DOI:10.12379/j.issn.2096-1057.2022.07.05

虚拟机软件保护技术综述

李成扬[1] 陈夏润[1] 张 汉[2] 文伟平[1]
[1](北京大学软件与微电子学院 北京 102600)
[2](武汉十月科技有限责任公司战略规划部 武汉 430073)
(lcymoon@pku.edu.cn)

Research on Virtual Machine Protection: A Survey

Li Chengyang[1], Chen Xiarun[1], Zhang Han[2], and Wen Weiping[1]
[1](*School of Software and Microelectronics, Peking University, Beijing* 102600)
[2](*Department of Strategies and Planning, Wuhan October Technology Co., Ltd., Wuhan* 430073)

Abstract Static and dynamic analysis of software is always present in software distribution. As an extension of code obfuscation, virtual machine software protection provides the possibility to defend against MATE(man-at-to-end) attacks. Due to the lack of review articles in this field, this paper reviews and collates this issue. The existing problems in the development of virtual machine protection were first pointed out. Then, the structure of virtual machine software protection was introduced, its security was analyzed by citing related articles. And at last, a summary of current work was given, and the future research directions were envisioned.

Key words software protection; virtual machine protection; virtualization-based obfuscation; code obfuscation; virtual machine

摘 要 在软件的分发过程中,要考虑针对软件的静态和动态分析.虚拟机软件保护作为代码混淆的延伸发展,为抵御MATE攻击提供了可能性,囿于目前该领域综述性文章空白的现状,对该问题进行综述整理.首先指出了现有虚拟机保护发展中存在的问题,然后介绍了虚拟机软件保护的结构,同时引述相关文章对其安全性进行分析和总结,并对未来的研究方向进行了展望.

关键词 软件保护;虚拟机保护;虚拟混淆;代码混淆;虚拟机

中图法分类号 TP311

自20世纪40年代至今,计算机(冯·诺依曼体系架构)的发展和应用已取得巨大成功[1],对其进行分析,应包含如下几点:1)足够的抽象性.可以被应用于各种实际场景,并进一步解放生产力.2)信息载体的便捷性.更低的存储成本,以及更快的传播速度.3)不可替代性.目前没有一个更好的可替代实体可以行使目前计算机的功能.而作为计算机的“核心”——软件,在发展的过程中始终处于攻与防的军事备赛中.一方面软件面临着盗版(software piracy)、逆向工程(reverse engineer)、

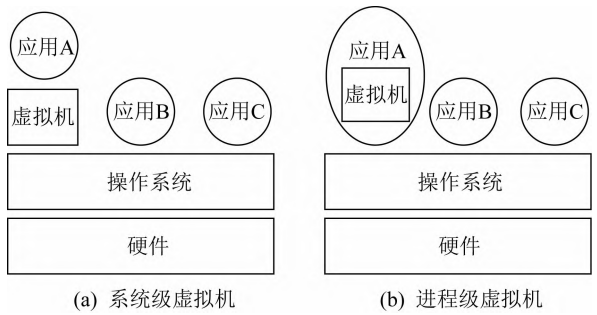
收稿日期:2022-05-26
基金项目:国家自然科学基金项目(61872011)
引用格式:李成扬, 陈夏润, 张汉, 等. 虚拟机软件保护技术综述[J]. 信息安全研究, 2022, 8(7): 675-684

代码篡改(code tamper)的威胁;另一方面是代码混淆(code obfuscation)、软件水印(software watermarking)、软件防篡改(software tamper-proofing)等保护措施[2].而虚拟机软件保护(virtual machine protection, VMP)[3-4]较大程度上阻止了针对软件的静态和动态分析,可将其表述为:针对程序中待保护的代码(源码或者二进制段),将其抽离转变为自定义指令,并同对应的解释器一起嵌入程序中,在程序运行时,通过解释器对自定义指令解释执行,在不暴露原有指令的前提下实现程序原有的语义.

在进一步讨论 VMP 的细节前,需要对进程级虚拟机和所处的攻击语境作一个铺垫说明.

1) 为了屏蔽硬件层的设计差异,进一步复用计算机资源,软件层级的虚拟化技术为使用者提供了更多的可能性.根据虚拟化对象的层级,可以进一步将虚拟机分为系统级虚拟机和进程级虚拟机[5],前者实现操作系统的虚拟化,后者实现单个应用的虚拟化,仅针对当前进程生效,如图 1 所示.虚拟机在实现虚拟化的过程中,实现的是将虚拟的客户系统映射到真实主机,本身并不涉及对细节的隐藏[5].但在代码保护领域提及的虚拟机软件保护(virtual machine protection)或者虚拟混淆(virtualization-based obfuscation),实现的是虚拟指令到真实指令的映射,同样的未隐藏、也未缺失某些指令功能,但对基于模式或者经验开展分析工作的自动化工具或分析人员而言,因其对虚拟指令集的不熟悉,可以实现代码逻辑或实现细节上的保护.本文语境下的 VMP 是进程级别的虚拟机,致力于提高软件安全性.

