

基于硬件的动态指令集随机化框架的设计与实现

杜三, 舒辉, 康绯

(信息工程大学数学工程与先进计算国家重点实验室, 河南 郑州 450001)

摘要: 针对现有的指令集随机化方法存在从代码段中剥离数据困难、静态指令集随机化密钥固定和伪随机数密钥不安全等问题, 设计并实现了基于硬件的动态指令集随机化框架(HDISR), 通过在装载程序时加密程序代码, 将指令集随机化引入内核层和应用层的安全防护, 内核使用单独的内核密钥, 不同的应用程序使用不同的用户密钥。实验结果表明, HDISR 能将代码注入攻击降级为拒绝服务攻击, 且额外硬件损耗少于2.57%, 每兆字节代码加密的启动延时0.31 s。

关键词: 代码注入攻击; 指令集随机化; 动态指令集随机化; 内核

中图分类号: TP309.1

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2017.00216

Design and implementation of hardware-based dynamic instruction set randomization framework

DU San, SHU Hui, KANG Fei

(State Key Laboratory of Mathematical Engineering and Advanced Computing,
Information Engineering University, Zhengzhou 450001, China)

Abstract: All the existing ISR methods have some defects including stripping data from code segment is hard to accomplish, static ISR has fixed key and pseudo-random key is not secure. To introduce ISR technology into the security protection of kernel layer and application layer, hardware-based dynamic instruction set randomization framework (HDISR) was designed and implemented, in which program code was encrypted at loading time. Kernel encryption uses kernel key and applications encryption uses a different user key per process. The experimental results show that HDISR can degrade code injection attack to Denial of Service attack with less than 2.57% additional hardware and 0.31s startup delay of each megabyte code encryption.

Key words: code injection attack, instruction set randomization, dynamic instruction set randomization, kernel

1 引言

常用的网络应用和操作系统不断曝光内存破坏类漏洞(如缓冲区溢出、悬空指针、格式字符串等), 代码注入攻击能够利用漏洞执行恶意代码, 是网络空间最常见的安全威胁之一^[1]。作为

移动目标防御(MTD, moving target defense)的重要组成部分, 指令集随机化(ISR, instruction set randomization)对程序使用的指令集进行随机化处理, 而攻击者无法得知当前使用的指令集, 构造有效的漏洞利用程序, 因此对代码注入攻击有很好的防御效果。

收稿日期: 2017-09-27; 修回日期: 2017-10-19。通信作者: 杜三, qq13882525704@sina.com

基金项目: 国家重点研发计划基金资助项目(No.2016YFB08011601)

Foundation Item: The National Key R&D Plan Program of China (No.2016YFB08011601)

指令集随机化技术也可以保护脚本语言, 通过随机化标准的操作集, 使注入的脚本无效, 从而抵御 SQL 注入攻击^[2]和 Javascript 注入攻击^[3]等威胁。同时, 在程序混淆中使用指令集随机化可以增加攻击者逆向分析的难度^[4,5], 在硬件混淆中使用指令集随机化能够防止供应链中注入的后门程序执行^[6]。与地址空间随机化 (ASLR, address space layout randomization) 和数据执行保护 (DEP, data execution prevention) 相比, 指令集随机化不仅能对抗代码注入攻击, 还具有抗逆向能力, 可以防止非授权代码的执行。

2003 年, Kc 等^[7]首次提出了对抗代码注入攻击的通用方法——指令集随机化技术, 设计了基于 Bochs 仿真器的 X86 原型, 验证了硬件实现指令集随机化的可行性。Portokalidis 等^[8,9]提出了支持共享库和全面防御未授权代码执行的指令集随机化方法, 使用不同的密钥对不同的程序进行随机化处理。王奕森等^[10]提出了基于编译置换的指令集随机化技术, 实现了操作码层次的指令集随机化, 利用该技术实现了 X86 架构下 Windows 系统程序的指令随机化。上述基于软件实现的指令集随机化方法均采用静态指令集随机化, 存在一系列问题, 包括指令翻译性能损耗大、攻击翻译软件可以旁路绕过指令集随机化防御、密钥固定易遭到密钥猜测攻击^[11]。

针对上述问题, Papadogiannakis 等^[12]首次提出了动态指令集随机化方法, 并设计了硬件支持的指令集随机化系统 ASIST, 可以为内核和用户程序提供指令集随机化支持。Sinha 等^[13]设计了 Polyglot 系统, 改进了 ASIST 对页共享技术支持的缺陷。上述基于硬件实现的指令集随机化方法采用了动态指令集随机化, 在程序崩溃之后使用新的随机密钥重新加密代码, 使已探测的密钥值在程序重启后无效, 可以阻止密钥猜测攻击。但上述方法在实现时需要修改源码和链接脚本来彻底分离代码段与数据段, 而从代码段中分离数据困难; 同时使用了软件产生的伪随机数密钥, 具有可重现性, 如果攻击者拥有足够的计算能力, 可以破解伪随机数加密, 从而危及整个密码应用的安全^[14,15]。

针对现有方法的缺陷, 本文提出了基于编译

结果比较的代码区域信息生成方法, 解决了从代码段中分离数据困难的问题; 提出了基于代码区域信息的动态指令集随机化技术, 解决了静态指令集随机化密钥固定的问题; 在 CPU 中实现真随机数密钥生成, 解决了软件生成的伪随机数密钥安全性不高的问题; 最后设计了基于硬件的动态指令集随机化框架 (HDIR, hardware-based dynamic instruction set randomization), 实现了内核动态指令集随机化与多密钥的应用程序动态指令集随机化。

