一种基于匹配字符串地址判定ARM固件装载基址的方法

朱瑞瑾[1],张宝峰[1],毛军捷[1],骆 扬[1],谭毓安[2,3],张全新[2,3]
(1. 中国信息安全测评中心,北京 100085;2. 北京理工大学计算机学院,北京 100081;
3. 北京市海量语言信息处理与云计算应用工程技术研究中心,北京 100081)

摘 要: 固件是嵌入式系统的灵魂,当对固件进行安全检测或者深入理解固件中的运行机制时,对固件进行反汇编是一个必经的步骤.对固件反汇编时,首先要确定固件的装载基址及其运行环境的处理器类型.通常我们可以通过拆解硬件设备或者查阅产品手册获得处理器类型,但目前尚没有自动化工具可获知固件的装载基址.鉴于目前大部分嵌入式系统中的处理器为ARM类型,本文以ARM固件为研究目标,提出了一种自动化方法来判定固件的装载基址.首先通过研究固件中字符串的存储规律及其加载方式,提出了两个算法可分别求出固件中字符串偏移量和LDR指令加载的字符串地址.然后利用这些字符串信息,提出了DBMAS(Determining image Base by Matching Addresses of Strings)算法来判定固件的装载基址.实验证明本文提出的方法可以成功判定使用LDR指令加载字符串地址的固件装载基址.

关键词: 装载基址;固件;ARM;反汇编
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112 (2017)06-1475-08
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.06.028

Determining Image Base of ARM Firmware Based on Matching String Addresses

ZHU Rui-jin[1],ZHANG Bao-feng[1],MAO Jun-jie[1],LUO Yang[1],TAN Yu-an[2,3],ZHANG Quan-xin[2,3]
(1. *China Information Technology Security Evaluation Center,Beijing* 100085,*China*;
2. *School of Computer Science and Technology,Beijing Institute of Technology,Beijing* 100081,*China*;
3. *Beijing Engineering Research Center of Massive Language Information Processing and Cloud Computing Application,Beijing* 100081,*China*)

Abstract: Firmware is the soul of an embedded system,and disassembly is a necessary step to understand the operational mechanism or detect the vulnerabilities of the firmware. When disassembling a firmware,it should first determine the processor type of running environment and the image base of firmware. In general,the processor type can be got by tearing down the device or consulting the product manual. However,at present there is still no automated tool that can be used to obtain the image base of firmware. Since the processors of majority embedded systems are ARM architecture,in this paper we focus on the firmwares in ARM and propose an automated method to determine the base address. Firstly,by studying the storage rule and loading mode of the string we present two algorithms to calculate the string offset and the string address loaded by LDR instruction. Then with these information,we proposed a DBMAS (Determining image Base by Matching Addresses of Strings) algorithm to determine the image base. Experimental results indicate the proposed method can successfully determine the image base of firmware that uses the LDR instruction to load string address.

Key words: image base;firmware;ARM;disassemble

1 引言

从随身携带的手机、智能手环、智能手表到路由器、交换机、固态盘等,嵌入式系统已经遍及社会的方方面面.最近暴露出多起与嵌入式系统固件的安全漏洞相关的事件,如NSA开发出入侵硬盘固件的恶意软件[1,2],Stuxnet病毒入侵伊朗核电站PLC[3,4],Cisco、D-LINK、Tenda、Linksys、Netgear等品牌多款路由器固件中

收稿日期:2015-11-05;修回日期:2016-11-28;责任编辑:梅志强
基金项目:国家自然科学基金(No.61370063);国家重点研发计划网络空间安全重点专项

被爆有后门[5~8]等等. 嵌入式系统固件的安全问题越来越受到重视,为了提高嵌入式系统的安全性,更好地保护用户的信息安全,需要对固件进行安全检测,反汇编是固件安全检测中一种常用和重要的方法. 而当反汇编器(如 IDA Pro[9])处理固件时需要已知固件的装载基址(即固件映射到嵌入式系统内存中的起始位置)及其运行环境的处理器类型. 正确装载基址可以使反汇编器建立准确的交叉引用[10],包括字符串引用,函数引用等,这些信息对于固件分析人员理解固件有重要意义. 同时,正确装载基址有助于在整体上理解固件的内存布局,而错误的装载基址导致引用立即地址的指令寻址错误[11]. 进行反汇编时,处理器类型通常可以通过拆解硬件设备或者查阅产品手册得到,而对于未知格式的固件文件,很难直接获得其装载基址.

