2012 年全国大学生信息安全竞赛

作品报告

作品名称：云环境下虚拟主机通信监控系统

参赛学校：

学院/系：

指导教师：周天阳

组　　长：张家齐

组　　员：王珏　亢鹏　张文涛

通信地址：

电　　话：15136873686

电子邮箱：zhangjqfriend@126.com

提交日期：2012-6-3

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用A4纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者也可以多加内容。

目　　录

第一章 摘要

当前互联网上云服务方兴未艾。而木马、病毒等恶意软件和DDoS攻击严重威胁云服务的安全。这些威胁的共同点是严重影响提供云服务的虚拟主机的正常运行，甚至使提供云服务的虚拟机实例崩溃，给使用云服务的用户带来了巨大的损失。

木马、病毒等恶意软件的网络入侵都伴随着恶意通信，DDoS攻击则是使用大量的网络连接瘫痪靶机。因此有效的通信监控能给云服务提供安全保障。而传统的通信监控工具由于权限较低，检测手段落后，受限于用户操作系统，已经不能有效地完成云环境下虚拟主机的通信监控功能。

硬件虚拟化技术的发展为云环境下虚拟主机通信监控提供了新的思路。云环境下虚拟主机通信监控系统利用VMM最高的特权级对虚拟机上的操作系统存在的网络连接进行完全监控。从而实现了隐蔽通信的检测及终止的功能，使得内核级Rootkit隐藏的通信能够被检测出来。从而有效保护了云服务的安全。

本系统不挂钩函数，不修改内核数据结构，保证了操作系统的完整性，稳定性。通过构建VMM，捕获网卡中断来监控虚拟主机的操作系统，在云环境下虚拟主机平台上实现了隐蔽通信检测和终止。利用硬件虚拟化的特权指令（VMCALL）和共享缓冲区的方式实现VMM和用户层之间的信息交互。

经实验测试表明，本系统能够在云环境下的虚拟主机中稳定运行，对虚拟主机的CPU和内存的性能影响较小。最重要的是能对虚拟主机的通信进行有效的监控，相较于当前主流的通信监控工具，技术优势明显，极好地适应了当前云计算安全领域的快速发展，在云服务应用越来越广泛的今天具有良好的市场应用前景。

第二章 作品介绍

2.1 研究背景

2006年3月，亚马逊（Amazon）推出弹性计算云（Elastic Compute Cloud；EC2）服务。2006年8月9日，Google首席执行官埃里克·施密特（Eric Schmidt）在搜索引擎大会（SES San Jose 2006）首次提出“云计算”（Cloud Computing）的概念。自此以来，云计算以及由云计算提供的服务即云服务在全球迅速发展。互联网流量调查机构DeepField公布的最新调查报告显示，亚马逊云服务的互联网使用流量占整个北美互联网流量的1%，1/3的互联网用户每天至少会使用一项亚马逊云服务功能。云服务已成为互联网中重要的一部分，但是互联网上充斥着各种安全威胁。而云服务提供商对其服务的安全性却关注不多。2011年著名云服务提供商亚马逊出现故障，包括Fousquare和Reddit在内的主要网站在周四都面临着网页崩溃或是访问速度过慢等问题，导致成千上网的用户无法访问他们喜爱的网站。不仅普通互联网用户无法上网，使用亚马逊云服务的网站更是遭受了巨大的损失。

针对云服务的攻击手段除了传统的木马和病毒外更主要的是近年来兴起的DDoS（分布式拒绝服务）攻击。DDoS攻击的基本原理是控制大量分布在互联网各处的主机，使其同时向攻击目标发送数据包，导致攻击目标死机或是无法响应正常的请求。可见要防范DDoS攻击，有效的通信监控是关键。

目前，存在用户态和内核态两种主机通信监控的技术。

在用户态下进行网络数据包的拦截有三种方法：Windows包过滤接口、WinsockLayeredServiceProvider(LSP)、替换系统自带的WINSOCK动态链接库。使用这些方法进行数据包拦截最致命的缺点就是只能在Winsock层次上进行，而对于网络协议栈中底层协议的数据包无法进行处理。因此，这些方法并不适合主机通信监控。

内核态技术主要是通过挂钩操作系统的TCP/IP驱动栈实现的。主要有TDI过滤驱动程序、NDIS中间层驱动程序、NDIS小端口驱动程序、NDIS协议驱动程序、Win2kFilter-HookDriver、NDISHookDriver等。由于它们是基于操作系统环境运行的，如果操作系统被恶意软件攻击，则这种技术提供的网络监控数据可信度就大大降低。

如果安全软件能够进一步提升自身权限并独立于操作系统进行检测，不仅可以完成检测功能，还能够有效地保护自身和系统的安全。

虚拟化技术的飞速发展为构建虚拟主机安全软件提供了契机。2006年，Intel在其CPU产品中加入了硬件虚拟化扩展技术——VT-x。VT-x引入了一种新型的CPU操作模式VMX，VMX进一步分为根模式和非根模式。虚拟机监控器（Virtual Machine Monitor，VMM）在VT-x的帮助下可以为上层操作系统提供一个虚拟的硬件环境。VMM运行在操作系统和底层硬件之间，处于根模式下，对各种系统资源具有完全控制权。操作系统运行在非根模式下，其运行状态受VMM的监控。

由此，本作品提出云环境下虚拟主机通信监控技术，利用VMM的高特权级，对客户机操作系统（Guest OS）进行实时监控，通过捕获上层操作系统的网卡中断等系统事件并模拟处理，从而实现隐藏通信的检测、创建监控、终止连接的功能，使得利用内核级Rootkit技术隐藏的恶意通信能够被可靠地检测出来，同时保证系统其它合法通信安全运行。

2.2 系统简介

本系统利用VMM捕获网卡中断，对此时网卡的数据缓冲区进行读取，通过分析其中的原始数据，得到当前的网络连接信息，达到通信监控的目的。并根据用户指令在VMM层对特定数据包进行过滤，达到终止通信的目的。