2) Collberg 等人[6]指出恶意的终端用户在软件安全保护中应当受到足够的重视,因为合法的用户也可以对软件程序进行次数或者时间上无限制的调试,甚至是修改.MATE 攻击(man-at-the-end attack)[7]和具体化的白盒攻击(white box attack)[8],对软件拥有完全访问权和控制权的用户,可以执行任意类型和次数的静态或动态分析,从而获得软件运行的内部信息或结果,所以如何在诸如此类的攻击场景中保证软件的安全,便是 VMP 试图解决的问题.



(a) 系统级虚拟机 (b) 进程级虚拟机

图 1 虚拟机类型

其次,对于 VMP 保护思路最早被提出的时间节点并没有统一的定论.文献[9-10]认为 VMP 保护最早可以追溯到 Maude 等人[11]提出抽离核心的代码放在硬件单元中保护执行;文献[12]认为 Collberg 在首次对代码混淆进行分类的文章中提出的“table interpretation”便是今日虚拟机软件保护的雏形,因为在相关的章节中已经明确提出“将代码转变为不同的虚拟机代码,并通过解释器进行解释执行”[13].虽然对于起源的时间节点没有明确的定论,但商业软件的成功推广,如 VMProtect[14], Themida[15], Code Virtualizer[16]等,让工业界和学术界更多地关注于该领域的发展.同时开源的 VMP 项目,如 VMPROTECT(https://github.com/eaglx/VMPROTECT), rewolf-x86-virtualizer(https://github.com/rwfpl/rewolf-x86-virtualizer),也进一步推动了相关技术的研究.

为了有效梳理现有的研究和工作成果,我们首先确定了 virtual machine protection, virtualization-based obfuscation, emulation-based obfuscation, software protection,虚拟机软件保护,虚拟机等关键词,然后通过如下论文数据库进行检索:The DBLP Computer Science Bibliography(https://dblp.uni-trier.de), Web of Science(https://apps.webofknowledge.com), Scopus (https://scopus.com), Semantic Scholar(https://www.semanticscholar.org),中国知网(https://www.cnki.net).

最后确定了直接相关的期刊论文共 46 篇(截至 2022 年 1 月).不同年份论文发表数量统计如图 2 所示,可以看出虽然每年的数量不一,但从总体看是呈上升趋势的,特别是近几年在文章数量上有了较大的增长.从研究机构看,国内西北大学、南开大学、国防科技大学等均在此领域有对应的研究成果,但发表在高质量的国内外期刊上的文章

数量还较少，同时针对 VMP 的综述文章更少，仅在 ARES(https://www.ares-conference.eu/)上有 1 篇针对 VMP 还原的综述[12]，因此本文汇总整理目前国内外在 VMP 的研究成果，并对 VMP 的发展给出一个探讨说明.

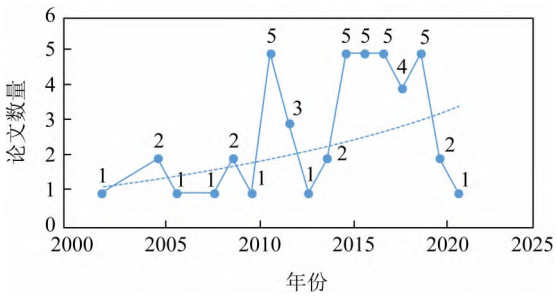


图 2　不同年份论文数量

本文的贡献在于梳理汇总了国内外目前的 VMP 研究，针对 VMP 的架构和安全性分析给出了阐述说明，有助于对该领域的进一步研究.

1　问题与挑战

为了有效标识文献内容，对其进行关键词提取，如图 3 所示，可得出如下结论：

1）攻击和保护方案均从不同的维度有对应的研究成果，其中左侧为攻击方式的研究，右侧为保护方式的研究；

2）相较于攻击的方式，增强保护呈现出多元化的状态，对于安全性增强的研究多于攻击一方；

3）相较于保护或者攻击维度的多元化，在效果的评估指标上并没有针对 VMP 的特定统一指标，基本上沿用了程序分析的指标，包括安全性的理论分析和性能分析.