国外学者 Igor Skochinsky[12] 给出了针对嵌入式系统固件中未知格式文件人工推理装载基址的常用线索,其中包括自重定位代码(Self-relocating Code)、初始化代码、跳转表等. Zachry Basnight 等人[11,13]给出了两种人工推理出装载基址的方法. 第一种方法利用 ARM 指令中的立即数和固件更新的配置文件人工推理出合理装载基址. 第二种方法使用硬件调试器连接 PLC 设备,当 PLC 挂起时使用调试器可从 PLC 设备中获得其内存转储,通过人工分析内存转储中常见的 ARM 指令模式可发现固件的装载基址. Italo Dacosta 等人[14]指出在 C 语言中当 switch 语句中 case 的值是连续且稠密时,这些 case 语句块的内存地址通常存储在一个跳转表中. 利用该跳转表可推导出附近代码的内存地址,从而利用代码的偏移量和内存地址得到文件的装载基址. Santamarta[15]给出了另外一种利用跳转表的方法,即利用跳转表中的内存地址之差得到 case 语句块的大小,人工分析出某个 case 语句块对应的内存地址,然后联合该 case 语句块在文件中的偏移量求出装载基址. Stefan[16]给出一个人工判定 MIPS 固件装载基址的方法,该方法假定有一定量的立即地址指向固件本身,通过重新设置固件的装载基址人工判定字符串或者函数序言是否正确匹配来测试当前设置的基址是否正确.

以上判定固件装载基址的方法均需要人工参与,且高度依赖固件分析人员的经验和直觉,目前尚没有方法可以自动化判定出固件的装载基址. 据统计当前大约有 63% 的嵌入式设备为 ARM 体系结构[17],因此本文以 ARM 体系结构下的嵌入式系统的固件文件为研究对象,研究了二进制文件中字符串的存储规律及其地址加载方式,针对使用 LDR 指令加载字符串地址的二进制文件,提出了一种判定装载基址的方法. 该方法共分为三个步骤,第一步,识别出二进制文件中所有字符串并输出其偏移量;第二步,识别出二进制文件中所有可能使用 LDR 指令加载的字符串地址;第三步,利用找到的字符串偏移量和 LDR 加载的字符串地址,判定出装载基址. 本文的贡献有以下两点:

(1)首次提出一种识别二进制文件中 LDR 指令的算法,该算法利用 LDR 指令的编码特点,可以有效识别出二进制文件中 LDR 指令并计算出其加载的地址.

(2)首次提出一种判定 ARM 固件中二进制文件装载基址的算法,该算法利用二进制文件中的字符串偏移量以及二进制文件中的 LDR 指令加载的字符串地址,判定出装载基址. 实验证明,本文提出的判定装载基址方法针对使用 LDR 指令加载字符串地址的二进制文件是有效的.

2 固件中的字符串及其地址

2.1 固件中的字符串及其识别方法

二进制文件一般都包含一些字符串(strings),主要为程序中的用户提示信息、程序出错信息,编译器版本等. 这些字符串的组成部分为可打印字符或者转义字符. 可打印字符包括字母、数字、标点符号等,其 ASCII 范围为(0x20,0x7E). 转义字符为换行符、制表符等,其 ASCII 范围为(0x09,0x0D). 所以字符串的 ASCII 范围为(0x09,0x0D)∪(0x20,0x7E). ARM 体系结构下嵌入式系统中一般使用 C 语言编程,在本文中只讨论 C 风格的字符串. C 语言中,字符串一般存储在字符数组中,字符数组最后一个元素存放字符串结束符'\0',对应的 ASCII 码为 0x00.

编译器在存储字符串时,一般把相邻代码中用到的字符串集中存储,如图 1 所示. 某些编译器出于性能考虑,在存储字符串时进行了对齐操作,即如果某个字符串的开始地址不是 4 的整数倍,则在字符串前补充一些填充字节(padding)以使得字符串的开始地址为 4 的整数倍,如图 1(*b*)所示.

从图 1 中可以看出,无论是非字对齐存储的字符串还是字对齐存储的字符串,均有相同的规律,即除了第一个字符串以外,其他的字符串前后都是 00. 根据该特点,我们提出了 FIND-String 算法来找出固件文件中所有的字符串偏移量. 因二进制文件中有部分指令的编码等同于可打印字符的 ASCII 码,为了提高字符串识别的准确率,在 FIND-String 算法中需设置字符串的最小长度参数 wnd,只有长度大于 wnd 的字符串才认为是有效字符串.

FIND-String 算法主要思想为扫描固件文件,找出距离大于 wnd 且之间均为可打印字符或者转义字符的相邻字符串结束符位置,这些位置之间存储内容即为字符串,并可得到字符串偏移量. 对于固件中集中存储的字符串,由于不容易判断第一个字符串的开始位置,故舍弃. 如算法 1 所示。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000395C0	20	25	69	2C	20	72	75	6E	20	3D	20	25	70	20	7D	00	%i, run = %p }
000395D0	4E	6F	77	3A	20	25	6C	75	00	4B	65	72	6E	65	6C	20	Now: %lu Kernel
000395E0	41	6E	69	6D	61	74	69	6F	6E	73	3A	00	41	70	70	20	Animations: App
000395F0	41	6E	69	6D	61	74	69	6F	6E	73	3A	00	6D	96	04	08	Animations: m

