·安全技术·　　文章编号：1000-3428(2019)12-0134-07　　文献标志码：A　　中图分类号：TP309

面向 RISC 处理器的控制流认证方案

李　扬，戴紫彬，李军伟

（信息工程大学，郑州 450001）

摘　要：为使 RISC 处理器平台具备检测代码重用攻击的能力，将控制流完整性机制与可信计算中的动态远程证明协议相结合，提出面向 RISC 处理器的硬件辅助控制流认证方案。以开源 RISC 处理器为基础，扩展与处理器紧耦合的硬件监控单元，同时给出控制流认证方案的证明协议，设计用于跟踪执行路径的硬件编码方法以实现信息压缩。实验结果表明，与 C-FLAT 方案相比，该方案传输延时小且资源消耗少，能够保证 RISC 处理器控制流的可信安全。

关键词：代码重用攻击；控制流完整性机制；远程证明；硬件；安全处理器

开放科学（资源服务）标志码（OSID）：

中文引用格式：李扬，戴紫彬，李军伟. 面向 RISC 处理器的控制流认证方案[J]. 计算机工程，2019，45(12)：134-140，146.

英文引用格式：LI Yang，DAI Zibin，LI Junwei. Control flow authentication scheme for RISC processor[J]. Computer Engineering，2019，45(12)：134-140，146.

Control Flow Authentication Scheme for RISC Processor

LI Yang，DAI Zibin，LI Junwei

（Information Engineering University，Zhengzhou 450001，China）

【Abstract】To enable RISC processor platforms to detect code reuse attacks，this paper proposes a hardware-assisted control flow authentication scheme for RISC processors，integrating Control Flow Integrity（CFI）mechanism with dynamic remote attestation protocols in trusted computing. Based on open-source RISC processors，the scheme extends hardware monitoring units that are tightly coupled with processors，and gives an attestation protocol of the control flow authentication scheme. It also designs a hardware encoding method for execution path tracking to compress information. Experimental results show that compared with the C-FLAT scheme，the proposed scheme can reduce transmission delay and resource consumption，and can ensure trusted security of control flows in RISC processors.

【Key words】code reuse attack；Control Flow Integrity（CFI）mechanism；remote attestation；hardware；security processor

DOI：10.19678/j.issn.1000-3428.0053415

0　概述

代码重用攻击[1]作为一种新型软件攻击技术，其利用漏洞改变内存中原有的程序代码达到攻击目的，严重威胁到 RISC 处理器的安全[2]。在针对代码重用攻击设计安全机制时，考虑到软件安全机制的局限性[3]，应结合 RISC 处理器结构适当引入硬件安全模块，以有效检测代码重用攻击并减少软件机制引起的性能开销。

目前，较多方案将硬件监控单元集成到处理器流水线中[4]，将需要的信息从流水线阶段转发到硬件监控单元，使硬件监控单元利用这些信息来验证控制流转移的有效性。文献[5]设计的 SCFP 架构是一种基于海绵结构密码算法[6]的控制流保护技术，其在编译阶段使用基于海绵结构的密码算法加密程序，并在CPU 运行过程中逐条对指令解密并执行。文献[7]提出采用 PUF 加密的硬件控制流检测技术，但其使用的汉明距离计算方法和线性加密机制过于简单，容易受到重放攻击。文献[8]提出的 SOFIA 是一种能够防御多种控制流攻击的硬件方法，其通过使用流密码和 MAC 加密验证及连接指令块，能够执行控制流完整性（Control Flow Integrity，CFI）机制，但由于全部采用垂直签名[9]，因此在检测过程中存在较大的延迟。文献[10]提出基于绑定消息认证码的硬件 CFI 机制 RAGuard，但其只是一种执行后沿的 CFI 机制。文献[11]将远程证明运用到程序控制流的合法性验证

基金项目：国家高技术研究发展计划。

作者简介：李　扬（1996—），男，硕士，主研方向为安全芯片设计；戴紫彬，教授、博士；李军伟，讲师。

收稿日期：2018-12-17　　**修回日期：**2019-01-18　**E-mail：**593935858@qq.com

中,提出 C-FLAT 方法,并在 ARM TrustZone[12] 平台上进行原型实现,但由于 C-FLAT 为软件方法,因此运行时会产生较大开销。

本文在 CFI 机制中结合硬件监控模块的设计思想,扩展与处理器紧耦合的硬件监控单元,同时引入动态远程证明协议,提出面向 RISC 处理器的硬件辅助控制流认证方案,并给出协议证明和降低开销的方法。

1　攻击模型与假设

RISC 处理器平台通常为程序数据和代码分别划分单独的存储区域,其中,数据存储区被设置为可读可写权限(rw),代码存储区被设置为可读可执行权限(rx),通过这种方式可确保数据不能被执行、代码不能被覆盖。此外,任何程序都可以通过其相应的控制流图(Control Flow Diagram,CFG)进行抽象。攻击模型如图 1 所示,图中的控制流图为程序在运行时应遵循的有效路径示例。

