嵌入式反病毒虚拟机①

Built-In Anti-Virus Virtual Machine

吴晓丹 李 曦 陈香兰 (中国科学技术大学 计算机科学与技术系 安徽 合肥 230027)

摘 要: 通过对传统反病毒虚拟机运行机制的分析，结合应用程序实际运行特征，指出传统反病毒虚拟机将执行部件包含在虚拟机内部的结构设计对虚拟机的模拟执行能力、实现代价和运行效率等多个方面造成的影响。继而提出将执行部件从虚拟机内部剥离的思想，并以 x86 体系结构 WINDOWS 操作系统环境为例，设计了嵌入式反病毒虚拟机，使传统反病毒虚拟机的结构得到简化、实现代价得到降低、扩展能力得到增强。

关键词: 虚拟机 嵌入式 动态翻译 反病毒 多线程 结构化异常

1 引言

随着病毒变形技术的不断发展，单一的特征码扫描和 API 调用静态分析已经不能满足反病毒应用的需要，越来越多的反病毒人员开始使用虚拟机技术来完成变形病毒的解密还原[1]和入侵行为分析[2]。尽管虚拟机技术[3]从 20 世纪 60 年代的 VM/370 发展至今已近半个世纪，但真正的反病毒虚拟机还处于起步阶段。

本文首先对传统反病毒虚拟机的运行机制进行了分析，针对反病毒虚拟机控制和执行部件未分离，严重影响反病毒虚拟机模拟效果和实现代价的现状，提出了将执行部件从虚拟机内部剥离的设计思路，并完成了 x86 体系结构 WINDOWS 环境下嵌入式反病毒虚拟机的设计。通过对嵌入式反病毒虚拟机上异常处理和多线程扩展实现方式的介绍，展示了嵌入式反病毒虚拟机在扩展能力上的优势。最后对嵌入式反病毒虚拟机进行总结并提出有待进一步研究的问题。

2 传统反病毒虚拟机运行机制分析

传统反病毒虚拟机，基于变形病毒一般首先运行解密循环、解密循环不超过 256 条指令的前提[1,4]，在设计时仅考虑对常用 CPU 指令进行模拟。虚拟机在对客户程序进行模拟运行的过程中，通过指令循环检测、内存修改检测和 API 调用监测等方法，发现解密循环，判断解密过程结束。

客户程序解密过程结束之后，虚拟机将对其进行已知病毒特征码扫描。考虑到病毒可能采用多层加密(如 hezi 病毒采用了 5 层加密变形)，因此每次虚拟机在完成对解密循环的模拟执行之后，将继续模拟运行 256 条指令，如发现解密循环，则继续对解密循环进行模拟执行，否则退出模拟执行开始已知病毒特征码扫描。传统反病毒虚拟机执行流程如图 1 所示。

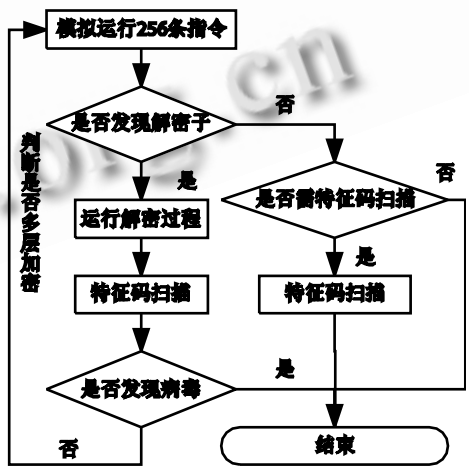

图 1 传统反病毒虚拟机执行流程图

虚拟机对指令模拟执行的实现方法有两种[4]:

① 利用操作系统提供的对应用程序的调试功能，创建调试进程，采用单步或断点跟踪法，捕获每一条

① 收稿时间:2009-03-26

指令执行前后的寄存器和内存的变化情况。

② 软件模拟 CPU。软件模拟的 CPU 具备和真实 CPU 一样的功能，能完成机器指令的取指、译码和执行。

由于在单步执行模式下 CPU 标志寄存器中的 TF 位会被置 1，断点跟踪时调试器需要在断点位置进行指令改写，因此客户程序只要简单检测 CPU 标记位、代码段的指令内容就能判断是否处于被调试状态，或者直接调用 IsDebugPresent 函数进行判断。基于上述原因，反病毒虚拟机对指令的模拟执行主要采用第二种方式。

