

增强型虚拟寄存器轮转算法

潘雁, 林伟

(数学工程与先进计算国家重点实验室, 河南 郑州 450001)

摘要: 为了对抗基于数据流逆向分析的语义攻击, 以虚拟寄存器为切入点, 提出了一种增强型虚拟寄存器轮转算法。该算法通过在解释执行中随机打乱部分虚拟寄存器与操作数的虚拟编译映射关系, 有效地增加了虚拟机在解释执行过程中的数据流复杂度; 同时, 随机采用3种机制对轮转长度进行设定, 增强了虚拟机代码保护系统的多样性。最后, 设计实现了采用增强型虚拟寄存器轮转算法的虚拟机代码保护原型系统, 验证了该算法的有效性。

关键词: 虚拟机代码保护; 寄存器轮转; 数据流分析; 语义攻击

中图分类号: TP311

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2018038

Enhanced method based on virtual registers rotation

PAN Yan, LIN Wei

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China

Abstract: Semantic attacks based on the data flow analysis bring big challenges to the code obfuscation. Concerning the data flow of virtual machine based (VM-based) code protection, the method transfers the mapping relation between the virtual registers and the op-code of the bytecode during executing, which means the uncertainty and complexity of the data flow during interpretive execution of the bytecode. In addition, three policies are proposed to address the problem that how to choose the length of rotation for each bytecode, which grows complexity of the protection. Finally, a prototype of VRR-VM (virtual machine protection system based on virtual registers rotation) was implemented. Experiment results show that the method is effective and applicable for anti-reversing.

Key words: VM-based code protection, virtual registers rotation, data flow analysis, semantic attacks

1 引言

数据流分析是逆向分析中重要的方法之一, 各类数据流分析工具与方法层出不穷^[1,2], 如二进制插桩框架 Pin^[3]、动态污点传播分析技术^[4]、符号执行技术等, 用以辅助逆向分析人员进行数据流分析。其中, 在对虚拟机代码保护的逆向分析中, Sharif^[5]对被保护程序执行过程中的数据流信

息进行记录, 然后采用动态数据流分析和污点分析的方法对记录的程序轨迹进行分析, 构建控制流图等数据结构, 为进一步分析奠定基础; 黄荷洁等^[6]提出一种基于动态数据流分析的破解方法, 通过 Pin 记录代码在虚拟机执行过程中的数据流信息, 整理分析可有效还原算法的控制流图, 辅助逆向分析人员的算法重构。

在代码混淆领域, 由于在汇编语言层面数据

收稿日期: 2018-03-09; 修回日期: 2018-04-27

通信作者: 潘雁, z11pany@qq.com

基金项目: 国家重点研发计划基金资助项目 (No.2016YFB0801601, No.2016YFB0801505)

Foundation Item: The National Key R&D Plan Program of China (No.2016YFB0801601, No.2016YFB0801505)

流混淆难以操作, 因此数据流混淆主要针对源代码混淆。而在虚拟机代码保护领域, 为对抗数据流分析, 徐方华等^[7]提出解释函数跳转表的随机轮转机制, 跳转表是维持整个虚拟机解释器执行流程的重要数据结构, 虚拟机调度器通过跳转表实现虚拟机字节码和虚拟指令解释函数之间的映射, 该方法通过在每次执行一条字节码后动态乱转跳转表, 增大逆向分析的跟踪难度; Wang 等在文献[8]中提出在虚拟指令中添加寄存器旋转指令 (RRI, register rotation instruction), 其主要功能是在执行过程中通过旋转改变虚拟寄存器之间的对应关系, 其虚拟指令为 RRI operand1, operand2, 其中第 2 个操作数表明其将会旋转几个单位。该指令可以在字节码序列中任意插入, 使虚拟寄存器与字节码之间的对应关系在该特殊指令前后发生变化, 能在一定程度上增加逆向分析的难度, 但缺乏隐蔽性, 逆向分析人员识别该特殊指令后可对其他的样本进行同样的分析。

符号执行技术、污点分析技术的发展提升了数据流分析的能力, 为更好地对抗基于数据流分析的语义攻击, 本文基于跳转表、虚拟寄存器轮转的思想, 详细分析虚拟寄存器在虚拟机代码保护过程中的关键作用, 改进寄存器旋转的机制, 提出一种增强型虚拟寄存器轮转算法。

