基于中断向量表重构的固件代码反汇编技术

崔 晨 李清宝 胡 刚 王 炜
(解放军信息工程大学信息工程学院 郑州 450002)

摘 要 反汇编是固件代码逆向分析的重要研究内容,其正确性直接影响固件代码逆向分析的准确性。固件代码结构具有特殊性,针对上层应用程序的反汇编算法大都不能直接用于固件代码的反汇编。中断向量表是固件代码的重要组成部分,从中断向量开始对中断服务子程序进行反汇编,可提高固件代码反汇编的精度。通过对固件代码结构特点的研究分析,介绍了中断向量表的重构方法,提出了一种基于中断向量表重构的固件代码反汇编技术。经测试分析,与传统的静态反汇编技术相比,基于中断向量表重构的固件代码反汇编技术不仅能够对固件代码中的主函数进行反汇编,还能够对中断服务子程序进行反汇编,反汇编精度平均提高了8.72%。

关键词 逆向分析,固件代码,反汇编,中断向量表
中图法分类号 TP309 **文献标识码** A

Firm-code Disassembly Technology Based on IVT Reconstruction

CUI Chen LI Qing-bao HU Gang WANG Wei
(Institute of Network Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract Disassembly is an important part of firmware reverse engineering analysis, whose correctness directly influences the precision of FREA. At present, most of the disassembly methods focus on practical program. However, these methods could not be directly used in firm-code disassembly due to its particularity. IVT (Interrupt Vector Table) is the core of firm-code. Effective interrupt vectors are available by reconstructing the IVT. The more interrupt vectors we obtain, the more precise the disassembly result is. The structural characteristics of firm-code were studied, and the IVT reconstruction method was introduced. Moreover, a disassembly technology based on the reconstruction of IVT was proposed. The experimental results show that the proposed technology can effectively improve the precision of firm-code disassembly, by which both of main function and interrupt subprograms could be disassembled, compared with traditional static disassembly methods. The disassembly precision is increased by 8.72% in average.

Keywords Reverse analysis, Firm-code, Disassembly, IVT

1 引言

随着计算机应用需求的发展,固件逆向工程已成为一个重要的研究领域,反汇编是固件代码逆向分析的重要研究内容,是固件程序进行代码还原的重要手段之一。固件代码的还原有利于整个嵌入式系统的分析,在安全检测、遗产系统的维护等方面有着重大的现实意义。

反汇编是将目标固件程序转换成可读取汇编代码的过程[1]。按是否动态运行目标程序,可以将反汇编技术分为静态反汇编和动态反汇编。动态反汇编需要一个可控的动态执行环境来执行目标程序。由于固件程序的硬件配置随着应用的不同而表现出很大的差异,构建这样一个动态执行环境相对困难,因此固件代码的反汇编通常采用静态反汇编技术。静态反汇编可分为线性策略[2]、递归策略[3-5]和启发式策略[6]3种。线性策略不能自动对数据与指令进行区分,不适用于冯诺依曼结构下的固件代码。递归策略利用目标程序中的控制流信息区分程序中的数据和指令,能够得到较为准确的结果,但对间接跳转目标地址计算问题一直难以彻底解决,也无法保证控制流之外的部分一定都是数据。启发式策略在一定程度上弥补了递归策略的这一缺陷,它针对疑似数据部分进行异常节点分析,将异常节点视为数据。

目前关于上层应用程序的反汇编研究较为成熟,但固件代码特殊性使得现有的反汇编技术不能直接应用于固件程序。固件程序包含中断向量表(Interrupt Vector Table,IVT)和中断服务子程序两部分,IVT中包含了中断服务子程序的入口地址,可以根据每个入口地址对所有的中断服务子程序进行反汇编。因此,得到的IVT越完全,能够提供的入口地址就越多,获取的有效中断服务子程序越多,代码反汇编的覆盖面就越广,从而可以有效提高固件代码反汇编结果与原始汇编代码的接近程度,即反汇编的精度。IVT中的有效中断

