doi: 10.3969/j.issn.0490-6756.2018.02.012

Windows 恶意代码动态通用脱壳方法研究

郭　文，王俊峰
(四川大学计算机学院，成都 610065)

摘　要：加壳技术为程序保护提供了一种新的思路，但同时也成为恶意代码的保护伞.恶意软件通过加壳可以批量、快速地生成海量变种，给分析人员带来极大的困扰.因此，研究脱壳技术成为解决该问题的一种有效方法. 传统的脱壳方法如 UPX、ASProtect 等针对的是特定种类的壳，因其不能应付壳的版本与种类的变化而逐渐无法适用，研究一种通用的动态脱壳方法是极为必要的.根据加壳程序执行时都要在内存中还原原始代码的特点，在动态二进制分析平台的基础上提出了一种基于内存标记的动态通用脱壳方法.实验表明，该方法无需先验知识就可以有效地定位加壳程序的原始入口点，提取出程序的原始代码，具有较好的脱壳效果.

关键词：恶意代码；脱壳；动态分析；Decaf-platform
中图分类号：TP309.7　　**文献标识码**：A　　**文章编号**：0490-6756(2018)02-0283-07

Research of dynamic general unpacking methods for Windows malware

GUO Wen, *WANG Jun-Feng*
(College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract: Code packing brings a new conception to protect software, but it also serves as an umbrella for malicious code. It has been intensified that malware using packing techniques to evade detection and it troubles analysts due to the massive variants of malware produced by code packing. Traditional unpacking methods based on feature matching gradually become inapplicable because they can't cope with the change of shell version and type, so a general unpacking method would be very useful. In this paper, the authors proposed a common unpacking method based on dynamic binary analysis platform, according to the property that packer will restore the original code during the process of executing. The experimental results show that this method can effectively locate the original entry point of the program, extract the code that has been hidden, and can get the accurate image size of the process in the memory, which can effectively realize dynamic unpacking of the shell code.

Keywords: Malware; Code unpacking; Dynamic analysis; Decaf-platform

1　引　言

逆向分析可以让恶意代码分析人员在没有源代码的情况下分析目标文件，通过反汇编、反编译和动态跟踪等技术挖掘恶意软件的行为和目的，找到查杀方案.现阶段，越来越多的恶意程序为了保护自己并增加逆向分析的难度，采用了反逆向技术来躲避查杀，其中最为典型的就是抗反汇编、反调

收稿日期：2017-06-12
基金项目：国家重点研发计划(2016YFB0800605，2016QY06X1205)；国家科技重大专项(2015ZX01040101-002)；四川省软科学计划(2016ZR0087)
作者简介：郭文(1993－)，女，湖北黄冈人，硕士生，研究方向为网络与信息安全. E-mail: guowen_1014@163.com
通讯作者：王俊峰. E-mail: wangjf@scu.edu.cn

试以及加壳技术的使用.据《瑞星2016年中国信息安全报告》称,瑞星"云安全"系统于2016年截获的病毒样本总量比2015年同期上涨了16.47%,网络安全态势不容乐观.在此期间,数量最多的依然是新增木马病毒,而加壳带来的变种是造成病毒数量急剧增加的重要原因.加壳给恶意代码的查杀与分析造成了很大的阻碍[1],因此提取恶意程序的原始代码是恶意软件分析的重点研究内容.

脱壳的目的就是还原程序的原始代码.脱壳主要分为手动脱壳和自动脱壳两种[2],手动脱壳对分析人员的技术要求较高,需要有专业的逆向工程经验才能完成;自动脱壳则只需脱壳人员具备最基本的计算机知识即可,而且能节省大量的时间,因此应用得更为广泛.目前的自动脱壳技术又分为定向脱壳和通用脱壳两大类,定向脱壳工具是基于特征匹配实现的,具有脱壳速度快的优点;而通用脱壳则是基于加壳程序执行时的共性实现脱壳的,相比而言,通用脱壳工具较好地克服了定向脱壳工具的局限性,可以脱多种类型的壳,适用范围更广.