2 虚拟机代码保护框架

2.1 虚拟机代码保护技术概述

虚拟机保护的核心思想是利用自定义指令系统和现有指令系统之间的差异性, 增加逆向分析的难度。它虚拟出一套自己设计的指令系统, 用于将一系列指令放在一个解释引擎中执行。在该指令系统中, 大部分常用的汇编指令, 都可以用虚拟机中的相应指令代替^[9]。

虚拟机保护机制如图 1 所示, 具体保护步骤如下。

Step1 提取待保护程序 P 中使用软件开发工具包 (SDK, software development kit) 标识的目标代码。

Step2 将目标代码翻译为虚拟指令序列, 称为虚拟指令翻译过程。

Step3 虚拟指令经虚拟编译为字节码, 称为虚拟编译过程, 与 Step2 合称为虚拟化过程。

Step4 构造解释函数集合和解释函数跳转表。

Step5 重建可执行程序文件, 将虚拟机上下文、字节码、解释函数、调度器重新构成新节或加至最后一节, 并在目标代码处填充垃圾代码。

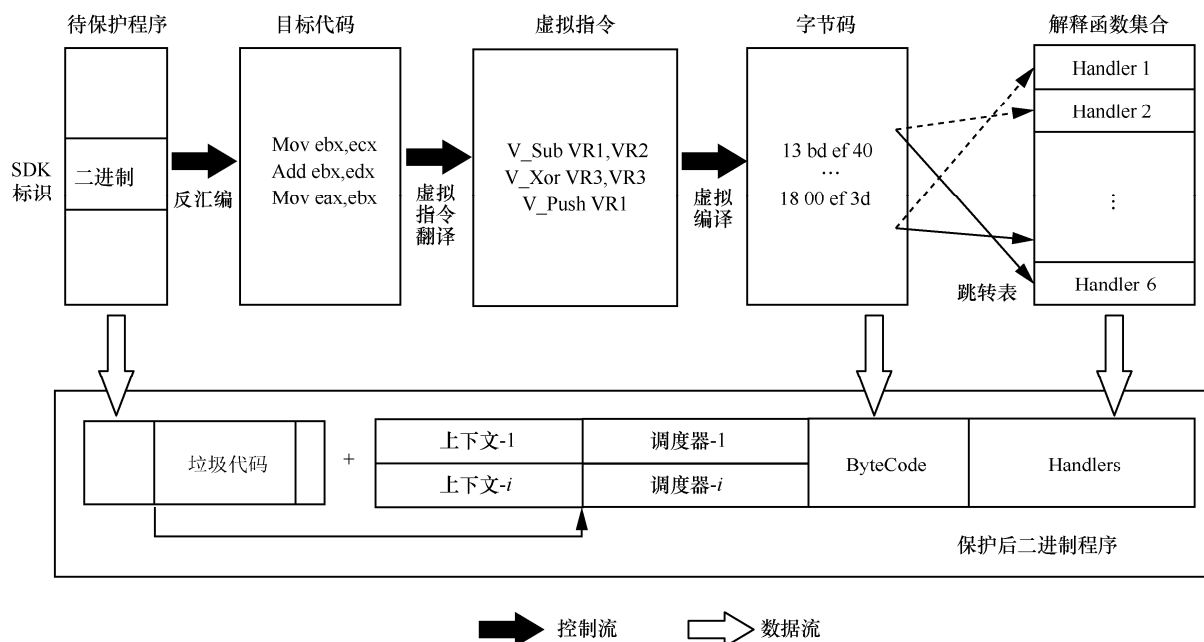


图 1 虚拟机代码保护机制

一个被虚拟机保护的程序主要由虚拟机字节码指令、虚拟机调度器、虚拟指令解释函数和虚拟机上下文组成。其中，虚拟机解释器由虚拟机调度器和虚拟指令解释函数组成。虚拟机保护通常采用中心解码结构，此外还会采用一些线性结构。在虚拟机运行过程中，虚拟机解释器先通过虚拟机调度器循环进行取指——解码操作，然后交由虚拟指令解释函数进行解释执行。

随着逆向分析技术的发展以及逆向分析者对虚拟机结构、解释函数的持续分析，虚拟机代码保护技术逐渐被攻破^[10-12]。与此同时，正向保护研究人员不断以虚拟机代码保护框架中的各个模块为对象研究新的混淆方法，并将随机化、动态多样化思想应用于虚拟机代码保护技术。

1) 解释函数：房鼎益等^[13]提出通过设计数据流混淆引擎对解释函数进行数据流混淆，增大数据流结构的复杂性；谢鑫等^[14,15]提出对所有解释函数进行变长切分和随机乱序，实现代码并行化与虚拟机多样化；Wang等^[16,17]提出指令等价替换来增强多解释函数序列间的差异性，并通过随机选择路径以抵御累积攻击。