(*a*) Pebble智能手表固件tintin_fw.bin

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0003CA00	61	67	65	5F	63	68	65	63	6B	00	00	00	73	65	63	5F	age_check sec_
0003CA10	62	61	74	5F	67	65	74	5F	61	64	63	5F	76	61	6C	75	bat_get_adc_valu
0003CA20	65	00	00	00	25	73	3A	20	45	72	72	6F	72	20	69	6E	e %s: Error in
0003CA30	20	41	44	43	0A	00	00	00	73	65	63	5F	62	61	74	5F	ADC sec_bat_
0003CA40	67	65	74	5F	74	65	6D	70	65	72	61	74	75	72	65	5F	get_temperature_
0003CA50	62	79	5F	61	64	63	00	00	25	73	3A	20	49	6E	76	61	by_adc %s: Inva

(*b*) Samsung智能手表固件wingtip_in.bin

字符串结束符 '\0'　　填充字节

图1　两种字符串存储类型实例

算法 1　FIND-String 算法

```
输入:二进制文件 binaryFile,字符串最小长度 wnd
输出:二进制文件中长度大于 wnd 的字符串在文件中的偏移量 offset
function FindString(binaryFile,wnd)
  bin[fileSize] ←binaryFile
  pos ← 1
  while(0 <pos <fileSize) do
    if(bin[pos-1] = = 0x00 && isprint(pos) = = TRUE)
      offset ←pos
      length ← 1
      while(isprint(pos + length) = =TRUE && bin[pos + length + 1]
      ! = 0x00) do
        length + +
      end while
      if(length >wnd && isprint(bin[pos +length]) = = TRUE)
        Output:offset
      end if
      pos ←pos +length
    else
      pos + +
    end if
  end while
end function
function isprint(c)
  if((c > = 0x09 &&c < = 0x0D) || (c > =0x20 && c < =
  0x7E))
    return TRUE
  else
    return FALSE
end
function
```

利用算法 1 可得到图 1(*a*)中的字符串偏移量:

Offset = 000395D0

Offset = 000395D9

Offset = 000395EC

这些字符串偏移量信息将在判定装载基址的算法中用到,下面介绍如何求解 LDR 指令中加载的字符串地址.

2.2　字符串地址的识别

固件中的字符串一般为错误提示信息、菜单信息、或者用户提示信息等. 实验发现,ARM 固件一般通过 LDR 指令将字符串的地址加载到寄存器中,下面举例说明字符串地址的加载过程. 图 2 为 Pebble 智能手表固件文件 tintin_fw.bin 的反汇编结果,如图所示,字符串 aNowLu 的地址是 0x080495D0,由于超出了 MOV 指令中立即数的范围而不能直接使用 MOV 指令把字符串地址保存到寄存器中,所以代码段 0x08016B58 处定义了一个地址值 0x080495D0,并使用 LDR 指令加载这个地址值到寄存器中. 同理,另一个字符串 aKernelAnimatio 的地址也是用 LDR 指令加载的. 字符串 aNowLu 和 aKernelAnimatio 实际存储见图 1 (*a*)所示.

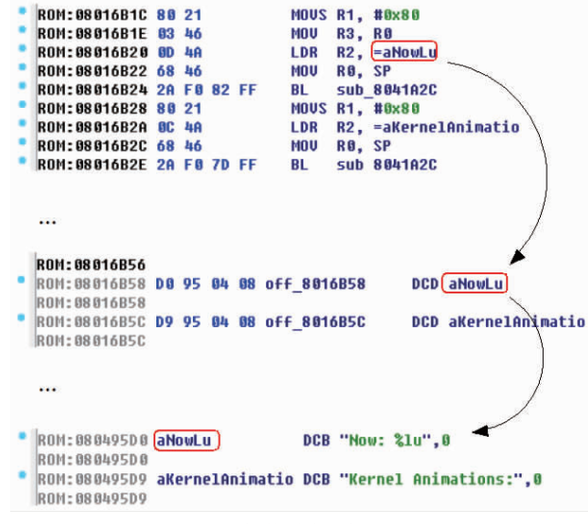


图2　Pebble智能手表固件tintin_fw.bin

下面分析 LDR 加载字符串的详细步骤,以字符串 aNowLu 为例,LDR 指令加载字符串 aNowLu 的指令内存地址为 0x08016B20,机器码为 0D 4A. 因为这个固件是小端存储,所以机器码实际为 4A 0D. LDR 指令有多

种语法格式,其中 Thumb 状态下加载立即数到寄存器的语法格式为[18]:LDR〈Rd〉,[PC,#〈immed_8〉*4],对应的指令编码格式如图 3 (*a*)所示.

