白盒环境中防动态攻击的软件保护方法研究

王怀军,房鼎益,董 浩,陈晓江,汤战勇
(西北大学信息科学与技术学院,西北大学-爱迪德信息安全联合实验室,陕西西安 710127)

摘 要: 运行态软件常常面临着核心算法被逆向和机密信息被泄漏的严峻威胁,急需研究有效的防动态攻击的软件保护方法.本文包含两方面研究内容:(1)对现有防动态攻击的软件保护方法进行深入分析和综合比较,针对当前主流的防动态攻击的四类保护方法,从实现难度,性能影响和安全性三个指标进行综合对比;(2)通过实例介绍我们在防动态攻击的软件保护方面两项研究工作.分别研究了基于变形引擎的动态软件保护方法和一种安全性增强的虚拟机软件保护方法.

关键词: 白盒环境; 防动态攻击; 软件保护; 虚拟机软件保护;
中图分类号: TP309.7 **文献标识码:** A **文章编号:** 0372-2112 (2014)03-0529-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2014.03.017

Research on Software Protection Defending Dynamic Attack in White-Box Environment

WANG Huai-jun, FANG Ding-yi, DONG Hao, CHEN Xiao-jiang, TANG Zhan-yong
(*NWU-Irdeto Network-Information Security Joint Laboratory (NISL), Department of Information Science, Northwest University, Xi'an, Shaanxi* 710127, *China*)

Abstract: Running software often faces serious threats, for example, the core algorithm is reversed or confidential information is leaked. So, it is needed to study effective protection methods against dynamic attack. The paper includes two aspects. First, do in-depth analysis and comprehensive comparison of the existing method of software protection against dynamic attacks. According to four mainstream software protection methods against anti-dynamic attack, we do comprehensive comparison of these four protection methods from three indicators of implementation difficulty, performance impact and security. Second, Introduce the work of our two studies in software protection against anti-dynamic attacks through examples. They are a software protection method based on dynamic deformation engine and a security-enhanced software protection based on virtual machine.

Key words: white-box attack environment; anti-dynamic attack; software protection; virtual machine based software protection

1 引言

软件应用已渗透到各个领域,但其自身面临严峻威胁.程序理解、代码分析、动态调试、运行跟踪等静态和动态软件分析方法常被黑客利用,破解技术和工具也随手可得.攻击者逆向软件核心算法或篡改软件版权信息,并通过网络、复制拷贝、共享等方式扩散.这对开发者和使用者都会造成严重危害:①对开发者而言,软件是一种知识密集型数字产品,凝聚着智慧和汗水,其安全性不仅关系到开发商的知识产权,更关系到软件行业市场竞争的公平性和合理性,影响软件产业健康发展;②对使用者而言,软件承载用户的隐私和机密信息,其安全性关系到信息的完整性和机密性,若被泄露、篡改,后果不堪设想.

随着软件攻击工具和攻击技术发展,软件面临的白盒环境[1](white-box context)给软件带来更大威胁,攻击者对软件执行过程完全可见,给予充分时间,一定会成功逆向.且白盒环境中保护者在与攻击者"博弈"中处于劣势地位:①保护者只能预测可能的攻击技术或工具,而攻击者可以肆意对软件进行攻击分析;②攻击者成功攻击后,立即就可应用软件核心技术,而保护者只能依据攻击影响的扩散察觉到保护方法已失效,具有延迟

收稿日期:2013-04-01;修回日期:2013-10-20;责任编辑:孙瑶
基金项目:教育部科学技术研究重点项目(No.21181);教育部博士点基金(No.20106101110018);国家科技支撑计划课题(No.2013BAK02B02);国家自科学基金(No.61070176, No.61170218, No.61272461, No.61202393);陕西省科技攻关(No.2011K06-07, No.2012K06-17);陕西省科技计划(No.2011K06-09);陕西省教育厅产业化培育项目(No.2011jg06);陕西省自然科学基础研究计划(No.2012JQ8049)

性.Matias 对白盒环境中软件攻击方法进行分类[2]:①静态攻击,未执行程序时的攻击分析;②动态攻击,执行程序时的攻击分析;③混合攻击,即有执行程序时,也有未执行程序时的攻击分析.目前,防静态攻击技术已较成熟,如加密等.软件面临主要威胁来自动态攻击,因此白盒环境中防动态攻击的软件保护方法成为研究热点.

2 防动态攻击的软件保护方法

针对软件所处的白盒环境,结合文献[3]提出白盒环境中的安全模式,下面介绍四类主流的防动态攻击的软件保护方法.

2.1 基于指令动态映射的软件保护方法

该方法即指在程序执行过程中,运行中的指令动态变化.主要基于自修改代码原理,利用动态加解密、代码伪装等技术保护软件.2008 年,Dube[4]对软件防逆向工程的方法进行总结,认为基于变形的动态保护是一种比较有效的保护手段.本节主要介绍该方法中的动态加解密和代码伪装两种技术.

2.1.1 动态加解密

Collberg 的 OBFCKSP 算法[5]对动态加解密技术进行描述,在调用函数之前对该函数解密,执行后再加密.显然,程序在执行过程中只有一块加密代码被解密成明文.

文献[6]中详细介绍动态加解密原理并给出保护实例.但保护后程序在执行时一定会被还原,攻击者可基于内存拷贝[7]等方法进行攻击.因此,抗动态攻击的能力较弱.

2.1.2 代码伪装技术

代码伪装主要使用自修改代码技术,在程序执行过程中,动态修改指令.该思想源于 Kanzaki[8,9]提出的指令伪装技术,文中分析了大量伪装代码增加了动态分析的复杂度,2010 年进行改进[10],根据代码段执行时间阈值判定程序是否被攻击,提高程序反调试能力.Madou 提出基于代码变形的软件动态保护方法[11],抽取程序中相似的代码集成束,束中的代码块通过 Script 被依次还原.相对于 Kanzaki 的方法,虽然该方法对软件中代码大量分析,但具有较高安全强度.