2 HDISR 总体框架

HDIR 总体框架如图 1 所示, 在 CPU 中增加了真随机数密钥生成模块与指令翻译模块, 前者产生内核密钥与用户密钥, 后者选择密钥, 解密读入的指令; 编译器支持模块通过基于编译结果比较的代码区域信息生成方法得到程序代码段中指令的具体位置信息——代码区域信息, 与加密密钥、选用的指令集随机化方法 (静态指令集随机化或者动态指令集随机化) 组成特征信息, 为内核与应用程序的指令集随机化提供支持; 在内核启动与应用程序装载时, 根据编译器提供的代码区域信息, 使用基于代码区域信息的动态指令集随机化方法, 加密内核与应用程序的代码; 引导程序在内核加载到内存后, 根据内核特征信息加密内核代码, 同时切换 CPU 执行模式为解密模式, 实现内核动态指令随机化; 可执行文件加载器装载可执行和可链接格式 (ELF, executable and linkable format) 文件时, 使用用户密钥加密 ELF 文件的代码, 在进程上下文切换的同时更新用户密钥, 实现多密钥的应用程序动态指令集随机化。

HDIR 的工作流程如下: CPU 产生真随机数密钥, 编译器支持模块生成代码区域信息; 根据代码区域信息, 引导程序使用内核密钥 (KernelKey) 加密内核代码, 内核使用用户密钥 (UserKey) 加密 ELF 文件的代码; 加密后的指令通过 CPU 中的指令翻译模块解密执行。通过 CPU、编译器、引导程序、内核和应用程序的相互配合, HDISR 将指令集随机化引入内核层与应用层的安全防护。

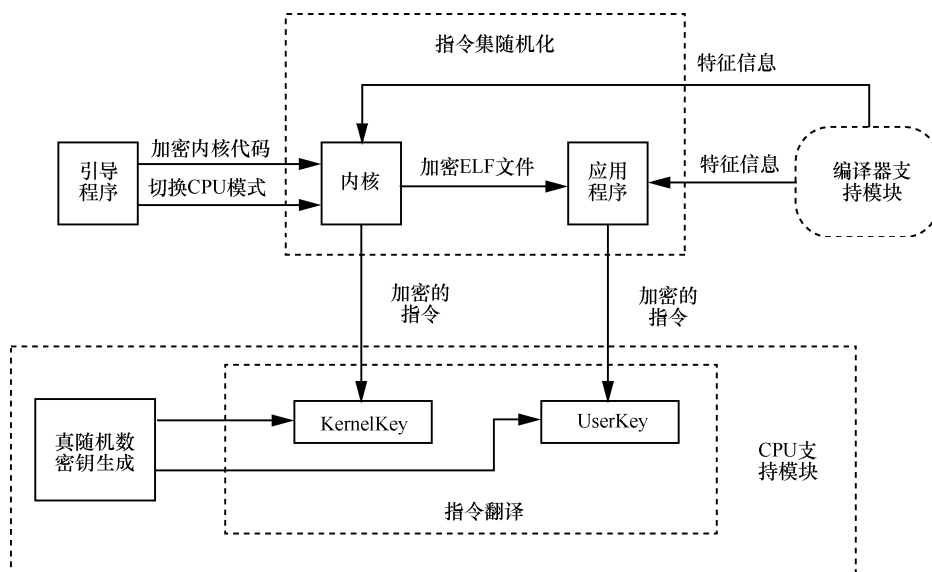


图 1 HDISR 总体框架

3 指令集随机化支持

3.1 CPU 支持模块

为了解决软件实现指令集随机化存在的性能损耗大和软件生成的伪随机密钥安全性不高等问题, HDISR 在 CPU 上实现了真随机数密钥生成和指令翻译, 此外, 设计了密钥生成指令、CPU 执行模式切换指令、读密钥指令、写密钥指令, 为内核与应用程序的动态指令集随机化提供支持。

3.1.1 真随机数密钥生成

指令集随机化的安全性依赖于加密密钥, 而软件生成的随机密钥依赖于初始化的种子, 不能产生真随机数。通过随机物理现象中获得随机性, 可以实现真随机数发生器, 产生具有不可预测性的随机序列。本文采用基于振荡器采样的真随机数生成方法, 设计了真随机数密钥生成模块, 生成供指令集随机化使用的密钥。

真随机数发生器原理如图 2 所示, 利用 D 触发器采样一个周期固定的快振荡器产生随机序列。首先使用 3 个反相器组成的环形振荡器作为噪声源, 产生自然抖动; 然后以 CPU 的系统频率作为采样频率 f_s , 使用 D 触发器对环形振荡器进行采样输出; 接着对 3 组环形振荡器的采样结果进行异或, 输出真随机数序列。此真随机数发生器产生的随机数序列通过了 FIPS 140-1 Poker 测

试; 最后使用有穷状态机, 将产生的随机数序列循环存入内部寄存器。

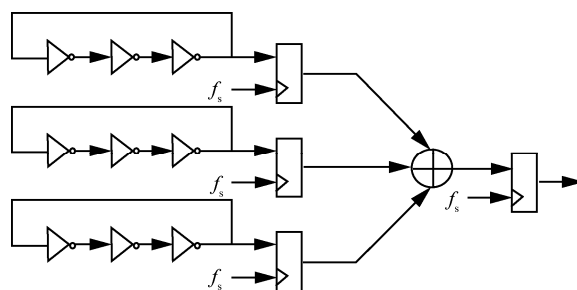


图 2 真随机数发生器原理

3.1.2 指令翻译

在 HDISR 框架中, 内核与应用程序的加密指令需要解密后才能正确执行。针对软件实现指令翻译存在性能损耗大和旁路绕过的问题, HDISR 在 CPU 中实现指令翻译。在取指令阶段与译码阶段之间加入指令翻译模块, 对读入的指令去随机化, 将翻译后的指令送入译码阶段, 保证指令的正常执行。