通过分析 LDR 指令的编码格式及其机器码可知,Rd = $(010)_2$ = 2, immed_8 = $(0000\ 1101)_2$ = 0x0D. Thumb 状态的 LDR 指令寻址地址为(PC&0xFFFFFFFC) + (immed_8 * 4),因为 ARM 处理器采用 3 级流水线技术,所以在 Thumb 状态下,PC 的取值为 PC = Current + 4,则 LDR 的寻址地址为:

$$\begin{aligned}\text{address} &= (\text{PC}\&0\text{xFFFFFFFC}) + (\text{immed\_8} * 4)\\ &= ((\text{Current} + 4)\&0\text{xFFFFFFFC}) + (\text{immed\_8} * 4)\\ &= ((0\text{x}08016\text{B}20 + 4)\&0\text{xFFFFFFFC}) + (0\text{x}0\text{D} * 4)\\ &= (0\text{x}0806\text{B}24\&0\text{xFFFFFFFC}) + 0\text{x}34\\ &= 0\text{x}08016\text{B}24 + 0\text{x}34\\ &= 0\text{x}08016\text{B}58 \end{aligned} \quad (1)$$

从而得到 LDR 的寻址地址为 0x08016B58. 如图 2 所示,在内存地址 0x08016B58 处开始的 4 个字节为 D0 95 04 08,因固件是小端存储,则为 0x080495D0,该地址即 LDR 指令要加载的地址. 如图 2 所示,0*x*080495*D*0 地址处存储内容为字符串实际内容.

ARM 状态的 LDR 语法及指令加载方法同 Thumb 状态类似,此处不再举例介绍,其指令编码如图 3(b)所示. 从中可知,ARM 状态下 LDR 指令的编码起始字节为$(1110\ 0101\ 1001\ 1111)_2$ = 0xE5 9F,以小端格式存储在固件中即为 0x9F E5.

根据上述分析和 LDR 指令的编码特点,本文提出算法 FIND-LDR 来识别二进制文件中 ARM/Thumb 状态下的 LDR 指令并计算其加载地址. 根据 ARM/Thumb 状态下 LDR 语法及加载方法的不同,该算法的第二个输入为 ARM 状态或者 Thumb 状态. 具体如算法 2 所示.

需要指出的是,使用 FIND-LDR 算法搜索到的部分 LDR 指令可能是错误的,这是因为固件中部分内容恰好匹配 LDR 指令编码格式,而这部分内容不是代码而是数据. 并且 Thumb 状态下 LDR 指令更短,所以 Thumb 状态下搜索到的错误 LDR 指令较 ARM 状态多一些. 但是这一小部分的错误 LDR 指令并不影响最后关于基址的判定.

算法 2 FIND-LDR 算法

```
输入:二进制文件 binaryFile、ARM/Thumb 状态 state
输出:二进制文件中 ARM/Thumb 状态下 LDR 指令加载的地址
function FindLDR(binaryFile,state)
  bin[fileSize] ←binaryFile
  offset ← 0
  if (state == STATE_THUMB)
    while(0 ≤offset <fileSize) do
      opcode ←bin[offset + 1]
      opcode ←opcode & (11111000)2
      if(opcode == (01001000)2)
        PC ←offset + 4
        immed_8 ← bit[7,…,0]
        address ← (PC & 0xFFFFFFFC) + (immed_8 * 4)
        Rd ← Memory[address,4]
        Output:Rd
      end if
      offset ←offset + 2
    end while
  else
    if (state == STATE_ARM)
      while(0 ≤offset <fileSize) do
        if (bin[offset + 2] == 0x9F &&bin[offset + 3] == 0xE5)
          PC ←offset + 8
          immed_12 ← bit[11,…,0]
          address ← (PC&0xFFFFFFFC) + (immed_12)
          Rd ← Memory[address,4]
          Output:Rd
        end if
        offset ←offset + 4
      end while
    end if
  end if
end function
```

(*a*) LDR指令Thumb编码格式

(*b*) LDR指令ARM编码格式

图3 LDR 指令编码格式

3　装载基址的判定

使用2.1小节介绍的FIND-String算法可以获得二进制文件中字符串偏移量的集合,使用2.2小节介绍的FIND-LDR算法可获得二进制文件中LDR指令加载的内存地址集合. 二进制文件中大部分字符串的地址将被加载到寄存器中(否则无法对字符串进行操作),如果字符串的地址使用LDR指令加载,则字符串的偏移量集合中的部分元素和LDR指令加载的内存地址集合中部分元素存在对应关系,可利用两个集合元素之间的对应关系判定出装载基址.

FIND-String算法得到的字符串集合$(s_1,s_2,\cdots,s_n)$记为S,对应的偏移量集合$(o_1,o_2,\cdots,o_n)$$(o_i<o_{i+1}(1\leqslant i\leqslant n-1))$记为$O$,其中$n$为固件中字符串的个数. 利用算法FIND-LDR得到的LDR指令加载的内存地址值去除重复元素后的集合为$(a_1,a_2,\cdots,a_m)$,记为A,其中m为LDR指令加载的地址值个数.