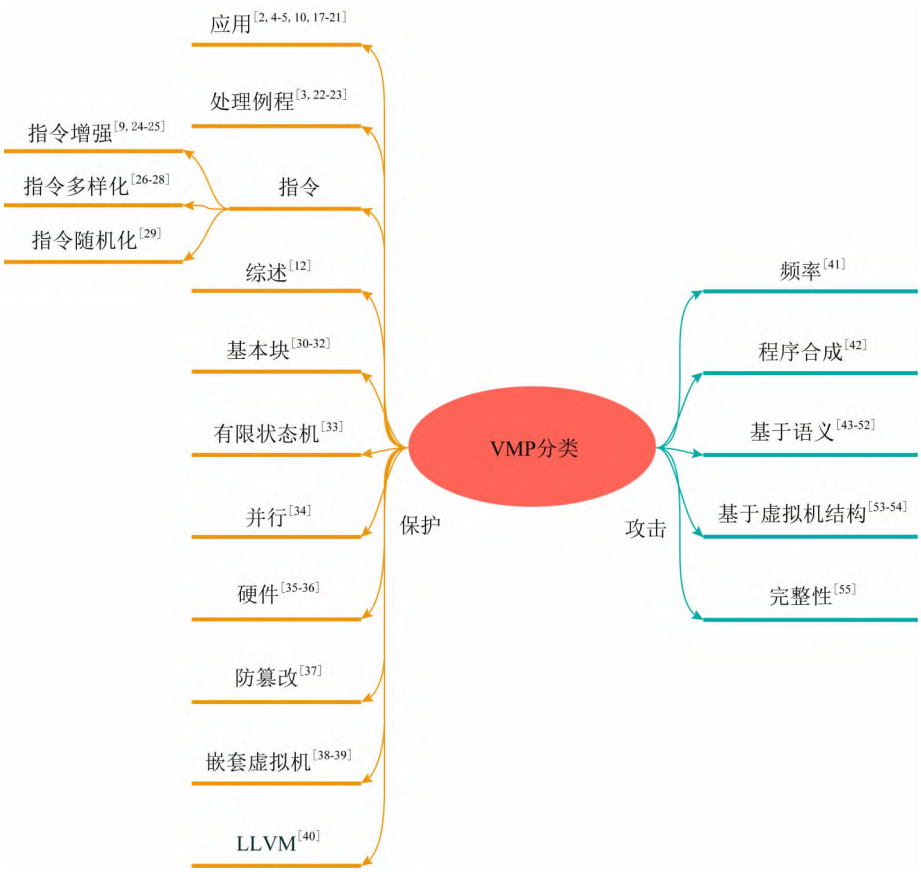


图 3　文献分类

虽然从统计的文章中可以看出对安全研究的文章从数量上多于攻击研究的文章，但是因为没有统一的客观评估指标，针对众多的安全加固方法，其真实效果是欠缺真实考量的.同时结合年份

进行分析，从图4可以得到如下结论：

1）近几年针对VMP的研究更为密集，无论是保护层面还是攻击层面；

2）攻击层面，更多地专注于基于语义的攻击模式；保护层面相较于早期并没有较大的理论创新，更多地是针对现有方法在某一层面上的加固或强化.

现有的VMP发展中，无论是攻击层面还是保护层面均有不同的学术成果的产出，但仍然存在一些不足和挑战：一方面是保护层面有待新理论的提出，在性能开销的降低和保护效果的增强上取得更好的效果；另一方面是对于方法的评估模型或指标的构建，结合方法本身的特性给出可量化的评估方案.

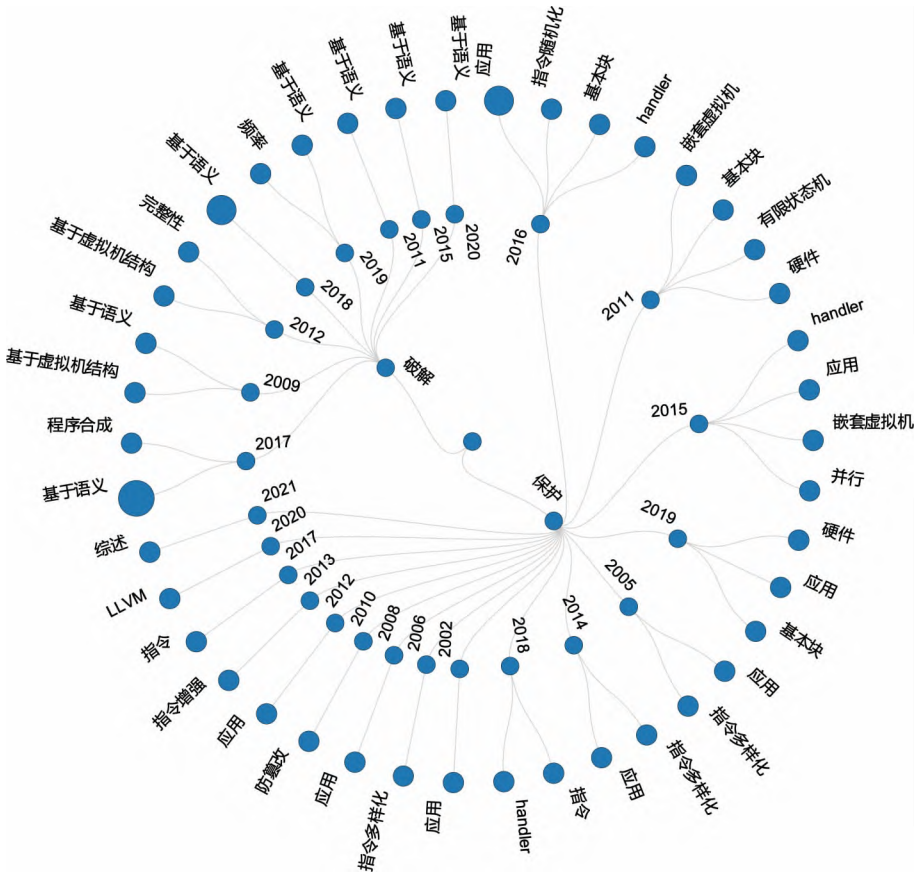


图4 文章类型分布

2 虚拟机结构

VMP作为软件保护的一种方法，被认为是代码混淆的一种延伸发展，归结于2点原因：其一，在处理手法上同混淆相似，对源程序进行变形处理，在保证原有语义的前提下，使得程序中核心的内容被有效隐藏；其二，VMP和混淆方法的亲和性，VMP中可以使用混淆方法进行自身效果的加强[17-18].从本质上讲，作为逻辑实体的VMP本身构成包括：虚拟机指令集（virtualization instruction set，VIS）、虚拟机解释器（virtualization interpreter）和虚拟上下文（virtualization context），其结构如图5所示.