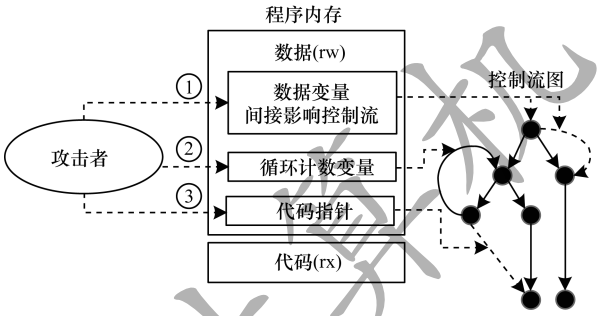

图 1　攻击模型

将运行时代码重用攻击划分为以下 3 类,在图 1 中用①~③表示。

1)针对数据变量的攻击。此类攻击通过攻击非控制数据,间接影响程序的控制流,例如修改条件分支中的条件和在控制流图有效路径内改变控制流执行的方向。

2)针对循环的攻击。此类攻击通过修改程序中循环的条件,在控制流图有效路径内改变循环路径的执行次数,例如面向循环编程(Loop Oriented Programming,LOP)攻击[13]。

3)针对代码指针的攻击。此类攻击主要破坏返回地址和函数指针,从而执行违反合法路径的恶意操作,例如面向返回编程(Return Oriented Programming,ROP)攻击[14]和面向跳转编程(Jump Oriented Programming,JOP)攻击[15]。

本文方案设计中对攻击者的假设包括以下 3 点:

1)攻击者可以利用内存漏洞修改任意位置的数据,但是无法在运行时修改程序代码。

2)本文方案仅检测软件攻击,尤其针对代码重用攻击,针对设备的物理攻击不在本文的讨论范畴。

3)本文方案能够检测影响程序控制流的攻击,但不能检测纯数据驱动的攻击,例如面向数据编程(Data Oriented Programming,DOP)攻击[16]。

2　硬件监控单元架构设计

图 2 显示了与 RISC 处理器紧耦合的硬件监控单元总体架构。本文设计需要简单修改处理器流水线中的部分硬件逻辑,使紧耦合的硬件监控单元能够跟踪处理器的控制流信息。硬件监控单元中主要包括分支过滤模块和循环监控模块。

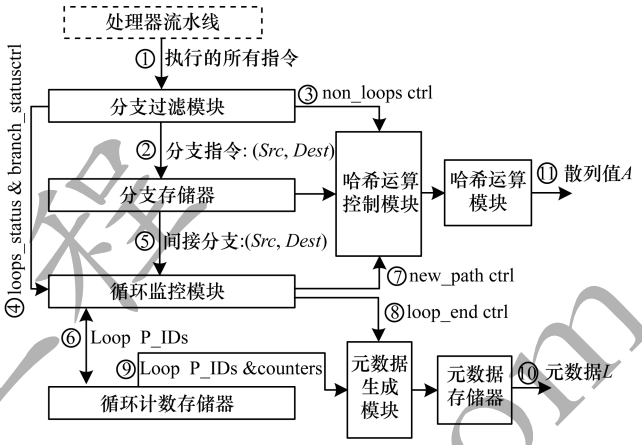

图 2　安全 RISC 处理器总体架构

2.1　分支过滤模块

分支过滤模块通过跟踪每个时钟周期执行的指令,能够筛选出与控制流相关的指令,包括调用、跳转、返回和条件分支,并将它们简明地表示为指令的源地址和目标地址对(*Src*,*Dest*)。在此基础上,检测该分支指令是否在程序的循环语句内,如果不在循环语句内,则向哈希运算控制模块发送 non_loops ctrl 信号,哈希运算控制模块接收信号后,控制哈希运算模块从分支存储器取出(*Src*,*Dest*),并计算散列值 *A*;如果在循环语句内,则需要对所在循环路径进行编码。

2.2　循环监控模块

循环监控模块能够对程序中的循环进行编码和计数。针对程序中循环语句的特殊处理,如果同一循环路径每次迭代都进行散列运算,会产生数量庞大的散列结果,占用大量的存储资源。因此,本文方案在连续多次执行相同的循环路径时,仅在首次执行时对循环路径中的(*Src*,*Dest*)进行散列处理,剩余的执行只记录循环次数,从而有效压缩与循环相关的执行信息。

3　控制流认证方案设计

本文方案在跟踪监控期间,由于硬件监控单元能够并行记录处理器中的控制流,因此不会造成处理器停顿,避免了利用软件技术的开销。在检查认证期间,单设备认证需要中断处理器,可以通过优化认证方案减小延迟(例如调用系统函数前认证),在多设备认证中则可以选择不中断处理器的运行,而是在认证失败时采取拒绝通信的方式将认证失败设备与其他设备进行隔离。

3.1 证明协议

本文控制流认证方案的证明协议如图 3 所示，其中，验证方 V 通过协议对证明方 P 上程序 S 的运行情况进行认证，即判断 P 上程序的执行路径是否可信。证明协议规定 V 和 P 都能够访问程序的二进制，并且通过传统的静态证明协议确保 P 正在执行的程序 S 是完整、未被修改的。