到稿日期:2011-08-31 返修日期:2011-12-10 本文受国家863项目核心芯片安全缺陷发现及其逆向分析模拟仿真系统(2009AA01Z434)资助。
崔 晨(1984-),女,硕士生,主要研究方向为固件代码缺陷分析,E-mail:ceciinsc@gmail.com;李清宝(1967-),男,博士,教授,主要研究方向为核心芯片漏洞分析、计算机体系结构;胡 刚(1984-),男,硕士生,主要研究方向为固件代码缺陷分析;王 炜(1975-),男,博士,讲师,主要研究方向为计算机系统结构。

向量是未知的，因此正确的重构 IVT[7] 是固件反汇编的关键。

本文分析了固件代码的结构特点以及 IVT 对固件反汇编正确性的影响，提出一种基于 IVT 重构的固件代码反汇编算法。最后，对基于 IVT 重构的反汇编算法进行了测试，并将其与传统的基于递归策略的反汇编算法进行了比较，测试结果表明本算法不仅能够对固件代码中的主函数进行反汇编，还反汇编了中断服务子程序，有效地提高了固件代码反汇编的精度。

2 基于 IVT 重构的固件代码反汇编

2.1 固件代码结构特点

固件逆向工程中的固件是指存储于具有永久存储功能的器件中的二进制程序，这些器件包括 ROM、PROM、FLASH 等永久存储器，以及具有永久存储功能的 FPGA/CPLD、SOC/SOPC 等。固件代码具有自身的特殊性，与传统应用程序相比，固件代码是仅包含数据和指令的纯二进制代码序列，直接控制硬件完成设备功能，其结构相对简单。一个固件通常由 IVT 和中断服务子程序组成。IVT 由中断向量组成，描述了固件代码的各中断服务子程序的入口地址。它在处理器存储空间的地址是固定的，长度由中断个数和中断向量长度决定。每个 IVT 中一定包含复位中断，它也是固件程序的主函数，开机或复位时执行。每个中断服务子程序包含一个或多个功能模块，功能模块完成与某一特定硬件交互的功能，同一个功能模块可以同时包含在不同的中断处理过程中[8]。固件代码的这种结构如图 1 所示。

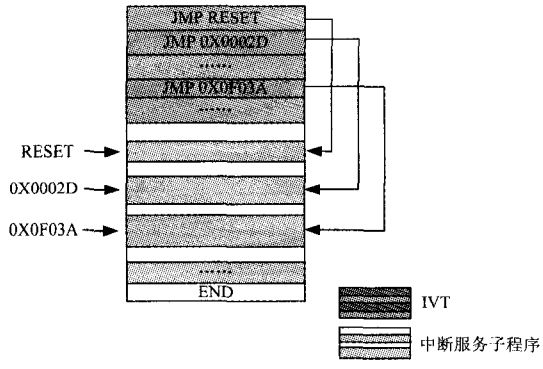

图 1 固件代码结构

2.2 IVT 在固件反汇编中的作用

固件代码本质上由两部分组成：IVT 和中断服务子程序。固件代码反汇编的目的是将 IVT 和中断服务子程序从二进制代码提升为汇编语言。由于 IVT 描述了各中断服务子程序的入口地址，因此，可以利用基于递归策略的反汇编算法从中断服务子程序的入口地址处对其进行反汇编，从而得到中断服务子程序的汇编代码。固件代码反汇编的流程分为两步：获取 IVT 和基于递归策略的反汇编。后者相对容易，因为基于递归策略的反汇编算法较为成熟，能够有效对固件代码中的中断服务子程序进行反汇编。但对于一个特定处理器类型的固件程序，往往无法确定 IVT 中的有效中断向量，也无法直接获得 IVT 的全部信息。IVT 中有效信息的缺失增加了固件代码反汇编的难度。