2) 虚拟机上下文：使用多个上下文代替单个，并动态跳转，增大数据流分析难度^[18]。

3) 虚拟机调度器：使用多个虚拟机调度器，并在每一个解释函数后插入一个控制单元指令，随机决定跳转的下一个调度器，将单一的循环调度结构扩充为循环结构与链式结构混合调度结构^[19,20]。

4) 指令集随机化：每次执行保护时打乱字节码与解释函数的对应关系，使不同的保护程序中字节码语义不同，延长攻击者的分析时间^[21,22]。

总结分析研究现状，本文以对抗语义攻击为目标，将关注点放在虚拟寄存器。

2.2 虚拟寄存器

机器指令由操作码与操作数构成，操作数主要包括寄存器、内存、立即数。虚拟机中字节码与机器指令类似，组成结构如图2所示，包含操作码与操作数，其中，操作码占2 byte，操作数则对应虚拟寄存器和立即数，因此，虚拟寄存器在虚拟机代码保护过程中记录数据流转，是极为重要的组成部分。在解释执行过程中，基于堆栈

的虚拟机通过开辟内存空间的方式存储虚拟机上下文，虚拟机上下文提供虚拟执行过程中所需的缓存空间，包括虚拟栈、虚拟寄存器、密钥等，每个部分随机分布在新开辟的内存空间。以本文研究的对象——虚拟寄存器为例，若干个虚拟寄存器顺序存储于内存。其中，每个虚拟寄存器对应的数字代表其在虚拟编译过程中的字节码，即其在内存中的偏移，称该对应关系为虚拟编译映射。例如，VR1→6，表示编译过程中VR1被虚拟编译为二进制0x06，该映射关系在每次混淆过程中都是随机生成的。在解释执行时，解释函数会依据每条字节码的操作数，根据偏移到内存中操作相应的虚拟寄存器，称此对应关系为解释执行映射。

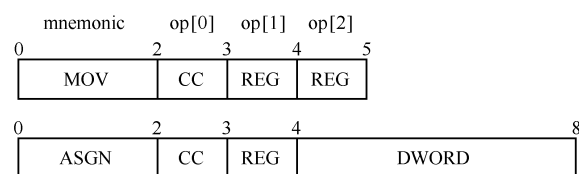


图2 VRR-VM 虚拟指令实例

3 轮转算法实现

寄存器轮转算法的基本原理如图3所示，由于虚拟寄存器在内存中是线性存储的，因此轮转算法需要在逻辑上模拟环形旋转，虚拟机在解释执行字节码时需要依据当前状态下的对应关系操作相应的内存，而轮转算法使对应关系在运行过程中是动态的。原始的对应关系如图3左侧，经左旋一个单位后，对应关系如右侧，对于虚拟寄存器VR3，在旋转之前，其对应的字节码为0，即在内存中的偏移的0*32，而经过旋转后其对应的字节码为1，即其在内存中的偏移为1*32。

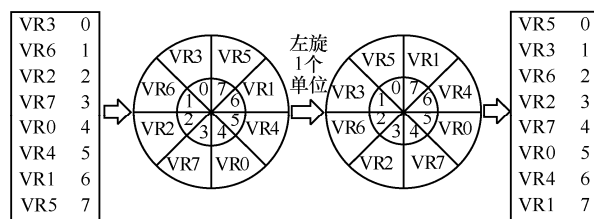


图3 寄存器轮转原理示意

依据寄存器轮转混淆算法的原理，该算法需在虚拟编译过程与解释执行过程分别对虚拟编译

映射和虚拟机上下文进行调整,以保证程序的正确执行。特别地,在解释执行过程中解释执行映射关系是不变的。在虚拟编译过程中,对于每一类虚拟指令都采用轮转策略,随机轮转 n 个单位长度,同时实时更新每条指令对应的虚拟编译映射关系;在解释执行过程中,每一类虚拟指令对应的解释函数执行时,伴随着内存的轮转操作,其轮转的单位长度同虚拟编译时相同。

而在逻辑上,字节码的生成过程是静态分析的过程,而在字节码的解释执行过程中存在跳转、循环等动态执行过程,因此,若对所有的虚拟寄存器进行旋转,其在静态分析过程中形成的旋转顺序,在动态执行中会被打乱。因此,虚拟寄存器轮转是相对的,在解释执行过程中必须要存在不变的虚拟寄存器,如图 4 所示。因此,对于每条字节码来说,虚拟寄存器是轮转的;对被保护的 X86 指令来说,虚拟寄存器是不变的,否则程序是无法正确执行的。而从逆向分析者的角度,由于每条 X86 指令都被虚拟化为多条虚拟指令,因此,从虚拟指令角度看,X86 指令是不存在分界线的。所以,对于逆向者虚拟寄存器始终是轮转的。