本系统主要分为初始化模块、中断捕获模块、网络操纵模块（隐藏通信检测子模块、通信创建监控子模块和通信终止子模块）。

初始化模块是本系统实现各项功能的基础，包括系统所有数据结构的初始化，事件处理的准备，构建VMM，以及进入根模式。

中断捕获模块是本系统技术实现的重点。该模块通过对VMCS（虚拟机控制域）相关控制位的设置实现中断的捕获，通过访问PCI配置空间和高级可编程中断控制器（APIC）得到网卡设备的中断向量号，将通过VMCS客户机状态域记录的捕获中断的向量号与之比对，判断出是否为网卡中断并做相关操作。对网卡信息进行读取后，最终通过中断注入机制和中断窗口机制交付给客户机操作系统进行处理。

网络操纵模块是实现本系统功能的主要模块，该模块又分为隐藏通信检测子模块、通信创建监控子模块、通信终止子模块。隐藏通信检测子模块采用直接网卡编程技术实现隐藏网络的检测，在VMM下能够准确、可靠地检测出Guest OS中所有通信；通信创建监控子模块则实时监控通信创建的信息，根据通信黑白名单数据库和用户选

择的信息，正常执行或者阻止通信的创建；通信终止子模块利用VMM最高的特权级对原始网络数据包的过滤以及对目标通信另一端发送含终止连接位的数据包，从而达到终止通信的目的。

2.3 应用前景

虚拟化技术的优势已经得到了业界的普遍认可，虚拟化技术所具备的提高资源利用率和节能环保的特性也得到了越来越多高性能计算机用户的青睐，然而利用虚拟化技术为云服务提供安全保障的研究还不多。

目前从事研发和生产虚拟化软件的厂商众多，包括VMware、Xen、MS等在内的厂商极力发展初期的虚拟化市场。云服务提供商也使用虚拟化技术动态生成虚拟机实例提供给云服务用户。本系统基于虚拟化技术进行通信检测和终止，相较其他通信监控软件有极大的优势。

第四届中国云计算大会于2012年5月23-25日在北京国家会议中心隆重举行。云计算理念已经深入人心，成为全球范围内整个电子信息产业的中心主题。中央政府高度重视云计算的发展，将云计算列为“十二五”规划中积极培育发展的战略性新兴产业，不断加大支持力度。随着云计算的飞速发展，本系统具有广阔的应用前景。同时，本系统的关键技术在云环境下的安全防护方面具有较好的借鉴意义，为安全软件开发提供了新的思路。

随着全球信息化建设的快速发展，病毒和恶意代码的更新及传播越来越快，不管是云服务提供商还是用户都急需安全检测软件对云服务和在其上运行的网络服务提供强有力保护，本系统能够对云环境下的虚拟主机进行通信监控，保证其系统安全，防止黑客入侵，有极好的应用前景。

第三章 实现方案

3.1 设计思路

在支持硬件虚拟化的CPU上成功加载VMM后，原有的操作系统变成客户机操作系统（Guest OS），VMM成为系统关键资源的真正拥有者，能够捕获Guest OS的外部中断等系统事件。通过VMCS区域的配置，VMM可以有选择地对Guest OS中发生的动作进行捕获。当Guest OS发生特定动作时，产生VM-exit，陷入到VMM中，VMM判断陷入原因，进入相应的处理函数进行处理，然后返回到Guest OS继续运行。整个系统的运行原理如图3-1所示：