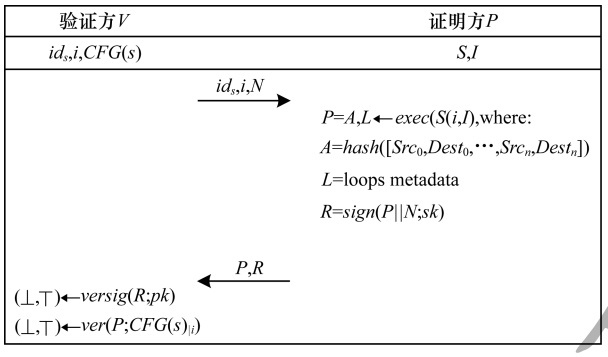

图 3 控制流认证方案的的证明协议

证明协议的执行流程具体如下：

1）V 通过对程序二进制进行静态和动态分析完成预处理，生成 S 的 CFG（包含循环语句执行的具体情况）。

2）V 通过向 P 发送程序认证输入 i、程序编号 id_s 和随机数 N 来启动协议，其中发送随机数 N 是为了防止重放攻击[17]。

3）P 使用认证输入 i 和不可信输入 I 来执行 S，不可信输入 I 可能会通过上文的代码重用攻击技术劫持控制流。在 S 的执行过程中，P 中硬件监控单元将捕获控制流中每个分支的源地址和目标地址对作为哈希运算模块的输入，以生成累积认证值 A。如果 P 对所有指令都进行认证，将产生巨大的存储开销、功耗和通信开销，特别是对于资源受限的 RISC 处理器设备，将严重影响设备的性能。因此，P 仅计算执行路径中分支指令的累积认证值，此外，P 还生成辅助元数据 L，用于认证程序中的循环路径和迭代次数（包括递归函数）。

4）将 A 和 L 合并为包含唯一程序路径信息的 P，在程序退出时，利用签名密钥 sk 对 P 和随机数 N 进行签名，生成报告 $R = sign(P \| N; sk)$，报告 R 被存储在受保护的存储器中，其只能被硬件监控单元访问。

5）P 将 R 发送到 V，V 在接收到 R 后，使用密钥 pk 对签名进行验签。验签无误后，V 检查 R 中的路径信息 P，与 CFG 在认证输入 i 下的合法路径是否一致，如果检查结果为真，则表示 P 没有受到代码重用攻击。

3.2 循环处理

本文针对 RISC 处理器中循环路径的执行特点，提出一套硬件编码方案，以跟踪和记录程序中循环语句的具体执行情况，为控制流认证中辅助元数据 L 的生成提供方法。

3.2.1 循环识别

为在运行过程中高效地识别程序中的循环语句，首先需要找到程序执行循环语句时的特征。程序中的循环结构包括 do-while、while 和 for，从它们的汇编代码中可以发现，向后跳转的条件分支指向一个循环的开始，本文将这个共有特征作为判断是否开始执行循环的依据，将 3 种循环结构统一为图 4 所示的概念结构，每当执行一次循环，判断一次循环条件，根据判断结果选择执行入口指令还是出口指令，当执行入口指令时，跳转到入口节点地址执行；当执行出口指令时，顺序执行内存中的下一条指令。

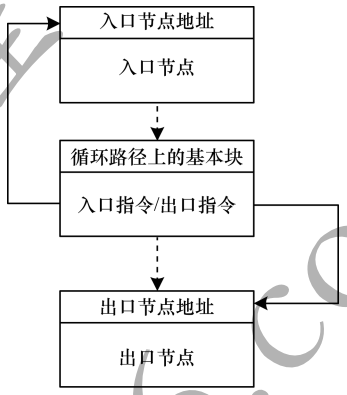

图 4 循环概念结构

循环的入口节点地址和出口节点地址被用于判断循环的迭代次数和循环深度。通过记录循环路径编码出现的次数来确定循环迭代的次数。当在执行当前循环路径中执行新循环路径的入口节点时，表示发生循环嵌套。当执行到循环出口节点时表示退出循环，退出循环的原因可能是程序顺序执行的结果，也可能是 break 跳出循环的结果。

3.2.2 循环中条件分支的编码

本文设计的跟踪机制规定，当识别到表示循环入口指令的分支类型时，循环监控模块中的循环路径编码器开始为当前循环路径编码。图 5 为一段包含循环的程序伪代码及其 CFG。示例伪代码中包含了带有 if-else 语句的 while 循环。对伪代码中的各个基本块进行编号，使它们分别对应 CFG 中的节点。CFG 中循环的入口节点为 N_2，出口节点为 N_7。在这个简单的循环中，只有 2 条有效循环路径，其中包括实线路径 $N_2 \rightarrow N_3 \rightarrow N_4 \rightarrow N_6 \rightarrow N_2$ 和虚线路径 $N_2 \rightarrow N_3 \rightarrow N_5 \rightarrow N_6 \rightarrow N_2$。