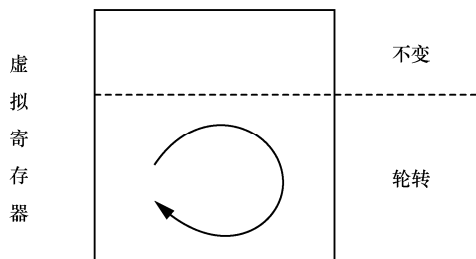


图 4 寄存器轮转示意

综上,虚拟寄存器轮转算法如下所示。

1) 虚拟编译过程轮转算法

- ① 初始化虚拟编译映射关系 `reg_opcode[VMR_NUM]`;
- ② 选取轮转的虚拟寄存器数目 `VMR_ROTATE_NUM`;
- ③ for 每条虚拟指令
- ④ 将 `reg_opcode[VMR_ROTATE_NUM]` 旋转 n 个单位;
- ⑤ 虚拟指令编译为字节码;
- ⑥ end for

⑦ 得到字节码序列

2) 解释执行过程轮转算法

- ① for 每条虚拟指令
- ② 将内存旋转 n 个单位;
- ③ 解释执行当前字节码;
- ④ end for
- ⑤ 完成解释执行

而对于每条指令轮转的单位长度,本算法提供 3 种机制进行随机选取,如下。

① 每一类虚拟指令拥有独立的轮转单位长度,在虚拟机初始化时,每一类虚拟指令随机设定固定值,而在解释执行时需将其硬编码至对应的解释函数。

② 每一类虚拟指令拥有独立的轮转单位长度,其轮转单位长度采用该类虚拟指令字节码的变换值,优点是轮转单位长度无需硬编码,同时保证随机性。

③ 每一条指令拥有独立的轮转单位长度,可以使用其操作码、操作数为基进行变换以获取其轮转的单位长度,优点是相较于前 2 种机制都更为复杂。

对于不同次混淆,其轮转单位长度的选取都是随机的;而对于同一种机制,每一类或每一条虚拟指令都会进行随机的轮转;而对于虚拟寄存器的轮转,在解释执行过程中,为保证内存中寄存器与编码的正确对应关系,解释函数的每次执行都伴随着内存中寄存器的旋转,增加了数据流的复杂性。

4 实验及分析

4.1 理论分析

在抗逆向分析方面,虚拟寄存器随机轮转算法较为隐蔽地打乱虚拟寄存器与操作数的虚拟编译映射关系,增加了数据流向的复杂性,混淆了逆向分析者对虚拟指令操作数的识别;同时轮转长度选取机制的随机性增强了虚拟机代码保护系统的多样性。

具体来说,假定寄存器初始化布局如图 3 左侧,某明文字节码序列如表 1 第一列所示,其中,字节码 A2 标识的虚拟指令为 `V_Pop`, 2D 标识的虚拟指令为 `V_Mov`, 对应寄存器的初始化布局,

第一条指令的第一操作数 03 对应的寄存器为 VR7, 由于每执行一条字节码, 寄存器对应关系将逆时针旋转 n 个单位, 类似于复式替换密码。假定该条虚拟指令执行时旋转了 3 个单位, 因此下一条指令的第一操作数 01 对应 VR1, 第二操作数对应 VR7, 因此其对应的助记符指令序列正确识别应为第三列, 而逆向分析者在未破解寄存器轮转之前会错误识别为第二列, 其实际意义将会被完全曲解。

即使逆向分析者发现了寄存器轮转这一策略, 解释函数在连续执行的过程中随机轮转虚拟寄存器, 充满了不确定性, 导致分析人员不容易分析清代码究竟访问的是哪一个寄存器。因此, 分析者静态还原虚拟机是比较困难的, 而动态调试过程中只要出现一个偏差, 就会导致后续的指令分析错误。

表 1 字节码与助记符指令序列对应关系

明文字节码 序列	助记符指令序列 差错识别	助记符指令序列 正确识别
A2 03	V_Pop VR7	V_Pop VR7
2D 01 06	V_Mov VR6,VR1	V_Mov VR1,VR7