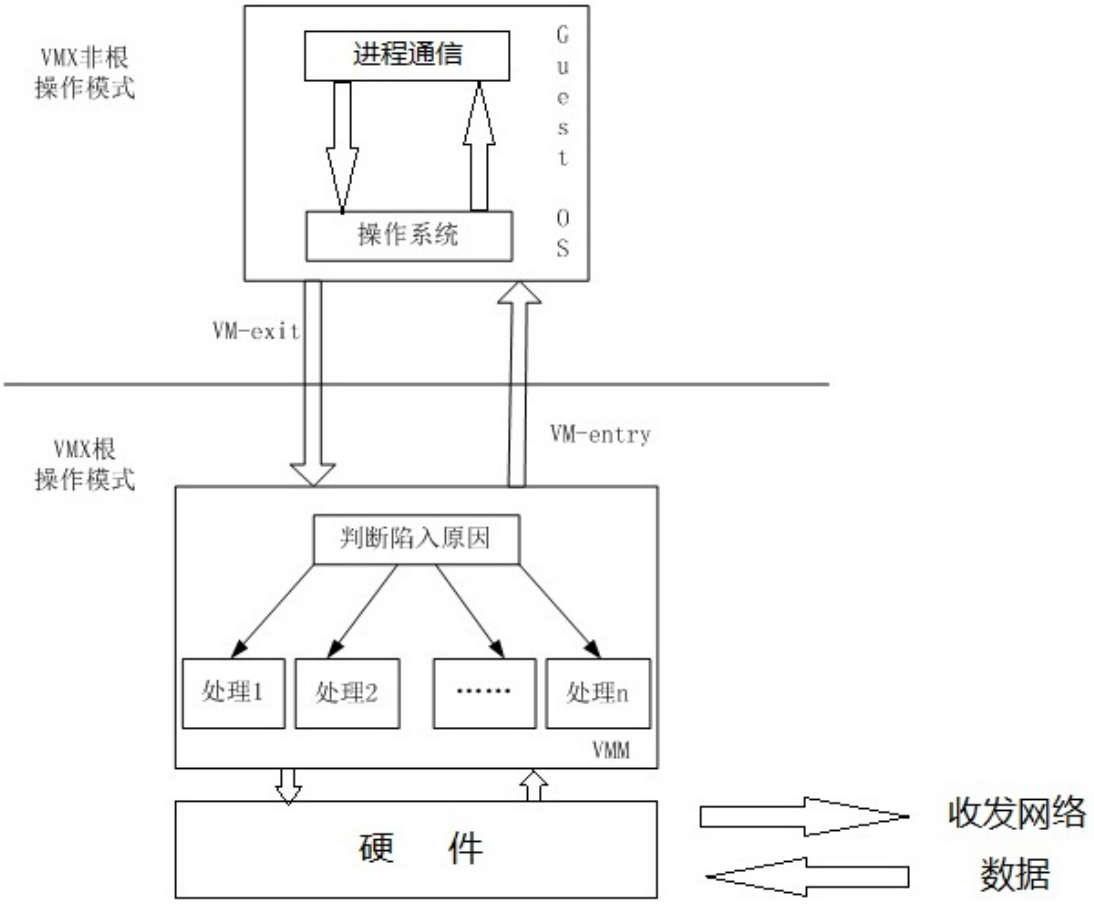


图3-1 系统的运行原理

Windows操作系统在发送接收网络数据包和新网络连接创建时要发生网卡中断，VMM捕获网卡中断的特定动作，通过对当前网卡缓冲区的分析，结合系统数据信息

对当前通信进行不同处理，从而达到隐藏通信检测，通信创建监控和通信终止的目的。

在本系统的设计思路中涉及到 VMM 对不同陷入原因采取不同处理，这个过程分以下几种情况：

1. 在隐藏通信检测时，VMM 通过分析获取当前网络连接信息，并判断当前通信是否在通信列表中，如果不在，则把当前的通信信息加入到通信列表。
2. 在通信创建监控时，如果创建通信在进程白名单中，直接返回 Guest OS，由 Guest OS 完成通信创建；如果创建通信在黑名单中，则 VMM 破坏通信创建的过程，使通信不能正常创建。如果该通信不在通信黑白名单中，则提示用户，根据用户选择采取相应的操作。
3. 在终止通信时，VMM 首先获取通信另一端的地址信息，然后对此来源数据包的过滤以及构造 RST 数据包终止网络连接。

3.2 系统方案

云环境下虚拟主机系统由初始化模块、中断捕获模块、网络操纵模块（包括隐藏通信检测子模块、通信创建监控子模块、通信终止子模块）组成。

初始化模块主要是配置VMCS区域，完成VMM的加载。初始化模块使得原有的操作系统变成客户机操作系统（Guest OS），VMM成为系统资源的真正拥有者，可以控制Guest OS在发生特定动作时陷入到VMM中，进而由其他模块完成相应的功能。

中断捕获模块通过对VMCS（虚拟机控制域）相关控制位的设置实现中断的捕获，通过访问PCI配置空间和高级可编程中断控制器（APIC）得到网卡设备的中断向量号，将通过VMCS客户机状态域记录的捕获中断的向量号与之比对，判断出是否为网卡中断并做相关操作。对网卡信息进行读取后，最终通过中断注入机制和中断窗口机制交付给客户机操作系统进行处理。

隐藏通信检测子模块通过VMM直接操作网卡，读取网卡数据缓冲区的原始数据然后分析数据包的相关信息得到Guest OS中所有的通信。通信创建监控子模块根据通信黑白名单和用户选择的相关信息进行判断，完成对通信的正常创建或阻止创建等操作。

通信终止子模块利用VMM底层优势实现对特定数据包过滤，并构造一个含终止连接位的数据包发送出去，实现终止连接。

数据管理模块完成对系统中数据的管理和维护，数据部分包括通信黑白名单，VMM和Guest OS的交互数据，系统日志记录等。

3.3 系统主要数据结构

3.3.1 虚拟机控制域VMCS

为了更好地支持硬件虚拟化技术，VT-x引入了虚拟机控制结构（VMCS）。VMCS是保存在内存中的数据结构，是一个最大不超过4KB的内存块。VMCS区域划分为客户机状态域，宿主机状态域，VM-entry和VM-exit控制域，VM-exit信息域。每个VMCS对应一个虚拟CPU（VCPU），其中客户机状态域记录了发生VM-exit时，对应VCPU的当前运行状态；宿主机状态域则用来保存根操作模式下CPU的运行状态；VM-entry控制域和VM-exit控制域是对VM-entry和VM-exit操作的具体行为进行控制规定的地方；VM-exit信息域存放VM-exit产生的原因和具体的分类细化指标。

3.3.2 共享内存处网卡数据

在Windows操作系统中，用户空间的虚拟地址0x7ffe0800和内核空间的虚拟地址0xffdf0800指向了同一个物理页，可以利用这个特殊的物理页来完成内核空间和用户空间的信息交互。本系统正是利用了这个物理页完成VMM和用户程序的进程信息交互。

共享内存处的数据结构如下：

| 信号位 | 长度 | 数据 |

3.3.3 网络数据包信息

为了处理数据包的方便，根据不同协议定义了相应的结构体，以对应相应的数据包头部。为处理方便，结构体中均未定义它们的选项域。结构体定义如下所示：

TCP头部：

```
struct TCP_HEAD
{
    unsigned short srcPort;
    unsigned short dstPort;
    unsigned int seq;
```

```
    unsigned int ack;
    unsigned char hlen;
    unsigned char flags;
    unsigned short winsiz;
    unsigned short checksum;
    unsigned short urgPointer;
};
```

UDP头部：

```
struct UDP_HEAD
{
    unsigned short srcPort;
    unsigned short dstPort;
    unsigned short length;
    unsigned short checksum;
};
```

IPv4头部：

```
struct IP_HEAD
{
    unsigned char VAI;
    unsigned char TOS;
    unsigned short tlen;
    unsigned short iden;
    unsigned short fragment;
    unsigned char TTL;
    unsigned char protocol;
    unsigned short checksum;
    unsigned int srcAddr;
    unsigned int dstAddr;
};
```

以太网帧头部：

```
struct MAC_HEAD
{
    unsigned char mac_srcaddr[6];
    unsigned char mac_dstaddr[6];
    unsigned short type;
};
```

IP规定，数据包分片的重组地点是信宿机。当目的主机收到一个数据报时，可以根据其片偏移量和MF位来判断它是否是一个分片。如果MF位是0，并且分片偏移量为0，则表明它是一个完整的数据报。如果片偏移量不为0，或者MF标志位为1，则表明它是一个分片，此时目的主机需要进行分片重组。根据数据报首部“标志”字段的值，可