对于每个条件分支，处理器进行判定：如果采用分支，跳转到计算得出的目标地址执行；如果不采用分支，处理器顺序执行内存中的下一条指令。本文设计通过跟踪这些分支行为，使用‘1’或‘0’对采用或不采用条件分支进行编码，对不存在分支的顺序路径不进行编码，即图 5（b）中“-”标记的路径。所有的分支信息由分支过滤模块从处理器流水线中提取，并被循环监控模块中循环编码器所利用，从而实现对每个循环路径的编码，编码结果记为 path_IDs，

则图 5(b)中虚线路径$N_2 \rightarrow N_3 \rightarrow N_5 \rightarrow N_6 \rightarrow N_2$ 的 path_IDs 被编码为“011”,实线路径$N_2 \rightarrow N_3 \rightarrow N_4 \rightarrow N_6 \rightarrow N_2$ 的 path_IDs 被编码为“0011”。

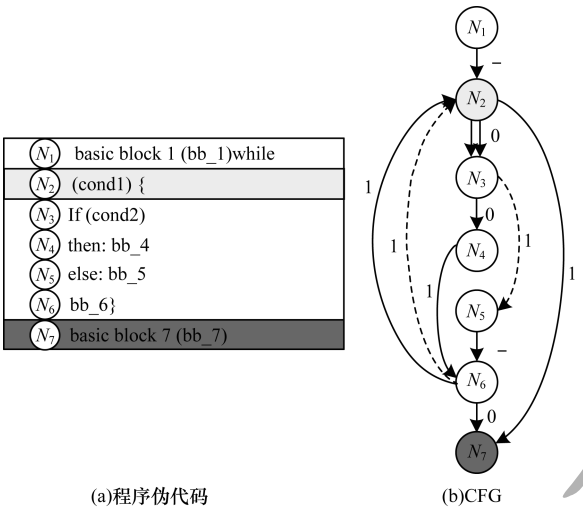

图 5　包含循环的程序伪代码及其 CFG

3.2.3　循环中间接分支的编码

循环中的间接分支可能涉及多个目标地址,不能简单地使用“1”或“0”进行编码,为对循环路径进行唯一编码,本文方案利用较少位数的n bit 对具有不同目标地址的间接分支进行编码,使得每次认证可用的间接分支编码数量为2^n个,如图 6 所示。间接分支的编码被存储在片上内存,当运行时出现超过预存的间接分支编码数量时,中断处理器的运行。当退出循环时,组装元数据 L。元数据 L 被存储在片上存储器中,其中包括每条循环路径出现的顺序,循环路径的编码,循环路径执行的次数以及循环中间接分支的编码。在最终的认证过程中,累积散列值 A 描述了程序顺序执行的情况,元数据 L 描述了程序中循环的执行情况,将两者相结合,能够有效表示程序在特定输入下执行路径的具体情况。

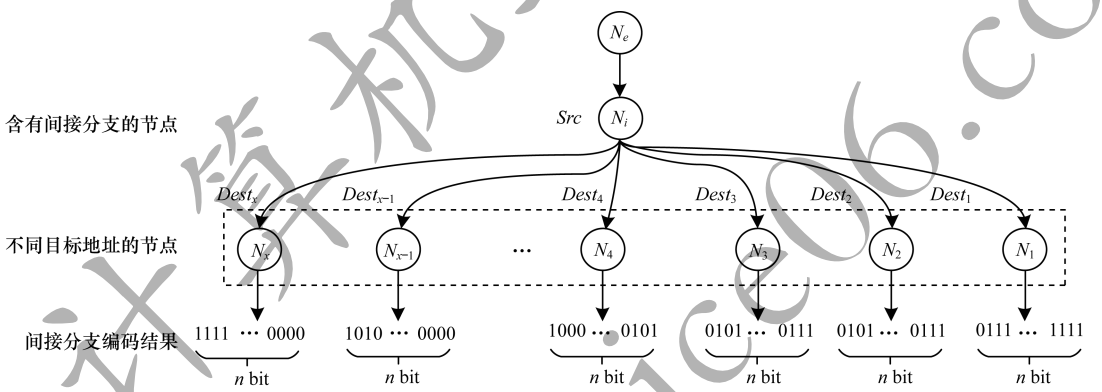

图 6　循环中间接分支的编码原理示意图

4　关键硬件单元设计

4.1　分支过滤模块

处理器与流水线紧耦合的分支过滤模块如图 7 所示,其利用从流水线各个阶段引出的信号,跟踪处理器分支的执行。

在选择的开源 RISC 处理器中[18],指令长度都是固定的 32 bit,图 8 为指令集中所有的分支指令,这些指令的操作码是对齐的,分支过滤模块利用取指阶段取出的 32 位指令信息的 0 ~ 6 bit 和 12 bit ~ 14 bit,筛选出流水线中与控制转移相关的指令。

