的漏洞攻击检测技术研究[D]. 成都: 电子科技大学, 2013.)
- [2] WIKIPEDIA. Reverse engineering [EB/OL]. [2014 - 04 - 20]. <http://zh.wikipedia.org/wiki/%E9%80%86%E5%90%91%E5%B7%A5%E7%A8%8B>.
- [3] HE J. The influence of big-end and little-end storage pattern on programming and its countermeasures [J]. *Journal of Yangzhou Polytechnic College*, 2009, 13(2): 39 - 42. (何剑. 大小端存储模式对编程的影响及对策[J]. 扬州职业大学学报, 2009, 13(2): 39 - 42.)
- [4] CHEN H, ZHOU Z. Study on the method of transforming between big-endian and little-endian based on the experiment of embedded system [J]. *Research and Exploration in Laboratory*, 2008, 27(5): 66 - 68. (陈辉, 周自立. 嵌入式系统实验关于大小端转换方法的探讨[J]. 实验室研究与探索, 2008, 27(5): 66 - 68.)
- [5] JIANG Z. Endianness processing in network programming [J]. *Network Technique*, 2004(2): 74 - 75. (蒋振宇. 网络编程中的字节序处理[J]. 网络技术, 2004(2): 74 - 75.)
- [6] GUAN H. A method of endianness adjustment in binary translation [J]. *Technology and Market*, 2011, 18(7): 569. (管海兵. 二进制翻译中的字节序调整方法[J]. 技术与市场, 2011, 18(7): 569.)
- [7] GUO J, YAN P, HE Y, *et al.* Software design for a high capacity storage system in embedded devices [J]. *Computer Engineering and Applications*, 2004, 40(14): 102 - 105. (郭建兴, 鄢萍, 何勇, 等. 嵌入式设备中一种大容量存储系统的软件设计[J]. 计算机工程与应用, 2004, 40(14): 102 - 105.)
- [8] XU R. Memory compression technology of the embedded system [J]. *Microcontroller and Embedded Systems*, 2004, 5(2): 17 - 20. (徐蓉. 嵌入式系统中的内存压缩技术[J]. 单片机与嵌入式系统应用, 2004, 5(2): 17 - 20.)
- [9] FU X, NI H, ZHU M. Memory compress mechanism for embedded system [J]. *Computer Engineering*, 2007, 33(24): 83 - 85. (付湘, 倪宏, 朱明. 嵌入式设备中的内存压缩机制[J]. 计算机工程, 2007, 33(24): 83 - 85.)
- [10] LU C. A memory allocation scheme of the embedded system [J]. *Microcontroller and Embedded Systems*, 2002, 3(12): 12 - 16. (卢春鹏. 一种嵌入式系统的内存分配方案[J]. 单片机与嵌入式系统应用, 2002, 3(12): 12 - 16.)
- [11] WIKIPEDIA. Endianness [EB/OL]. [2014 - 04 - 20]. <http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82%E5%BA%8F>.
- [12] MA Y, LIU Y, ZHANG X. The situation and developing prospects of embedded systems [J]. *Information Technology*, 2001(12): 57 - 59. (马义德, 刘映杰, 张新国. 嵌入式系统的现状及发展前景[J]. 信息技术, 2001(12): 57 - 59.)

(上接第3510页)