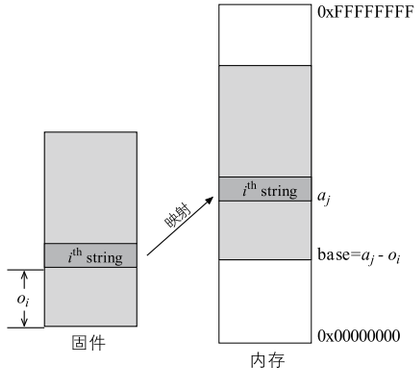


图4　固件加载示意图

如图4所示,如果某一个字符串在文件中的偏移量为o_i,加载到内存中位置为a_j,假设装载基址为base,则:

$$o_i+\text{base}=a_j \tag{2}$$

由此可知固件的基址为base = a_j-o_i. 现在已知集合S中部分字符串的地址将被LDR指令加载,集合A中部分地址为字符串地址,所以集合O和集合A中的部分元素满足式(2). 当base为某个内存地址时,集合O和集合A中满足式(2)的元素个数最多,则认为该内存地址即固件的装载基址. 然而,在32位系统中内存范围较大$(0\sim2^{32})$,如果采用枚举算法则需将该内存范围内每一个值作为基址base,计算集合O和集合A中满足式(2)的元素个数,当base为某个内存地址时,满足式(2)的元素个数最多,则认为该内存地址即为固件的装载基址. 但枚举方法的效率较低,其时间复杂度为$O(2^{32}*n*m)$. 为此,我们设计了一个算法可快速计算出这个内存地址,算法主要思想为:首先确定二进制文件装载基址的范围,其最小值为0,在32位嵌入式系统中装载基址的最大值为0xFFFFFFFF-fileSize,其中fileSize为二进制文件的大小. 用集合A中每个元素a_j依次减去集合O中每个元素o_i,如果差值在装载基址的范围内则保存,否则舍弃. 然后统计每个差值出现的次数,按出现次数进行降序排序并将结果输出.

假设base为某个内存地址时集合O和集合A中有k对元素符合$o_i+\text{base}=a_j$. 当用集合A中第j个元素减去集合O中每个元素时,得到一个集合:

$$(a_j-o_1,a_j-o_2,\cdots,a_j-o_n) \tag{3}$$

这样集合A中每个元素a_j减去集合O中每个元素o_i得到一个矩阵$\boldsymbol{M}$:

$$\begin{bmatrix} a_1-o_1 & a_1-o_2 & \cdots & a_1-o_n \\ a_2-o_1 & a_2-o_2 & \cdots & a_2-o_n \\ \vdots & \vdots & & \vdots \\ a_j-o_1 & a_j-o_2 & \cdots & a_j-o_n \\ \vdots & \vdots & & \vdots \\ a_m-o_1 & a_m-o_2 & \cdots & a_m-o_n \end{bmatrix} \tag{4}$$

我们已经假定有k对元素满足$o_i+\text{base}=a_j$,但这k对元素的具体下标是未知的. 假设(a_x,o_y)为k对元素中任意一对,此时在矩阵$\boldsymbol{M}$的第x行y列的元素$M_{[x,y]}$即存储着a_x-o_y的值. 然后我们对矩阵$\boldsymbol{M}$统计出每个元素的出现次数,按照元素的出现次数进行降序排序,并将结果输出,出现次数最多的元素即为候选基址. 候选基址的实际意义为在这个内存地址处,集合O与集合A中有最多的元素存在着对应关系,即集合S中有最多的字符串地址被加载到寄存器中. 根据上述分析,我们提出判定装载基址的DBMAS算法(Determining image Base by Matching Addresses of Strings),为便于统计,算法首先将矩阵$\boldsymbol{M}$所有的元素初始化为-1.

算法3　DBMAS算法

```
输入:固件中字符串偏移量集合 O = (o1,o2,…,on)
固件中 LDR 指令加载的地址集合 A = (a1,a2,…,am)
二进制文件的大小 fileSize
输出:矩阵 M 中元素及其出现次数排序后的结果
function DBMAS(O,A,fileSize)
  将矩阵 M 中每个元素初始化为 -1
  max ← 0xFFFFFFFF-fileSize
  for all aj ∈ A do
    for all oi ∈ O do
      if 0 < aj - oi && aj - oi < max
        M[j,i] ← aj - oi
      end if
    end for
  end for
  统计矩阵 M 中每个元素出现的次数
```

对元素及其出现次数组成的键值对按出现次数降序排序
删除排序结果中矩阵 $\boldsymbol{M}$ 的初始值 -1 及其出现次数
Output:排序后的元素及其出现次数
end function

DBMAS 算法的时间复杂度为 $O(n*m)$,其中为集合 O 中字符串偏移量的个数,m 为集合 A 中字符串地址的个数. 可见 DBMAS 算法的时间复杂度远小于枚举算法的时间复杂度.