IVT 是固件程序的重要组成部分，由中断向量组成，记录了各中断服务子程序的起始地址。IVT 中包含的中断类型和中断向量的最大个数是由处理器类型决定的，如 8051 有 5 个中断源，ARM 有 7 个，8086 中有 256 个。用户无法对其进行修改，但不同的固件程序中实现的中断服务子程序是不同的，在程序中实现了的中断服务子程序的中断向量称为有效中断向量。有效中断向量的类型和个数是由固件需要实现的功能决定的。IVT 的重构就是确定 IVT 中有效中断向量个数和地址的过程。因此，通过 IVT 重构来获取有效中断向量，获取的有效中断向量越多，固件代码反汇编的精度就越高。

2.3 IVT 重构

通过 IVT 重构来获取有效中断向量，是固件程序进行正确反汇编的前提之一。其重构过程如下：从中断向量表中获取主函数入口地址，遍历主函数控制流图。首先判断是否存在中断调用指令，若存在，则记录其中断向量号，并在 IVT 相应的位置作标记，获取中断号对应的中断向量，遍历该中断服务子程序的控制流图，若在期间发现中断调用指令，则做同样的处理。经过上述步骤后的 IVT 中已有一部分中断向量被标记，即，这些已标记的中断向量一定为有效中断向量。然后，采用启发式方法确定剩余的中断向量是否为有效中断向量。假设它们都是有效的，逐个验证。从 IVT 中获取其对应的起始地址，扫描中断服务子程序，在扫描过程中，若发现未知指令，或控制转移指令的目标地址超出固件程序的逻辑空间，则认为它是无效的，否则为有效，并且在 IVT 相应的位置上做标记。最终，IVT 中所有被标记的向量就为该目标固件程序的有效中断向量集合。

假设有效中断向量编号集合用 INT_AVAILBLE 表示，目标固件的所有中断向量编号集合为 INT_ALL，待定有效的中断向量编号集合为 INT_UNCERTAIN，INT_UNCERTAIN 中的某个中断编号为 i，算法流程如图 2 所示。

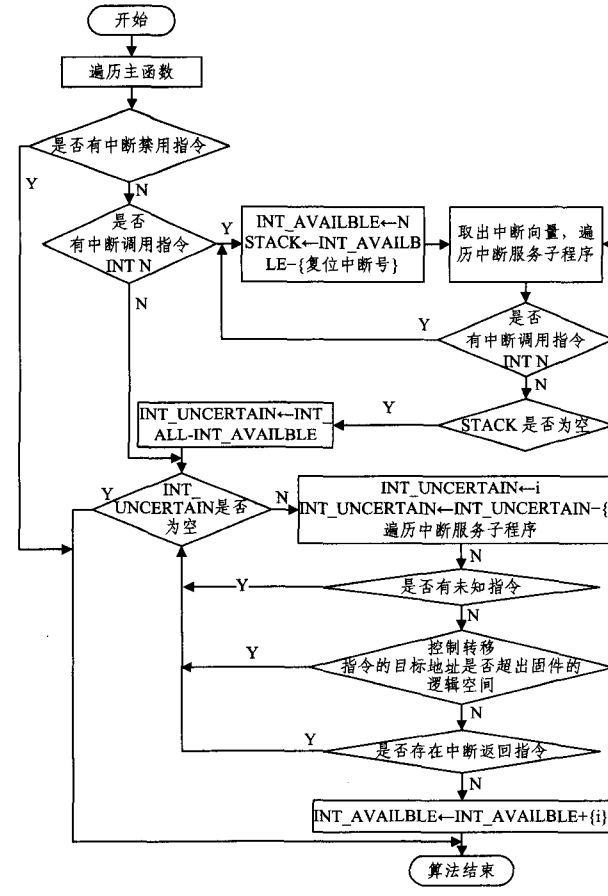

图 2 IVT 重构算法

该重构算法利用了处理器类型的相关信息，采用启发式的方法来确定目标固件程序中的有效中断向量，为基于 IVT 重构的固件代码反汇编算法奠定了基础。

2.4 基于 IVT 重构的固件代码反汇编算法

本节在 IVT 重构的基础上描述了固件代码反汇编的策略。该策略的主要思想是，首先递归扫描主函数，若扫描到中断调用指令 INT N，则在 IVT 中进行标记；然后根据 IVT 中提供的函数入口地址，利用递归策略的反汇编算法，得到该中断服务子程序的反汇编；最后对于 IVT 中未进行标注的中断向量，采用启发式方法进行扫描，得到对应函数的反汇编结果。算法描述如图 3 所示。