2.1 虚拟机指令集

虚拟机指令集约定了本地指令和虚拟指令的映射关系，作为解释器的输入，指定了程序原生的语义逻辑.现有的虚拟机实现中，包括基于栈（stack-based）和基于寄存器（register-based）的实现方式[17,57]，这里的分类依据便是指令集的设计.基于栈式的虚拟机实现中，将原生的指令转变为基于堆栈的操作，对虚拟机上下文传入的数据以

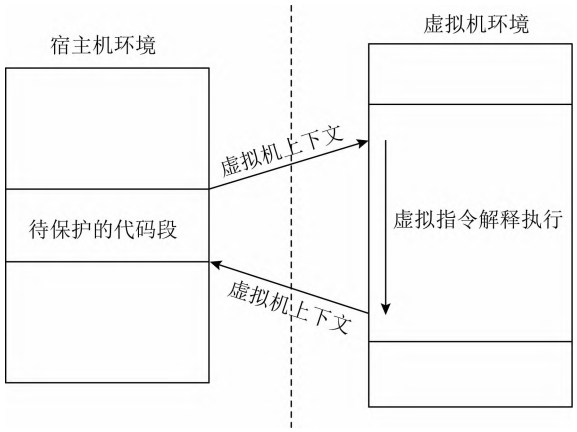


图 5　虚拟机结构

堆栈的形式进行计算，通过压栈和出栈实现对应的逻辑运算.从虚拟机的设计上分析，栈式虚拟机是简单易于实现的，因为没有寄存器的分配问题，简化了指令集的设计难度，如 VMProtect 的指令架构便是基于栈；基于寄存器的虚拟机，使用类似精简指令集的形式，模拟出原生的指令功能，因此，在单条指令的长度上长于基于栈的指令，但实现相同的功能点时，对应的总体指令数量会少于栈式实现.因此，文献[17]对 2 种实现方式进行对比，通过实验证明，基于寄存器的解释器相较于基于栈的实现方式，能有效减少虚拟指令数量，获得更小的运行时间开销.

在指令集的设计上，考虑到虚拟机本身的开销较大[13,33]，以及因为一些原生指令的使用频率较低，如 x86 架构的一些平台相关的指令，鉴于对原生指令全部模拟的不必要性，所以一般自定义的指令并不是图灵完备的，因此必然涉及虚拟机环境和宿主机真实环境的切换，包括寄存器、堆栈信息和标志位的状态切换.

2.2　虚拟机解释器

虚拟机解释器和虚拟机指令集是 VMP 的核心，整体的安全性和执行效率均受解释器影响.虚拟机解释器是以虚拟机指令为输入，使用解释器内部的逻辑处理实现对应的原生语义.内部的处理方式归结于 2 个主要组件：分发器(dispatcher)和处理例程[34](handler，也称之为原子处理函数[56]).基于程序指令的取指-译码-执行的模式，当解释器面对接收到的虚拟指令时，首先通过 dispatcher 找到对应的 handler 处理单元，handler 再从虚拟机上下文环境中提取对应的数据，完成相应的逻辑处理[41].

现有 dispatcher 的设计可以分为 2 大类：基于译码-执行(decode-dispatcher)的方式和基于线索化的方式[17,57].前者的架构图如图 6(b)所示，解释器接收到虚拟机指令，通过分发器进行译码，确定对应的 handler 进行处理执行，重复这个过程，直至处理完虚拟机指令或者遇到错误，记录此时的虚拟机上下文中的数据，从虚拟机环境返回到宿主机环境，并重新对程序的堆栈、寄存器和标志位等信息进行赋值.一般的实现方式为 while 循环内部嵌套 switch 结构，通过 switch case 进行有效的分发，好处是具有跨平台性，移植性更好，但对应的是处理逻辑上因为使用 switch 存在较多的跳转，包括 dispatcher 和 handler 之间的跳转，原生程序同 dispatcher 之间的跳转.所以为了提高程序的执行效率，另一种实现方式是使用线索化(threaded).