算法 DBMAS 对矩阵 $\boldsymbol{M}$ 中每个元素统计次数,删除排序结果中矩阵 $\boldsymbol{M}$ 的初始值 -1 及其出现次数后,再输出按出现次数降序排序的结果,该结果中第一个内存地址(即出现次数最多的地址)认为是候选装载基址. 一般情况下,如果候选基址出现的次数远远大于其他元素的出现次数,该候选基址即为正确的装载基址. 否则,该候选基址不是正确的装载基址,DBMAS 算法不适用于此二进制文件.

4 实验结果与分析

针对本文所涉及的判定二进制文件装载基址的评估,目前尚没有一套公开的测试集. 为此,本文收集了互联网上的多个嵌入式设备固件作为测试集,其中包括智能手表、智能手机、MP3 播放器、固态盘、PLC 等. 实验采用计算机硬件环境为 Pentium Dual-Core 3.0GHz 处理器,4GB 内存,软件环境为 Microsoft Windows7 SP1.

4.1 字符串和 LDR 指令中地址识别实验结果

在实验中,我们选取了测试集中 12 个二进制文件作为实验对象,然后分别使用 FIND-String 算法识别文件中字符串,使用 FIND-LDR 算法识别文件中 LDR 指令加载的地址. 实验结果如表 1 所示,Strings 列为文件中字符串的个数;ARM_LDR 列、Thumb_LDR 列分别为使用 FIND-LDR 算法识别出的 ARM、Thumb 状态下 LDR 指令加载地址去重后的个数,ALL_LDR 列为使用 FIND-LDR 算法识别出的所有状态 LDR 指令加载地址去重后的个数. 我们经过大量实验统计发现,文件中绝大部分可打印字符串长度都超过 5,所以我们在 FIND-String 算法试验中参数 wnd 取值为 5.

表 1 实验结果

Model	File	Strings	ARM_LDR	Thumb_LDR	ALL_LDR	Match	Base
Pebble Smart Watch	tintin_fw.bin	1580	0	3152	3152	957	0x08010000
Garmin Forerunner 225	forerunner225_240.rgn	2355	1	3092	3093	336	0x00000F88
Garmin Fenix3	fenix3_440.rgn	3195	2	9516	9518	667	0x00000F88
Samsung Gear Fit	wingtip_in.bin	6956	0	4334	4334	461	0x08004000
Sony SBH52	sbh52_firmware.bin	348	4	1564	1568	N/A	N/A
iAudio E2 MP3	e2_eu_fw.sb	2394	297	15975	16224	69	0x00059000
iAudio 10 MP3	iaudio10_fw.bin	3878	9754	3981	13492	919	0x20000000
HTC One	pn07diag.nbh	6196	1649	802	2420	780	0x0ADFFCD8
Samsung 830 SSD	cxm03b1q.enc	1	0	704	704	N/A	N/A
SanDisk Ultra X2316RL	firmware.bin	3668	2183	833	2844	523	0x7FFFF400
Western Digital My Book	20080111.bin	33	17	1702	1714	N/A	N/A
Rockwell 1769-L3x	pn-275814.bin	4978	3023	1734	4627	124	0x00D00000