由于加壳会对程序的函数列表、指令代码等内容进行加密,静态分析加壳恶意软件得到的有用信息非常有限,动态分析有利于更好地捕捉加壳程序的真实意图.现有的动态脱壳方法集中在基于调试器和基于虚拟机两个方面,前者由于调试器自身的局限性,无法进行复杂的程序行为监控,后者需要提前对程序入口点的范围有一个粗略的判断,实用性较差.本文在动态二进制分析平台decaf-platform[3]的基础上,利用加壳程序会在执行过程中动态还原原始代码的特征,通过开发插件对内存读写与执行的状态进行监控,获取程序入口点及进程镜像大小,无需先验知识即可实现对加壳恶意代码的脱壳,同时本文的方法还能有效的判断程序是否加壳.

2 相关工作

2.1 加壳技术

加壳是采用特殊的算法,对二进制文件中的资源进行隐藏,得到一个新的可执行文件,其表现形式为对原始程序的压缩或加密.加壳的主要目的是隐藏程序的真正入口点OEP(Original Entry Point),伪装自己的真实行为.与普通的磁盘文件加密不同的是,加壳程序在运行时才在内存中完成解压或解密工作,而这一过程对外界是完全透明的.加壳程序执行时,壳程序先取得程序控制权,到达程序的真正入口点后将控制权递交给原始代码,真正的程序才得以执行.

加壳技术从最初产生以来不断发展,从最初的仅仅具备压缩或加密功能到现在的各种反调试技术的使用,虽然对程序版权的保护起到了良好的效果,但同时也助长了恶意代码的变种行为,大大降低了恶意代码分析人员的工作效率.脱壳的目的就是提取加壳程序的原始代码,分析程序的真实执行行为.

2.2 壳识别技术

壳识别技术可以帮助脱壳者确定壳的种类和版本,进而采用相应的脱壳方法.在传统的静态脱壳中,壳识别是至关重要的一步.PEiD是一款典型的壳识别工具[4],它维持着一个需要不断更新的壳特征库,通过提取程序特征并进行匹配来完成壳识别,确定程序所加壳的种类和版本.由于有了壳特征库作为基础,PEiD最大的优点就是针对性强,其自带的脱壳插件处理速度较快;但也由于壳特征库的束缚,无法识别特征库以外的壳和经过版本更新的壳,也不能处理未知壳.

在动态通用脱壳中,脱壳方法不依赖于壳的种类和版本,因此只需要检测出程序是否加壳即可.文献[5]提出了基于静态特征的PE文件加壳检测方法,根据PE文件中节区的属性、表项的个数以及PE文件的熵等特征进行加壳判定;文献[6]以明可夫斯基距离作为评判标准对程序进行分类,检测程序是否加壳,也达到了较好的检测效果.

2.3 脱壳技术

现有的脱壳方法根据实现方式可分为手动和自动两大类.手动脱壳利用调试工具对加壳程序进行手动分析查找OEP,常见的确定OEP的方法有单步调试、ESP平衡定律、内存镜像、模拟跟踪等[7],找到程序入口点后再利用调试工具自带的脱壳插件或专门的内存转储工具进行脱壳,最后再根据脱壳结果确定是否需要进行IAT修复.

手动脱壳对分析人员的技术要求较高,需要脱壳人员具备专业的计算机知识和较丰富的调试经验,不能进行广泛的应用.自动脱壳通过提供一个自动化工具就可以将输入的加壳程序进行脱壳处理并返回一个可执行程序.目前已有多种自动脱壳工具,可分为基于调试器的脱壳工具和基于虚拟机的脱壳工具两大类[8].

基于调试器的典型脱壳工具有OllyBone[9]和Universal PE Unpacker[10].