判断哪些分片属于同一个原始数据报；片偏移量则用来确定分片在原始数据报中的位置。

本系统采用基于“洞”的算法来进行分片重组。该算法的核心结构是“洞”，它是指已经部分重组的IP数据包中那些未到达的分片。故定义一个指向“洞”的指针结构体链表，存放每个要重组的IPv4数据包数据区的未填充部分（洞）：

```
struct hole
{
    unsigned short first;
    unsigned short last;
    hole *next;
};
```

定义“洞”结构体，存放每个重组的IPv4数据包：

```
struct re_head
{
    IP_HEAD *iphead;
    hole *le;
    unsigned char ip[7000];
    re_head *next;
    unsigned short len;
};
```

定义一个结构体，存放每个数据包中我们所需要的基本信息，如MAC地址，IP地址，端口等网络连接的相关信息以便用户查看：

```
struct PACKET_READ
{
unsigned char MAC_SRC[6];
unsigned char MAC_DST[6];
unsigned int IP_SRC_ADDR;
unsigned int IP_DST_ADDR;
unsigned char protocol;
unsigned short SRC_PORT;
unsigned short DST_PORT;
unsigned int SEQ;
unsigned int ACK;
unsigned short TCP_DATA_LEN;
};
```

该结构体图示如下所示：

```
|0+++++++|1+++++++|2+++++++|3+++++++|4+++++++|5+++++++|6+++++++|
|MAC_SRC|++++++++++++++++++++++++++++++++++++++++|MAC_DST|
```

```
|++++++++++++++++++++++++++++++++++++++++++|IP_SRC_ADDR++++|
|++++++++++++++++|IP_DST_ADDR++++++++++++++++++++|protocol++|
|SRE_PORT++++++++|DST_PORT++++++++|SEQ++++++++++++++++++++|
|++++++++|ACK++++++++++++++++++++++++++++++|TCP_DATA_LEN+++|
```

定义一个结构体链表，存放所有的网络连接信息：

```
struct PACKET_SAVE
{
unsigned int count;
PACKET_READ *packet_read;
PACKET_SAVE *next;
};
```

3.4 功能实现原理

3.4.1 网卡中断捕获原理

2006年，Intel在其CPU产品中加入了硬件虚拟化扩展技术——VT-x。VT-x引入了一种新型的CPU操作模式VMX，VMX进一步分为根模式和非根模式。虚拟机监控器（Virtual Machine Monitor，VMM）在VT-x的帮助下可以为上层操作系统提供一个虚拟的硬件环境。

在支持硬件虚拟化的CPU上成功加载VMM后，原有的操作系统变成客户机操作系统（Guest OS），VMM成为系统关键资源的真正拥有者，通过VMCS区域的配置，VMM可以对Guest OS中发生的外部中断进行捕获。在相关函数中加入对外部中断陷入的判别，并对其进行处理。

目前，网卡一般为PCI设备。本系统通过访问PCI配置空间和高级可编程中断控制器（APIC）可得到网卡设备的中断向量号。VMM捕获外部中断陷入之后，本系统通过VMCS客户机状态域记录的捕获中断的向量号与之比对，判断出是否为网卡中断并做相关操作。

3.4.2 隐蔽通信监控原理

针对隐蔽通信，目前通信监控工具通常通过使用 Windows API，挂钩系统函数和内存扫描等方式获取通信列表。但是间谍软件利用 Rootkit 通过摘除自身连接在相应链表中的结构，恢复挂钩函数等手段来达到逃避检测的目的。恶意软件甚至利用在内核级的 Rootkit 主动破坏检测工具，使得检测工具不能正常运行。

本系统提出的隐蔽通信监控不同于传统的隐蔽通信监控，它不依赖于系统中静态的结构，不挂钩任何系统函数，其设计思路如下：

成功加载 VMM 后，原有操作系统成为 Guest OS，VMM 可以捕获 Guest OS 中发生外部中断等事件。Guest OS 中发送接收数据包前会发生网卡中断，如果 VMM 捕获 Guest OS 的网卡中断，并通过分析得到当前的通信信息，进而可靠地枚举 Guest OS 中的通信列表。

Windows 发送接收数据包前会发生网卡中断，而通过对 VMCS 的设置，VMM 可以对 Guest OS 中的网卡中断进行捕获，这样 VMM 捕获了 Guest OS 中每次发送或接收数据包的动作。在捕获 Guest OS 中发送接收数据包的动作后，VMM 对网卡中的数据缓冲区进行分析得到当前的通信数据。该数据具有以下特点：

1）通过分析该数据的内容，可以得到当前网络连接的具体信息，这是完成隐蔽通信监控功能的基本要求。

2）该数据在操作系统处理之前已被本系统读取，难以被人为的修改，保证获取通信数据的可靠性。

3）该数据存在于网卡中，与操作系统无关，保证了本系统的通用性。

有数据从网线上“流入”的时候，网卡把它放到内部FIFO中，同时进行DMA传输到接收环形缓冲区。之后网卡发出中断。VMM捕获网卡中断，此时客户机操作系统尚没有意识到网络数据的到来，我们通过读取难以更改的网卡缓冲区数据得到最原始的通信信息，同时检测程序运行于VMM中，这样保证了通信列表的可信度，同时大大降低了检测程序本身被攻击的可能性。

3.4.3 通信终止设计原理

该系统通过在VMM层直接网卡编程读取数据获取网络数据。VMM读取的网卡数据直接来源于硬件，处于最底层，恶意软件的通信难以隐藏，所以它获取的数据全面可靠，可信度高。

我们根据以上特点，对系统监控数据进行分析，建立起用户通信列表。在系统界面上，系统会显示每个数据包的详细信息，并通知给用户。用户根据所得信息，可以选择性终止可疑连接。当用户进行终止连接操作时，构建RST包，向所选目标发送；并通过在VMM中直接对网卡进行操作，设置过滤规则，处理指定来源数据包，在客户机操作系统毫无知觉的情况下将可疑数据包的过滤，以达到中断连接的目的。

3.5 主要技术及其具体实现

本系统采取了一系列技术和机制来保证对通信的监控，包括构建VMM技术、网卡中断的捕获和处理技术、直接网卡编程技术和数据包分析技术。下面将对这几种关键技术的具体实现进行分析。