在时间与空间开销方面, 在构成的虚拟机代码保护系统中, 该算法在解释函数中增添了一个内存旋转单元, 在控制旋转长度的前提下, 其带来的多余空间开销相较整个系统较小, 而由于内存旋转单元将对内存进行旋转, 所以带来的时间开销将由平均旋转长度决定。

4.2 实验验证

本文实验环境为 Windows 7 操作系统, Intel(R) Core i7-6700 CPU @ 3.40 GHz 处理器, 32 GB 内存, 编译环境为 Visual Studio 2012, 采用的实例为 Validity.exe, 结合源码以及逆向分析工具为 IDA 与 OllyDbg 来验证虚拟寄存器随机轮转算法的有效性。

实验 1 有效性分析

对保护代码中的第一条指令 xor eax, eax 进行跟踪。在虚拟化过程中, 目标指令经由虚拟指令翻译得到 11 条虚拟指令, 而后通过虚拟编译生成对应的字节序列, 如图 5 所示, 其中, 第 7 条虚拟指令处理了真实的 xor 功能, 其余虚拟指令用以实现对操作数和符号的处理。

容易发现, 在虚拟指令中第 2 条与第 7 条虚拟指令的目标操作数同为虚拟寄存器 VR41, 而在虚拟编译过程中, 第 2 条的 VR41 编译为 14, 而第 7 条的 VR41 编译为 0C, 在未采用寄存器轮转算法的虚拟机代码保护系统中, 两条语句的 VR41 将被编译为相同的二进制。相同地, 第 3 条与第 7 条虚拟指令也存在相同的情况。逆向分析者在仅获取字节码的前提下, 必须分析虚拟寄存器的轮转策略, 以获取虚拟指令的语义, 否则差之毫厘, 谬以千里。

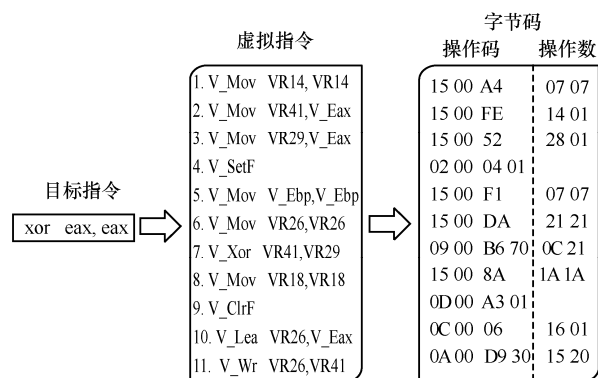


图 5 X86 指令到字节码的转换

在解释执行过程中, 为对应虚拟编译过程中的轮转策略, 需对内存进行旋转。虚拟机寄存器在内存中的分布如图 6 所示, 前 11 个 DWORD 是在执行过程中不进行旋转操作的寄存器, 后 32 个 DWORD 是进行旋转操作的寄存器。由于第一条虚拟指令是垃圾指令, 因此从第二条指令开始分析。第二条指令执行结束, 偏移位 20 处被赋值, 在下一条指令执行之前内存已被修正, 向左旋转了一个单位, 效果如图 7 所示, 若内存中数值较多, 会极大地影响数据的分析。同时, 由于轮转长度的不确定性, 逆向分析人员需跟踪每条虚拟指令, 若想通过自动化工具进行分析, 则必须完全理解轮转的机制, 包括轮转的算法与长度选择的机制。

00033F50	06 02 00 00	42 38 72 5C	00 E0 FD 7E	00 00 00 00
00033F60	01 00 00 00	E0 FC 41 00	00 00 00 00	E0 FC 41 00
00033F70	04 FC 41 00	90 90 2D 01	00 00 00 00	00 00 00 00
00033F80	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00033F90	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00033FA0	42 38 72 5C	00 00 00 00	00 00 00 00	00 00 00 00
00033FB0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00033FC0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00033FD0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00033FE0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00033FF0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