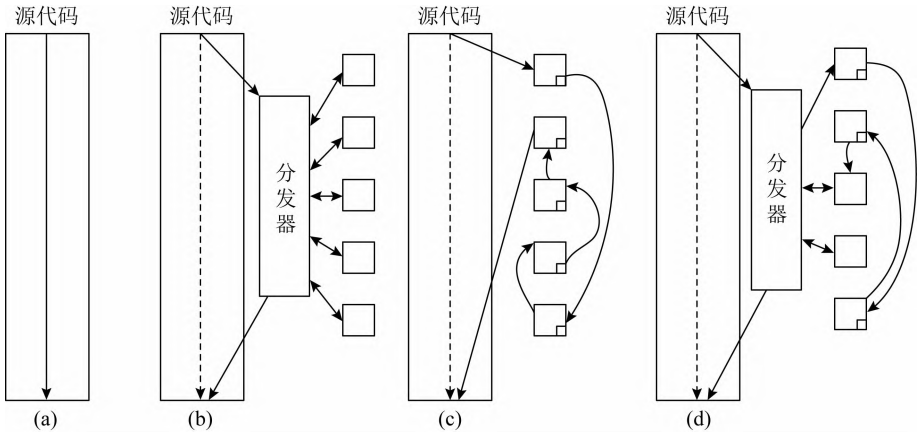


图 6　dispatcher 类别

基于线索化的解释器是将原本 dispatcher 内的间接跳转替换为 handler 尾部添加的直接跳转，在执行完对应的 handler 逻辑后，直接跳转到下一个 handler 进行处理，为用空间换取时间的策略，VMProtect 中便有该技术的使用[42]，如图 6(c)所示.但对于复杂的指令序列，涉及多 handler 处理的情况下，这种空间开销反而成为一种负担，因此出现了混合式，即对于简单的指令序列使用线索化的方式，对于复杂的指令使用分发的方式，即文献[56]中提到的分派式加链式结构，亦即图 6(d)所示.

3 虚拟机安全性分析

VMP 保护的安全性涉及 2 种安全实体：首先是保护后的程序是否足够安全，以抵抗 MATE 等攻击形式；其次是 VMP 保护程序本身是否足够安全、是否可以抵抗外在的分析攻击.应当说，前者的安全是后者安全的超集，因为 VMP 还可以结合其他保护方法进行效果的强化，如结合加密[18,30-32,35]，防篡改[37]，但 VMP 自身软件的安全性对于程序的整体安全性而言是重要的.

3.1 针对虚拟机的攻击方式

目前已有的针对虚拟机的破解方法主要有 2 类：1)基于虚拟机结构；2)基于语义分析.

基于虚拟机结构的破解方式指代的是利用虚拟机本身的结构特性，如译码-执行模式下 handler 与 dispatcher 的跳转关系，识别出具体的 handler 范围；使用 switch 对应的跳转变量识别出虚拟机的执行流[43]；使用虚拟机上下文试图推导虚拟机在切换过程中的数据流等措施[44].相应地，也有众多的文献[9,19,22-23]基于上述的分析手法进行了加强保护.

Rolles[53]首次提出基于虚拟机结构的攻击，分为 6 步逐步将 x86 架构的虚拟机代码还原为 x86 指令：1)对虚拟机进行逆向分析.获取虚拟机的执行语义，并构建等价的中间语言.这个过程只执行 1 次.2)检测虚拟机的入口.3)构建反汇编器.识别基于同套指令模板的指令集异同，并使用正则匹配约简指令的差异.4)将虚拟机操作码转变为中间代码.5)对生成的中间代码进行优化处理.6)生成 x86 代码.虽然是基于 VMProtect 的软件特性设计了具体的步骤，但是提供了一种破解虚拟机的范式.类似地，文献[54]也是基于虚拟机结构对 VMP 进行静态分析，以实现功能上的破解.

基于语义分析的攻击[45]，不需要针对虚拟机的架构有前提假设，以程序中的数据流和控制流为基准，使用动态的符号执行、污点分析跟踪程序中变量的执行流[46-47]，从而实现虚拟机指令与实际指令映射关系的重构.Coogan[48]基于恶意软件必须使用系统调用这一前提，针对虚拟化后的软件，以系统调用的相关参数为基准，对其相关的指令进行递归处理，从而实现对原生程序中重要指令的识别.特别地，在指令的识别过程中，基于用户-使用链(use-define chains)，为了减小指令对解释器的依赖作用，先通过对虚拟化的指令进行数据分析，再基于识别出的指令结合汇编级指令语义的方程式推理进行控制流分析.不同于其他文章力求还原出源程序的结构，该文在对解释器无任何前提假设的情况下，致力于有效识别出程序中重要的指令.文献[49]针对 switch 分发模式的虚拟机保护，使用前后向的数据流分析、污点分析对虚拟机保护后的程序进行分析，通过识别变量、归类变量和对指令进行语义约简实现将自定义的指令逐步还原为 x86 指令，并构造出程序控制流图.同时考虑到现有的 VMP 安全性增强实现中，对于符号执行会有一定的抵抗作用，因此一些攻击方案中针对符号执行作了强化处理，使用符号执行结合时间戳的实现方案进行破解[50]；文献[51]借助于编译器的作用从而实现对虚拟指令的有效约简.需要注意的是基于语义的分析并不只是动态的分析方式，也可以结合静态方法以提高指令的识别效率[52].

除了上述提到的方法外，针对 VMP 的破解方式也有结合频率分析的，如文献[44]提出针对虚拟机的频率攻击，实现对虚拟指令和本地指令映射关系的还原；但对于频率分析的使用场景，受限于指令映射的复杂性，正如文献[12]指出的，如果对指令进行分割、合并或者重复等混淆操作，频率分析便会受到一定的冲击.同时，在探究 VMP 安全性时，更多的文章关注的是机密性，而忽视了其完整性，而文献[55]对此通过实验验证，现有的 VMP 完整性方面易受攻击.