算法 FCRecursiveDisam

1. 调用 ReconstructInterruptTable ()，获取 IVT //获得中断服务子程序的入口。
2. 对于 S 中的每一个中断向量 s，调用 RecursiveDisam (s) //调用递归处理函数 RecursiveDisam。

函数：RecursiveDisam (s)

1. 如果地址 s 已经被访问过，则返回。
2. I=在地址 s 处反汇编所得到的指令。
3. 将 s 标记为已访问。
4. 判断 I 的类型。

4a. 如果 I 是一条分支指令，那么对于 I 的每一个可能目标 t 递归调用 RecursiveDisam(int s)；

4b. 否则 s:=s+指令 I 的长度。

图 3 基于 IVT 重构的固件代码反汇编算法

基于 IVT 重构的固件代码反汇编算法的优点有两个：首先，由于该算法是基于静态反汇编策略的，因此相对动态反汇编策略来说，反汇编结果更加完整，并且避免了构建固件代码动态执行环境的困难。其次，目前大多反汇编算法都是针对应用软件代码的，本算法从固件代码的特点出发，结合反汇编策略，能够得到有效的、较为完整的固件代码反汇编结果。

3 测试及分析

为了检验本文提出的算法在实际固件反汇编中的效果，分别使用传统的基于递归策略的反汇编算法和基于 IVT 重构的反汇编算法对属于 MCS-51、ARM、I386 的 8 个不同长度的目标代码进行了反汇编测试。

判断反汇编算法准确性的指标是反汇编结果与原始汇编代码的接近程度，以及反汇编过程需要的时间。用 t 表示完成一个固件代码到符号汇编程序转换的时间，单位是 s。用 p 表示反汇编后得到的汇编代码与原始汇编代码的接近程度，即反汇编精度。假设目标固件程序中数据部分的长度为 d，指令部分的长度为 i，反汇编后有效数据长度为 d'，有效指令长度为 i'，则反汇编算法的精度 p 为：$p=(\frac{d'+i'}{d+i})\times 100\%$。

假设基于 IVT 重构的反汇编算法的反汇编精度用 p_i 表示，反汇编后有效数据长度为 d_i'，有效指令长度为 i_i'；传统基于递归策略的反汇编算法的反汇编精度用 p_t 表示，反汇编后有效数据长度为 d_t'，有效指令长度为 i_t'，则：

$$p_i=(\frac{d_i'+i_i'}{d+i})\times 100\%$$

$$p_t=(\frac{d_t'+i_t'}{d+i})\times 100\%$$

假设 p_i 较 p_t 的精度提高率为 w，则：

$$w=(\frac{d_i'-d_t'}{d+i}+\frac{i_i'-i_t'}{d+i})\times 50\%$$

$$=(\frac{d_i'+i_i'}{d+i})\times 50\%-(\frac{d_t'+i_t'}{d+i})\times 50\%$$

$$=p_i-p_t$$

下面以 8 个固件程序为测试样本，对文中提出的基于 IVT 重构的反汇编算法进行了测试，并将该算法与基于递归策略的反汇编算法进行了比较，测试结果如表 1 所列。

表 1 反汇编测试结果

指令集类型	长度(kB)	反汇编时间(s)		反汇编精度(%)	
		$T_{Recursive}$	T_{IVT}	p_t	p_i
MCS-51	2.134	3	4	80.30	90.04
MCS-51	4.212	6	8	84.22	92.35
ARM	10.238	10	15	82.69	94.12
ARM	45.239	17	23	80.30	89.23
ARM	110.324	77	82	76.81	85.73
I386	3.090	5	9	89.92	91.48
I386	8.512	7	17	73.71	80.23
I386	10.242	10	22	70.98	85.49