软件模拟的 CPU 在完成取指、译码之后，进行模拟执行时，必须保证：

① 指令效果的正确模拟。

② 执行流程的正确转移。

③ 系统和用户信息在模拟执行过程中的安全。

上述 3 点，①被称为“指令的执行”，②、③则被称为“指令执行的控制”。

从计算机体系机构图(图 2)中可以看到，应用程序的运行除了执行硬件(CPU)的执行能力外，还需要操作系统提供的运行支持。

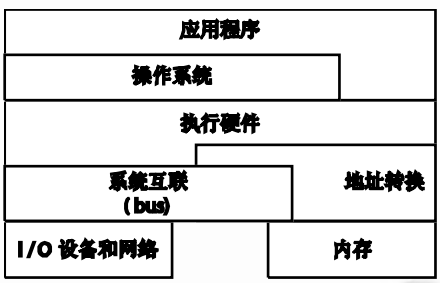

图 2　计算机体系结构图

传统反病毒虚拟机基于变形病毒发展初期的特征，仅实现对了 CPU(执行硬件)功能的模拟。由于只考虑应用程序、CPU、操作系统三者中应用程序和 CPU 的关系，传统的反病毒虚拟机将“指令的执行”作为软件模拟 CPU 的组成部分实现在虚拟机内部。相应的，执行部件的操作对象(寄存器和内存空间)也由虚拟机在其内部进行模拟或映射。

尽管在计算机体系机构图中，操作系统是位于硬件之上直接为应用程序提供服务的，但应用程序在实际使用操作系统的服务时，都必须借助 CPU 中断指令进行跳转。传统反病毒虚拟机将执行部件及其操作对象都封闭在虚拟机内部的设计，阻断了客户程序和操作系统之间的联系，使得反病毒虚拟机必须代替操作系统在模拟运行过程中向客户程序提供必要的运行支持。

传统的反病毒虚拟机在扩展支持异常处理机制时，将异常处理过程中的异常识别和异常处理回调函数跳转都简化的实现在虚拟机内部。由于异常的触发，并非仅仅是被 0 除、异常地址访问这么简单，任何错误的指令、计算结果的上溢和下溢、精度问题、甚至字节对齐都有可能导致异常的触发。而是否需要调用相应的(中断)处理函数，还需要对不同的标记位进行比较分析。因此，在反病毒虚拟机内部实现对异常的识别，最终导致两种结果：

① 能对所有异常进行有效识别，但需要在指令模拟执行前后进行大量检测，这样的设计实现复杂而且运行效率较低。

② 能对少数常用的异常进行识别，但当客户程序中包含其他异常时，则不能进行识别。

病毒技术和反病毒技术的发展总是互相追赶的，随着反虚拟机技术的不断发展[5]，多线程支持也逐步纳入反病毒虚拟机的扩展方向。不难想象，反病毒虚拟机在对线程创建等 API 和系统调用提供支持的同时，还需要对线程间通信和线程调度提供模拟。由于客户程序被虚拟机完全封闭，操作系统不能为客户程序创建线程相关的数据结构，因此线程数据结构的创建和其他功能的模拟都只能由虚拟机来完成。模拟的完备性、实现的代价和运行效率等问题将再次出现。

3　执行分离的反病毒虚拟机

执行分离的反病毒虚拟机，改变传统反病毒虚拟机控制和执行都在虚拟机内部完成的设计思路，在对应用程序执行环境、CPU 和操作系统运行机制等多方面因素综合考虑的基础上，将虚拟机模拟执行部件从虚拟机内部分离出来，把执行任务交给操作系统和真实 CPU。这样的设计简化了虚拟机内部结构，降低了扩展难度和实现代价。

反病毒虚拟机对客户程序的控制能力体现执行流程的控制和执行效果的控制：

① 流程的控制。模拟执行时客户程序中所有执行流程的跳转都在虚拟机的监控之下。

② 效果的控制。客户程序中所有恶意的行为不会

在模拟执行时对系统和用户信息造成损害。

执行部件的分离，意味着客户程序代码是运行在真实 CPU 之上的。如何确保虚拟机对客户程序在执行流程和效果的控制？指令动态翻译是解决的办法。所有的客户程序代码在被 CPU 运行之前，都必须经过一次代码翻译。由于动态翻译并不将翻译结果作用在客户程序代码上，因此不会对客户程序的代码内容和结构造成影响。