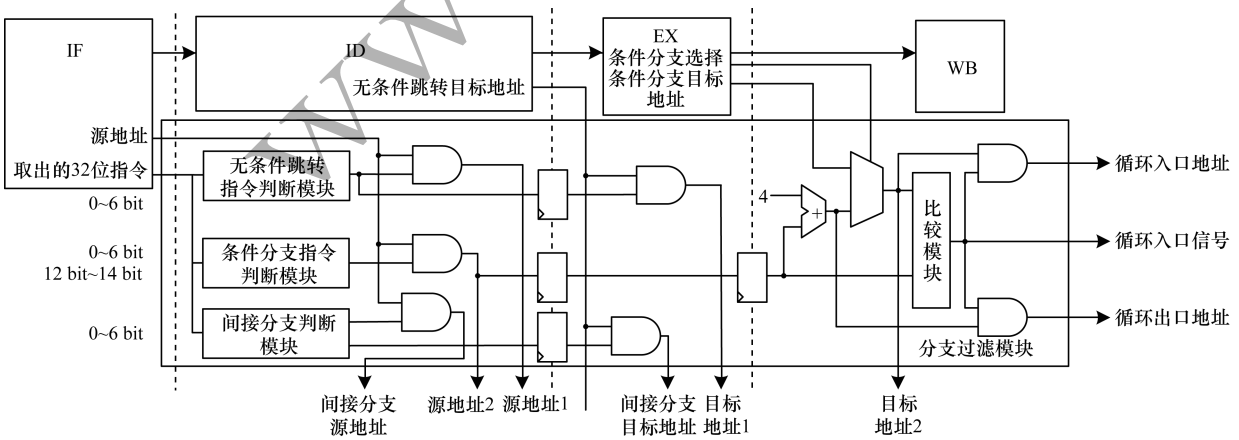

图 7　分支过滤模块结构

31 25	24 19	15 14	12 11 7	6 0	
imm[20|10:1|11|19:12]				1101111	JAL
imm[1:0]		rs1	000	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

图 8 指令集中的分支指令

所有分支指令的源地址在取指阶段产生,所有分支指令的目标地址在译码阶段产生,需要注意的是,无条件跳转指令 jal 和 jalr 一定会跳转到目标地址执行,而条件分支指令是否跳转到目标地址执行,取决于在执行阶段对指令中条件的判断,当条件分支指令不跳转到目标地址执行时,将顺序执行内存中的下一条指令,此时将程序计数器加 4 指向的地址作为条件分支的目标地址,以便于硬件监控单元跟踪控制流的执行。

当跟踪无条件跳转指令时,在译码阶段采集指令的源地址 1,在执行阶段采集指令的目标地址 1;当跟踪条件分支指令时,在译码阶段采集指令的源地址 2,在写回阶段采集指令的目标地址 2。条件分支指令目标地址的 2 个可能值作为二选一数据选择器的输入,执行阶段引出的条件分支选择信号作为数据选择器的选择信号,从而输出实际执行指令的地址。

本文选取的 RISC 处理器没有配备硬件分支预测器,为保证性能,采用默认的静态分支预测机制,即:如果是向后跳转的条件分支指令,则预测为“跳”;如果是向前跳转的条件分支指令,则预测为“不跳”,并且要求编译器也按照这种默认的静态分支预测机制来编译生成汇编代码,从而使该处理器能够得到较好的性能。如上文所述,在执行阶段才会对条件分支指令中的条件进行判断,即在执行阶段才能判断静态分支预测结果是否正确,当预测失败时,将冲刷掉处理器取指阶段和译码阶段中的指令信息,处理器在停顿 2 个时钟周期后才能继续工作,为防止分支过滤模块跟踪实际不执行的分支指令,分支过滤模块在接收到预测结果不正确的信号后,接下来的 2 个时钟周期不访问分支存储器,即使按照静态分支预测机制预先取出的 2 条指令中包含分支指令,它们的(*Src*,*Dest*)也不会被存储在分支存储器中,不会对它们进行散列运算。

分支过滤模块还具备判断循环状态的功能,根据 3.2.2 节提出的识别循环入口的方法,分支过滤模块通过比较已跟踪到的条件分支的目标地址是否大于源地址,来判断该条件分支指令是否为循环入口指令,如果是入口指令,则将该条指令的目标地址作为循环入口节点地址,将该条指令的下一条指令地址作为循环出口节点地址,并把它们存储起来用于对循环状态的判断。此外,为便于循环监控模块对循环中间接分支的编码,将间接分支的目标地址单独引出。

4.2 循环监控模块

循环监控模块结构如图 9 所示。将分支过滤模块的输出作为其输入,模块内包含地址寄存器、循环路径编码器、IBE 缓存和计数器等子模块。寄存器 *P* 用于发现循环嵌套最内层的循环是否在连续执行相同的循环路径,当比较单元发现新循环路径的编码结果与寄存器 *P* 中的编码相同时,则循环计数器中的计数值增加 1,此循环路径中分支不被用于散列运算,当寄存器 *P* 中没有存储编码结果,或新的编码结果与寄存器 *P* 中的编码结果不同时,则使用新的编码更新寄存器 *P*。