针对该保护方法,Wu[12]利用修改代码段页的属性为"只读"的思想定位程序中所有自修改代码;Dux[13]则利用改进的控制流图可视化描述自修改代码位置.但还原所有被伪装代码,攻击者依然需要耗费大量时间和精力.

2.2 基于混淆的软件保护方法

Collberg 最早对混淆进行定义[14]:T 表示从原代码 P 到目标代码 P' 的映射,在此过程中,P 与 P' 保持语义等价.在运行过程中,如果 P 无法中止或者以错误的状态中止,则 P' 可以中止也可以不中止;否则,P' 必须中止,且与 P 有相同输出.Barak 分析了混淆的适用性[15],证明不是所有程序都可利用基于混淆的保护方法.本节主要介绍三种在抗动态攻击方面有明显效果的混淆方法.

2.2.1 白盒加密

主要思想是对现有一个分组密码选定密钥,将明文和密文之间的映射关系进行置乱编码运算,用查找表的方式表示置乱后的映射结果,仅从查找表中分析,攻击者无法获取密钥信息,即将白盒密码的加解密过程转换为一系列表格查找过程.

白盒加密研究热潮源于 2003 年 Chow 分别基于 DES[1]和 AES[16]提出的白盒加密算法.针对其中 DES 白盒加密,Jacob 提出了注入故障攻击[17];而针对 Chow 的 AES 白盒加密,Billet 提出 BGE 攻击[18],可在时间复杂度最多为 2^{30}时间内得到密钥.随后,Link 对白盒 DES 算法[19]进行改进,改变白盒 DES 的设计结构,加入混淆信息,然后将结果用查找表表示,可抵抗 Jacob 注入故障攻击,不能抵抗 Wyseur[20]利用混淆轮转的差异密码分析法的攻击.此外,Bringer[21]和 Xiao[22]也提出了白盒加密的改进方法.

当然,现有白盒加密算法都存在一定问题,其安全性主要与实现的结构有关.因此,针对白盒加密实现的结构仍需持续研究和改进.

2.2.2 控制流混淆

控制流代表程序在逻辑上的相关性,清晰的控制流有助于攻击者理解和分析.2007 年,Birrer 提出了基于程序切片的变形保护方法[23],对程序进行切片,随机散布到整个程序,在执行时,通过调用切片管理函数或者切片地址跳转表进行跳转控制.且在执行过程中,切片顺序动态变化,混淆攻击者分析.

控制流混淆一定程度上增加攻击者动态攻击难度.对于该保护方法,一般攻击过程:在程序执行过程中动态提取程序控制流图,分析管理函数或跳转表.

2.2.3 数据混淆

动态分析中,数据对理解程序起着很重要的作用.如攻击者逆向 Code Virtualizer[24]保护的软件时,通过监视栈和堆中数据分析 Code Virtualizer 解释过程.数据混淆的对象是程序执行中的数据,包括简单类型和复杂类型的常量和变量.Collberg 提出了数据结构的混淆[25].Anckaert 研究的数据位置混淆[26]和基于数据流平展的数据流保护[27],利用 MMU(Memory Management Unit),在程序执行时隐藏数据在内存中实际存储地址,增加静态数据流分析和动态数据流跟踪的复杂度,增加攻击者利用数据分析难度.数据同态加密[28]也是一

种有效的数据保护技术.对于数据混淆的攻击一般基于数据的处理方法进行等价还原.

2.3 基于“代码隔离”的软件保护方法

T Maude[29]等人于 1984 年提出一种可选择的隐藏程序解释执行过程的方法,从目标程序中“切分”出关键代码片段,通过基于硬件的方式隐藏程序解释执行过程.而当前利用软件实现“分离保护”的方法主要有两种方式.

2.3.1 基于服务器的“代码隔离”

即将“切”出的关键代码置于服务器中执行,攻击者只能根据关键代码部分的输入和输出进行分析.2009 年,Herzberg 设计了 WBRPE(White Box Remote Program Execution)白盒远程程序执行架构,程序的执行过程依赖于本地和远程服务器[30].2010 年他对 WBRPE 架构进行了改进[31],利用混淆器对传至服务器的代码或数据进行混淆,进一步提高程序执行时的安全性.这种方法具有较强安全性,但这种方法对软件性能造成较大影响,也限制了软件的应用环境.

攻击该方法保护后的软件,需要利用黑盒攻击方法,通过大量输入输出进行测试实验,并对数据进行分析.

2.3.2 基于虚拟执行环境的“代码隔离”

该方法也是当前的研究热点,它将目标软件中被保护代码译成虚拟指令,而虚拟指令需要在虚拟解释器中执行.目前流行的商业产品代表:Code Virtualizer[24],Themida[32]和 VMProtect[33]等在工业界已得到广泛应用.如图 1 为 VMProtect 基本原理示意图.

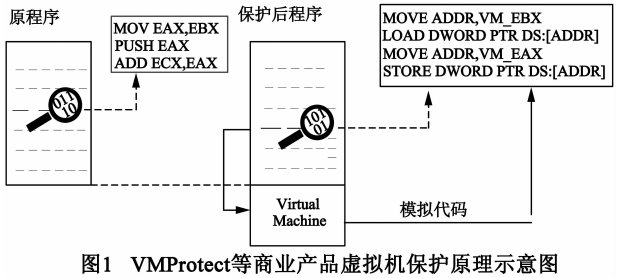

图1 VMProtect等商业产品虚拟机保护原理示意图

ASProtect 是一种商业加壳保护软件[34],其中设计了多个虚拟机分别用于保护关键代码、stolen code 和 ASProtect 保护的输入表部分.Fang 对传统的虚拟机进行了改进[35],提出多级分层保护思想,关键代码段使用不用的虚拟机解释器嵌套保护,攻击者必须分析每一层虚拟机解释器才能成功得到关键代码. 虚拟机软件保护[36]提高了软件保护力度,但由于对性能的影响,仅适用于少量代码的保护.