图 6 虚拟寄存器在内存中的分布

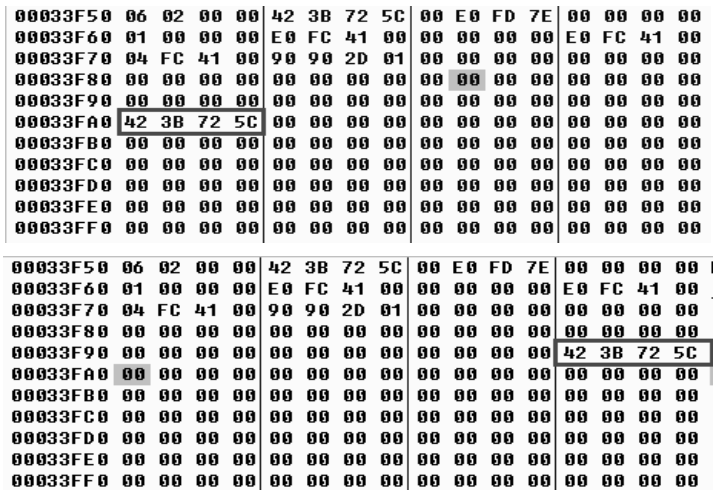


图 7 虚拟寄存器轮转效果

实验 2 算法带来的额外时间与空间开销

为验证算法带来的时间与空间开销,本节将采用该算法与未采用该算法虚拟机代码保护系统进行比较,其中,轮转算法的平均轮转长度为 2,测试用例采用有效性程序 Validify.exe、希尔排序 ShellSort.exe、插入排序 InsertionSort.exe、冒泡排序 BubbleSort.exe 和快速排序 QuickSort.exe。

表 2 是 2 种策略保护前后文件大小变化,表 3 是目标代码执行时间开销比较,可见该算法增加的空间开销极小,但时间开销增长较大,且与平均轮转长度正相关,如图 8 所示。

表 2 有无轮转算法策略保护前后文件大小变化

被保护软件	保护前 文件大小/kB	保护后文件大小/kB	
		无轮转算法	有轮转算法
ShellSort.exe	19	34	36
InsertionSort.exe	19	33	35
BubbleSort.exe	19	33	36
QuickSort.exe	19	34	36

表 3 有无轮转算法策略保护前后目标代码执行时间变化

被保护软件	保护前目 标代码执 行时间/ms	保护后目标代码执行时间/ms		增长 比例
		无轮转 算法	有轮转 算法	
ShellSort.exe	0.39	611	1 391	1.28
InsertionSort.exe	1.51	3 457	8 525	1.47
BubbleSort.exe	3.84	93 456	20 480	1.19
QuickSort.exe	0.37	339	726	1.14

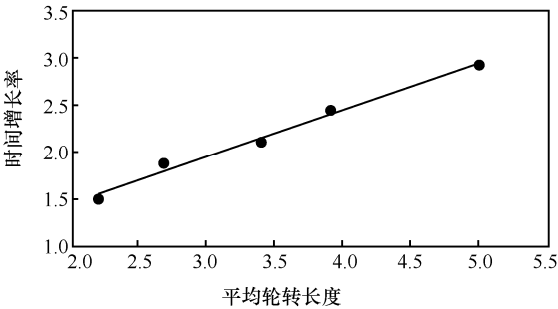


图 8 时间增长率与平均轮转长度相关性

实验 3 原型系统与商业软件的对比情况分析

该实验从时间和文件大小 2 个角度来对比分析原型系统与商业软件的性能,其中,CV (code virtualizer) 版本号为 2.2.1.0,使用的虚拟机类型为 Tiger32 White; VMP (virtual machine protection) 版本号为 2.13.8,采用最快速度策略进行虚拟机保护。测试用例选用 4 类排序程序,如表 4 所示,其中,目标代码指令数为静态反汇编得到的指令数;排序的输入为 500 位随机数;同时,为消除计算机产生的时间误差,执行时间为将目标代码循环执行 10 次的结果。

表 4 测试用例描述

测试程序	目标代码指令数	Key-Code 执行时间/ms
ShellSort.exe	58	0.39
InsertionSort.exe	38	1.51
BubbleSort.exe	41	3.84
QuickSort.exe	58	0.37

表 5 为保护前后文件大小变化,表 6 为保护前后目标代码执行时间变化,其中,执行时间与

指令膨胀率正相关，由于每次执行时间都有所不同，图表中执行时间为10次执行时间的平均值。为了更加直观地比较不同方法之间的差别，将数据进行一定的处理后做出下列比较，如图9和图10所示，其中，将CV保护的文件大小缩小5倍，图10中采用的数据是3种方法与保护前运行时间的比例。由图表可知，原型系统时间与文件的增加比例与商业软件VMProtect接近，而相较于CV则有文件开销上的优势。

表5 保护前后文件大小变化

被保护软件	保护前 文件大小/kB	保护后文件大小/kB		
		CV2.2.1.0	VMP2.13.8	X86VM
ShellSort.exe	19	725	31	36
InsertionSort.exe	19	727	30	35
BubbleSort.exe	19	709	30	36
QuickSort.exe	19	722	30	36

