

## 基于语义规则的污点传播分析优化方法

林伟\*, 蔡瑞杰, 祝跃飞, 石小龙

(数学工程与先进计算国家重点实验室, 郑州 450002)

(\*通信作者电子邮箱 leebeep@126.com)

**摘要:** 离线污点分析中的针对轨迹记录文件的污点传播分析的时间开销非常巨大, 因此研究快速高效的污点传播分析具有重要意义。针对上述问题, 提出了一种基于语义规则的污点传播分析优化方法。该方法定义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。实验结果表明, 所提方法能够有效降低污点传播分析的时间开销, 仅占传统基于中间语言污点分析的 14% 左右, 提高了分析效率。

**关键词:** 离线污点分析; 轨迹跟踪; 语义规则; 污点传播; 时间开销

**中图分类号:** TP393.08      **文献标志码:** A

### Optimization method of taint propagation analysis based on semantic rules

LIN Wei\*, CAI Ruijie, ZHU Yuefei, SHI Xiaolong

(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou Henan 450002, China)

**Abstract:** Time overhead of the taint propagation analysis in the off-line taint analysis is very large, so the research on efficient taint propagation has important significance. In order to solve the problem, an optimization method of taint propagation analysis based on semantic rules was proposed. This method defined semantic description rules for the instruction to describe taint propagation semantics, automatically generated the semantics of assembly instructions by using the intermediate language, and then analyzed taint propagation according to the semantic rules, to avoid the repeated semantic parsing caused by repeating instructions execution in the existing taint analysis method, thus improving the efficiency of taint analysis. The experimental results show that, this method can effectively reduce the time cost of taint propagation analysis, only costs 14% time of the taint analysis based on intermediate language.

**Key words:** off-line taint analysis; track recording; semantic rule; taint propagation; time overhead

## 0 引言

污点分析<sup>[1-3]</sup>作为一种新兴的程序分析技术,在协议格式逆向解析、恶意代码分析、数据结构恢复、隐私数据保护等安全领域发挥着重要作用。它的基本思想是把“不可信数据”或程序中需要分析的数据标记为污点<sup>[4]</sup>,比如系统环境变量、命令行参数、用户输入数据、文件数据及网络通信数据等,并对其的传播和使用过程进行监控和记录分析。

污点分析依照分析的时分为在线污点分析和离线污点分析。在线污点分析是指在执行过程中进行污点分析并输出结果。离线污点分析是指先记录程序的运行轨迹,然后依据轨迹文件进行运行时状态的还原以便污点分析。当程序对执行时间有要求时,如网络通信的超时判断、反调试的执行时间判断等,需要使用离线污点分析。

传统的离线污点分析中,首先在程序执行时记录指令信息为轨迹文件,然后根据轨迹文件中记录的每条指令的静态信息和运行时信息,还原任意时刻寄存器和内存的状态,提取每条指令的语义信息,然后再根据语义信息进行污点分析。

然而传统方法记录的指令中大部分均为循环导致的重复的指令,不重复的指令仅占指令总数的百分之一左右。因此对全部指令进行语义信息提取,再进行污点分析的方法因为重复的语义解析次数太多,降低了污点分析的效率。

目前,国内对如何提高污点分析中效率的研究相对较少,王铁磊<sup>[5]</sup>提出了使用有序二元决策图(reduced ordered Binary Decision Diagram, roBDD)的结构来进行影子内存的存储和管理,降低了污点分析中内存需求,提高了污点分析的性能,但在污点分析过程中依然需要对指令语义进行重复处理。黄昭<sup>[6]</sup>在污点数据标记模块中使用二级表动态标记方法替代了TaintCheck中的回溯链表,可以减少污点分析过程中的内存使用量,但增加了查找时间,并且对语义分析的重复性没有进行考虑。国外Patil等<sup>[7]</sup>对并行程序的执行进行分析和优化,研发了PinPlay平台进行轨迹重放,主要针对轨迹回放进行优化。但针对污点分析的轨迹记录优化的相关研究还处于起步阶段。

针对当前已有的污点分析方法时间开销过大的问题,提出了一种基于语义规则的高效污点传播分析方法,采用动态

收稿日期: 2014-07-07; 修回日期: 2014-09-16。

基金项目: 国家科技支撑计划项目(2012BAH47B01); 国家自然科学基金资助项目(61309007); 郑州市科技创新团队项目(10CXTD150)。

作者简介: 林伟(1986-),男,湖南常德人,博士研究生,主要研究方向: 逆向分析、网络协议分析; 蔡瑞杰(1990-),男,河南郑州人,硕士研究生,主要研究方向: 逆向分析、网络协议分析; 祝跃飞(1963-),男,河南郑州人,教授,博士生导师,主要研究方向: 密码学、网络信息安全; 石小龙(1988-),男,湖北襄樊人,硕士研究生,主要研究方向: 网络协议分析。