Sharif[37]和 Rolles[38]的方法对攻击虚拟执行环境保护有一定效果.一般攻击流程是:首先逆向虚拟机解释器,然后识别并提取虚拟执行环境中处理函数执行序列,分析执行序列,还原被保护指令的逻辑含义.

2.4 基于校验的防篡改软件保护方法

篡改是攻击者常用的攻击手段,通常有两个目的[39]:①绕过访问控制机制非授权访问软件;②嵌入恶意代码或修改删除部分代码破坏代码完整性.针对攻击者动态分析软件时的篡改行为,基于校验的防篡改技术是检测是否被篡改的有效手段.

完整性检测手段[40]主要有校验和、hash 值和数字签名等方法.响应方式可分为隐式和显式两种方式,显式响应如程序崩溃,死机或关机等.隐式如诱导攻击者进入设计的假流程、自恢复或通过多执行流程的“表决”确定执行结果等.

Anckaert 提出基于备份代码段随机执行的防篡改的方法[41].Li 提出基于软件行为自校验的防篡改保护方法[42],利用 hash 函数得到程序的行为指纹,利用行为模型确定行为特征.Hoi 等人提出 guard 保护软件的思想[43],在多个位置设置 guard,利用哨兵(guard)(即一段代码)保护程序.Dedi′c 则利用图博弈模型[44],在检测到攻击者时,推测攻击者意图,诱使攻击者学习并参与“图形”游戏,这种启发式推测及引导还需完善.Abadi 对程序执行时的流程完整性进行校验[45],判断流程是否完整或被修改.由于破解、注入或恶意代码会修改运行时进程,Gilbert 设计的 DYMO[46],可以跟踪代码运行时完整性,监控代码是否被篡改.

针对该方法保护后软件,攻击者需要分析校验检测方法,对于显式响应机制,容易定位响应位置;对于隐式响应机制,需要通过代码分析区分程序功能和响应指令.

2.5 进一步分析

综上所述,不难看出这些保护方法针对攻击者的动态分析起着一定的防御作用.下面基于攻击经验和实验,从三个方面对上述四类保护方法进行总结分析:①实现难度,即设计这种保护方法的难度,一般情况下,实现难度越大的保护技术的抗攻击性越好;②性能影响,即利用该方法对软件保护之后,对软件性能造成的影响;③安全性,即被该方法保护后,软件的安全性提高的程度.每一个方面的评估都使用三个级别:一般、高和较高.对比分析结果如表 1 所示.

由表 1 可看出,白盒加密、数据混淆、虚拟执行环境和部分自校验的保护方法实现难度较高.代码隔离保护方法对被保护软件性能影响最大,故仅适用于少量关键代码保护.同时,白盒加密和代码隔离保护技术的保护强度较高.当然,在实际应用防动态攻击的保护方法时,一般是多种保护技术交互应用,协同保护.

表 1 防动态攻击保护方法的比较

保护方法		实现难度	性能影响	安全性
指令动态映射	动态加解密	一般	一般	一般
	自修改代码	一般	高	一般
混淆技术	白盒加密	较高	高	较高
	控制流混淆	高	一般	高
	数据混淆	较高	高	高
代码隔离	基于服务器	高	较高	较高
	基于虚拟执行环境	较高	较高	较高
校验的防篡改保护		较高	高	高

还有其他一些简单实用的软件保护技术可用于防御攻击者动态分析.例如,反调试技术可对调试器的使用加以抑制或制止,文献[47]中概括了检测程序被调试的基本方法,并分析了反调试能力;文献[48]中通过介绍了反 VMware 虚拟机执行的方法;Schneider[49]利用多线程技术保护注册算法,这种技术会增加攻击者的动态分析复杂度;API 保护和反模拟器也是软件防动态分析保护中常见的技术,等等.当然许多恶意软件也利用上述方法保护自身,防止被杀毒软件等识别,如文献[50]研究动态污点分析方法识别混淆的恶意代码变种.

此外,我们也对防动态攻击的软件保护方法进行深入研究,下面分别介绍.

3 基于变形引擎的软件保护方法

基于代码伪装技术,提出一种基于指令变形切片和计时切片的软件保护机制,实现指令变形和控制流程变形的动态保护技术.从软件中删除待保护软件的重要代码片段,利用自修改变形技术构造这些代码片段的变形引擎,在软件执行过程中,将它们依次还原并执行;同时,基于一些代码片段执行时间调度变形子引擎;并利用动态加解密技术保护变形子引擎.

3.1 该保护方法的基本机理

基于变形引擎的动态保护方法主要有四部分组成:变形切片 mc(metamorphic code)、计时切片 tc(time code)、变形引擎 me(metamorphic engine)和辅助函数 af(auxiliary function).

其中,变形切片 mc 是程序中需要被隐藏的代码片段,即是需要被保护的关键代码片段;记录正常情况下计时切片 tc 的执行时间阈值$[\mathrm{tmin}_i, \mathrm{tmax}_i]$,被保护程序执行时,通过 tc 实际执行时间是否位于阈值$[\mathrm{tmin}_i, \mathrm{tmax}_i]$之间,判断程序是否被调试;指令变形引擎 me 是由变形子引擎组成,变形子引擎用于还原第 i 个变形切片 $\mathrm{mc}_i(i\geqslant 1)$,若程序未被调试,则变形引擎正确调度变形子引擎,正确还原指令变形切片,否则还原为错误指令.辅助函数 AF 用于保存变形中还原的指令变形切片 mc,正常情况下,变形引擎依次将 $\mathrm{mc}_{i-1}(i\geqslant 1)$还原为 mc_i,其中 mc_0 为 AF 初始化时指令.此外,引入动态加解密机制保护变形子引擎.