3.2 解释器:一个核心

如果 VMP 保护的程序可以脱离解释器进行执行,则自定义指令集的优势必然消失,因为可以借助其他语言,如中间语言,重新优化代码组织形式,建立更直接的接近于宿主机指令的映射关系,因此,如果可以绕过解释器对虚拟机指令进行转换或者去除,本质上相当于脱壳后的程序.因此,众多的文献研究重点在于加强解释器的安全性.而解释器由 dispatcher 和 handler 构成,进而安全性的加强落于这两者安全性的增强.进一步讲,解释器实现的任务是对约定的映射关系进行有效的处理,所以这种保护又可以归于 2 阶段的映射关系,即虚拟指令和 dispatcher 之间的映射,dispatcher 和 handler 之间的映射,因此众多的文章在指令层级实现了多种增强方案.如文献[24]提出设计防篡改指令、反调试指令以增强指令的安全性.文献[27-29]均从指令多样化和随机化入手,以期复杂化映射关系,提高安全性.另一方面,为了提高抵抗污点分析和符号执行的能力,文献[25]通过提出使用污点漂白和异常机制处理等针对性的设计.也有结合有限状态机实现对指令语义进行隐藏[33]的处理方案.

同时,因为指令层级的保护方案带来的一定开销,一些文章试图以基本块为基本单位进行保护[30-32],在兼顾效率的同时,使用块级的加解密提供更高的安全性.以及牺牲一部分性能,使用嵌套虚拟机[38-39]或多套虚拟机解释器[34]的实现方式,而为了更有效地隐藏解释器内的映射关系,文献[36]提出不依赖于混淆,而是通过将分发器隐藏在 CPU 中实现对 dispatcher 的有效隐藏,从而实现对现有虚拟机破解方法的抵抗.

目前国内外的文献所讨论的虚拟机方案大多作用于 x86 架构,有少数文章致力于跨平台的实现方案,如文献[40]提出基于 LLVM 实现虚拟机,从而实现跨平台性.

最后需要补充说明的是,在软件保护及相关的应用[20-21]中,VMP 的应用领域不仅可以针对可执行文件,也可以如源代码层级的保护,如针对 C#,在源码层级实现了虚拟化混淆的工具[58];对 JavaScript 代码进行虚拟机保护,将 JavaScript 代码转变为 WebAssembly,然后针对 WebAssembly 代码进行虚拟化处理[59]等方案,在实现安全的解释器的同时提供可靠的虚拟机安全性.

4 未来研究方向

作为软件保护技术的 VMP,其未来的研究方向可以归为以下几点:

1) 统一的评估指标.虽然文献[12]从预期效果、使用方法的类别、自动化程度和方法的通用性层面对已有的虚拟机破解方法给出了一种评判标准,但针对虚拟机保护还没有形成统一的评判标准.现有的众多的方案设计中,仅从软件的可用性、时空开销或者单一的 handler 数量进行效果的评定,缺乏整体性和针对性.借鉴代码混淆的评估指标或者密码学的形式化证明,可能更有助于效果的评估.

2) 开销和保护的粒度.结合硬件加密或软件加密,一定程度上取得了更高的安全性.而如果将 VMP 本身的处理逻辑视为一种加解密方式,嵌套虚拟机便是加密算法的叠加使用.但两者均面临程序带来的不可忽视的运行开销.合理地选择保护的粒度,如指定保护范围,针对基本块而非每条指令,可降低开销,但对应的安全性是否仍足以抵抗现有的语义攻击,粒度的选择上对安全性的影响还需具体的实验论证.

3) 同机器学习的结合.VMP 本身的优势在于自定义指令集的难理解,如果可以有效利用机器学习生成指令集和解释器,利用其本身的不可解释性,将是 VMP 未来自动化和市场化中可以研究的方向.

4) 跨平台性.目前的软件和硬件平台愈发呈现出多样化的趋势,针对于具体的平台进行逐个开发的难度和时间成本都不容小觑,因此,借助于 LLVM 等平台,利用中间语言的特性实现跨平台的方案存在具体的实际需求.但如何在缺失平台特性的情况下,开发高可用的 VMP 仍是目前的难点.

5 结 论

虚拟机软件保护有助于提高软件的安全性,防止被恶意的静态和动态分析.基于国内外目前的

研究成果，本文首先论述了目前虚拟机保护面临的问题，包括评估指标和优化方法上的创新局限性，随后介绍了虚拟机结构，并通过引述文献，分析了其安全性，最后总结现有成果，对未来进行了展望.

参 考 文 献

[1] Campbell-Kelly M, Aspray W, Ensmenger N, et al. Computer: A History of the Information Machine [M]. New York: Taylor and Francis, 2018: 1-360

[2] 张丽娜，阎文斌. 基于虚拟机的软件保护研究与设计[J]. 计算机工程与应用，2012，48(26)：66-70，161

[3] 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

[4] Blunden B. Virtual Machine Design and Implementation in C/C++ with Cdrom [M]. Texas, USA: Wordware Publishing Inc., 2002: 1-500

[5] Smith J E, Nair R. Virtual Machines: Versatile Platforms for Systems and Processes [M]. 北京：机械工业出版社，2006：1-382

[6] Collberg C S, Thomborson C. Watermarking, tamper-proofing, and obfuscation-tools for software protection [J]. IEEE Trans on Software Engineering, 2002, 28(8): 735-746

[7] Akhunzada A, Sookhak M, Anuar N B, et al. Man-at-the-end attacks: Analysis, taxonomy, human aspects, motivation and future directions [J]. Journal of Network and Computer Applications, 2015, 48: 44-57