二进制插桩(Dynamic Binary Instrument, DBI)工具来记录程序的执行轨迹,首先进行去重处理得到静态指令序列,针对指令序列的每条指令初始化污点状态进行基于中间语言的污点传播分析,生成指令语义规则,再对整个轨迹文件中的每条指令依据语义库进行污点分析,最后得到污点处理结果。通过对 Windows XP 下的三款常用软件进行测试和性能分析,结果表明所提方法能够有效降低离线污点分析的时间开销,提高分析效率。

## 1 问题和解决思路

动态二进制插桩(DBI)技术是指在不影响程序正常执行流程的前提下,可选择性地在每条指令执行前或执行后插入额外的分析代码对指令进行实时分析。目前,DBI 技术广泛用于程序的动态分析,研究人员开发了多种支持 DBI 技术的二进制分析平台,比如 Pin<sup>[8]</sup>、DynamoRIO<sup>[9]</sup>、Valgrind<sup>[10]</sup>等。

其中 Pin 是 Intel 公司开发的一个动态二进制分析平台,提供了丰富的 API,使得开发者能够轻松构建各种二进制代码分析工具。同时基于 Pin 开发的工具运行效率很高。例如,使用 Pin 统计程序的基本块(Basic Block, BBL)速度要比 Valgrind 快 3.3 倍,比 DynamoRIO 快 2 倍。因此本文选取了动态二进制插桩工具 Pin 作为轨迹记录的基础平台。其工作流程如图 1 所示。

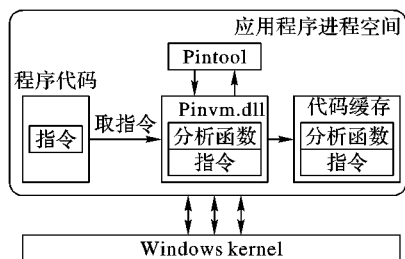


图1 Pin 工作流程

记录每条指令的静态信息和运行时信息,静态信息包括指令地址、指令机器码,运行时信息包括当前的线程号、指令读写的寄存器值以及读写的内存地址和内存值等。当记录程序执行轨迹后进行离线污点分析一般有两种方法。

1) 基于汇编指令的污点分析。这种方法的基本思想是解析指令的语义,根据指令的语义进行污点传播分析,比如对指令 `mov al, cl`, 分析指令的语义后可知该指令执行后会将操作数 `cl` 的污点状态赋值给 `al`。

2) 基于中间语言的污点分析。这种方法的基本思想是先将汇编指令翻译至一种中间语言表示,再基于中间语言进行污点分析。中间语言指令类型很少,语义功能非常简单,便于程序自动化分析。

上述两种方法的基本框架都如图 2 所示,逐指令读取执行轨迹中的指令,解析指令的语义,然后进行污点传播分析。这种方法存在的问题是如果是循环操作,同一条指令被执行多次,那么就需要对同一指令多次执行语义解析操作。比如统计了计算器软件 `calc.exe` 从启动到关闭所执行的指令数目,指令总数超过 1600 万,但是不重复的指令仅有 15 万左右,如果按照现有的方法,就需要解析全部 1600 万条指令的语义,其中重复的语义解析次数太多,降低了污点分析的效率。

因此提出了一种基于语义规则的污点分析优化方法,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,首先对执行轨迹所有不重复的指令自动生成其语义规则,然后再根据语义规则进行污点传播分析,减少重复性的语义解析工作,提高污点分析的效率,方法的基本框架如图 3 所示。

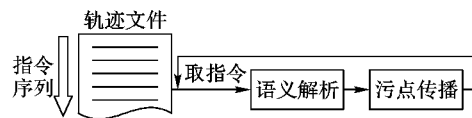


图2 现有污点分析方法框架

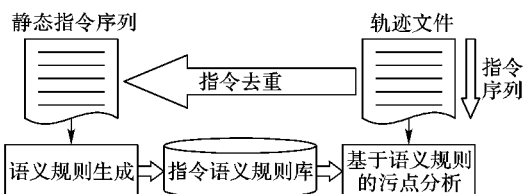


图3 基于语义规则污点分析优化方法基本框架

## 2 基于语义规则的优化方法

### 2.1 语义规则生成

污点分析是根据指令的语义来传播污点状态,因此其结果也会蕴含指令的语义信息,比如对指令 `mov al, cl` 来说,假设指令运行前寄存器 `al` 和 `cl` 所依赖的污点标签集合分别为  $\{1\}$  和  $\{2\}$ , 进行污点分析之后,寄存器 `al` 和 `cl` 的污点状态都变为  $\{2\}$ , 因此通过对比污点分析前后污点状态的变化, `al` 的状态由  $\{1\}$  变为了  $\{2\}$ , 而  $\{2\}$  是指令执行前 `cl` 的污点状态, 因此可以根据污点分析的结果推测该条指令的污点传播规则为  $T(al) \leftarrow T(cl)$ , 其中  $T(x)$  表示  $x$  的污点状态。