3.2 被保护软件的执行过程

保护后的软件布局如图 2 所示.

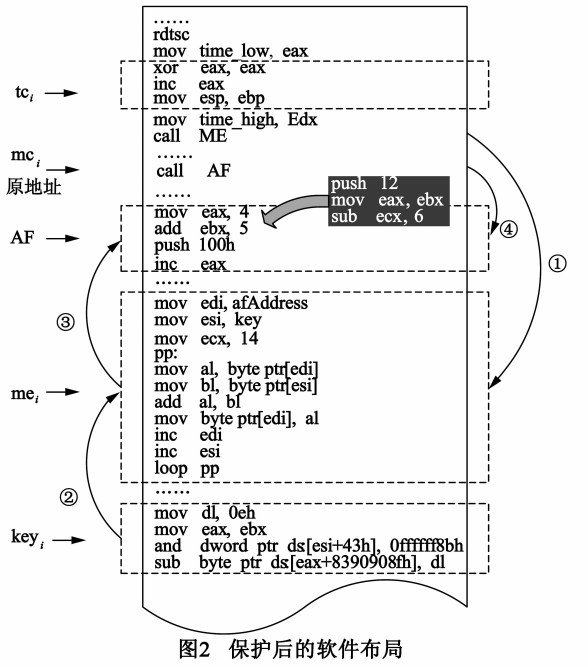

图2 保护后的软件布局

根据图 2 可以看出,保护后软件的执行主要有以下四个步骤:

步骤 1:调用变形引擎.程序执行计时切片 tc_i 后,触发指令变形引擎 ME;

步骤 2:取解密密钥.根据相关参数提取密钥;

步骤 3:指令变形引擎 ME 根据 tc_i 的执行时间判断是否对辅助函数进行变形,此时如果程序被调试或者分析,必定会影响计时切片 tc_i 的执行时间,进而导致变形引擎 ME 不能正确调度变形子引擎 me_i;在正常情况下,利用解密密钥解密指令变形子引擎 me_i 并将辅助函数 AF 还原为相应的指令变形切片 mc_i.

步骤 4:程序继续执行计时切片 tc_i后面的指令;在程序执行到删除指令变形切片 mc_i 的位置时,跳转到辅助函数 AF 处并执行;最后返回继续执行 mc_i 原位置后面的指令.

通过执行过程可以看出,变形切片 mc_i 在程序中原来的位置都在调用辅助函数 AF,迷惑软件攻击者,在一定程度上提高软件安全性.而且,指令变形引擎是在计时切片 tc_i 中触发,而不是在指令变形切片 mc_i 的提取位置,很好地隐藏计时切片 tc_i 和指令变形切片 mc_i 和指令变形引擎的对应关系.

3.3 安全性分析

基于变形引擎的软件保护方法,一方面隐藏了原软件指令变形切片 mc 中的信息,使得该代码片段只有在执行的时候才被还原,有效防止攻击者根据关键信息的静态分析.另一方面,利用计时切片执行时间进行指令变形子引擎的调度分配,将反调试机制、控制流程变形、变形引擎和动态加解密等技术有机结合,在一定程度上增加了攻击者动态分析的难度.

攻击者成功分析被保护软件,需满足以下要求:正确估算每个计时切片有效执行时间、正确分析指令变形引擎调度支配算法、正确获取每个指令变形子引擎的加密密钥、正确根据指令变形子引擎还原对应指令变形切片程序.

3.4 时空开销实验及分析

实验环境:Win XP SP3 操作系统,3.0GHz 处理器,4GB 内存,VS 6.0 开发环境;

实验对象:C++开发的简易计算器 Calculator.exe.其中实现对输入的任意两个数分别进行加减乘除及乘方运算.

实验过程:分别设置变形切片指令条数及变形切片数量,对 Calculator.exe 进行保护,记录保护后软件大小和输入为 15 和 10 时的执行时间.保护前软件大小为 164KB,执行时间为 254μs.保护后软件大小及执行时间如表 2 所示.

表 2　保护后 Calculator 软件大小 Size(KB)和执行时间 Time(μs)

变形切片指令条数区间	项目	变形切片数量			
		3	6	9	12
[1,5]	Size	410.834	412.350	416.112	418.561
	Time	261	268	272	281
[6,10]	Size	413.527	417.323	422.645	427.170
	Time	269	286	297	317
[11,15]	Size	415.425	421.700	427.688	433.090
	Time	279	308	330	352

由表 2 可知,变形切片指令条数与变形切片数量的增长对软件大小影响较小,保护后软件大小基本与待保护指令总条数成正比.变形切片与计时切片是一一对应关系,因此变形切片数量对软件大小的影响包括待保护指令总条数和计时切片中用于计时指令总条数两方面因素.

由于该保护方法在软件执行过程中对内存进行改写,因此对软件性能会造成一定影响.需要还原(改写)的指令越多,影响越大.此外,变形引擎的调度、变形子引擎的动态加解密对软件性能也有影响.

4 虚拟机软件保护方法

基于 2.3.2 节对当前的虚拟执行环境技术机理的分析,通过逆向攻击虚拟机保护后软件,从两个方面对传统虚拟机保护技术进行研究,设计了 VMBP(Virtual Machine Based Protection)保护系统.

4.1 虚拟寄存器位置交换指令

在虚拟机软件保护技术中,VM_Context 是虚拟执行环境,保存虚拟寄存器值的临时结构,获取 VM_Context 是攻击者分析处理函数的前提.在传统虚拟机保护技术中,VM_Context 中的虚拟寄存器值总是按照一定顺序存放,容易获取.虚拟寄存器位置交换指令是在保护后软件执行过程中,动态无序的交换 VM_Context 中虚拟寄存器位置,增加攻击者理解处理函数的复杂度和难度.