3.5.1 构建VMM技术

本系统构建的 VMM 基于 Intel VT-x 技术。构建 VMM 的目的是为了让正在运行的目标操作系统迁移到 VM 中成为 Guest OS，从而 VMM 可以对目标操作系统中发生的某些特定动作进行捕获，进而通过分析处理，完成相应的功能，它是完成系统功能的基础。Intel 为开启 VMX 操作模式作了详细的规定和流程，本系统正是基于此构建了一个轻量级的 VMM。构建 VMM 的主要步骤主要包括：

- 分配并初始化VMCS，VMXON和VMM堆栈的内存空间。
- 判断当前系统是否支持并且允许开启VMX操作模式。
- 在VMCS中的客户机状态区域保存当前操作系统完整的状态，以保证当该系统置于客户VM中时，能够以正确的状态继续运行。
- 初始化VMCS中的宿主机状态区域，以保证VMM的正常工作。
- 初始化VMCS的控制区域，设置产生VM-exit的条件。产生VM-exit的条件是保证本系统完成功能的最小集合，以保证整个系统的效率。
- 执行VMLAUNCH指令，将当前操作系统迁移到客户模式。

3.5.2 网卡中断的捕获和处理技术

3.5.2.1 外部中断的捕获和处理

本系统对通信的监控是通过捕获网卡中断实现的。VMCS 域中有一部分称为 VM 执行控制域，通过配置这块区域可以在虚拟机在非根操作模式运行的时候，控制处理器非根操作模式退出到根操作模式；网卡中断属于“外部中断”，为了实现 VMM 对外部中断的捕获，需要设置 VMCS 域中 VM-EXECUTION CONTROL FIELDS 的相关位。把 External-interrupt exiting 位置为 1，外部中断就会引发陷入。

SetBit(&temp32, 0);

WriteVMCS(0x00004000, temp32);

在 VMMEntryPointer 函数中加入对外部中断陷入的判别，并对其进行处理。通过 VMREAD 指令，从 VMCS 的某些区域读出 VMExit 的一些信息，比如陷入原因，客户机状态等。其中最重要的是 ExitReason，即陷入原因。

通过对 VM_EXIT_REASON 的值的分析，得知相应的陷入事件，执行相应的处理事件：

External interrupt：

if(ExitReason == 0x00000001)

VMCALL：

if(ExitReason == 0x00000012)

INVD ：

if(ExitReason == 0x0000000D)

……

处理中断是通过“中断窗口机制”配合“中断注入机制”实现的。“中断注入”即把陷入的中断请求原封不动地注入回客户机处理（根据需要再对某些中断作特殊处理）。

中断窗口机制：

在中断注入前，需要先辨别客户机状态是否满足注入条件。当客户机的 IF 位为 0 时，状态不满足注入要求，我们通过设置 VM 执行控制域，停止中断捕获，打开 “中断窗口”；将此时到来的中断挂起。这样，一旦 IF 位变为 1，即引发“中断窗口陷入”，在这个陷入的处理中实现对挂起中断的正常注入。

if ((GuestEFLAGS & 0x00000200) == 0x00000000)

{

```
            vt_exint_pass(TRUE);
        vt_exint_pending(TRUE);
    }
    else
    {
        vt_exint_pass(FALSE);
            vt_exint_pending(FALSE);
            DeliveryInt();
    }
```

中断注入机制：

将分析得到的中断信息、中断错误码等写入 VM-ENTRY CONTROL FIELDS 中的相应位置：

```
// vmentry interrupt information
WriteVMCS(0x00004016,g_Event.vmcs_intr_info.v);
if (g_Event.vmcs_intr_info.s.err==INTR_INFO_ERR_VALID)
{
// vmentry interrupt error code
    WriteVMCS(0x00004018,g_Event.vmcs_exception_errcode);
}
```

对中断的后续处理工作并没有必要再使用专门的中断处理函数，所以可以把它交付给客户机操作系统完成。这样，通过“中断窗口机制”的配合，就能成功在实现中断的捕获后并执行自己的一些动作，并使其得到正确处理。

3.5.2.2 网卡中断的识别

VMM捕获外部中断之后需要判断其是否为网卡中断。Windows操作系统是通过中断向量（Vector）来识别不同中断的。本系统也根据中断向量（Vector）来判断捕获的外部中断是否为网卡中断。

1） 找到主机中网卡的IRQ

目前主机中网卡一般挂在PCI总线之上，为PCI总线设备。PCI有三个相互独立的物理地址空间：设备存储器地址空间、I/O地址空间和配置空间。配置空间是PCI所特

有的一个物理空间。由于PCI支持设备即插即用，所以PCI设备不占用固定的内存地址空间或I/O地址空间，而是可以由操作系统决定其映射的基址。

系统加电时，BIOS检测PCI总线，确定所有连接在PCI总线上的设备以及它们的配置要求，并进行系统配置。所以，所有的PCI设备必须实现配置空间，从而能够实现参数自动配置，实现真正的即插即用。

PCI总线规范定义的配置空间总长度为256个字节。在256个字节的配置空间中有Class Code（类代码）和Interrupt Line（IRQ 编号）两个字段信息。类代码共三个字节，分别是基类代码、子类代码和编程接口。基类代码可用于区分设备类型。

DDK（Driver Development Kit）提供了内核函数HalGetBusData用于读取PCI设备的配置空间。PCI设备主要靠三个信息定位，即Bus号、Device号和功能号。分别对这三个数进行排列组合，就能枚举出系统内的所有PCI设备。在枚举的同时通过判断BaseClass是否等于0x02（网络控制器的基类代码）找到网卡，找到网卡之后就可得到网卡的IRQ。关键代码如下：