OllyBone 通过将 OEP 所在页的属性设置为不可执行从而引发跳转时的内存访问异常，通过捕获该异常来确定程序的 OEP. Universal PE Unpacker 是交互式静态反编译软件 IDA Pro 的一个插件，它是利用加壳程序执行时需要调用 LoadLibrary 和 GetProcAddress 函数组合的特征来确定 OEP. 这两种脱壳工具是基于调试器实现的，具有一定的通用性，缺点在于都需要事先提供入口点的范围，当加壳人员把 OEP 和解壳代码放在同一个代码段，或者以其他函数实现 GetProcAddress 的功能时，以上两种工具就会失效.

基于虚拟机的脱壳工具有 PolyUnpack[11] 和 MalUnpack[12].

PolyUnpack 通过将当前正在执行的指令与预先静态反汇编生成的指令进行比对，若发现有新生成的指令，则结束当前进程，提取目标程序. 但静态反汇编和逐条比对指令带来了较大的时间复杂度，整个脱壳过程实施起来相当繁琐. MalUnpack 利用静态分析与动态执行结合的方法识别并去除恶意程序的环境敏感性，结合污点分析和符号执行技术实现二进制代码的通用脱壳，该方法在一定程度上可以解决虚拟环境易被发现的问题，但不能应用于采用了新的环境敏感技术的加壳软件的脱壳.

本文在总结了现有脱壳工具的不足后，在动态二进制分析平台的基础上，提出了基于内存标记的动态通用脱壳方法，该方法利用加壳程序执行的固有特征实现，能准确地判断程序是否加壳，无需先验知识，并且适用于多种壳.

3　本文动态通用脱壳方法的实现

脱壳流程一般可分为壳识别、查找 OEP、内存转储以及程序修复几个步骤[13]，如图 1 所示. (1) 利用查壳工具确定程序是否加壳以及加的是何种壳；(2) 通过单步跟踪或利用 ESP 平衡定律等方法来确定程序的 OEP；(3) 在 OEP 处将程序的进程内容从内存空间转储到磁盘上；(4) 根据需要完成导入地址表 IAT(Import Address Table)重建和程序修复.

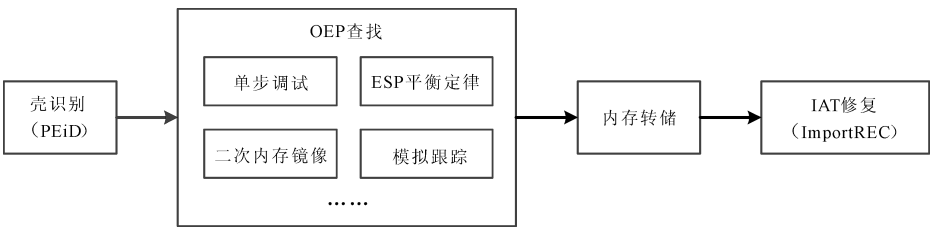

图 1 通用脱壳流程图
Fig. 1　General shelling model

如图 1，加壳只是改变了程序的结构而并未改变其功能，加壳程序在最终运行时还是要执行最原始的代码. 当程序运行到原始入口点，原始代码开始执行，此时就能进行脱壳，因此脱壳最关键的一步就是确定程序的 OEP.

根据加壳程序的“功能不变性”以及加壳程序在内存中“先写后执行”的特点，本文提出了一种基于内存标记的动态通用的脱壳方法，该方法利用的是所有加壳程序的固有特征，不依赖于壳的种类和版本，并且无需先验知识，具有较好的通用性. 在此基础上，本文提出了如图 2 所示的脱壳模型. 该模型主要由预处理模块、入口点查找模块、内存转储模块和程序修复模块四部分组成. 首先对程序进行查壳处理，然后开始运行加壳程序，查找程序的真正入口点，到达入口点后将内存中的进程镜像转储出来，最后根据需要对程序进行 IAT 修复.