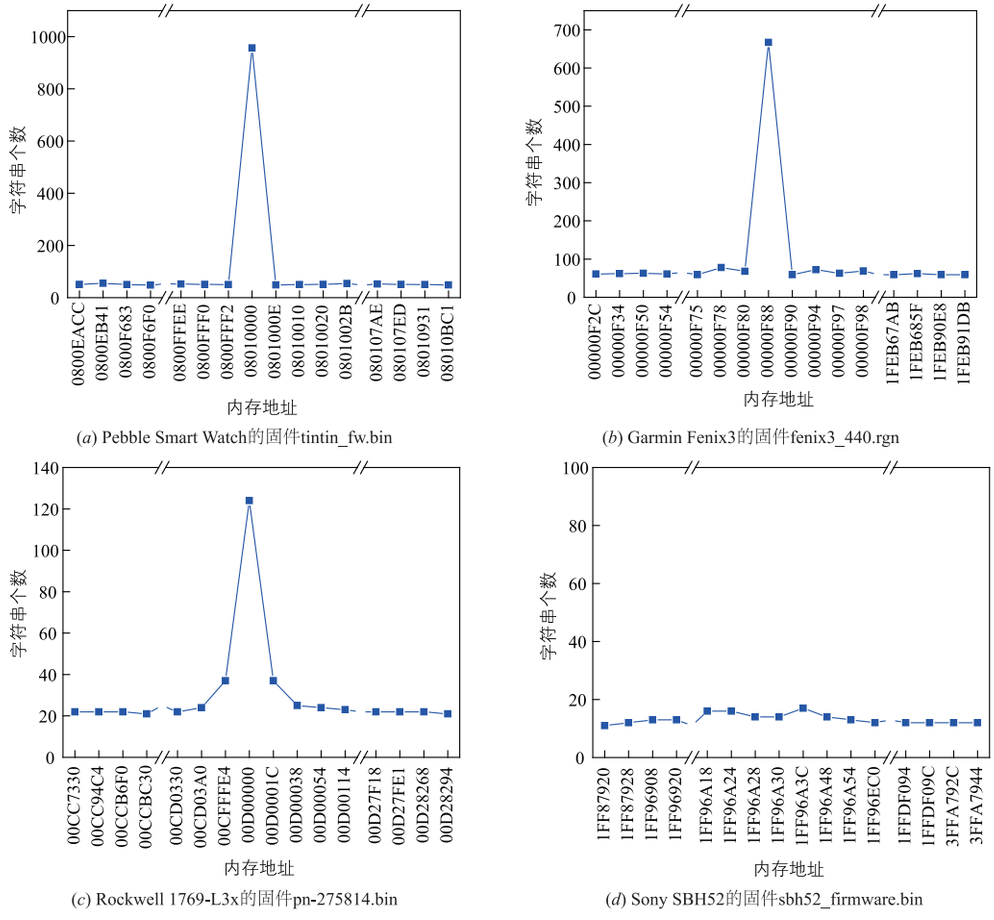
4.2 判定装载基址的实验结果

判定装载基址实验结果如表 1 所示,Base 列为使用 DBMAS 算法判定出的二进制文件装载基址,Match 列为 LDR 指令加载的字符串地址数目,即矩阵 $\boldsymbol{M}$ 中装载基址出现的次数,符号 N/A 代表本文提出的方法不适用这个固件,实验失败的原因将在 4.3 小节介绍.

我们选择 Pebble 智能手表的固件 tintin_fw.bin 文件作为案例分析,根据 4.1 小节实验结果,FIND-String 算法在 tintin_fw.bin 文件中识别出 1580 个字符串,FIND-LDR 算法识别出 0 个 ARM 状态 LDR 指令加载地址以及 3152 个 Thumb 状态 LDR 指令加载地址,合计去重后为 3152 个 LDR 指令加载地址,使用 DBMAS 算法判定基址的结果如图 5(a)所示. 从图中可以看出最大值点为 X = 0x08010000, Y = 957, 即内存地址 0x08010000 在矩阵 $\boldsymbol{M}$ 中出现了 957 次,远远大于其他的内存地址出现的次数. 其实际意义为在内存地址 0x08010000 处,有 957 对数据满足 o_i + base = a_j,DBMAS 算法输出内存地址 0x08010000 为候选装载基址.

得到装载基址 0x08010000 后可人工验证该基址是否正确,具体做法为使用 IDA Pro 加载 tintin_fw.bin 文件,设置处理器类型为 ARM Little-endian,设置装载基址为 0x08010000,可以看出反汇编代码中关于绝对内存地址的交叉引用是正确的,LDR 指令加载字符串地址时可显示出正确的字符串名称,这说明内存地址 0x08010000 为正确的装载基址,同时验证了文件中 1580 个字符串有 957 个字符串的地址通过 LDR 指令加载到寄存器中. 图 5(b)(c)分别为 Garmin Fenix3 智能手表的固件文件 fenix3_440.rgn,Rockwell 1769-L3x 的

固件文件 pn-275814.bin 的基址判定结果,其基址分别为 0x00000F88、0x00D00000,经人工验证均为正确的装载基址. 图 5(*d*)为 Sony SBH52 蓝牙耳机固件文件 sbh52_firmware.bin 的基址判定结果,从图看出,矩阵 ***M*** 中并没有某一个内存地址比较突出,经过人工验证算法输出结果并不是正确的装载基址,说明本文提出的算法不适用这个文件.



(*a*) Pebble Smart Watch的固件tintin_fw.bin　　(*b*) Garmin Fenix3的固件fenix3_440.rgn

(*c*) Rockwell 1769-L3x的固件pn-275814.bin　　(*d*) Sony SBH52的固件sbh52_firmware.bin

图5　判定装载基址实验结果

4.3　基址判定失败的原因分析

从表 1 中可以看出,部分型号的 ARM 固件中二进制文件并没有判定出装载基址,使用本文提出的方法判定装载基址失败的原因有以下可能方面:

(1)部分厂商的固件文件是加密的或者压缩的,此类文件必须解密、解压缩后,才可使用本文提出的算法进行判定,如 Samsung 830 SSD 的 cxm03b1q.enc 文件即为一个加密的固件文件.

(2)部分二进制文件中使用 ADR 指令加载字符串地址到寄存器,ADR 指令是位置无关代码(position independent code,PIC),是基于 PC 的相对寻址方式[18],并不需要字符串的内存地址,因此使用 FIND-LDR 算法搜索到的 LDR 指令加载的地址中并不包含字符串地址. 本文提出的算法需要利用 LDR 指令中加载的字符串地址,故对此类二进制文件无效,如 Sony SBH52 蓝牙耳机固件 sbh52_firmware.bin 文件及 Western Digital My Book 的固件 20080111.bin 文件.