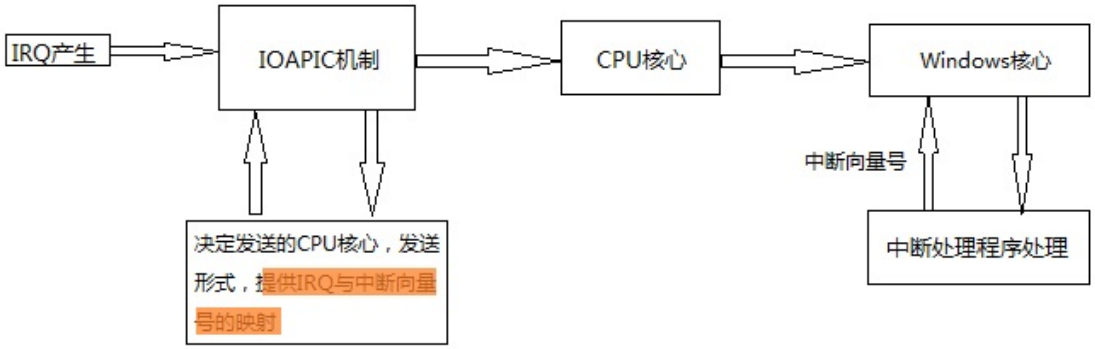
```
for(bus = 0; bus <= PDI_BUS_MAX; ++bus)
 {
      for(dev = 0; dev <= PDI_DEVICE_MAX; ++dev)
      {
            for(func = 0; func <= PDI_FUNCTION_MAX; ++func)
            {
                  SlotNumber.u.AsULONG = 0;
                  SlotNumber.u.bits.DeviceNumber = dev;
                  SlotNumber.u.bits.FunctionNumber = func;

                  RtlZeroMemory(&PciConfig,
                                    sizeof(PCI_COMMON_CONFIG));
                  Size = HalGetBusData(PCIConfiguration,
                                          bus,
                                          SlotNumber.u.AsULONG,
                                          &PciConfig,
                                          PCI_COMMON_HDR_LENGTH);
  if(PciConfig.BaseClass==0x02)
      Irq = PciConfig.u.type0.InterruptLine;
```

2） 找到网卡中断的中断向量

APIC是可以用于多个核心CPU的新型高级可编程中断控制器。它的作用是当一个IRQ产生时，它负责决定将IRQ发送给哪个CPU核心，以及以何种方式发送等。绝大多数x86系统或者依赖于i8259A可编程中断控制器（PIC），或者依赖于i82489A高

级可编程中断控制器（APIC）的一个演变；大多数计算机都包含一个APIC。PIC只能在单处理器上工作，它有15条中断线。APIC可以在多处理器系统上工作，它们有256条中断线。APIC实际上是由几个部件构成的：一个专门接收设备中断的I/O APIC，以及一些本地APIC，和一个与8259A兼容的中断控制器。本地APIC在一条私有的APIC总线上接受来自I/O APIC的中断，而8259A兼容的中断控制器则负责将APIC输入转换成与PIC等价的信号。I/O APIC负责实现中断跳转（IRQ和IDT的映射）。



IOAPIC通过读/写相应的寄存器来进行编程。IOAPIC的寄存器是不能像通用寄存器一样直接访问的，而是通过映射成内存的物理地址进行访问。Windows把访问IOAPIC系列寄存器所需要的两个寄存器映射到物理地址0xfec00000和0xfec00010的位置，这在Windows下是固定的。

Table 1. Memory Mapped Registers For Accessing IOAPIC Registers

Memory Address	Mnemonic	Register Name	Access	D/I# SIgnal
FEC0 xy00h	IOREGSEL	I/O Register Select (index)	R/W	0
FEC0 xy10h	IOWIN	I/O Window (data)	R/W	1

NOTES:
xy are determined by the x and y fields in the APIC Base Address Relocation Register located in the PIIX3. Range for x = 0-Fh and the range for y = 0,4,8,Ch.

IOREGSEL——I/O 选择寄存器。这个寄存器可以通过内存物理地址 0xfec00000 直接访问，可读可写，长度为 32 位，但只有低 8 位起作用，其他的位被保存。它的作用是选择一个要操作的 IOAPIC 寄存器。

unsigned char *io_reg_sel;（长度为 8 位）

IOWIN——I/O 窗口寄存器。这个寄存器可以通过内存物理地址 0xfec00010 直接访问，可读可写，长度为 32 位。它的作用是用来读/写由 I/O 选择寄存器指定的 IOAPIC 寄存器。

unsigned long *io_win;（长度为 32 位）

因为这里谈的都是真正的物理地址，是不能够直接访问的，在 Windows 下访问则需要对物理地址进行一个映射（实际上就是添加一个页表项来描述这个映射关系），在 wdk 中提供了一个这样的映射函数：

```
PVOID MmMapIoSpace(
  IN PHYSICAL_ADDRESS  PhysicalAddress,
  IN SIZE_T  NumberOfBytes,
  IN MEMORY_CACHING_TYPE  CacheType
  );
```

经过映射后，我们就可以对其进行操作了。

paddr = MmMapIoSpace(phys, 0x14, MmNonCached);// MmMapIoSpace 把物理地址映射为系统空间的虚拟地址。0x14 是这片空间的长度。

IOAPIC 重定位表（跳转表）标示每个 IRQ 被重定位到哪个中断处理函数。除了中断处理函数之外，还有一些其他的信息，这里忽略。这个表一共有 24 项，每一项用两个 IOAPIC 寄存器来存取（64 位）。

这个表的寄存器偏移从0x10开始，一直到0x3f为止，一共48个。例如IRQ1（键盘中断）对应的表项所在的寄存器偏移位0x12和0x13，而中断向量保存在0x12寄存器的低8位（1字节），由此可以读出中断向量。

程序实现为：

```
for (*io_reg_sel = 0x10;*io_reg_sel <= 0x3e;*io_reg_sel += 0x02)
{
ch = *io_win;
ch &= 0xff;
DbgPrint("irq = %2x.\r\rvector = %x.\r\n",i++,(unsigned long)ch);
}
```

3.5.3 直接网卡编程技术

3.5.3.1 基本原理

网卡的种类各种各样，不同厂商生产的网卡其内部结构也有所不同。但是网卡的工作原理及实现上却都是大同小异。不失一般性，本文选用RTL8139为例，对网卡的直接编程技术原理进行说明。