虚拟寄存器位置交换指令的指令格式如下:

RDS	Operand1	Operand2

其中,RDS 是操作码,$Operand_1$ 代表虚拟寄存器位置交换方式,当 $Operand_1 = 0$ 时,表示虚拟寄存器整体下移,最底部移至顶部;$Operand_2$ 代表移动幅度,$Operand_2 \in [1,6]$.如执行如下虚拟指令"rri 0h,4h"前后虚拟寄存器位置变化如图 3 所示.

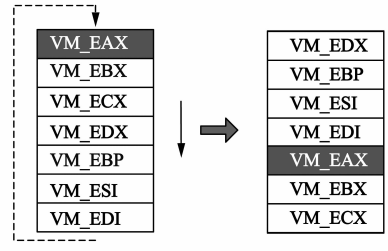

图3　执行指令"rri 0h,4h"前后的虚拟执行环境

当 $Operand_1 \in [1,7]$时,表示两个虚拟寄存器位置交换,其中 $Operand_1$ 代表要交换位置的其中一个虚拟寄存器位置;$Operand_2$ 代表另一个虚拟寄存器位置,因此 $Operand_2 \in [1,7]$.如执行虚拟指令"rri 2h,5h"前后虚拟寄存器位置变化如图 4 所示.

可见,虚拟寄存器位置交换指令前后虚拟机执行环境动态变化.无序的随机组合及应用,可使变化难以预测.

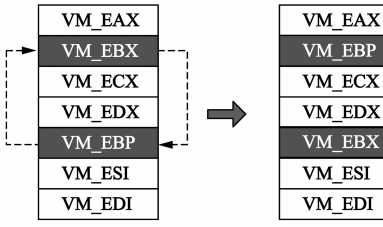

图4　执行指令"rri 2h,5h"前后的虚拟执行环境

4.2 虚拟机多解释路径的设计

传统虚拟机保护技术中,x86 指令与虚拟指令、虚拟指令与处理函数之间是一对多关系,故一条 x86 指令只被一组处理函数序列解释执行.假设已知 x86 指令 I_{x86},解释该指令的虚拟指令序列为$\langle V_1, V_2, \cdots, V_i\rangle$,解释虚拟指令 V_i 的处理函数序列为$\langle h_{v_i}^1, h_{vi}^2, \cdots\rangle$.

那么,解释 x86 指令 I_{x86}的处理函数序列为:

$$\langle h_{v1}^1, h_{v1}^2, \cdots, h_{v1}^l, h_{v2}^1, h_{v2}^2, \cdots, h_{v2}^m, \cdots, h_{vi}^1, h_{vi}^2, \cdots, h_{vi}^n\rangle \tag{1}$$

此时,攻击者通过多次执行,不断积累攻击经验,逐步实现攻击目的,或通过共享攻击经验,协助其他攻击者攻击该虚拟机保护的其他软件.为应对上述问题,研究了多解释路径的虚拟机保护技术.

虚拟机多解释路径是指一条 x86 指令由多组虚拟指令序列解释,同时,一条虚拟指令可由多组处理函数序列解释.多解释路径的虚拟机保护技术主要是通过 x86 指令变形函数 $f(x)$和处理函数变形引擎 $g(y)$实现,利用 x86 指令变形函数 $f(x)$对 I_{x86}进行变形,设每条 x86 指令都有 p 种等价变形指令,则:

$$f(I_{x86}) = \{I_{x86}^1, I_{x86}^2, \cdots, I_{x86}^p\} \tag{2}$$

利用处理函数变形引擎 $g(y)$对处理函数进行变形,设每一个处理函数都有 k 种等价变形,则:

$$g(h_{vi}^j) = \{h_{vi}^{j1}, h_{vi}^{j2}, \cdots, h_{vi}^{jk}\} \tag{3}$$

根据式(3),可得序列(1)的等价序列个数有:

$$k^l \times k^m \times k^n = k^{(l+m+n)} \tag{4}$$

根据式(2)和(4)可得每条 x86 指令的处理函数解释序列个数:$X = (p+1) \times k^{(l+m+n)}$.故解释 Y 条 x86 指令的序列个数为 X^Y.可见,即使参数较小,处理函数解释序列状态空间依然很大.

VMBP 是基于栈的虚拟机,根据对栈的操作及等价逻辑变换,对 x86 指令设计的等价变形模板个数也有限,在 VMBP 中,$0 \leqslant p \leqslant 3$,当 $p=0$ 时表示该 x86 指令没有等价指令.处理函数的变形引擎,一方面与处理函数中指令的变形模板有关;另一方面与处理函数中指令的迭代变形策略有关.其中,迭代次数取任意正数,因此,利用处理函数的变形引擎可构造无限个等价处理函数,即 k 值可无限大.但 k 值越大,造成时空开销越大.所以,在 VMBP 中每个处理函数的 k 值不能过大.处理函数变形引擎根本目的是构造差异度较大的等价处理函数,在变形引擎 $g(y)$中,通过变形模板选择算法、变形指令的选择和迭代次数,构造等价处理函数.其复杂度主要由迭代变形次数控制,防止过大性能消耗.

4.3 安全性分析

虚拟寄存器位置交换指令动态改变虚拟机执行环境,攻击者要正确分析提取的处理函数,首先需要确定所有虚拟寄存器位置交换指令,分析虚拟执行环境的变化.这是正确分析和理解处理函数中指令的基础,提高攻击者根据寄存器值变化分析指令行为的难度,增强 VMBP 虚拟机解释器自身安全,提高被 VMBP 保护软件的安全.

VMBP 中的多解释路径使攻击者在不同次执行中分析不同的指令,减弱攻击者通过多次执行,利用累积经验攻击的效果.同时,由于不同攻击者在分析被 VMBP 保护的相同软件时,执行的路径也有差异,因此降低了攻击者共享攻击经验的可能性.一定程度上限制了恶意行为的扩散,提高该技术的保护强度.