指令翻译原理如图 3 所示, 其中, KernelKey 和 UserKey 由真随机数密钥生成模块产生, 存放在自定义的内部寄存器、内核密钥寄存器与用户密钥寄存器。特权模式 (Supervisor) 位指示当前执行指令的特权模式, 为 Linux 系统的特权模式提供支持。根据 Supervisor 位判断指令的特权模式, 从而选择用户密钥或内核密钥作为解密密钥

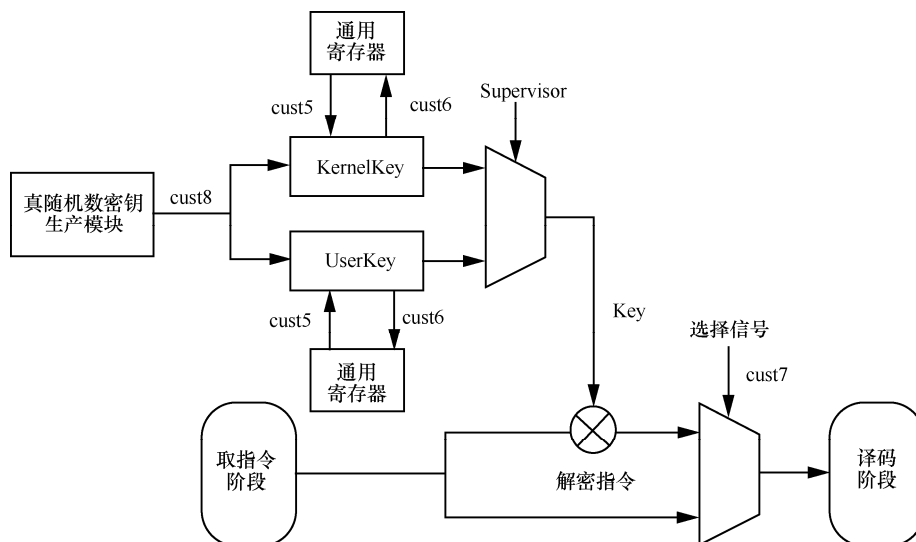


图3 指令翻译原理

解密当前执行的指令。在 OpenRisc 处理器中，特殊寄存器 SR 中的 SM 位作为 Supervisor 位。当 SM 为 1 时，表示当前执行指令处于内核模式；当 SM 为 0 时，表示当前执行指令处于用户模式。在取值阶段读入指令后，指令翻译的处理流程如下。

1) 取出指令后，根据选择信号判断当前指令是否需要解密。

① 如果不需要，直接执行步骤 2)。

② 否则，根据指令的 Supervisor 位，选择解密密钥。

a) 如果 Supervisor 等于 0，当前指令处于用户模式，选择 UserKey 作为解密密钥，解密指令。

b) 否则，当前指令处于内核模式，选择 KernelKey 作为解密密钥，解密指令。

2) 送入译码阶段，保证指令的正常执行。

3.1.3 自定义指令

在 HDISR 框架中，为了支持内核与应用程序的动态指令集随机化，CPU 增加了 4 条指令：密钥生成指令、CPU 执行模式切换指令、读密钥指令以及写密钥指令。

1) 密钥生成指令：cust8 K，真随机数密钥生成模块产生 32 bit 真随机数，存入相应的密钥寄存器。K=0 时，生成用户密钥，存到用户密钥寄存器；K=1 时，生成内核密钥，存到内核密钥寄存器。

2) CPU 执行模式切换指令：cust7，使能选择信号从而启动指令翻译功能，解密后的指令送入

译码阶段。

3) 读密钥指令：cust6 rD, K，读取密钥寄存器，存入通用寄存器 rD。K=0 时，读取用户密钥寄存器；K=1 时，读取内核密钥寄存器。

4) 写密钥指令：cust5 rD, K，将 rD 寄存器中的值写入密钥寄存器。K=0 时，写入用户密钥寄存器；K=1 时，写入内核密钥寄存器。

3.2 编译器支持模块

现有的指令集随机化方法假设代码段与数据段是严格区分的，但是代码段中可能夹杂数据，限制了此类方法的通用性。通过分析编译器的汇编过程，HDISR 提出了通用的静态指令集随机化生成方法和基于编译结果比较的代码区域信息生成方法，为内核与应用程序的动态指令集随机化提供支持。

3.2.1 静态指令集随机化生成

虽然 ELF 文件默认是将代码与数据分离的，但还是存在代码中混入数据的情况。例如，在内嵌汇编代码中定义数据会使编译生成的程序代码段夹杂着数据。现有的静态指令集随机化方法使用 objcopy 直接加密可执行文件的代码段，不仅会加密其中的指令，还会加密其中夹杂的数据。但是代码段中的数据不会被指令翻译模块解密，会导致程序执行异常。

在编译过程中，汇编器将指令翻译为对应的机器码。通过跟踪机器码生成过程，可以在产生指令机器码的同时加密机器码，生成静态加密的

可执行文件。在 gcc 编译器中, as 程序作为汇编器将汇编文件翻译成目标文件。分析 as 的处理流程之后发现, 指令汇编处理函数 `md_assemble()` 实现了每条指令的汇编。HDIRS 通过劫持指令的汇编过程, 在指令翻译为机器码后使用指定的密钥对指令的机器码加密, 可以生成静态指令集随机化的 ELF 文件。因此, 在汇编过程中只会对指令的机器码加密, 而不会对代码段中的数据加密, 避免了整个代码段加密引起的数据加密问题。