5　结论以及进一步工作

针对嵌入式系统固件的反汇编是对嵌入式系统安全性分析的必要步骤,大部分格式未知的嵌入式系统固件无法直接获得装载基址,使得反汇编工作不能顺利展开. 本文研究 ARM 固件中字符串的存储规律及字符串地址常见加载方式,提出了一种利用固件中字符串偏移量和 LDR 指令加载的字符串地址来判定固件装载基址的方法,实验结果表明本文提出的方法对使用 LDR 指令加载字符串地址的固件是有效的. 下一步的研究方向是探索 ARM 指令的编码规律,提出针对其他类型的 ARM 固件的基址判定方法.

参考文献

[1] Kaspersky Lab. Equation:The Death Star of Malware Galaxy [EB/OL]. https://securelist.com/blog/research/68750/equation-the-death-star-of-malware-galaxy/,2015.

[2] Kaspersky Lab. A Fanny Equation:"I am your father,Stuxnet" [EB/OL]. https://securelist.com/blog/research/68787/a-fanny-equation-i-am-your-father-stuxnet/,2015.

[3] Langner R. Stuxnet:Dissecting a cyberwarfare weapon[J]. IEEE Security & Privacy,2011,9(3):49-51.

[4] Falliere N,Murchu L O,Chien E. W32. Stuxnet Dossier [R]. USA:Symantec Corp,2011.9.

[5] Trend Micro. Netis Routers Leave Wide Open Backdoor[EB/OL]. http://blog.trendmicro.com/trendlabs-security-intelligence/netis-routers-leave-wide-open-backdoor/,2015.

[6] Craig Heffner. Reverse Engineering a D-Link Backdoor [EB/OL]. http://www.devttys0.com/2013/10/reverse-engineering-a-d-link-backdoor/,2015.

[7] Zach Cutlip. Netgear Root Compromise via Command Injection [EB/OL]. http://shadow-file.blogspot.lt/2013/10/netgear-root-compromise-via-command.html,2015.

[8] Craig Heffner. From China,With Love [EB/OL]. http://www.devttys0.com/2013/10/from-china-with-love/,2015.

[9] Hex-Rays SA. The IDA Pro Disassembler and Debugger [EB/OL]. https://www.hex-rays.com/products/ida/index.shtml,2016.

[10] Schuett C,Butts J,Dunlap S. An evaluation of modification attacks on programmable logic controllers[J]. International Journal of Critical Infrastructure Protection,2014,7(1):61-68.

[11] Basnight Z,Butts J,Lopez Jr J,et al. Firmware modification attacks on programmable logic controllers[J]. International Journal of Critical Infrastructure Protection,2013,6(2):76-84.

[12] Skochinsky I. Intro to embedded reverse engineering for PC reversers[R]. Montreal,Canada:RECON,2010.

[13] Basnight Z H. Firmware Counterfeiting and Modification Attacks on Programmable Logic Controllers[D]. Ohio:Air Force Institute of Technology,2013.

[14] Dacosta I,Mehta N,Metrock E,et al. Security analysis of an IP phone:Cisco 7960G[A]. Principles,Systems and Applications of IP Telecommunications [C]. Berlin:Springer-Verlag,2008.236-255.

[15] Santamarta R. Reverse Mode-Reversing Industrial Firmware for Fun and Backdoors I[EB/OL]. http://www.reversemode.com/index.php? option=com_content&task=view&id=80&Itemid=1,2015.

[16] Viehbock S. Reverse engineering an obfuscated firmware image E02 - analysis[EB/OL]. https://sviehb.wordpress.com/2011/09/09/reverse-engineering-an-obfuscated-firmware-image-e02-analysis/,2015.

[17] Costin,A,et al. A large-scale analysis of the security of embedded firmwares[A]. Proceedings of the 23rd USENIX Conference on Security Symposium[C]. San Diego,California:USENIX Association,2014.95-110.

[18] ARM Limited. ARM Architecture Reference Manual[M]. Cambridge,England:ARM Limited,2014.

作者简介

朱瑞瑾 男,1985 年生,山东菏泽人. 中国信息安全测评中心助理研究员,研究方向为软件安全、嵌入式系统安全、固件安全、逆向工程. E-mail:ruijinzhu@gmail.com

张宝峰 男,1983 年生,山东日照人. 中国信息安全测评中心副研究员,研究方向为网络安全、嵌入式设备安全.

毛军捷 男,1984 年生,北京人,中国信息安全测评中心副研究员,研究方向为可信计算、固件安全.

骆 扬 男,1984 年生,河北石家庄人,中国信息安全测评中心做博士后,研究方向为硬件木马、硬件蠕虫、智能芯片安全性.

谭毓安 男,1972 年生,重庆巫溪人,北京理工大学计算机学院教授,北京理工大学计算机实验教学中心主任,中国计算机学会高性能计算专委会委员,信息存储技术专委会委员. 研究方向为信息安全、网络存储、嵌入式系统.

张全新(通信作者) 男,1974 年生,山东德州人,北京理工大学计算机学院讲师,研究方向为存储算法、移动计算. E-mail:zhangqx@bit.edu.cn