3.1　预处理

在执行脱壳流程之前，先对程序进行静态预处理，预处理的主要目的是判断程序是否加壳，从而剔除未加壳程序，以免带来不必要的后续处理. 为了提高检测的效率，此步骤采用文献[5]提出的方法进行加壳检测，该方法提取了 PE 文件的 9 个静态特征，利用多种机器学习算法生成分类器来进行加壳检测，检测率高并且误报率和漏报率较低，同时，使用静态特征也可以保证较快的识别速度. 加壳程序在本文中提出的脱壳模型中运行时，会有转储文件生成，该特征可以作为程序加壳与否的验证. 经预处理识别为未加壳的程序无需进行后续的流程，而已加壳的程序则需要继续脱壳.

3.2　入口点查找

3.2.1　*确定入口点候选值*　加壳程序启动后，壳代码会优先取得程序控制权，当壳代码执行完，原始代码已经被释放出来，完成控制权的交接，程序

到达原始入口点,真正的代码开始执行.利用加壳程序的这一固有执行特点,在原始代码被完全释放出来以后再进行内存转储和输入表修复等工作,脱壳流程就基本完成.因此入口点查找是否成功直接影响到脱壳流程能否顺利进行,是整个脱壳流程中最重要的模块.

本文提出的脱壳模型的入口点查找是基于内存监控和动态标记实现,如图 3 所示.在加壳程序执行前,先在主机中开辟一片存储区域来记录客户机的内存状态变化,该区域的每一个字节都代表着加壳进程在内存中的一个页面,大小为 4096 kB.

当客户机中的加壳程序被加载后,主机会记录该进程启动时的相关语义信息,包括进程名、进程 Pid 和进程的 CR3 值等.

在加壳进程执行过程中,若发生内存写操作,"写操作标记模块"的回调函数就会捕获该行为并在该写指令的目的地址对应的主机存储区域做上标记;同时"写——执行识别模块"会不断检查当前指令是否被标记过,若被标记说明该指令是在运行过程中新生成的,则该处可能为程序的原始入口点.