4.4 时空开销实验及分析

实验目的:①验证虚拟机保护的时空开销.②验证保护多样性效果.

实验环境:Win XP SP3 操作系统,3.0GHz 处理器,4GB 内存,VS 6.0 开发环境.

实验对象:见表 3.

表 3 被保护软件大小(KB)和被保护指令的执行时间(μs)

软件名称	待保护指令描述	文件大小	待保护指令执行时间
Csnake	贪吃蛇中定位游戏界面算法的 60 条指令.	233.563	1.654
compress	文件压缩算法中 110 条指令.	229.447	1334
Hanoi	汉诺塔关键算法中 82 条指令,盘子数为 5.	528.503	2388

实验一:验证虚拟机保护的时空开销

分别利用虚拟机保护软件 VMBP,CV(Code Virtualizer)1.3.1.0 和 VMP(Virtual Machine Protector)1.7.0 对实验对象进行保护,保护强度均设置为高.为分析 VMBP 保护时 k 值影响,对每个处理函数取相同 k 值,且 k 分别取 1、2、3 时.保护前后软件大小和被保护指令的执行时间如表 4 所示.

表 4 保护后软件大小(KB)和被保护指令执行时间(μs)

保护方法	项目	VMBP			CV	VMP
		$k=1$	$k=2$	$k=3$		
Csnake	Size	253.952	274.432	293.855	278.884	114.688
	Time	248.587	296.609	304.331	330.956	326.426
compress	Size	249.856	270.900	295.120	290.432	150.012
	Time	42590	43880	43951	31320	28769
Hanoi	Size	569.344	589.824	615.401	567.343	200.704
	Time	4832.324	5283.562	5305.817	4944.398	4760.427

由表 4 可知,随着 k 值增加,VMBP 保护后软件大小基本增加 20KB,是因为每个处理函数都增加了一个

等价变形,但由于变形引擎 $g(y)$变形程中指令选取的随机性及不同指令有不同长度的变形模板,增量有差异.VMBP 与 CV 对软件大小影响基本相同,但 VMP 保护后软件小于原始软件大小,是因为 VMP 在高强度保护中采用文件压缩技术.

三种虚拟机保护技术对被保护指令执行时间的影响处于相同数量级,其差异主要由于这三种虚拟机高强度时的控制参数(如处理函数等价变形的迭代次数)设置不同.$k=2$ 比 $k=1$ 时 VMBP 对被保护指令执行时间影响明显,是因为选择处理函数时,增加了判定指令.而 $k=3$ 与 $k=2$ 对其影响基本相同,是因为 $k=3$ 时只增加备选的处理函数,对执行时间基本无影响.

实验二:验证保护多样性效果

将被 VMBP 保护后的软件执行 6 次($k=2$),利用 IDC 脚本收集被保护代码每次执行时的指令,分别计入集合 $m_1, m_2, m_3, m_4, m_5, m_6$,然后计算被保护代码相邻两次执行时差异度 $\Delta H(m_x, m_{x+1})$.

$$\Delta H(m_x, m_{x+1}) = \frac{2 \times H(m_x, m_{x+1})}{Numm(m_x) + Numm(m_{x+1})} \quad (5)$$

其中,$Numm(m_x)$表示 m_x中指令总条数.$H(m_x, m_{x+1})$记为指令集合 m_x与 m_{x+1}中有差异的指令总条数.

各被保护代码每次执行时指令条数如表 5 所示.

表 5　被保护代码块每次执行的指令条数

被保护软件	m_1	m_2	m_3	m_4	m_5	m_6
Csnake	14700	16098	14931	15165	15002	16293
compress	43921	40700	36379	37296	39408	40007
Hanoi	18600	20281	19432	21014	20418	19208

显然,被保护代码每次执行指令条数不同,是由不同执行路径引起的.利用 BCompare 对比工具分析,可得 $H(m_x, m_{x+1})$,见表 6.

由表 6 可以看出,相同被保护指令块不同次执行的指令有较大差异,基本在 40% 左右.因此会给攻击者多次执行软件进行分析时造成一定困难.差异度的影响因素主要有 x86 变形和处理函数变形.随着指令模板函数增加,差异度会越来越大.

4.5　小结

上述两个实例,分别从“指令动态映射”和“代码隔离”研究了防动态攻击的软件保护技术.基于变形引擎的软件保护方法使攻击者逆向分析软件时,必须对指令变形引擎、指令变形子引擎、密钥传递及计时切片等进行大量分析,大大增加攻击开销.虚拟机软件保护方法通过虚拟寄存器位置交换指令和虚拟机解释器多处理路径,一方面需要攻击者根据动态变化的虚拟执行环境理解虚拟机解释器的处理函数序列,增加分析复杂度;另一方面不同攻击者或同一攻击者不同次执行,虚拟指令解释过程不同,有效防御攻击者基于经验的累积攻击技术.

表 6　被保护代码块相邻两次执行有差异的指令条数及差异度

被保护软件	项目	(m_1, m_2)	(m_2, m_3)	(m_3, m_4)	(m_4, m_5)	(m_5, m_6)
Csnake	$H_1(m_x, m_{x+1})$	6481	7201	6272	5868	6863
	$\Delta H_1(m_x, m_{x+1})$	0.421	0.464	0.417	0.389	0.439
compress	$H_2(m_x, m_{x+1})$	13286	13770	10284	15643	14722
	$\Delta H_2(m_x, m_{x+1})$	0.314	0.357	0.279	0.408	0.371
Hanoi	$H_3(m_x, m_{x+1})$	7742	7918	7716	8657	8510
	$\Delta H_3(m_x, m_{x+1})$	0.398	0.399	0.382	0.418	0.430

参考文献

[1] Chow S, Eisen P, Johnson H, Van Oorschot P. A white-box DES implementation for DRM applications[A]. Revised Papers of ACM CCS-9 Workshop, DRM[C]. Berlin, Heidelberg: Springer-Verlag, 2003, 2696: 1 – 15.