注：p_i 代表基于 IVT 重构的反汇编算法；p_t 代表传统的基于递归策略的反汇编算法。

数据显示，基于 IVT 重构的反汇编算法与传统的基于递归策略的反汇编算法相比，前者对上述 8 个固件程序反汇编的平均精度达到 88.58%，后者反汇编平均精度为 79.86%。

根据实验数据计算，利用本文提出的基于 IVT 重构的反汇编算法对上述 8 个固件程序进行反汇编，反汇编精度平均提高了 8.72%。由这两种算法各自的性质可知，基于 IVT 重构的反汇编算法在反汇编时利用了 IVT 的信息，不仅对固件代码中的主函数进行反汇编，还反汇编了中断服务子程序，所以算法精度比传统的基于递归策略的反汇编算法的高。

基于 IVT 重构的反汇编算法在反汇编过程中进行了 IVT 重构以及利用中断向量中有效中断向量的地址进行递归解码，因此，所花费的时间较多，多余的时间开销是由 IVT 重构和中断服务子程序反汇编产生的。总之，反汇编的关键在于反汇编出来的程序与原程序之间的相近程度，即反汇编的精度。本文提出的基于 IVT 恢复的反汇编算法可以得到更加精确的反汇编结果。

结束语 代码反汇编是固件代码逆向工程的关键，反汇编的完整度是进行精确的二进制程序分析与理解的重要保证。由于固件代码的特殊性，使得现有的反汇编技术不能直接应用于固件程序。本文在分析固件代码结构的基础上，研究了 IVT 在固件代码反汇编中的重要作用，即获取的 IVT 越完全，能够获取的程序入口地址便越多，得到的有效中断服务子程序就越多，则反汇编的精度越高。在此思想的基础上，提出了一种基于 IVT 的固件代码反汇编技术。经过测试验证，该技术能够有效提高固件代码反汇编的完整度。下一步工作中，将继续研究一些相关问题，如若 IVT 中存在不符合编码规范或是存在类似于指令的数据，则启发式策略失效等。

参 考 文 献

[1] Eilam E. Reversing: Secrets of Reverse Engineering [M]. New Jersey: Wiley Publishing, 2005: 3-4

(下转第 316 页)

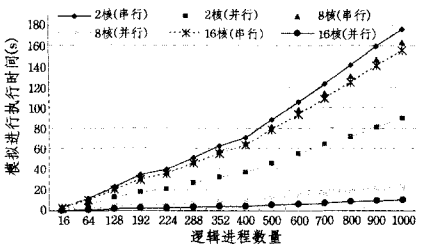

图 6 串/并行内核的执行时间

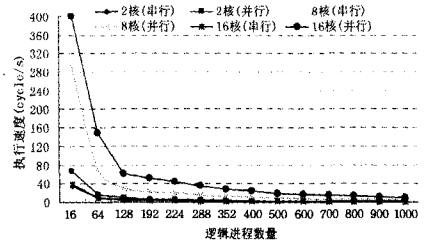

图 7 串/并行内核执行速度

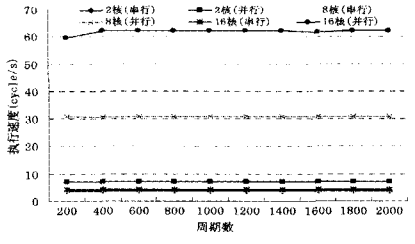

图 8 周期变化下执行时间

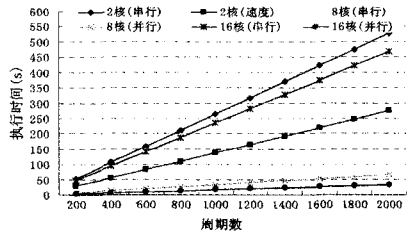

图 9 周期变化下的运行速度

结束语 并行模拟内核主要利用多核 CPU 并行计算能力,以及操作系统并行执行线程的机制改进调度策略来并行执行逻辑进程,其依然保留了串行内核的离散事件模拟(DES)特性,从而导致在主线程同步事件通知和更新事件的过程中其他线程被挂起而耗费模拟时间。为进一步提升仿真性能,并行离散事件模拟(PDES)算法代替目前的模拟方式,时钟模型也由单一的改为分布式的,每个逻辑进程都拥有相应的本地时钟,进程间通过带有时间信息的数据来同步时钟以及传递信号。这样,所有的调度线程就可以不停地扫描循环可执行的逻辑进程并运行,进一步减少由于事件同步和通知导致线程挂起的时间。