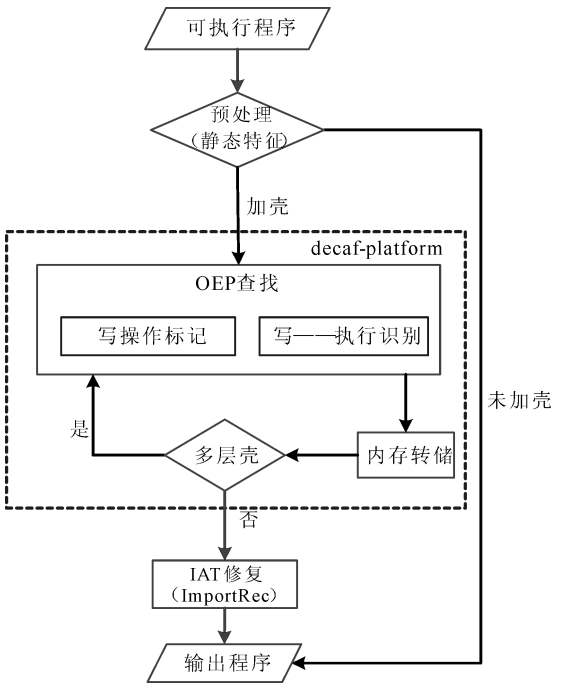

图 2 基于内存标记的动态通用脱壳模型

Fig. 2 Dynamic general shelling model based on memory marking

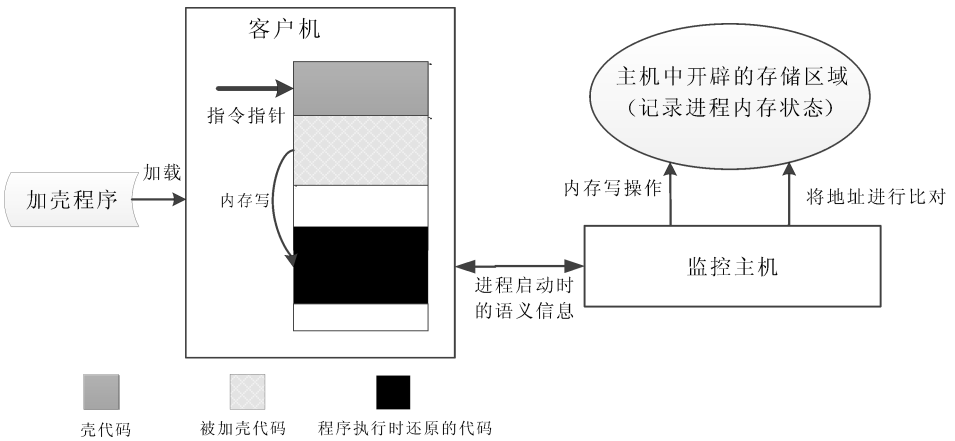

图 3 入口点查找模块示意图

Fig. 3 Themodule diagram of finding OEP

3.2.2 入口点筛选 由于程序在执行过程中可能会发生多次写操作,经过上述的"写操作标记模块"和"写——执行识别模块"得到的入口点可能有多个候选值,而程序的真正入口点只有一个,所以还需要根据相关规则对上述入口点进行筛选.

根据加壳程序的执行特点,本文采用了 3 种方法来筛选入口点.

(1) ESP 平衡定律:加壳工具为了实现通用性,在程序到达原始入口点时必须保证堆栈状态和原始程序相同,否则程序在执行时会出现异常.所以在程序刚加载时先记录好 ESP 寄存器的值,然后将候选入口点处的 ESP 寄存器的值与初始值进行比对,若二者相同则该处就可能是程序的真正 OEP.

(2) 大跨度跳转:在加壳时,程序通常会构造一个新的区段来存放原始代码,因此在执行原始代码前也就是转到 OEP 处时会有一个大跨度的指令跳转,据此也能进一步确定程序的真正入口点,因

此本文对前面得到的候选入口点进行扫描，若发现该地址是由一个大跨度跳转得到的，则可以确定该处即为程序真正的 OEP.

(3) PUSHAD 与 POPAD 组合：对于部分壳如“UPX”、“ASPack”等，壳代码在执行前有一个入栈的操作“PUSH AD”，壳代码执行结束跳转到 OEP 前会有一个出栈操作“POP AD”，根据这一特点也能对入口点的候选值进行筛选. 该方法实现起来较为简单，经过第一步的壳识别确定为相关类型的壳可以采用该方法进行快速的入口点筛选.

综合利用这三种方法可以最终确定程序唯一的 OEP：首先根据 ESP 平衡定律对候选入口点进行初步筛选，对于满足该定律的入口点若满足大跨度跳转则可以确定为真正的 OEP；对于 UPX 等壳可以直接利用指令组合特征进行快速筛选确定 OEP.

3.3　转储和修复

程序运行到 OEP 后，原始代码已经被完全释放出来时，此时需要将进程镜像从内存中抓取出来，以文件的形式存储在磁盘上. 在确定 OEP 后，可以通过进程环境控制块的信息来获取进程的镜像大小，然后在磁盘中申请对应大小的存储空间，接着将镜像内容保存到文件中完成内存转储. 在转储完成后，若发现加壳进程存在多层壳，则继续转入内存标记模块查找入口点，否则就进行最后的程序修复.

一个完整的 PE 文件除了存储代码和数据等的各种节区外还有文件头，文件头中存有保证程序正常运行的导入地址表等信息. 某些加壳工具尤其是 ASProtect 等加密壳会对程序的导入地址表进行加密处理，因此经过内存转储得到的程序一般无法正常运行，所以脱壳后还需要对转储文件进行修复. 程序修复的原理是：当加壳程序运行到原始入口点处，导入地址表已经被释放出来，因此修复的方法就是在此时构造一个新的区段来存放导入地址表，构成一个可执行的 PE 文件.

4　实验分析

4.1　实验环境

在上述脱壳模型的基础上，本文实现了一个完整的脱壳方案，并通过实验证明了该方案的有效性. 本文方法是在动态二进制分析平台 decaf-platform 的基础上实现的. DECAF(Dynamic Executable Code Analysis Framework)是一个基于 QEMU 进行二次开发的二进制框架，在 QEMU 的基础上又增加了污点跟踪等动态分析功能，并且允许用户开发自定义的插件来实现自己的功能.