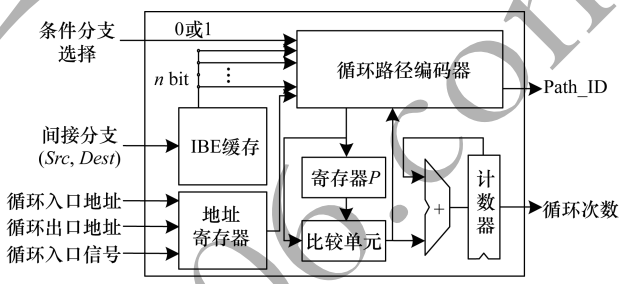

图 9 循环监控模块结构

循环监控模块支持深度为 3 的循环嵌套,当其接收到一个循环入口信号时,将相应的循环入口地址和出口地址存储在地址寄存器组中,同时循环路径编码器开始编码,在编码过程中,如果出现新的循环入口信号,首先与存储在寄存器组中的最新循环入口地址比较,如果相同,则上一个循环的编码结束,循环中所有分支的(*Src*,*Dest*)被用于散列运算,编码结果与存储在寄存器 *P* 中的编码进行比较(或将编码结果存储在寄存器 *P*),并清空循环路径编码器中的已有编码;如果不同,上一个循环的编码同样结束,已执行的分支(*Src*,*Dest*)同样被用于散列运算,循环路径编码器中的已有编码同样被清空,但不同之处在于对于寄存器 *P* 的访问只能为清空其中的编码结果。当发现执行的指令地址是循环的出口地址时,清空地址寄存器中对应的循环入口地址和出口地址,如果循环路径编码器中有编码结果,则将编码结果和计数值都存储到循环计数存储器中,同时清空循环路径编码器中的编码结果。

循环路径编码器的本质是由数据选择器和寄存器组成的循环移位寄存器,条件分支的编码位数为 1 bit,通过串行输入进入循环移位寄存器,间接分支的编码为 *n* bit,通过并行输入进入到循环移位寄存器,通过图 10 所示的循环编码器结构,实现一次向

循环移位寄存器中写入1 bit 或n bit 的功能,达到对循环路径编码的目的。

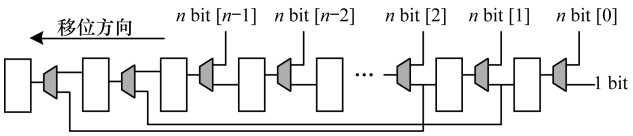

图 10　循环编码器结构

为减小循环中间接分支编码引入的开销,在循环监控模块中添加间接分支编码缓存(Indirect and Branch Encoding Cache,IBECache),如图 11 所示,IBECache 包含 2 个内部数据结构:*IBEIndex* 和 *IBELookUp*。*IBELookUp* 中包含了间接分支和对应的编码结果,*IBEIndex* 则保存了 *IBELookUp* 的索引。

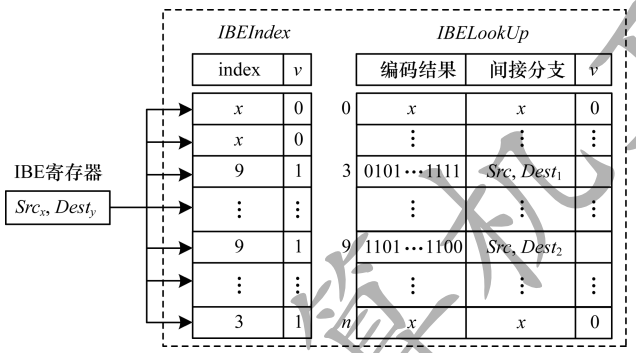

图 11　IBECache 结构

IBELookUp 是一个特殊的内存数组,数组中的每个条目包含四部分内容,分别为编码结果(n bit)、间接分支的源地址(32 bit)、间接分支的目标地址(32 bit)和有效位(1 bit)。*IBELookUp* 的深度根据具体的平台确定,可以在所选平台上运行一组基准程序,根据不同间接分支目标地址出现次数的最大值确定 *IBELookUp* 的深度。当 *IBELookUp* 有效位为 1 时,表示对应条目为实时存储区域,存储一个编码结果和一个(*Src*,*Dest*);当 *IBELookUp* 有效位为 0 时,表示对应条目的存储区域未被利用,或已被释放。

IBEIndex 是一组影子寄存器,IBE 寄存器与这些影子寄存器相关联,每个影子寄存器包含两部分内容:索引和有效位。当 *IBEIndex* 有效位为 1 时,表示 IBE 寄存器中(Src_x,$Dest_y$)的编码结果存在于 *IBELookUp* 中;*IBEIndex* 有效位为 0 时,则需要从存储器中取出新的编码结果,并将编码结果和当前(Src_x,$Dest_y$)一同存入 *IBELookUp* 中,同时,将正在更新条目的索引存储到 *IBEIndex* 中的对应位置,并置有效位为 1。