3.2.2 代码区域信息生成

为了解决从代码段中剥离数据困难的问题, HDIRS 提出了基于编译结果比较的代码区域信息生成方法, 为动态指令集随机化提供准确的代码位置信息, 具有更好的通用性。本文定义了代码区域信息来描述 ELF 文件中指令的具体位置, 考虑到代码段中夹杂着数据, 指令区域是不连续的, 因此定义代码区域信息数据结构如下。

```
typedef struct codelayout_info
{
    uint32 TotalNum;
    vector < uint32, uint32> Intervals; // Offset, Size
}
```

其中, *TotalNum* 为代码段中指令区域块的总数, *Offset* 为指令区域块的文件偏移, *Size* 为指令区域块的大小。指令区域块集合 *Intervals* 以 *Offset* 和 *Size* 为元组组成, 元组的个数为 *TotalNum*。

在静态指令集随机化生成的 ELF 文件中, 代码段中只有指令是加密的。通过比较原始的 ELF 文件和静态指令集随机化生成的 ELF 文件, 根据代码段中的内容加密与否判断是指令还是数据, 再聚合代码段中的指令位置得到代码区域信息。ELF 文件的代码区域信息生成算法描述如下。

算法 1 代码区域信息生成算法

输入 原始的 ELF 文件、静态指令集随机化生成的 ELF 文件、加密密钥

输出 ELF 文件的代码区域信息

- 1) 解析静态指令集随机化生成 ELF 文件, 得到可执行属性节的节偏移与大小。
- 2) 根据节偏移与大小, 遍历代码段中的指令。

3) 利用加密密钥, 解密静态指令集随机化生成的 ELF 文件中指令的机器码。

4) 判断解密后的机器码与原始 ELF 文件对应位置的指令机器码是否相等; 如果相等, 遍历 ELF 文件的代码区域信息, 判断是否存在指令区域块, 使当前指令位置在指令区域块的尾部。

① 如果存在, 则将此指令区域块的 *Size* 加上当前指令的大小。

② 否则, 建立新的指令区域块元组 $\langle \text{Offset}, \text{Size} \rangle$, 将指令偏移存入 *Offset*, 指令大小存入 *Size*。

代码区域信息生成算法同时也适用于内核映像。内核映像通过 ELF 文件 *vmlinux* 生成, 使用算法 1 可以得到 *vmlinux* 的代码区域信息, 再根据 *vmlinux* 与内核映像的对应关系, 得到内核映像的代码区域信息。

3.2.3 特征信息描述

为了对内核与应用程序的指令集随机化提供支持, 定义了特征信息数据结构, 将代码区域信息、密钥与指令集随机化的模式作为文件的特征。其中, *ISRmode* 字段标识文件采用的指令集随机化方法 (静态指令集随机化或动态指令集随机化); *Key* 字段存放文件使用的密钥; *TotalNum* 和 *Intervals* 表示文件的代码区域信息, 其具体含义与 *codelayout_info* 中一致。特征信息数据结构定义如下。

```
typedef struct feature_info
{
    uint32 ISRmode;
    uint32 Key;
    uint32 TotalNum;
    vector < uint32, uint32 > Intervals; // Offset, Size
}
```

4 内核动态指令集随机化

HDIRS 设计了基于代码区域信息的内核动态指令集随机化方法, 使用内核密钥加密内核代码, 将指令集随机化引入内核层的安全防护。本文通过修改引导程序加载内核的过程与指令操作码判断的处理流程, 实现了内核指令集随机化。

4.1 引导程序

引导程序在内核代码加载完成后, 会将控制权转交给内核执行。通过修改引导程序启动内核的流程, 在内核映像加载到内存之后, 根据内核代码区域信息, 使用内核密钥加密内核代码; 在跳转到内核执行前, 切换 CPU 执行模式为解密模式, 保证加密的内核指令能在 CPU 中解密执行。嵌入式系统常使用 Uboot 作为引导程序, 下面以 Uboot 引导加载 uImage 的过程介绍内核指令集随机化的启动过程。

4.1.1 内核代码加密

内核映像的代码段中夹杂着数据, 不能直接加密整个代码段。在 HDISR 框架中, 通过基于编译结果比较代码区域信息生成方法得到内核的代码区域信息, 再使用硬件生成的内核密钥对内核的代码区域加密, 解决了对整个代码段加密引起的数据加密问题。引导程序加载内核时通过随机产生的内核密钥对内核代码重新加密, 实现了动态指令集随机化。

分析 Uboot 引导启动过程之后发现, 函数 `do_bootm()` 实现内核映像的加载。通过劫持 `do_bootm()` 函数, 加密内核的代码区域。内核指令集随机化的启动过程如图 4 所示, 通过环境变量 `FeatureOffset` 指向 `uImage` 末尾的内核特征信息, 在映像文件加载到内存后解析内核特征信息, 根据 `ISRmode` 选择指令集随机化方法。

如果 `ISRmode` 值为 0, 表示采用静态指令集随机化, 内核映像通过静态指令集随机化生成方法得到, 内核代码已经加密, 此时需要通过 `cust5` 指令将加密密钥存入内核密钥寄存器, 保证加密代码与解密执行密钥的一致性。

如果 `ISRmode` 值为 1, 表示采用动态指令集随机化, 此时需要根据代码区域信息对内核代码加密。首先 `cust8` 指令生成 `KernelKey`, 接着 `cust6` 指令取出内核密钥, 最后在 `encrypt_code()` 函数中使用 `KernelKey` 对内核代码区域加密。如果内核被压缩过, 则在内核正确解压后再加密内核的代码。