表6 保护前后目标代码执行时间变化

被保护软件	保护前目标 代码执行 时间/ms	保护后目标代码执行时间/ms		
		CV2.2.1.0	VMP2.13.8	X86VM
ShellSort.exe	0.39	232	1 076	1 391
InsertionSort.exe	1.51	1 175	7 978	8 525
BubbleSort.exe	3.84	3 795	18 940	20 480
QuickSort.exe	0.37	140	932	726

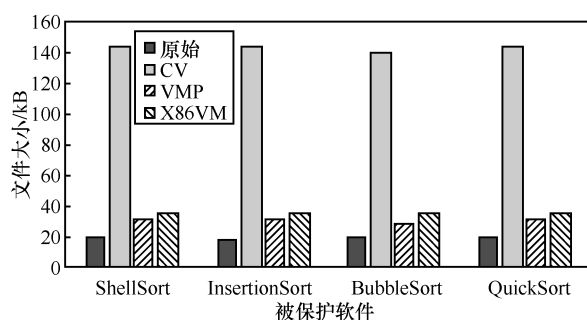


图9 文件开销比较

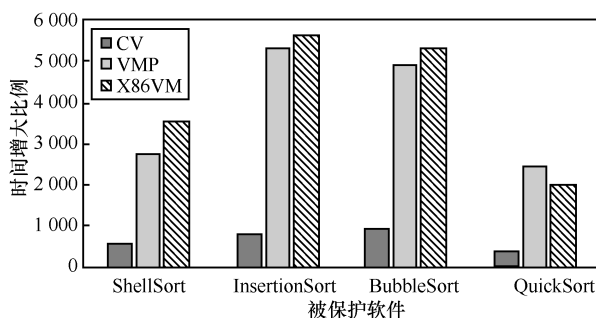


图10 时间开销比较

5 结束语

恶意的逆向分析给软件保护带来严峻的挑战，代码混淆与虚拟机代码保护技术能对软件中的代码与数据进行保护，增加程序的控制流与数据流的复杂度，以对抗逆向分析。本文关注虚拟机代码保护框架中的数据流，以虚拟寄存器为着力点，通过在虚拟机解释执行过程中变换虚拟寄存器与字节码中操作数的对应关系，增大字节码在执行过程中的不确定性，有效增加了虚拟机在解释执行过程中的数据流复杂度；与此同时，随机采用3种机制对轮转长度进行设定，增强了虚拟机代码保护系统的多样性。最后，设计并实现了基于虚拟寄存器轮转的虚拟机代码保护原型系统，通过有效性用例验证了算法的有效性与可行性，并借助性能测试用例对比了其相对于不添加虚拟寄存器轮转的原型系统的时间与空间开销，通过控制平均轮转长度能将时间开销控制在一定比例之内；对比商业软件则具有相近的时间与空间开销。

以该算法为基础，今后可研究的方向包括：

- 1) 轮转策略具有一定的改进空间，可以采取更复杂的轮转策略替代较为简单的旋转；
- 2) 由于虚拟寄存器是存放在新的内存空间，因此可扩展至对虚拟机上下文轮转，具有更强的混淆效果；
- 3) 本文的原型系统是堆栈机，相较而言，基于寄存器的虚拟机具有更复杂的数据流信息，若将寄存器轮转算法应用于基于寄存器的虚拟机，可能将会拥有更好的混淆效果。

参考文献：

- [1] NEWSOME J, SONG D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[J]. Chinese Journal of Engineering Mathematics, 2005, 29(5):720-724.
- [2] 徐欣. 动态数据流分析技术在恶意软件分析中的应用研究[D]. 合肥: 中国科学技术大学, 2016.
XU X. Research of dynamic data flow analysis technology application in malware analysis[D]. Hefei: University of Science and Technology of China, 2016.
- [3] REDDI, JANAPA V, ALEX, et al. PIN: a binary instrumentation tool for computer architecture research and education[C]//The Workshop on Computer Architecture Education, 2004:22.
- [4] CHOW J, PFAFF B, GARFINKEL T, et al. Understanding data