[2] Madou M, Anckaert B, De Sutter B, De Bosschere K. Hybrid static-dynamic attacks against software protection mechanisms [A]. Proceedings of the 5th ACM Workshop on Digital Rights Managemen[C]. New York, US: ACM, 2005. 75 – 82.

[3] Gu Y X, Larose G, Liem C. Software protection patterns: A new language of security[A]. Proceedings of ACM SIGPLAN Software Security and Protection Workshop[C]. Beijing, China: ACM, 2011. 1 – 5.

[4] Dube T E, Birrer B D, Raines R A, Baldwin R O, Mullins B E, Bennington R W, Reuter C E. Hindering reverse engineering: Thinking outside the box[J]. IEEE Security & Privacy, 2008, 6 (2): 58 – 65.

[5] Collberg C, Nagra J, Myilibrary. Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection[M]. USA: Addison-Wesley Professional, 2010.

[6] 段钢.加密与解密[M].第三版.北京:电子工业出版社, 2009.

[7] Suzaki K, Iijima K, Yagi T, Artho C. Software side channel attack on memory deduplication[A]. Proceedings of the 23rd ACM Symposium on Operating Systems Principles, Poster[C]. Cascais, Portugal: ACM, 2011. 1 – 5.

[8] Kanzaki Y, Monden A, Nakamura M, Matsumoto K. Exploiting self-modification mechanism for program protection[A]. Proceedings of the 27th IEEE Computer Software and Applications Conference[C]. Dallas, USA: IEEE, 2003. 170 – 179.

[9] Kanzaki Y, Monden A, Nakamura M, Matsumoto K. Program camouflage: A systematic instruction hiding method for protecting secrets[A]. Proceedings of World Academy of Science, Engineering and Technology[C]. Heidelberg, Germany: WASET,

2008.557 – 563.
[10] Kanzaki Y, Monden A. A software protection method based on time-sensitive code and self-modification mechanism[A]. Proceedings of the IASTED International Conferences on Informatics Software Engineering and Applications (SEA)[C]. Marina Del Rey, USA: EBSCO, 2010.325 – 331.
[11] Madou M, Anckacrt B, Moseley P, Debray S, De Sutter B, De Bosschere K. Software protection through dynamic code mutaion[A]. Proceedings of the 6th International Conference on Information Security Applications[C]. Berlin, Heidelberg: Springer-Verlag, 2006.194 – 206.
[12] Wu Y, Zhao Z, Chui T. An attack on SMC-based software protection[J]. Proceedings of the 8th International Conference on Information and Communications Security[C]. Heidelberg: Springer-Verlag, 2006, 4307: 352 – 368.
[13] Dux B, Iyer A, Debray S, Forrester D, Kobourov S. Visualizing the behavior of dynamically modifiable code[A]. Proceedings of 13th International Workshop on Program Comprehension (IWPC)[C]. Louis, MO, USA: IEEE, 2005.337 – 340.
[14] Collberg C, Thomborson C, Low D. A taxonomy of obfuscating transformations[R]. New Zealand: The University of Auckland, 1997, 1173 – 3500.
[15] Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, Yang K. On the (im) possibility of obfuscating programs[A]. Advances in Cryptology, CRYPTO'01, LNCS 2139[C]. California, USA: Springer Verlag, 2010.1 – 18.
[16] Chow S, Eisen P, Johnson H, Van Oorschot P. White-box cryptography and an AES implementation[A]. Proceedings of the 9th Annual International Workshop on Selected Areas in Cryptography[C]. London, UK: Springer-Verlag, 2003.250 – 270.
[17] Jacob M, Boneh D, Felten E. Attacking an obfuscated cipher by injecting faults[A]. Proceedings of ACM CCS-9 Workshop Digital Rights Management[C]. London, UK: Springer-Verlag, 2003.16 – 31.
[18] Billet O, Gilbert H, Ech-Chatbi C. Cryptanalysis of a white box AES implementation[A]. Proceedings of the 11th International Conference on Selected Areas in Cryptography[C]. India: Springer-Verlag, 2005.227 – 240.
[19] Link H E, Neumann W D. Clarifying obfuscation: Improving the security of white-box encoding[A]. Proceedings of International Conference on Information Technology: Coding and Computing-ITCC[C]. Washington, DC: Springer, 2005.679 – 684.
[20] Wyseur B, Michiels W, Gorissen P, Preneel B. Cryptanalysis of white-box DES implementations with arbitrary external encodings[A]. Proceedings of the 14th International Workshop on Selected Areas in Cryptography[C]. Ottawa, Canada: Springer, 2007.264 – 277.
[21] Bringer J, Chabanne H, Dottax E. White box cryptography: Another attempt[J]. IACR Cryptology ePrint Archive, 2006, 22(2011): 468 – 482.
[22] Yaying X, Xuejia L. A secure implementation of white-box AES[A]. Proceedings of the 2009 2nd International Conference on Computer Science and its Applocations[C]. Jeju, Korea: IEEE eXpress Conference Publishing, 2009.410 – 415.
[23] Birrer B D, Raines R A, Baldwin R O, Mullins B E, Bennington R W. Program fragmentation as a metamorphic software protection[A]. Proceedings of the Third International Symposium on Information Assurance and Security[C]. USA: IEEE Computer Society, 2007.369 – 374.
[24] Oreans. Code Virtualizer[OL]. http://www.oreans.com/codevirtualizer.php, 2013.
[25] Collberg C, Thomborson C, Low D. Breaking abstractions and unstructuring data structures[A]. Proceedings of International Conference on Computer Languages[C]. Chicago, IL: IEEE Computer Society, 1998.28 – 38.
[26] Anckaert B, Jakubowski M H, Venkatesan R, Saw C W N. Practical data location obfuscation[R]. USA: Microsoft, 2009.
[27] Falk R, Goudalo W, Chen E Y, Savola R, Popescu M, Anckaert B, Jakubowski M H, Venkatesan R, Saw C W. Runtime protection via datflow flattening[A]. Proceeings of the Third International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)[C]. Washington DC: IEEE Computer Society, 2009.242 – 248.
[28] 李顺东, 王道顺. 基于同态加密的高效多方保密计算[J]. 电子学报, 2013, 41(4): 798 – 803
Li Shundong, Wang Daoshun. Efficient secure multiparty computation based on homomorphic encryption[J]. Acta Electronica Sinica, 2013, 41(4): 798 – 803. (in Chinese)
[29] Maude T, Maude D. Hardware protection against software piracy[J]. Communications of the ACM, 1984, 27(9): 950 – 959.
[30] Herzberg A, Shulman H, Saxena A, Crispo B. Towards a theory of white-box security[J]. Emerging Challenges for Security, Privacy and Trust, 2009, 297: 342 – 352.
[31] Herzberg A, Shulman H. Robust combiners for software hardening[J]. Proceedings of the Third International Conference on Trust and Trustworthy Computing[C]. Berlin, Heidelberg: Springer-Varlag, 2010, 6101: 282 – 289.
[32] Oreans Technologies. Themida[OL]. http://www.oreans.com/themida.php, 2012.
[33] Co. VMProtect. VMProtect[OL]. http://vmpsoft.com, 2012.
[34] Co. ASProtect. ASProtect[OL]. http://www.aspack.com, 2012.
[35] Fang H, Wu Y, Wang S, Huang Y. Multi-stage binary code obfuscation using improved virtual machine[A]. Proceedings of the 14th International Conference on Information Security