4.1.2 CPU 执行模式切换

内核代码加载完成后, 引导程序会跳转到内核代码执行, 将控制权转交给内核。在跳转到内核代码执行之前, 需要切换 CPU 执行模式为解密模式, 保证加密的内核代码能在 CPU 中正常执行。分析 Uboot 之后发现, 内核启动函数 `do_bootm_linux()` 调用函数 `kernel()`, 实现向内核跳转执行。HDISR 需要先使能指令翻译模块, 再跳转到内核执行。`cut7` 指令切换 CPU 执行模式为解密模式, 再通过 `kernel()` 跳转到内核执行。

4.2 指令操作码

在内核代码加密之后, 不能直接使用原始的指令操作码来判断指令类型。此时, HDISR 需要先解密随机化的指令, 再判断指令的类型, 以保

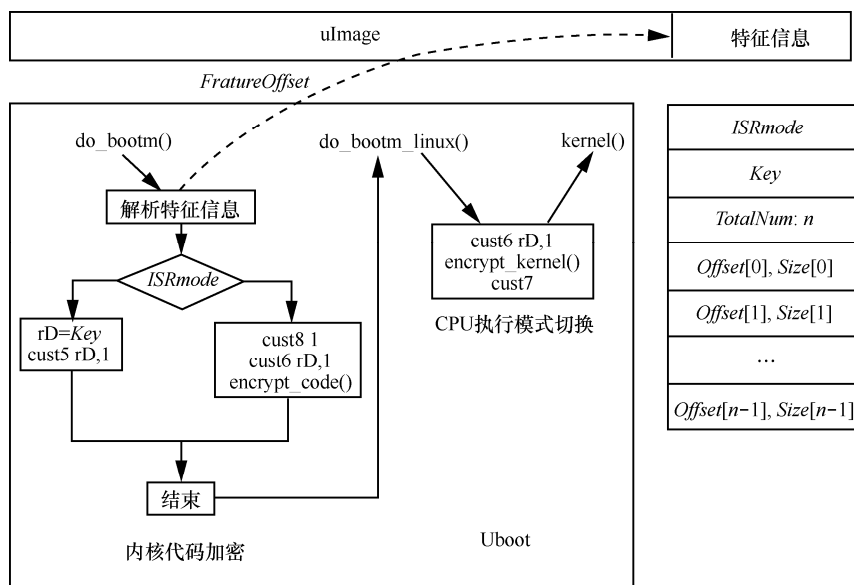


图4 内核指令集随机化启动过程

证系统的正常运行。例如，entry.s 中的异常处理_data_page_fault_handler 需要取出引发数据缺页的指令，根据指令的操作码判断是否为跳转指令。此时不能直接根据操作码判断跳转指令类型，如图 5 所示，HDISR 会根据引发缺页的指令的特权模式，选择指令的密钥（内核密钥或用户密钥）解密指令，再通过原始指令集的操作码判断指令类型。

```
l.sfeqi r6,0//1.j
l.bf 8f
l.sfeqi r6,1//1.jal
l.bf 8f
l.sfeqi r6,3//1.bnf
l.bf 8f
l.sfeqi r6,4//1.bf
l.bf 8f
l.sfeqi r6,0x11//1.jr
l.bf 8f
l.sfeqi r6,0x12//1.jalr
l.bf 8f
```

(a) 原始汇编代码

```
l.lwz r6,PT_SR(r3)
l.andi r6,r6,SPR_SR_SM
l.sfeqi r6,SPR_SR_SM //SM位
l.bf DS_kernel
l.nop
l.cust6 r21,0 //UserKey
l.j DS_end
l.nop
DS_kdrnel:
l.cust6 r21,1 //KernelKey
DS_end:
l.nop
//sfeqi r6,0 //1.j
l.srli r23,r21,26
l.xor r23,r23,r6
l.sfeqi r23,0 //1.j
l.bf 8f
```

(b) 支持ISR的汇编代码

图 5 指令操作码判断

5 应用程序动态指令集随机化

HDISR 设计了基于代码区域信息的应用程序动态指令集随机化方法，使用用户密钥加密 ELF 文件的代码，将动态指令集随机化引入应用层的安全防护。此外，在进程上下文切换的同时切换进程密钥，实现了多密钥的应用程序动态指令集随机化。

5.1 ELF 文件格式

为支持基于代码区域信息的动态指令集随机化，HDISR 修改了 ELF 文件的格式，如图 6 所示，在 ELF 文件中新增一节，存放 ELF 文件的特征信息，包括指令集随机化模式、密钥以及代码区域信息。指令集随机化模式（*ISRmode*）表示采用的指令集随机化方法；密钥（*Key*）表示 ELF 文件使用的加密密钥；代码区域信息提供代码段中指令的准确位置信息。

5.2 ELF 文件代码加密

在 Linux 内核中，装载函数 `load_elf_binary()` 只是为可执行文件和虚拟地址空间建立映射关系，并未真地将 ELF 文件的内容读入内存，只有在真正用到时才会以页故障方式读入内存。在 HDISR 框架中，通过基于编译结果比较的代码区域信息生成方法得到 ELF 文件的代码区域信息，在装载 ELF 文件前使用随机生成的用户密钥对 ELF 文件的代码区域加密，确保装载完成后通过页故障读入的代码是加密的，加密后的指令通过 CPU 解密执行。

ELF header
Program header Table
.text
.rdata
...
.data
.shstrtab
Section header Table
.symtab
.strtab

(a) 原始ELF文件格式

ELF header
Program header Table
.text
.rdata
...
.data
mydata
.shstrtab
Section header Table
.symtab
.strtab