在事务级建模中,可以充分利用 TLM 所提供的发起者和目标者的概念,提供并封装相应的接口简化建模方式。在并行离散模拟内核情况下,需要做一部分封装内核的工作以为事务级提供一致的接口,这样在事务级的代码就可以完全向后兼容了。

参 考 文 献

[1] Open SystemC Initiative[EB/OL]. http://www.systemc.org

[2] Pérez D G, Mouchard G, Temam O. FastSysC: A Fast SystemC Engine[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE). Paris, 2004

[3] Guindi R S, Naguib Y N. SplitPro: A Tool to Over- come SystemC Scheduling Inefficiencies[C]// Microelectronics (ICM), 2010 International Conference. Cairo, 2010: 347-350

[4] Mello A, Maia I, Greiner A, et al. Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE). Dresden, 2010: 606-609

[5] Combes P, Caron E, Desprez F, et al. Relaxing Synchronization in a Parallel SystemC Kernel[C]// The 2008 IEEE International Symposium on Parallel and Distributed Processing With Applications. Sydney, 2008: 180-187

[6] Viaud E, Pecheux F, Greiner A. An Efficient TLM/T Modeling and Simulation Environment Based on Conservative Parallel Discrete Event Principles[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE). Munich, 2006: 1-6

[7] Chandy K M, Misra J. Distributed simulation: A Case Study in Design and Verification of Distributed Programs[J]. IEEE Trantions on Software Engineering Software, 1979, 5(5): 440-452

[8] 李挥,陈曦. SystemC 电子系统级设计[M]. 北京:科学出版社, 2010

[9] 陈曦,徐宁仪. SystemC 片山系统设计[M]. 北京:科学出版社, 2003

[10] Bhasker J. SystemC 入门(第 2 版)[M]. 夏宇闻,甘伟,译. 北京:北京航空航天大学出版社, 2008

[11] Ezudheen P, Chandran P. Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machine[C]// 23rd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation. New York, 2009: 80-87

[12] Fujimoto R M. 并行与分布仿真系统[M]. 李革,刘宝宏,张耀程,译. 北京:电子工业出版社, 2010

[13] Stallings W. 操作系统:精髓与设计原理(第 6 版)[M]. 陈向群,陈渝,译. 北京机械工业出版社, 2010

[14] MSDN library[EB/OL]. http://msdn.microsoft.com/library

(上接第 304 页)

[2] Linn C, Debray S. Obfuscation of Executable Code to Improve Resistance to Static Disassembly[C]// Proceedings of the 10th ACM Conference on Computer and Communications Security. New York: ACM, 2003: 290-299

[3] Gough K J, Cifuentes C, Corney D, et al. An experiment in mixed compilation/interpretation[C]// Proceedings of the Fifteenth Australian Computer Science Conference. Hobart: Australian Computer Society, 1992: 315-327

[4] Cifuentes C, Van Emmerik M, Ramsey N, et al. Experience in the Design Implementation and Use of A Retargetable Static Binary Translation Framework [R]. CA, USA: Sun Microsystems, 2002

[5] 曾鸣,赵荣彩,姚京松. 一种基于重定位信息的二次反汇编算法[J]. 计算机科学, 2007, 34(7): 284-287

[6] 蒋烈辉,陈亮,等. 基于控制流和数据段分析的反汇编策略研究[J]. 计算机工程, 2007, 33(2): 94-96

[7] 胡刚. 固件程序代码逆向分析关键技术研究[D]. 郑州:郑州信息工程学院, 2010

[8] 张翠艳. 固件代码安全缺陷分析技术研究[D]. 郑州:郑州信息工程学院, 2010