当(Src_x,$Dest_y$)被加载到 IBE 寄存器中时,对(Src_x,$Dest_y$)与 *IBELookUp* 中所有条目的间接分支部分进行比较,如果存在匹配项,则将匹配条目的索引存储到 *IBEIndex*;如果不存在匹配项,则从存储器中获取新的编码,并将其存储在 *IBELookUp* 中未利用的条目中,同时置条目的有效位为 1。

为防止使用相同的编码结果对不同的(Src_x,$Dest_y$)进行编码,IBECache 缓存除了为设计降低开销和减小延迟,最重要的是通过限制条目数量保证一个编码结果只被一个(Src_x,$Dest_y$)使用。设计使得预存的编码数量与 IBECache 的深度相同,当 IBECache 发出缓存填满的信号时,如果(Src_x,$Dest_y$)在 *IBEIndex* 中没有对应的索引,在 *IBELookUp* 中没有对应的间接分支,则向处理器发送中断信号,进一步确诊产生异常的原因。原因可能包括以下 2 个方面:1)预存的编码数量不足,即 IBECache 缓存的深度不够,这可能是因为运行的基准程序不能代表所有在该平台运行的应用程序,有的应用程序中的间接分支的目标地址数目可能更多;2)攻击者利用大量的控制转移构建代码重用攻击,导致用于间接分支的编码数量不足。

5　实验与验证

5.1　实验平台

实验的硬件平台为 ZYNQ XC7Z020-CLG484,本文实验通过修改开源 SoC PULPino 实现如图 2 所示的硬件架构[19],其中包括具有 4 级流水线的开源基础处理器和硬件监控单元。在内存中预存 16 个由 4 bit 的间接分支编码,*IBEIndex* 和 *IBELookUp* 都被实现为 16 个条目,*IBEIndex* 每个条目的的索引位都是 4 bit;循环路径编码器被实现为 128 bit 的循环移位寄存器。通过 IP 复用在架构中添加哈希运算模块,哈希运算模块作为通用模块,不考虑其占用的资源。

实验的软件平台为 FreeRToS 实时操作系统,通过在系统上运行基准例程测试硬件监控单元的性能,修改例程的执行路径模拟对系统发起代码重用攻击,测试硬件监控单元的功能。

5.2　实验结果

表 1 为基础处理器和硬件监控单元占用 FPGA 资源情况(其中数字表示资源数),以基础处理器的资源占用量为参考基准,硬件监控单元使总体资源占用量中的 flip flops、LUTs 和 BRAMs 分别增加了 7.9%、3.8% 和 8.2%。flip flops 主要被用于组成循环路径编码器和 IBECache,LUTs 主要被用于分支过滤模块和其他模块中组合逻辑。此外,设计中增加了较多的 BRAM,占用了较大的资源,BRAM 被配置为内容地址存储器(CAM),主要用于组成以循环路径编码 path_IDs 为索引的循环计数存储器。在 65 nm 工艺库下,使用 Design Compiler 综合工具对基础处理器和硬件监控单元进行综合分析,其面积

和频率结果如表 2 所示,可见硬件监控单元能够与基础处理器在同频率下工作。

表 1 FPGA 综合结果

测试对象	flip flops	LUTs	BRAMs
基础处理器	24 152	57 175	588
硬件监控单元	1 916	2 187	48

表 2 ASIC 综合结果

测试对象	面积/μm^2	频率/MHz
基础处理器	55 918	100
硬件监控单元	11 830	100

C-FLAT 需要通过软件插桩工具将程序中的分支指令替换为多条指令,才能在程序执行时跟踪控制流路径信息,而本文设计能够在处理器运行过程中,并行提取和压缩控制流信息,因此,本文设计在监控期间不会造成性能开销,但是在认证期间,可能需要中断处理器的运行,这是因为在分支过滤模块内部会产生 2 个时钟周期的延迟,在循环监控模块中会产生 5 个时钟周期的延迟,需要注意的是,在这 2 个模块延迟期间,可能还不需要执行认证,所以能够通过优化认证方案,在认证期间抵消较多的性能开销。

为直观地观察循环监控模块对控制流信息压缩的效果,在存在无循环监控模块的情况下,以基准例程中的 DES 密码算法为例,对哈希运算的数据量进行比较,如图 12 所示。可以看出,加入循环监控模块可以有效减少哈希运算的数据量。此外,本文方案还能准确检测出模拟发起的多种代码重用攻击。

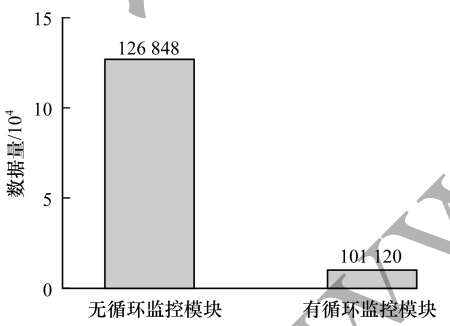

图 12 控制流信息的数据量比较

6 结束语

本文结合 CFI 机制与硬件监控模块,提出一种面向 RISC 处理器的硬件辅助控制流认证方案。通过添加分支过滤模块和循环监控模块,实时提取和压缩运行时处理器中的控制流信息,在无需修改程序二进制的情况下,完成对控制流的认证,达到检测代码重用攻击的目的。下一步将把该方案应用于 RISC 处理器设备群的控制流认证中,进一步扩大其适用范围。

参考文献

[1] LIU Tong, SHI Gang, MENG Dan. A survey of code reuse attack and defense mechanisms[J]. Journal of Cyber Security, 2016, 1(2): 15-27. (in Chinese)
柳童,史岗,孟丹. 代码重用攻击与防御机制综述[J]. 信息安全学报,2016,1(2):15-27.