ISR mode
Key
TotalNum:n
Offset[0], Size[0]
Offset[1], Size[1]
...
Offset[n-1], Size[n-1]

(b) 修改后ELF文件格式

图 6 ELF 文件格式修改

ELF 文件的代码加密流程如图 7 所示, 首先解析 ELF 文件, 获取 mydata 节的内容, 然后根据文件的特征信息数据结构解析此节。如果 *ISRmode* 值为 0, 表示 ELF 文件的代码已经加密, 实现静态指令集随机化; 如果 *ISRmode* 值为 1, 表示需要随机生成新的用户密钥, 重新加密 ELF 文件的代码区域, 实现动态指令集随机化。在重新加密 ELF 文件的代码时, HDISR 先使用 ELF 文件的 *Key* 解密代码, 再用新生成的用户密钥加密代码, 最后用新密钥更新 ELF 文件特征信息的 *Key*, 保证 ELF 文件的加密代码与密钥的一致性。

5.3 进程密钥切换

为了支持不同的应用程序使用不同的密钥, 在进程控制块 (PCB, process control block) 中增加了存放进程密钥的 *key* 字段, HDISR 在 ELF 文件装载时将加密密钥存入 *key* 中。此外, 进程上下文切换的同时需要将新进程的密钥写入用户密钥寄存器, 保证执行代码与指令翻译模块中用户

密钥的一致性。

在 HDISR 框架中, ELF 文件的装载过程如图 8 所示, 其中已加密的 ELF 文件 A 实现静态指令集随机化, 不需要加密文件 A 的代码; 未加密的 ELF 文件 B 实现动态指令集随机化, 使用随机生成的用户密钥重新加密文件 B 的代码。最后在装载函数 *load_elf_binary()* 中读取文件特征信息中的加密密钥, 并将其存入 PCB 的 *key* 中, 保证进程密钥与进程代码的一致性。

在上下文切换函数 *context_switch()* 调用 *switch_to()* 函数切换进程的寄存器集合的同时, HDISR 会使用 *cust5* 指令将新进程的 *key* 存入用户密钥寄存器。因此, 新进程的代码执行时, CPU 使用相应的用户密钥解密代码。需要注意的是, ELF 文件装载完成后, 如果首次调入执行的是此次装载的进程, 此时不会切换进程的上下文和进程密钥, 且 HDISR 会在 *load_elf_binary()* 函数中将进程密钥写入用户密钥寄存器。