根据上面的分析,可以利用污点分析的方法来分析指令的语义信息,因为基于汇编指令的污点分析需要前期对指令集进行语义化建模,工作量较大,因此选择基于中间语言的污点分析来自动生成所有不重复指令的语义规则,总体流程如图 4 所示,其中静态指令序列是指在轨迹记录时记录的静态指令信息,每条指令只记录一次,也就是执行轨迹的指令序列去重之后的所有指令。对一条指令,生成其语义规则的基本流程是首先初始化污点状态,然后基于中间语言进行污点传播分析,最后对比找到污点状态发生变化的变量,生成指令语义规则。

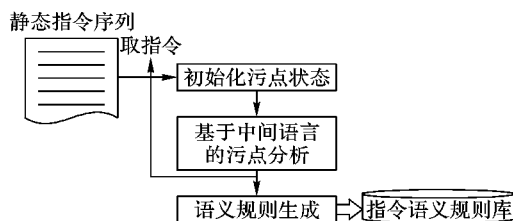


图4 语义规则生成流程

#### 1) 初始化污点状态。

将指令的寄存器和内存操作数初始化为污染源,首先对指令所有的操作数进行统一编号,使程序能快速从编号映射到实际的变量。

对于指令访问的寄存器,根据 Pin 对寄存器的编号,进行了如表 1 所示的扩展,对寄存器的每个字节赋予了一个唯一的编号,比如寄存器 `al` 的编号是 110, 根据这种编码方式,给定一个编号能够迅速定位到对应的寄存器字节,比如对编号 81, 高位的字节 8 表示寄存器 `ebx`, 低位的 1 表示第 2 个字节,

即寄存器 bh。

表1 语义规则中寄存器编码方式

寄存器	Pin 头文件中的序号	寄存器编号扩展
eax	11	110, 111, 112, 113
ebx	8	80, 81, 82, 83
ecx	10	100, 101, 102, 103
⋮	⋮	⋮

对于指令访问的内存,按照如下方式进行编号:若是读访问,则按内存地址从低到高,依次编号为1000,1001等;若是写访问,依次编号为2000,2001等。这样同寄存器一样,根据编号并结合指令实际访问的内存地址,则可以迅速定位到对应的内存单元。

根据以上编号规则,指令操作数的每个字节单元都分配了唯一的标签,污点传播结束时,根据依赖的标签就可以知道各字节单元所依赖的寄存器和内存单元。

## 2) 基于中间语言的污点传播。

基于中间语言污点分析的基本步骤是先将汇编指令翻译至中间语言,再解析中间语言的每条语句,进行污点传播分析。当前自动化程序分析有很多中间语言可以选择,比如 VEX、Vine IL<sup>[11]</sup>、BIL<sup>[12]</sup>等,中间语言会被设计成精简指令集,含有较少的指令类型和简单清晰的语法结构,并且能够处理隐式的操作数,比如指令 push eax 中的操作数 esp,同时有些中间语言会对 eflags 寄存器进行显示的处理,有效地解决了模拟 eflags 的难题。

以中间语言 Vine IL 为例来说明基于中间语言的污点传播过程。Vine IL 的语法规则如图5所示,其中包含8种语句结构(stmt)和8种表达式类型(exp),经过分析其中只有 MOVE 类型的语句会使污点传播,MOVE 语句格式为: LeftExp = RightExp, LeftExp 可能的类型有 TEMP 和 MEM,根据 RightExp 的类型,MOVE 语句的污点传播规则如表2所示。其中 T(x) 表示取 x 的污点状态。

program	::= decl* stmt*
decl	::= var var;
stmt	::= lval = exp;   jmp(exp);   cjmp(exp, exp, exp);   halt(exp);   assert(exp);   label label:   special string;   { decl* stmt* }
label	::= identifier
lval	::= var   var[exp]
exp	::= (exp)   lval   name(label)   exp ◊ b   exp   u   exp   const   let lval = exp in exp   cast(exp)   cast_kind   τ <sub>reg</sub>
cast_kind	::= Unsigned   U   Signed   S   High   H   Low   L
var	::= identifier; τ
◊ b	::= +   -   *   /   %   &   <<   >>   @   >   &   ^       =   <   >   <=   >=   <\$   <=\$   >\$   >=\$
◊ u	::= -   !
const	::= integer; τ <sub>reg</sub>
τ	::= τ <sub>reg</sub>   τ <sub>mem</sub>
τ reg	::= reg1_t   reg8_t   reg16_t   reg32_t   reg64_t
τ mem	::= mem32_t   mem64_t   τ <sub>reg</sub> [const]