在捕获网卡中断VMM陷入之后，操作系统尚未对网卡缓冲区的数据进行处理，此时网卡缓冲区里的数据就是通信的原始数据。本系统读取原始通信数据进行分析，

故结果可信度很高。

RTL8139的主要工作原理：发送数据时，计算机把要传输的数据并行写到网卡的缓存，网卡对要传输的数据进编码（10M以太网使用曼切斯特码，100M以太网使用差分曼切斯特码），串行发到传输介质上。接收数据时则相反。网卡包括硬件和固件程序（只读存储器中的软件例程），该固件程序实现逻辑链路控制和媒体访问控制的功能，还记录唯一的硬件地址即 mac地址，网卡上一般有缓存。网卡须分配中断irq及基本I/O端口地址，同时还须设置基本内存地址（base memory address）。本系统通过读取PCI配置空间，得到网卡的I/O地址范围。将这段映射为虚拟地址，即可对网卡的相关寄存器进行读写操作。

3.5.3.2 发送缓冲区读取

RTL8139 的发送路径由 4 个描述符（descriptors）组成，每个描述符由 2 个寄存器组成，分别是“发送起始地址寄存器”（TSAD）和“发送状态寄存器”（TSD）。每个描述符有一个固定的偏移地址，4 个描述符循环使用。对某个描述符进行写操作后，PCI 接口会通过 DMA 方式将描述符所指定的主存中的数据包复制到 RTL8139 内部的一个 2K 字节的发送缓冲队列中。当发送缓冲队列中数据量达到某一规定数值（此数值大小也由描述符指定）时，便会从发送缓冲队列中逐位发送到网线上。

首先根据四个“发送起始地址寄存器”里值的大小来判断出当前使用的描述符，在相应的“发送状态寄存器”中可以读取到包的状态以及大小等信息。然后根据所得信息，将“发送起始地址”通过 MmMapIoSpace 函数映射为虚地址，这样就可以对其中的数据信息进行读取。

在读取数据前，先根据网卡中断状态域（ISR）判断中断类型——是否是数据发送中断，再根据命令寄存器（CR）判断是发送缓存区是否为空。如果数据包发送正常，则可以读出接收缓冲区数据，送入共享内存。

3.5.3.3 接收缓冲区读取

RTL8139的接收路径是一个环行缓冲区，这个环行缓冲区是主存储器中的一块物理上连续的区域。从网线上传来的数据首先被存储到RTL8139内部的接收缓冲队列中，当接收缓冲队列中数据量达到某一门槛数值时，接收到的数据便会从接收缓冲队列中拷贝到环行缓冲区中，缓冲区写指针寄存器（CBA）保存了当前被复制的数据在

环行缓冲区中的地址，而缓冲区读指针寄存器（CAPR）则保存了已被读取的数据的地址指针。同时，接收状态被加到了收到的数据包的包头中。当接收缓冲队列中数据量达到某一门槛数值时就产生一个中断，告诉系统收到了数据包。

其接收缓存区是一个大小为64k+4字节的环形区域，数据在环形缓冲区的存放格式如下：

| 长度 | 状态位 | 内容 | 长度 | 状态位 | 内容 |...

对缓冲区数据的读取涉及以下寄存器和数据结构：

RBStart: Receive Buffer start address.

CR(BufferEmpty): Indicate if driver is empty.

CAPR: Buffer read pointer.

CBP: Buffer write pointer.

ISR/IMR(ROK, RER, RxOverflow, RxFIFOOverflow)

RCR： Receive configuration register.

Packet Header

当RJ-45那个接口有数据从网线上“流入”的时候，网卡把它放到内部FIFO中，同时进行DMA传输到接收环形缓冲区。之后网卡发出中断。中断后进入VMM的接收处理函数。这时通过对网卡内部状态寄存器和接收地址寄存器的读取，得到其接收缓存区起始地址和当前偏移。将这块接收缓冲区映射为虚拟地址，通过当前偏移指针对新数据包进行读取。

在读取数据前，需先根据网卡中断状态域判断中断类型——是否是数据到来中断，再根据命令寄存器判断是接收缓存区是否为空。如果数据包接收正常，则读取出前4个字节——包头部信息（Packet Header），再依据其读出接收缓冲区数据，送入共享内存。

3.5.4 VMM与用户层同步数据交互技术

Windows 操作系统中，所有进程的虚拟地址 0x7ffe0800 和内核空间的虚拟地址 0xffdf0800 指向同一个物理页，所有的进程都可以读取这个物理页的内容。本系统正是利用这个物理页作为共享缓冲区以实现 VMM 到应用程序的单向通信。

在数据交互的过程中，还存在类似于“生产者和消费者”的问题——VMM和应用程序间的同步问题。当VMM读取到网卡数据时，需要“提醒”用户层读取该数据。这里将共享内存起始位置处作为“信号位”，令用户层开辟读取线程不断读取信号，约定当信号置为‘Z’时表明有数据到来，进行数据读取。这样，当VMM捕获到网络数据写入共享内存后，置信号为‘Z’；用户层收到通知后读取数据，读取完通过调用专门驱动程序‘WriteA’的IoControl将信号清除（写入‘A’覆盖）。这样便实现了VMM和用户层的同步通信。

3.6 性能指标

1）能够有效的检测出操作系统中隐藏的通信。

2）能够终止系统中的恶意通信，有效保护系统。

实现了可视化的用户操作界面，用户能够清晰的看到系统中的隐蔽通信及相关信息，能够对系统中的恶意网络连接进行有效的终止，根据通信黑白名单和用户选择等信息，能够对系统中的通信进行严格的管控。