根据指令操作对象的不同，CPU 指令动作可以被划分为：普通寄存器操作、内存操作、控制跳转操作、I/O 操作、标志寄存器操作、段寄存器操作等。

通过对上述 6 类指令动作的动态翻译，反病毒虚拟机即可实现对客户程序模拟执行流程和效果的完全控制。

(1) 普通寄存器操作。普通寄存器包括通用寄存器、浮点寄存器、MMX 寄存器和 XMM 寄存器。客户程序对于这些寄存器的操作，并不会直接对系统和用户信息造成损害，因此对它们并不做特别的指令翻译。

(2) 内存操作。内存是操作系统和应用程序的活动空间，尽管操作系统和用户数据都是存放在磁盘等存储设备上的，但它们在运行和使用时都必须首先被加载到内存中，因此内存操作是病毒程序破坏操作系统、窃取用户信息的最主要途径。由于并非所有的内存操作都会造成对系统的破坏或者用户信息的泄露，因此对内存操作的动态翻译时，将放过目标地址为普通安全级别的内存操作，对于目标地址为重要安全级别的内存操作，虚拟机会在指令翻译时修改其目标地址，将操作转移到安全的内存区域。

(3) 控制跳转操作。跳转的目标有两种：已知功能代码和未知功能代码。对于到已知功能代码的跳转，虚拟机通过对跳转目标和参数内容的分析即可确定该跳转是否有害，从而决定是否执行。对于到未知功能代码的跳转，虚拟机需要继续对跳转之后的代码进行控制，因此在指令翻译时，虚拟机把客户程序指令流程的转移翻译为虚拟机模拟执行目标的转移。

(4) I/O 操作。I/O 操作是用户信息泄露的另一途径，由于 I/O 操作可以被映射为内存操作，因此对 I/O 操作的翻译，将等同安全级别为重要的内存操作。

(5) 标志寄存器操作。同普通寄存器。

(6) 段寄存器操作。同普通寄存器。

由于 CS 和 EIP 的特殊性，虚拟机将创建专门的对象(VCS 和 VIP)。VCS 和 VIP 受客户程序指令的影响，而 CPU 中真实 CS 和 EIP，则只受虚拟机模拟运行流程的控制。

4　嵌入式防病毒虚拟机

下面以 x86 体系结构 WINDOWS 环境为例，讨论嵌入式反病毒虚拟机的构造和工作机制。

在 WINDOWS 环境下，创建一个进程，需要经历如下的六个主要阶段：

① 打开将要在该进程中执行的映像文件。

② 创建 WINDOWS 执行体进程对象。

③ 创建初始线程。

④ 通知 WINDOWS 子系统新进程已经创建，为新进程和线程做好准备。

⑤ 开始执行初试线程。

⑥ 在新进程和线程环境中，完成地址空间的初始化，并开始执行程序。

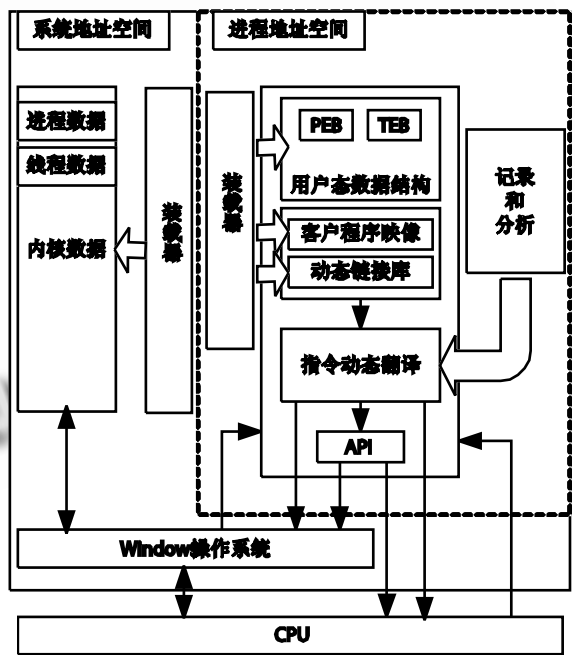

图 3　嵌入式反病毒虚拟机结构图

在嵌入式反病毒虚拟机中，客户程序不是以新进程的形式被装载的，而是映射客户程序文件到虚拟机内存空间，通过修改上述过程中相关的数据结构，将虚拟机伪装成需要模拟运行的客户程序。在运行过程中，虚拟机如同嵌入在客户程序体内的控制器，因此称之为嵌入反病毒式虚拟机。