图5 Vine IL 的语法规则

表2 污点传播规则

LeftExp	RightExp	传播规则	举例说明
TEMP/MEM	TEMP	T(L) = T(R)	L = R
TEMP/MEM	BINOP	T(L) = T(R1) op T(R2)	L = R1 op R2
TEMP/MEM	UNOP	T(L) = T(R)	L = ! R
TEMP/MEM	CONSTANT	T(L) = false	L = 0x1234
TEMP/MEM	MEM	T(L) = T(mem) *	L = [ mem ]

## 3) 语义规则生成。

语义规则生成的方式是找到污点分析之后污点状态发生变化的那些变量,比如以 xor eax, ebx 为例来说明语义规则生成方法。如图6(a)是根据编号规则初始化的污点状态;图6(b)中阴影部分表示污点分析之后污点状态发生变化的变量,比如 edx 的低8位依赖的污点标签由 {90} 变为了 {90, 110}, 因此可以生成规则“90 = 90, 110”,表示 dl 依赖于 dl 和 al;图6(c)是全部四条语义规则。

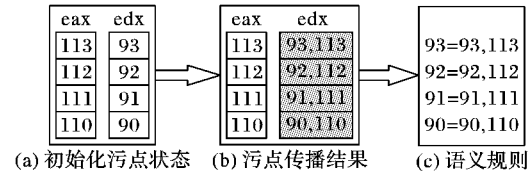


图6 xor eax, ebx 语义规则生成流程

对于一些特殊指令比如 CPUID 等,指令执行后寄存器的值会发生“突变”,中间语言也无法对这些指令进行处理,当识别到这些指令时,直接生成污点清除规则,比如“90 = 0”表示将寄存器 dl 的污点状态清除。

## 2.2 基于语义规则的污点分析

生成所有不重复指令的语义规则库后,逐条读取轨迹文件中的指令,根据指令地址查询相应的语义规则,再根据规则传播污点。比如对指令 xor eax, ebx, 查询到指令的语义规则如图6(c)所示,“90 = 90, 110”表示 dl 依赖于 dl 和 al,将 al 和 dl 的污点状态取并集然后赋值给 dl 即可。再比如对指令 mov al, byte ptr [esi] 的语义规则为“110 = 1000”,表示 al 依赖于指令读取的最小内存地址,根据记录的执行轨迹,得到该内存地址,查询该内存单元的污染状态,赋给寄存器 al。

## 3 实现与测试

实现了一个原型系统,采用动态二进制插桩工具来记录程序的执行轨迹,以 BBL 为单位,将寄存器作为程序的初始状态,记录内存访问的值,在轨迹记录过程中,仅在不确定事件发生时才记录寄存器的变态状态。最后对轨迹记录文件进行污点分析。为了验证该方法的有效性,对 Feiq 软件进行了分析。测试环境如表3所示。

表3 测试环境

项目	参数值	项目	参数值
操作系统	Windows XP sp3	CPU	i5-2400
内存	2.85 GB	硬盘	460 GB

实现了基于中间语言 Vine IL 的细粒度污点分析方法和基于语义规则的优化方法,选取了几款应用软件进行测试,分别采用两种方法对轨迹文件进行污点传播分析,结果如表4所示。可以看出去掉重复的语义解析后,污点分析效率提升了很多。基于 Vine IL 的污点分析之所以效率这么低,通过对不同模块分别计时,发现 80% 的时间消耗在中间语言翻译上,20% 的时间消耗在污点传播上,Vine IL 中间语言翻译的效率不高导致中间语言翻译占用了大部分时间,这种情况下,基于语义规则的优化方法优势更加明显。

基于语义规则的污点分析的时间消耗主要集中在四个方面:指令去重、语义规则生成、查找语义规则库和基于语义规则的污点分析。其中指令去重以内存地址为关键字,百万条指令消耗时间小于 0.1 s,可以忽略。另外两项与传统的基于

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%B>.
- [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)