[C]. Xi'an, China: Berlin Springer-Verlag, 2011. 168 – 181.

[36] Blunden B. Virtual Machine Design and Implementation in C/C + + [M]. USA: Wordware Pub, 2002.

[37] Sharif M, Lanzi A, Giffin J, Lee W. Automatic reverse engineering of malware emulators [A]. Proceeings of the 30th IEEE Symposium on Security and Privacy [C]. Oakland, California: IEEE Computer Society, 2009. 94 – 109.

[38] Rolles R. Unpacking virtualization obfuscators [A]. Proceeings of the 3rd USENIX Workshop on Offensive Technologies (WOOT) [C]. Montreal, Canada: ACM, 2009. 1 – 17.

[39] Protections H B. A survey of anti-tamper technologies [J]. CrossTalk: The Journal of Defense Software Engineering, 2004, 12(11): 12 – 16.

[40] 王朝坤,付军宁,王建民,余志伟. 软件防篡改技术综述 [J]. 计算机研究与发展,2011,48(6):923 – 933.
Qiankun Wang, Junning Fu, Jianmin Wang, Zhiwei Yu. Survey of software tamper proofing technique [J]. Journal of Computer Research and Development, 2011, 48(6): 923 – 933. (in Chinese)

[41] Anckaert B, Jakubowski M, Venkatesan R, De Bosschere K. Run-time randomization to mitigate tampering [J]. Advances in Information and Computer Security, 2007, 4752: 153 – 168.

[42] Li D, Hu Y, Hu X, Ling H. Self-checking tamper-proofing based on software behavior model [A]. Proceeings of the Fourth International Conference on Frontier of Computer Science and Technology [C]. Shanghai, China: IEEE Computer Society, 2009. 639 – 643.

[43] Chang H, Atallah M. Protecting software code by guards [A]. Proceedings of the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management [C]. London, UK: Springer-Verlag, 2002. 160 – 175.

[44] Dedic N, Jakubowski M, Venkatesan R. A graph game model for software tamper protection [A]. Proceedings of the 9th International Workshop on Information Hiding [C]. Saint, Malo: Springer-Verlag, 2007. 80 – 95.

[45] Abadi M, Budiu M, Erlingsson, Ligatti J. Control-flow integrity principles, implementations, and applications [J]. ACM Transactions on Information and System Security (TISSEC), 2009, 13(1): 1 – 40.

[46] Gilbert B, Kemmerer R, Kruegel C, Vigna G. Dymo: Tracking dynamic code identity [A]. Proceeings of the 14th International Symposium, RAID [C]. Menlo Park, CA: Springer-Verlag, 2011. 21 – 40.

[47] Gagnon M N, Taylor S, Ghosh A K. Software protection through anti-debugging [J]. IEEE Security & Privacy, 2007, 5(3): 82 – 84.

[48] Sikorski M, Honig A. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software [M]. USA: No Starch Press. 2012.

[49] Schneider T. Hardening registration number protection schemes against reverse code engineering with multithreaded petri nets [A]. Proceedings of Recon [C]. Montreal, Canada: Recon, 2005. 1 – 7.

[50] 王蕊,苏璞睿,杨轶,冯登国. 一种抗混淆的恶意代码变种识别系统 [J]. 电子学报,2011,39(10):2322 – 2331.
Wang Rui, Su Purui, Yang Yi, Feng Dengguo. An anti-obfuscation malware variants identification system [J]. Acta Electronica Sinica, 2011, 39(10): 2322 – 2331. (in Chinese)

作者简介

王怀军　男,1981 年 6 月出生,山东滕州人. 博士生,主要研究领域为软件安全与保护,软件攻击技术,软件保护方法有效性评测.

房鼎益　男,1959 年 4 月出生,陕西汉中人. 教授,博士生导师. 博士毕业于西北工业大学. 主要研究领域为网络与信息安全,软件安全与保护,无线传感网关键技术及其应用研究.
E-mail: dyfang@nwu.edu.cn