测试环境的主机：操作系统为 Ubuntu 12.04；处理器：Intel(R) Core(TM) i5-2450M CPU @ 2.50 GHz；内存：6 G；客户机：操作系统为 Windows XP SP3 的虚拟机；程序实现：C 语言.

4.2　实验样本与内容

为了验证本文提出的脱壳方法的有效性，本文选取了多种原始程序，并采用 15 种不同的加壳工具进行加壳得到了 37 个实验样本. 先利用 PEiD 工具和本文使用的静态特征两种方法分别对样本进行加壳检测，然后利用静态分析工具 PETool 进行入口点识别，再利用基于调试的动态脱壳工具 PackerBreaker 和本文方法对分别对加壳样本进行脱壳处理，得到实验结果.

4.3　实验结果与分析

实验结果如表 1 所示，其中静态特征加壳检测结果列的“√”表示检测到程序被加壳，PEiD 查壳结果列的“√”表示能正确识别壳的类型，“Unknown”表示识别出程序被加壳但无法确定壳的类型；后三列的“√”表示入口点查找正确或者脱壳成功，“×”表示入口点查找有误或脱壳失败.

从表 1 可以看出，在 37 个实验样本中，PEiD 能检测到所有的程序均被加壳但能正确识别出壳类型的只有 26 个，本文使用的静态特征检测方法也能检测到所有的加壳程序. 在入口点识别与脱壳处理方面：PETool 能正确识别出的入口点为 3 个，PackerBreaker 正确识别出入口点并完成脱壳的为 27 个，而本文方法识别出的正确入口点并进行脱壳的样本有 30 个. 由结果可知，本文方法要明显优于 PETool 工具，经分析是由于该工具是静态分析工具，是基于 PE 文件结构中的相应字段来获取入口点的，而程序经过加壳后入口点一般都被隐藏，因此识别效果较差；本文方法相比于 PackerBreaker 脱壳工具也有一定的优势，实验过程中发现，PackerBreaker 依赖于壳的版本与类型，对无法识别的壳无法进行处理，而本文方法是基于加壳程序的固有特征实现的，具有更好的通用性.

表 1 本文方法与其他脱壳工具对比结果

Tab. 1 The unpacking result compared with other unpacking tools

程序名称	加壳工具	静态特征加壳检测结果	PEiD 查壳结果	PETool 入口点识别结果	PackerBreaker 脱壳效果	本文方法脱壳效果
c. exe	ACProtect	√	Unknown	×	√	×
c. exe	ASPack	√	√	×	√	√
c. exe	ASProtect	√	√	×	×	×
c. exe	EXE32Pack	√	√	×	×	√
c. exe	MEW	√	√	×	√	√
c. exe	MPress	√	Unknown	×	√	√
c. exe	NSPack1. 3	√	√	×	√	√
c. exe	NSPack2. 4	√	Unknown	×	√	√
c. exe	NSPack3. 7	√	√	×	√	√
c. exe	PECompact	√	√	√	×	√
c. exe	PESpin	√	Unknown	×	×	×
c. exe	UPX	√	√	×	√	√
c. exe	WinUpack	√	Unknown	×	√	√
note. exe	ACProtect	√	Unknown	×	√	×
note. exe	ASPack	√	√	×	√	√
note. exe	ASProtect	√	√	×	×	×
note. exe	EXE32Pack	√	√	×	√	√
note. exe	MEW	√	√	×	√	√
note. exe	MPress	√	Unknown	×	√	√
note. exe	NSPack1. 3	√	√	×	√	√
note. exe	NSPack2. 4	√	Unknown	×	√	√
note. exe	NSPack3. 7	√	√	×	√	√
note. exe	PECompact	√	√	√	×	√
note. exe	PESpin	√	Unknown	×	×	×
note. exe	UPX	√	√	×	√	√
note. exe	WinUpack	√	Unknown	×	√	√
gzip124. exe	ASProtect	√	√	×	×	×
gzip124. exe	FSG	√	√	×	√	√
gzip124. exe	NSPack3. 7	√	√	×	√	√
gzip124. exe	PECompact	√	√	×	√	√
gzip124. exe	UPX	√	√	×	√	√
gzip124. exe	WinUpack	√	Unknown	×	√	√
bzip2. exe	FSG	√	√	×	√	√
bzip2. exe	PECompact	√	√	√	×	√
bzip2. exe	UPX	√	√	×	×	√
Notepad. exe	EZIP	√	√	×	√	√
qqspirit. exe	FSG	√	√	×	√	√
总计		37/37	26/37	3/37	27/37	30/37