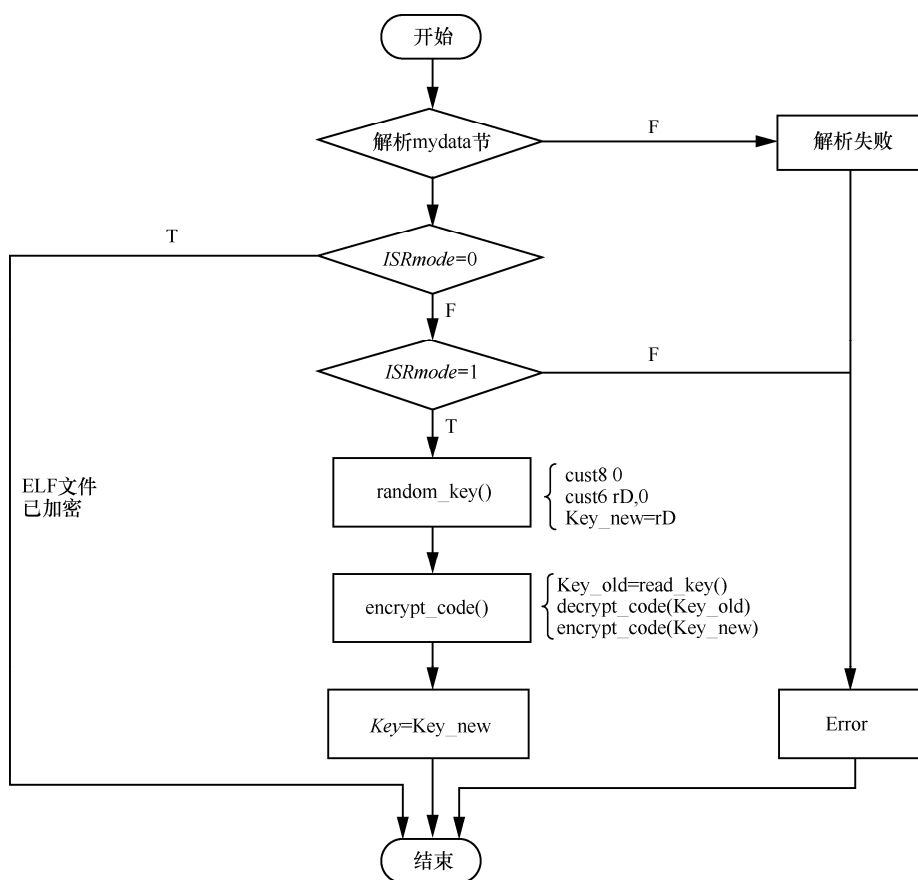


图 7 ELF 文件代码区域加密流程

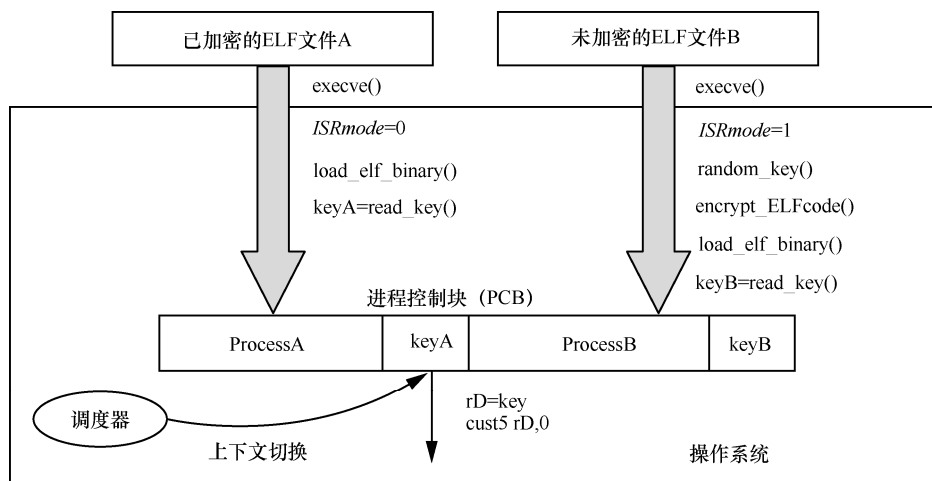


图 8 ELF 文件装载过程

6 测试

本文选取 OR1200 处理器与 Linux 3.4 版本实现了 HDISR 框架,OR1200 是基于 OpenRisc 1000 架构的 32 位 RISC 处理器。加密算法采用 32 位的异或加密,将修改后的 OR1200 处理器下载到 Xilinx atlys 开发板中的 FPGA 芯片中,并在开发板上运行支持指令集随机化的 Linux 系统。通过公开的内核漏洞与应用程序漏洞的攻击实验,验证了 HDISR 对代码注入攻击的防御效果。最后选取 SPEC 测试集测试了 HDISR 的性能损耗,实验结果表明,HDISR 能够抵御代码注入攻击,而且性能损耗较低。

6.1 防御效果测试

为了验证 HDISR 对代码注入攻击防御的有效性,本文选择了表 1 中的 5 个漏洞,其中包括 2 个内核漏洞和 3 个用户级缓冲区溢出漏洞。在原始 Linux 系统和应用程序上测试,能成功执行注入的漏洞利用代码。在 Linux 系统和应用程序经过 HDISR 保护后,注入的代码执行失败,程序异常崩溃。HDISR 会对 Linux 系统和应用程序使

用的指令集进行随机化处理,而注入的漏洞利用代码仍然依据原始指令集生成,执行错误指令集的代码会引起非法指令异常,导致程序崩溃。实验结果表明,HDISR 可以有效地阻止代码注入攻击,将代码注入攻击降级为拒绝服务攻击,对现有的内核级别和用户级别的远程代码执行漏洞具有较好的防御效果。

6.2 硬件资源损耗

本文选取 FPGA 中的 FF 寄存器和查找表资源来度量硬件损耗,以原始 OR1200 处理器综合后的资源损耗作为基准计算 HDISR 的资源损耗增加的百分比。HDISR 额外的硬件损耗如表 2 所示,额外硬件消耗均低于 2.57%,其中 FF 寄存器资源损耗增加了 2.57%,查找表资源损耗增加了 1.71%。

6.3 性能测试

选用 SPEC CPU2006 测试集^[16]来测试 HDISR 的性能损耗,包括运行时性能损耗和启动延时两方面。运行时性能损耗来源于指令翻译和密钥切换,通过静态指令集随机化运行耗时来度量。运行时性能损耗 S 的定义如下。

表 1 HDISR 防御代码注入攻击的测试集

CVE 编号	漏洞类型	漏洞描述	受影响的程序
CVE-2013-1763	本地提权漏洞	通过 sock_diag_handlers 数组下标越界取得 root 权限	Linux Kernel 3.1~3.7
CVE-2016-10229	远程代码执行漏洞	通过触发不安全的二次校验和计算执行任意代码	Linux Kernel before 4.5
CVE-2002-1549	栈溢出漏洞	通过长 HTTP GET 请求执行任意代码	Light HTTPd (lhttpd) 0.1
CVE-2002-1496	堆溢出漏洞	通过 Content-Length HTTP 头部的负值执行任意代码	Null HTTPd Server 0.5.0
CVE-2013-5019	栈溢出漏洞	通过 HTTP 请求中的长资源名执行任意代码	MINI HTTPD 1.21

表2 HDISR 额外的硬件损耗

综合后的处理器	FF 寄存器资源损耗	查找表资源损耗
OR1200	6 686	12 425
HDISR	6 858 (2.57%)	12 638 (1.71%)

$$S = \frac{T_s - T}{T}$$

其中, T 为原始程序的运行时间, T_s 为静态指令集随机化保护的程序运行时间。

HDISR 加密代码会带来程序的启动延时, 延时与程序的代码大小有关, 通过启动延时与代码大小的比值来度量代码加密效率。加密效率 L 的定义如下。

$$L = \frac{T_D - T_s}{C}$$

其中, T_D 为动态指令集随机化程序的运行时间, T_s 为静态指令集随机化程序的运行时间, C 为 ELF 文件代码的大小。

HDISR 的性能测试结果如表 3 所示, 结果表明, 平均运行时性能损耗为 0.566%, 且所有测试程序的运行时性能损耗都低于 0.9%; 程序的启动延时与二进制文件中的可执行代码的大小相关, 每兆字节的代码加密平均带来 0.31 s 的延时。

7 结束语

本文设计并实现了基于硬件的动态指令集随机化框架 HDISR, 将指令集随机化方法引入内核层和应用层的安全防护, 提高了系统安全性。内

核使用单独的内核密钥, 同时每个应用程序都有各自的密钥, 防止单个密钥丢失而导致整个系统的沦陷。HDISR 面临内存代码泄露的威胁, 由此降低了加密机制分析和密钥猜解难度, 下一步将研究限制代码可执行但不可读的方法以应对内存代码泄露风险。