嵌入式反病毒虚拟机包括装载器、指令动态翻译和记录分析三个部分(图 3),本文仅对前两个部分进行讨论。

4.1 装载器

嵌入式反病毒虚拟机的装载器分为用户态和内核态两部分，分别完成进程地址空间和系统地址空间下的数据装载和维护。

虚拟机实现对客户程序正确、真实模拟的前提是完成映像文件到内存空间的正确加载。尽管 PE 文件格式提供了对 relocation 的支持，但并非所有的应用程序都对其进行了设置。另一方面，病毒在感染宿主程序时也不一定会对自身数据和代码进行 relocation 设置。因此用户态的装载器必须将映像文件加载到 ImageBase 指定的映射地址。

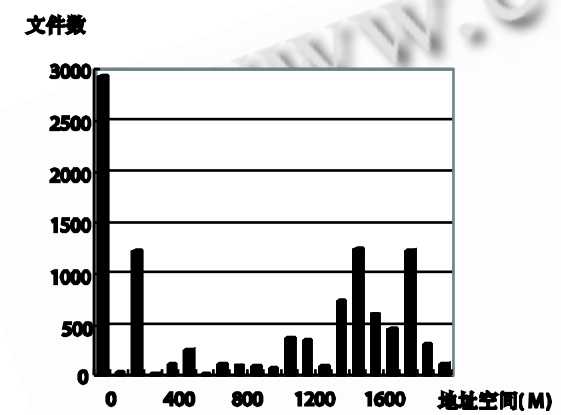

图 4　PE 文件内存映射空间分布统计图

通过分析 10436 个 PE 文件，得到内存映射空间分布统计图(图 4)和以下结论：

(1) 在 2G 的进程地址空间内，每百 M 的地址区间都有被程序映射使用的记录。为了保证客户程序被加载到 ImageBase，虚拟机可能需要进行自身的搬移。

(2) 程序对内存空间的映射区间分布严重不均衡，通过为虚拟机选择适当的 ImageBase，可以从一定程度上减少虚拟机自身搬移的几率。

装载器在开始映射映像文件前，首先对映像文件进行扫描，获取 ImageBase 和段表信息，根据客户程序将要占用内存的地址范围确定是否进行虚拟机自身搬移，之后再开始对映像文件的映射工作。对于客户程序将要使用到的动态链接库，装载器不会进行内存地址范围计算和自己搬移，因为动态链接库都对 relocation 数据进行了配置，不会出现因不能装载到默认映像基址而无法正常工作的情况。在扫描映像文件时，装载器还会获取客户程序堆栈大小等信息，对于超过虚拟机现有堆栈能力的，装载器会修改堆栈参数，重新启动虚拟机后再开始映像文件的映射。

完成对映像文件的映射之后，以驱动形式存在于内核态装载器，将系统地址空间内进程、线程数据结构中与虚拟机映像文件对应的信息修改为与客户映像文件对应，虚拟机映像相关信息被配置为客户程序加载的一个模块。

最后是用户态下，对执行环境进行初始化，完成 PEB、TEB 信息修改。包括堆、线程局部存储区数组的初始化，所有必要 DLL 的加载和导入表中 FirstThunk 所指向的 Import Address Table 中函数地址的计算等。

4.2 指令动态翻译

嵌入式反病毒虚拟机对客户程序的模拟运行是以指令动态翻译为核心的循环过程。虚拟机从客户程序代码内读取 VIP 所指向的指令，分析、翻译、封装、执行，然后开始下一条指令。在整个运行流程中，虚拟机为数据记录、病毒扫描和行为分析保留了接口，能够根据行为分析的需要在运行过程中对敏感信息进行记录，能够在扫描条件达成时调用病毒扫描工具。