5 结 论

本文利用加壳程序运行时要在内存中还原原始代码的共性,对加壳进程的内存写操作的目的地址进行标记,根据先写后执行的特点确定程序的原始入口点,在入口点处完成内存转储并进行导入表重建和程序修复. 本文提出的脱壳方法已经以插件的形式在 decaf-platform 二进制动态分析平台上实现,实验证明,该方法可以脱多种壳,突破了传统的基于特征匹配的脱壳方法的局限性,且无需脱壳

先验知识，具有较好的通用性.

本文提出的基于内存监控和动态标记的脱壳方法为程序脱壳提供了新的思路.但本文方法对一些强加密壳处理效果并不是很好，因此如何提高脱壳尤其是加密壳的效率是后续工作的研究重点；动态脱壳的预处理过程只需要确定程序是否加壳而不关注壳的种类，因此本文后期将会研究效率更高的加壳检测方法；同时由于本文目前是利用现有的修复工具进行 IAT 修复，如何提高程序修复的效率也是今后的研究方向.

参考文献：

[1] 郑荣锋，方勇，刘亮. 基于动态行为指纹的恶意代码同源性分析[J]. 四川大学学报：自然科学版，2016，53：793.

[2] 梁光辉，庞建民，戴超，等. 基于内存监控的动态脱壳系统[J]. 信息工程大学学报，2015，16：749.

[3] Henderson A，Yan L，Hu X，*et al*. DECAF：A platform-neutral whole-system dynamic binary analysis platform[J]. IEEE Trans Softw Eng，2014，99：1.

[4] Pandey S K，Mehtre B M. Performance of malware detection tools：a comparison[C]// International Conference on Advanced Communication Control and Computing Technologies. India：IEEE，2015.

[5] 佘春东，刘达富，王俊峰，等. 基于静态特征的 pe 文件加壳检测方法[P]. CN:102024112 B，2012.

[6] 吴丽娟，李阳，梁京章. 一种基于明可夫斯基距离的加壳 PE 文件识别方法[J]. 现代电子技术，2016，39：80.

[7] 李露，刘秋菊，徐汀荣. PE 文件中脱壳技术的研究[J]. 计算机应用与软件，2010，27：279.

[8] 赵中树. 基于 Windows 平台的脱壳技术研究与实现[D]. 成都：电子科技大学，2012.

[9] Babar K，Khalid F. Generic unpacking techniques[C]// International Conference on Computer，Control and Communication. Karachi，Pakistan，Pakistan：IEEE，2009.

[10] Raber J. Columbo：High perfomance unpacking[C]// International Conference on Software Analysis，Evolution and Reengineering. Klagenfurt，Austria：IEEE，2017.

[11] Royal P，Halpin M，Dagon D，*et al*. Polyunpack：Automating the hidden-code extraction of unpack-executing malware[C]// Computer Security Applications Conference. Miami Beach，FL，USA：IEEE Computer Society，2006.

[12] 王志，贾春福，鲁凯. 基于环境敏感分析的恶意代码脱壳方法[J]. 计算机学报，2012，35：693.

[13] 彭小详，户振江，龚涛，等. 恶意代码自动脱壳技术研究[J]. 信息网络安全，2014，13：41.