参考文献:

- [1] NIST. USA national vulnerability database[EB/OL]. <http://web.nvd.nist.gov/view/vuln/statistics>.
- [2] CHEN P, WANG J, PAN L, et al. Research and implementation of SQL injection prevention method based on ISR[C]//IEEE International Conference on Computer and Communications. 2017: 1153-1156.
- [3] BARUA A, ZULKERNINE M, WELDEMARIAM K. Protecting Web browser extensions from JavaScript injection attacks[C]//The International Conference on Engineering of Complex Computer Systems. 2013: 188-197.
- [4] 汤战勇, 李光辉, 房鼎益, 等. 一种具有指令集随机化的代码虚拟化保护系统[J]. 华中科技大学学报(自然科学版), 2016, 44(3): 28-33.
- [5] TANG Z Y, LI G H, FANG D Y, et al. Code virtualized protection system with instruction set randomization[J]. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2016, 44(3): 28-33.
- [6] DANGER J L, GUILLEY S, PRADEN F. Hardware-enforced protection against Software reverse-engineering based on an instruction set encoding[C]//ACM Sigplan on Program Protection and Reverse Engineering Workshop. 2014: 5.
- [7] LIU B, WANG B. Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks[C]//Design, Automation and Test in Europe Conference and Exhibition. 2014: 1-6.
- [8] KC G S, KEROMYTIS A D, PREVELAKIS V. Countering code-injection attacks with instruction-set randomization[C]//ACM Conference on Computer and Communications Security. 2003:

表3

SPEC 测试集测试结果

测试程序	代码大小/kB	原始运行时间/s	静态指令集随机化		动态指令集随机化		S	$L/(s \cdot MB^{-1})$
			运行时间/s	增加时间/s	运行时间/s	增加时间/s		
400.perlbench	2 430.016	15.540	15.649	0.109	16.386	0.737	0.702%	0.310
401.bzip2	960.451	17.984	18.11	0.126	18.407	0.297	0.701%	0.316
429.mcf	779.939	8.288	8.344	0.056	8.575	0.231	0.672%	0.304
445.gobmk	1 659.258	92.072	92.885	0.813	93.379	0.494	0.883%	0.305
453.povray	2 114.715	1.471	1.479	0.008	2.143	0.665	0.534%	0.321
456.hmmer	3 488.832	0.350	0.351	0.002	1.421	1.069	0.473%	0.313
454.calculix	1 067.371	8.749	8.769	0.020	9.095	0.326	0.23%	0.313
471.omnetpp	2 206.734	1.301	1.305	0.004	1.941	0.635	0.33%	0.295
平均	1 838.415	18.219	18.362	0.1423	18.918	0.557	0.566%	0.310

- 272-280.
- [8] PORTOKALIDIS G, KEROMYTIS A D. Fast and practical instruction-set randomization for commodity systems[C]//Computer Security Applications Conference. 2010:41-48.
- [9] PORTOKALIDIS G, KEROMYTIS A D. Global ISR: toward a comprehensive defense against unauthorized code execution[M]//Moving Target Defense. New York: Springer, 2011: 49-76.
- [10] 王奕森, 舒辉, 谢耀滨, 等. 基于指令集随机化的代码注入型攻击防御技术[J]. 计算机应用与软件, 2016, 33(5): 312-316.
WANG Y S, SHU H, XIE Y B, et al. Defense technology against code-injection attacks based on instruction set randomization[J]. Computer Applications and Software, 2016, 33(5): 312-316.
- [11] SOVAREL A N, EVANS D, PAUL N. Where's the FEEB? the effectiveness of instruction set randomization[C]//USENIX Security Symposium. 2005: 10.
- [12] PAPADOGIANNAKIS A, LOUTSIS L, PAPAESTATHIOU V, et al. ASIST: architectural support for instruction set randomization[C]//ACM SigSAC Conference on Computer & Communications Security. 2013: 981-992.
- [13] SINHA K, KEMERLIS V P, SETHUMADHAVAN S. Reviving instruction set randomization[C]//IEEE International Symposium on Hardware Oriented Security and Trust. 2017.
- [14] 欧海文, 赵静, 于慧红, 等. 基于振荡器的真随机数发生器的研究[J]. 通信技术, 2011, 44(12):153-155.
OU H W, ZHAO J, YU H H, et al. Study on oscillator-based truly random number generator[J]. Communications Technology, 2011, 44(12): 153-155.
- [15] 郭弘, 刘钰, 党安红, 等. 物理真随机数发生器[J]. 科学通报, 2009(23):3651-3657.
GUO H, LIU Y, DANG A H, et al. Physical true random number generator[J]. Chinese Science Bulletin, 2009(23): 3651-3657.
- [16] HENNING J L. SPEC CPU2006 benchmark descriptions[J]. ACM Sigarch Computer Architecture News, 2006, 34(4): 1-17.

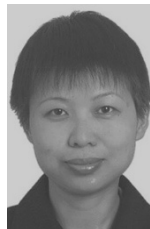
作者简介:



杜三 (1993-), 男, 四川射洪人, 信息工程大学硕士生, 主要研究方向为网络安全与嵌入式系统安全。



舒辉 (1974-), 男, 江苏盐城人, 博士, 信息工程大学教授, 主要研究方向为网络安全、嵌入式系统分析与信息安全。



康绯 (1972-), 女, 河南周口人, 硕士, 信息工程大学教授, 主要研究方向为网络信息安全。