[2] CHECKOWAY S, DAVI L, DMITRIENKO A, et al. Return-oriented programming without returns[C]// Proceedings of the 17th ACM Conference on Computer and Communications Security. New York, USA: ACM Press, 2010: 559-572.

[3] WU Chenggang, LI Jianjun. Evolution of control flow integrity[J]. China Education Network, 2016(4): 52-55. (in Chinese)
武成岗,李建军. 控制流完整性的发展历程[J]. 中国教育网络,2016(4):52-55.

[4] CLERCQ R D, VERBAUWHEDE I. A survey of hardware-based control flow integrity(CFI)[EB/OL]. (2017-07-31)[2018-12-10]. https://arxiv.org/ftp/ar xiv/papers/1706/1706.07257.pdf.

[5] WERNER M, UNTERLUGGAUER T, SCHAFFENRATH D, et al. Sponge-based control-flow protection for IoT devices[EB/OL]. (2018-02-19)[2018-12-10]. https://arxiv.org/pdf/1802.06691.pdf.

[6] CHANG D. Sufficient conditions on padding schemes of sponge construction and sponge-based authenticated-encryption scheme[C]//Proceedings of International Conference on Cryptology in India. Berlin, Germany: Springer, 2012: 545-563.

[7] ZHANG Jiliang, QI Binhang, QIN Zheng, et al. HCIC: hardware-assisted control-flow integrity checking[J]. IEEE Internet of Things Journal, 2018, 6(1): 458-471.

[8] CLERCQ R D, KEULENAER R D, COPPENS B, et al. SOFIA: software and control flow integrity architecture[C]// Proceedings of Design, Automation and Test in Europe Conference and Exhibition. Washington D. C., USA: IEEE Press, 2016: 1172-1177.

[9] WILKEN K, SHEN J P. Continuous signature monitoring: low-cost concurrent detection of processor control errors[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1990, 9(6): 629-641.

[10] ZHANG Jun, HOU Rui, FAN Junfeng, et al. RAGuard: a hardware based mechanism for backward-edge control-flow integrity[C]//Proceedings of Computing Frontiers Conference. New York, USA: ACM Press, 2017: 27-34.

[11] ABERA T, ASOKAN N, DAVI L, et al. C-FLAT: control-flow attestation for embedded systems software[C]// Proceedings of 2016 ACM SIGSAC Conference on Computer and Communications Security. New York, USA: ACM Press, 2016: 743-754.

[12] ARM Ltd. ARM security technology: building a secure system using TrustZone® technology[EB/OL]. [2018-12-10]. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

(下转第 146 页)

(上接第 140 页)

[13] LAN Bingchen, LI Yan, SUN Hao, et al. Loop-oriented programming: a new code reuse attack to bypass modern defenses[C]//Proceedings of IEEE Trustcom/BigDataSE/ISPA. Washington D.C., USA: IEEE Press, 2015: 190-197.

[14] CHECKOWAY S, DAVI L, DMITRIENKO A, et al. Return-oriented programming without returns [C]// Proceedings of ACM Conference on Computer and Communications Security. New York, USA: ACM Press, 2010: 559-572.

[15] XING Xiao. Automated construction of Jump-Oriented Programming attacks[D]. Nanjing: Nanjing University, 2012. (in Chinese)
邢骁. 自动化构造 Jump-Oriented Programming 攻击[D]. 南京: 南京大学, 2012.

[16] HU Hong, SHINDE S, ADRIAN S, et al. Data-oriented programming: on the expressiveness of non-control data attacks[C]//Proceedings of 2016 IEEE Symposium on Security and Privacy. Washington D. C., USA: IEEE Press, 2016: 969-986.

[17] LIU Jiafen, ZHOU Mingtian. Research and taxonomy of replay attacks on security protocol[J]. Application Research of Computers, 2007, 24(3): 135-139. (in Chinese)
刘家芬, 周明天. 对安全协议重放攻击的分类研究[J]. 计算机应用研究, 2007, 24(3): 135-139.

[18] RISC-V Foundation. The free and open RISC instruction set architecture[EB/OL]. [2018-12-10]. https://riscv.org/specifications.

[19] GitHub Inc. An open-source microcontroller system based on RISC-V [EB/OL]. [2018-12-10]. https://github.com/pulp-platform/pulpino.

编辑 金胡考