[8] Chow S, Eisen P, Johnson H, et al. A white-box DES implementation for DRM applications [G] //LNCS 2696: Digital Rights Management. Berlin: Springer, 2003: 1-15

[9] Wang H, Fang D, Li G, et al. NISLVMP: Improved virtual machine-based software protection [C] //Proc of the 9th Int Conf on Computational Intelligence and Security. Piscataway, NJ: IEEE, 2013: 479-483

[10] Wang H, Fang D, Dong H, et al. Research on software protection defending dynamic attack in white-box environment [J]. Acta Electronica Sinica, 2014, 42(3): 529-537

[11] Maude T, Maude D. Hardware protection against software piracy [J]. Communications of the ACM, 1984, 27(9): 950-959

[12] Kochberger P, Schrittwieser S, Schweighofer S, et al. SoK: Automatic deobfuscation of virtualization-protected applications [C] //Proc of ACM Int Conf Proceeding Series. New York: ACM, 2021: 1-15

[13] Collberg C, Thomborson C, Low D. A taxonomy of obfuscating transformations [R]. Arizona: The University of Auckland, New Zealand, 1997

[14] VMProtect Software. VMProtect software protection[EB/OL]. (2022-01-01) [2022-02-20]. https://vmpsoft.com/

[15] Oreans Technologies. Oreans: Themida overiew [EB/OL]. (2022-01-01) [2022-02-20]. https://www.oreans.com/Themida.php

[16] Oreans Technologies. Oreans: Code virtualizer overview [EB/OL]. (2022-01-01) [2022-02-20]. https://www.oreans.com/codevirtualizer.php

[17] Wang H, Fang D, Li G, et al. TDVMP: Improved virtual machine-based software protection with time diversity [C] //Proc of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014. New York: ACM, 2014: 1-9

[18] Fang H, Wu Y, Wang S, et al. Multi-stage binary code obfuscation using improved virtual machine [C] //Proc of the 14th Int Conf on Information Security. Berlin: Springer, 2011: 168-181

[19] Shi Y, Casey K, Ertl M A, et al. Virtual machine showdown: Stack versus registers [J]. ACM Trans on Architecture and Code Optimization, 2008, 4(4): 1-36

[20] Fang D, Gao L, Tang Z, et al. A software protection framework based on thin virtual machine using distorted encryption [C] //Proc of the 2011 Int Conf on Network Computing and Information Security. Piscataway, NJ: IEEE, 2011: 266-271

[21] Xie X, Liu F, 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

[22] 雷远晓. 针对虚拟机软件保护的攻击方法研究[D]. 西安：西北大学，2013

[23] Cheng X Y, Lin Y, Gao D B, et al. DynOpVm: Vm-based software obfuscation with dynamic opcode mapping [C] //Proc of the 17th Int Conf on Applied Cryptography and Network Security (ACNS). Berlin: Springer, 2019: 155-174

[24] Zaleski M, Brown A D, Stoodley K. YETI: A graduallY extensible trace interpreter [C] //Proc of the 3rd Int Conf on Virtual Execution Environments. New York: ACM, 2007: 83-93

[25] Blazytko T, Contag M, Aschermann C, et al. Syntia: Synthesizing the semantics of obfuscated code [C] //Proc of the 26th USENIX Security Symp (USENIX Security 17). Berkeley, CA: USENIX Association, 2017: 643-659

[26] Suk J H, Lee D H. VCF: Virtual code folding to enhance virtualization obfuscation [J]. IEEE Access, 2020, 8: 139161-139175

[27] Lee J Y, Suk J H, Lee D H. VODKA: Virtualization obfuscation using dynamic key approach [C] //Proc of the 19th World Int Conf on Information Security and Applications(WISA). Berlin: Springer, 2018: 131-145

[28] Kiperberg M, Leon R, Resh A, et al. Hypervisor-based protection of code [J]. IEEE Trans on Information Forensics and Security, 2019, 14(8): 2203-2216

[29] Gan J, Kok R, Kohli P, et al. Using virtual machine protections to enhance whitebox cryptography [C] // Proc of the 1st Int Workshop on Software Protection. Piscataway, NJ: IEEE, 2015: 17-23

[30] Ghosh S, Hiser J D, Davidson J W. A secure and robust approach to software tamper resistance [C] // Proc of the 12th Int Conf on Information Hiding. Piscataway, NJ: IEEE, 2010: 33-47

[31] Xu D, Ming J, Fu Y, et al. VMHunt: A verifiable approach to partially-virtualized binary code simplification [C] // Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2018: 442-458

[32] Xie H, Zhang Y, Li J, et al. Nightingale: Translating embedded vm code in x86 binary executables [G] //LNCS 10599: Proc of Int Conf on Information Security. Berlin: Springer, 2017: 387-404

[33] Xie X, Liu F, Lu B, et al. Virtual machine protection based on handler obfuscation enhancement [J]. Computer Engineering and Application, 2016, 52(15): 146-152

[34] Kuang K Y, Tang Z Y, Gong X Q, et al. Enhance virtual-machine-based code obfuscation security through dynamic bytecode scheduling [J]. Computers & Security, 2018, 74: 202-220

[35] Kuang K, Tang Z, Gong X, et al. Exploiting dynamic scheduling for vm-based code obfuscation [C] //Proc of the 2016 IEEE Trustcom/BigDataSE/I SPA. Piscataway, NJ: IEEE, 2017: 489-496