嵌入式反病毒虚拟机通过开辟空间为客户程序维护 CPU 寄存器和运行栈的方式，在一个线程中运行两个线程的代码，虚拟机线程在内部通过切换寄存器内容(VCS 和 VIP 不会影响 CS 和 IP)和运行栈，实现虚拟机和客户程序的并行运行。尽管用户态下标志寄存器 EFLAGS 中的很多标记位是不能被设置的，但这些标记位所表示的线程调试状态、中断许可、特权域等属性对于虚拟机和客户程序都是相同的，在切换时并不需要进行修改，因此不会影响程序的正确运行。

没有将指令的翻译封装和执行分到两个线程运行的原因在于，程序中平均每执行 4 条指令中就有一条跳转指令，翻译线程在完成对条件跳转指令的翻译后，必须等待执行线程完成对跳转指令之前所有指令的执行、正确设置标志寄存器后，才能确定下一个基本块的地址；执行线程这时也将停下来等待翻译线程提供新的指令去执行。将两个线程融合为一个线程能降低

系统因线程频繁切换和通信带来的资源消耗，提升虚拟机的运行效率。

5 异常处理和多线程扩展

嵌入式反病毒虚拟机由于已经将执行部件剥离，因此在扩展时仅需要考虑虚拟机在模拟执行流程和效果上对客户程序的控制。

5.1 异常处理扩展

由于客户程序代码在翻译之后都会运行于真实的 CPU 之上，因此异常触发时将直接被 CPU 捕获，随后搜索并调用虚拟机对应的异常处理函数。因此在对异常处理进行扩展时，只需要确保虚拟机定义的异常处理函数位于异常处理函数链的最顶部，并愿意处理所有类型的异常。这样既保证了在指令执行过程中对所有异常的正确识别，又保证了客户程序的执行流程不脱离虚拟机的控制。当虚拟机异常处理函数获得控制权后，根据异常类型，在异常处理函数链中搜索匹配的处理函数，转移客户程序模拟执行的流程。

5.2 多线程扩展

嵌入式反病毒虚拟机在对多线程问题提供支持时，只要是完成多线程相关的系统调用和 API 调用的指令动态翻译实现。以 CreateThread 调用的翻译为例，反病毒虚拟机将客户程序压栈的 lpStartAddress 和 lpParameter 所指向内容封装进结构体，再将结构体地址重新赋给 lpParameter，而 lpStartAddress 将被赋值为虚拟机指令动态翻译执行循环的开始地址。客户程序创建线程运行 func 的行为被转变为嵌入式反病毒虚拟机创建新的指令动态翻译执行线程对客户程序 func 函数进行虚拟执行。线程的调度、同步和通信都由操作系统提供支持。

6 总结

本文对传统反病毒虚拟机运行机制进行分析，针对其结构上的不足，提出了将执行部件从虚拟机内部分离的思路，讨论了嵌入式反病毒虚拟机的构造和主要模块，并阐述了相关功能扩展的实现。嵌入式反病毒虚拟机的提出，解决了传统反病毒虚拟机内部结构复杂、模拟效果有限、扩展能力差等问题，为反病毒虚拟机的发展提供了新的思路和方法。

在嵌入式反病毒虚拟机的设计中，首先考虑的是客户程序的执行正确性，并未对执行效率进行细致的研究。回顾目前主要的反虚拟机技术不难发现，执行效率低也是影响着反病毒虚拟机应用范围的一个重要因素。另一方面，如何增加虚拟机对客户程序执行流程的主动控制，在模拟运行时对所有可能的运行情况进行有效覆盖，提供对客户程序更全面的分析，也是有待继续研究的内容。

参考文献

1 曾宪伟,张智军,张志.基于虚拟机的启发式扫描反病毒虚拟机.计算机应用与软件, 2005,9:125－126.

2 Laureano M, Maziero C, Jamhour E. Intrusion detection in virtual machine environments. Proc. of the 30th EUROMICRO Conference. September 2004.

3 Rosenblum M, Garfinkel T. Virtual machine monitors: Current technology and future trends. IEEE Computer, 2005,5:39－47.

4 卢勇.反病毒虚拟机的研究与实现[硕士学位论文].成都:电子科技大学, 2006.

5 刘运,殷建平,蒋晓舟.反虚拟机技术的分析与对策.计算机科学, 2004,31:597－599.