- lifetime via whole system simulation[C]//Usenix Security Symposium, 2004:321--336.
- [5] SHARIF M, LANZI A, GIFFIN J, et al. Automatic reverse engineering of malware emulators[C]//2009 30th IEEE Symposium on Security and Privacy. 2009: 94-109.
- [6] 黄荷洁, 康绯, 舒辉, 等. 基于动态数据流分析的虚拟机保护破解技术[J]. 计算机工程, 2014, 40(9):59-65.
- HUANG H J, KANG F, SHU H, et al. Reverse technology of virtual machine protection based on dynamic dataflow analysis[J]. Computer Engineering, 2014, 40(9):59-65.
- [7] 徐方华. 基于虚拟堆的虚拟保护技术的研究[D]. 昆明:云南大学, 2013.
- XU F H. Research on virtual protection technology based on virtual heap[D]. Kunming: Yunnan University, 2013.
- [8] WANG H, FANG D, LI G, et al. NISLVM: improved virtual machine-based software protection[C]//The Ninth International Conference on Computational Intelligence and Security. 2013:479-483.
- [9] 舒柏程, 李毅超, 曹跃. 基于虚拟机的软件保护技术研究[J]. 计算机工程与科学, 2008, 30(A1): 25-28.
- SHU B C, LI Y C, CAO Y. Research on software protection based on virtual machine[J]. Computer Engineering & Science, 2008, 30(A1): 25-28.
- [10] GHOSH S, HISER J, DAVIDSON J W. Replacement attacks against VM-protected applications[C]//ACM Sigplan/sigops Conference on Virtual Execution Environments. 2012:203-214.
- [11] COOGAN K P. Deobfuscation of packed and virtualization- obfuscation protected binaries[D]. Arizona: University of Arizona, 2011.
- [12] COOGAN K, LU G, DEBRAY S. Deobfuscation of virtualization-obfuscated software: a semantics-based approach[C]//ACM Conference on Computer and Communications Security. 2011: 275-284.
- [13] 房鼎益, 张恒, 汤战勇, 等. 一种抗语义攻击的虚拟化软件保护方法[J]. 四川大学学报(工程科学版), 2017, 49(1):159-168.
- FANG D Y, ZHANG H, TANG Z Y, et al. DAS-VMP: a virtual machine-based software protection method for defending against semantic attacks[J]. Journal of Sichuan University (Advanced Engineering Sciences), 2017, 49(1): 159-168.
- [14] 谢鑫, 刘粉林, 芦斌, 等. Handler混淆增强的虚拟机保护方法[J]. 计算机工程与应用, 2016, 52(15):146-152.
- XIE X, LIU F L, LU B, et al. Virtual machine protection based on Handler obfuscation enhancement. Computer Engineering and Applications, 2016, 52(15):146-152.
- [15] 谢鑫, 刘粉林, 芦斌, 等. 一种基于代码并行化和虚拟机多样化的软件保护方法[J]. 小型微型计算机系统, 2015, 36(11): 2588-2593.
- XIE X, LIU F L, LU B, et al. Software protection scheme based on code parallelization and virtual machine diversity[J]. Journal of Chinese Computer Systems, 2015, 36(11):2588-2593.
- [16] 房鼎益, 赵媛, 王怀军, 等. 一种具有时间多样性的虚拟机软件保护方法[J]. 软件学报, 2015, 26(6):1322-1339.
- FANG D Y, ZHAO Y, WANG H J, et al. Software protection based on virtual machine with time diversity[J]. Journal of Software, 2015, 26(6): 1322-1339.
- [17] WANG H J, FANG D Y, LI G, et al. TDVMP: Improved virtual machine-based software protection with time diversity[C]//ACM Sigplan on Program Protection and Reverse Engineering Workshop. 2014:1-9.
- [18] WANG H J, FANG D Y, LI G, et al. NISLVM: improved virtual machine-based software protection[C]//The Ninth International Conference on Computational Intelligence and Security. 2013:479-483.
- [19] KUANG K, TANG Z, GONG X, et al. Exploiting dynamic scheduling for vm-based code obfuscation[C]//IEEE Trustcom/bigdata/isp. 2017:489-496.
- [20] KUANG K, TANG Z, GONG X, et al. Enhance virtual-machine-based code obfuscation security through dynamic bytecode scheduling[J]. Computers & Security, 2018, 74: 202-220.
- [21] TANG Z, LI G, FANG D, et al. Code virtualized protection system with instruction set randomization[J]. Journal of Huazhong University of Science & Technology, 2016.
- [22] XU J F, ZHANG W, SUN B. Research on software protection based on virtual machine[J]. Journal of China Universities of Posts & Telecommunications, 2012, 19(S1):122-126.

[作者简介]



潘雁 (1995-), 男, 安徽安庆人, 信息工程大学硕士生, 主要研究方向为网络与信息安全、软件保护与逆向。



林伟 (1986-), 男, 湖南常德人, 博士, 信息工程大学讲师, 主要研究方向为软件保护与分析、网络安全。