[36] Rolles R. Unpacking virtualization obfuscators [C] //Proc of the 3rd USENIX Conf on Offensive Technologies. Berkeley, CA: USENIX Association, 2009

[37] Kinder J. Towards static analysis of virtualization-obfuscated binaries [C] //Proc of Working Conf on Reverse Engineering, WCRE. Piscataway, NJ: IEEE, 2012: 61-70

[38] Salwan J, Bardin S, Potet M L. Symbolic deobfuscation: From virtualized code back to the original [G] //LNCS 10885: Proc of Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2018: 372-392

[39] Yadegari B, Johannesmeyer B, Whitely B, et al. A generic approach to automatic deobfuscation of executable code [C] //Proc of the 2015 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2015: 674-691

[40] Lin J, Liu C, Zhang X, et al. VMRe: A reverse framework of virtual machine protection packed binaries [C] //Proc of the 4th IEEE Int Conf on Data Science in Cyberspace(DSC). Piscataway, NJ: IEEE, 2019: 528-535

[41] Coogan K, Lu G, Debray S. Deobfuscation of virtualization-obfuscated software: A semantics-based approach [C] //Proc of the 18th ACM Conf on Computer and Communications Security. New York: ACM, 2011: 275-284

[42] Sharif M, Lanzi A, Giffin J, et al. Automatic reverse engineering of malware emulators [C] //Proc of the 30th IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2009: 94-109

[43] Li H, Zhan Y, Jianqiang W, et al. SymSem: Symbolic execution with time stamps for deobfuscation [G] //LNCS 10631: Proc of Int Conf on Information and Communications Security. Berlin: Springer, 2020: 225-245

[44] Liang M, Li Z, Zeng Q, et al. Deobfuscation of virtualization-obfuscated code through symbolic execution and compilation optimization [G] //LNCS 12020: Proc of Int Conf on Information Security and Cryptology. Berlin: Springer, 2018: 313-324

[45] Kalysch A, Götzfried J, Müller T. VMAttack: Deobfuscating virtualization-based packed binaries [C] // Proc of the 12th Int Conf on Availability, Reliability and Security. New York: ACM, 2017: Article No.2

[46] Ghosh S, Hiser J, Davidson J W. Replacement attacks against vm-protected applications [J]. ACM Sigplan Notices, 2012, 47(7): 203-214

[47] Tang Z, Li M, Ye G, et al. VMGuards: A novel virtual machine based code protection system with vm security as the first class design concern [J]. Applied Sciences (Switzerland), 2018, 8(5): 771-794

[48] Anckaert B, Jakubowski M, Venkatesan R. Proteus: Virtualization for diversified tamper-resistance [C] //Proc of the ACM Workshop on Digital Rights Management. New York: ACM, 2006: 47-58

[49] Sovarel A N, Evans D, Paul N. Where's the FEEB? the effectiveness of instruction set randomization [C] //Proc of the 14th USENIX Security Sysposium. Berkeley, CA: USENIX Association, 2005: 10-16

[50] 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

[51] 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]. 工程科学与技术，2017, 49(1): 159-168

[52] Ghosh S, Hiser J D, Davidson J W. Matryoshka: Strengthening software protection via nested virtual machines [C] //Proc of the 1st Int Workshop on Software Protection. Piscataway, NJ: IEEE, 2015: 10-16

[53] 杨明, 黄刘生. 一种采用嵌套虚拟机的软件保护方案[J]. 小型微型计算机系统, 2011, 32(2): 237-241

[54] Averbuch A, Kiperberg M, Zaidenberg N J. An efficient vm-based software protection [C] //Proc of the 5th Int Conf on Network and System Security. Piscataway, NJ: IEEE, 2011: 121-128

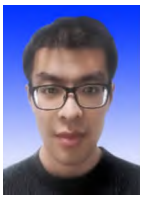
[55] 张晓寒, 张源, 池信坚, 等. 基于指令虚拟化的安卓本地代码加固方法[J]. 电子与信息学报, 2020, 42(9): 2108-2116

[56] 孙伟, 薛临风, 张凯寓, 等. 软件合法性保护技术及应用研究综述[J]. 信息安全研究, 2017, 3(5): 451-461

[57] 贺江敏, 相里朋. 代码安全性审查方法研究[J]. 信息安全研究, 2018, 4(11): 977-986

[58] Banescu S, Lucaci C, Krämer B, et al. VOT4CS: A virtualization obfuscation tool for C# [C] //Proc of the 2016 ACM Workshop on Software PROtection. New York: ACM, 2016: 39-49

[59] Wang W, Li M, Tang Z Y, et al. Invalidating analysis knowledge for code virtualization protection through partition diversity [J]. IEEE Access, 2019, 7: 169160-169173



李成扬

硕士研究生.主要研究方向为代码混淆、代码保护.

lcymoon@pku.edu.cn

陈夏润

硕士研究生.主要研究方向为漏洞挖掘、软件安全防护.

xiar_c@pku.edu.cn

张　汉

硕士.主要研究方向为移动安全.

zhanghanpku@pku.edu.cn

文伟平

博士,教授,博士生导师.主要研究方向为系统与网络安全、大数据与云安全、智能计算安全.

weipingwen@pku.edu.cn