3.7 特色

1）本系统不挂钩操作系统中的任何函数，不修改操作系统中的任何数据完成隐蔽通信监控和终止。

2）成功将硬件虚拟化技术应用到云环境下虚拟主机安全检测领域，利用VMM的最高权限有效保护了虚拟主机的安全。

3）利用VMM捕获网卡中断对虚拟主机中的通信进行监控，使监控的可信度和准确度大大提高。

第四章 性能测试

4.1 测试方案

4.1.1 测试内容

4.1.1.1 功能测试

a) 隐蔽通信监控：在支持虚拟化的PC平台上实现隐蔽通信的监控，检测包括隐蔽通信信息的网络连接。
b) 中断网络连接：根据用户的需求，如果不是用户信任的连接，用户发出中断网络连接的命令，实现对异常连接的中断。
c) 通信白名单/黑名单：对于检测出的异常连接，如果是用户可信的连接，则加入白名单，下次不再进行报警；如果是不可信的链接，则加入黑名单，再有该连接时直接中断。

4.1.1.2 性能测试

a) 正常系统性能测试：云环境下虚拟主机通信监控系统未启动时，系统的各项性能指标。
b) 非正常系统性能测试：云环境下虚拟主机通信监控系统启动后，系统的各项性能指标。
c) 性能比:非正常时的各项性能与正常时的各项性能的比例,性能比越接近于1，云环境下虚拟主机通信监控系统对主机的影响越小。

4.1.1.3 稳定性测试

云环境下虚拟主机通信监控系统能够在Windows XP下正常运行。

4.1.2 测试步骤

1) 在支持硬件虚拟化的PC中，进入BIOS，开启虚拟化支持。
2) 在PC未开启云环境下虚拟主机通信监控系统前，用PCMark 05测试软件对系统的各项性能进行测试。

3) 启动云环境下虚拟主机通信监控系统，检查系统管理界面工作是否正常。

4) 用 PCMark 05 测试软件对启动云环境下虚拟主机通信监控系统后的各项系统性能进行测试。

5) 开启远程控制木马，进行隐蔽的网络连接，对目标主机进行远程控制。

6) 分别用 Windows 防火墙、360 防火墙、瑞星等工具进行检测。

7) 对所有测试数据详细分析，得出测试结果。

4.2 测试环境

测试平台：采用主机作为测试平台，其配置如下：

CPU：Intel(R) Core(TM)2 Duo (2.66GHz) 32 processor

内存：1GB

硬盘：256 GB

网卡：RTL8139c

具体测试环境: Windows XP sp1 、Windows XP sp2、Windows XP sp3

（注：本系统将对 RTL8139c 网卡进行验证性操作，在需要时可以快速在使用其他型号网卡的机器上部署运行。）

4.3 测试数据及结果分析

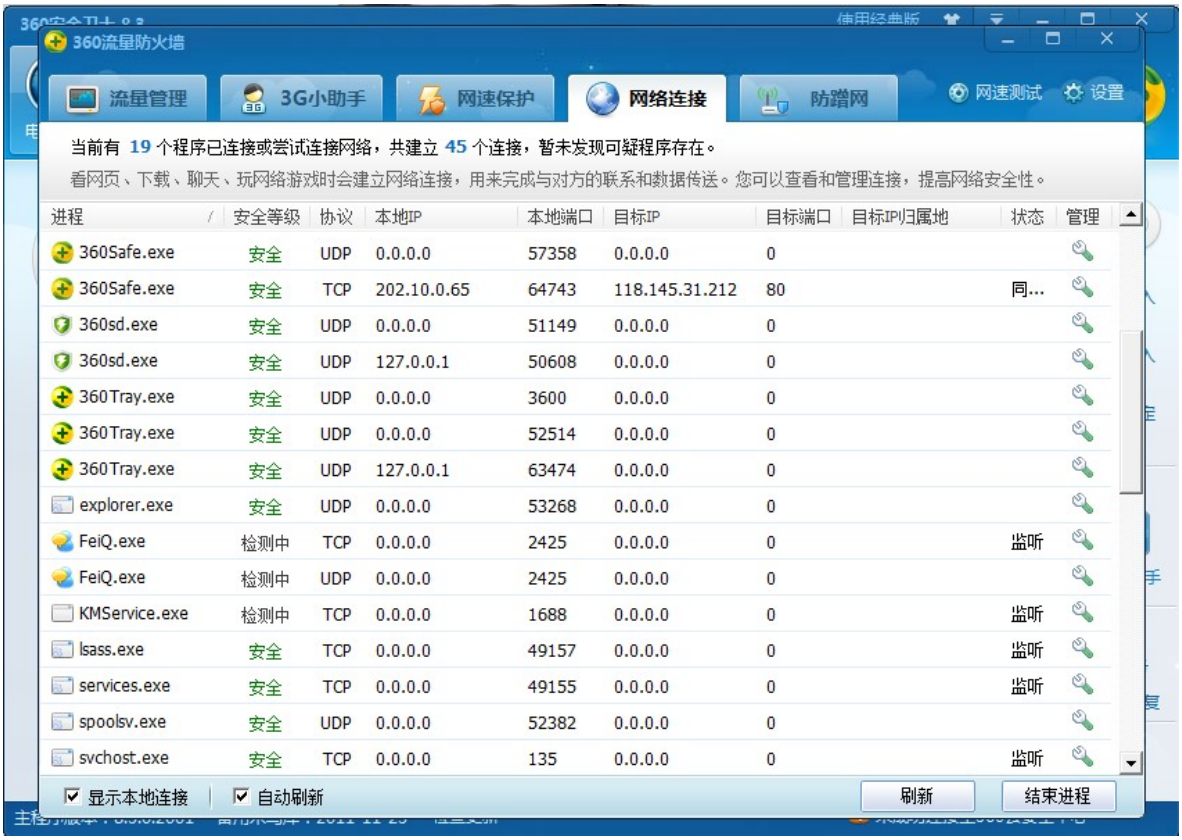
4.3.1 功能测试

4.3.1.1 隐蔽通信监控功能测试

测试木马，byshell。

在对云环境下虚拟主机通信监控系统进行测试时，我们使用隐藏通信连接的远程攻击工具 byshell 对远程主机进行攻击，在远程主机（IP 202.10.0.135）加载服务端 ntboot.exe,同时开启 Windows 防火墙和 360 防火墙,客户端主机(IP 202.20.0.64）对远程主机进行远程控制，可以查看远程主机系统信息、进程列表、远程关闭进程、下载/上传远程文件、远程命令，以及远程关机等操作。

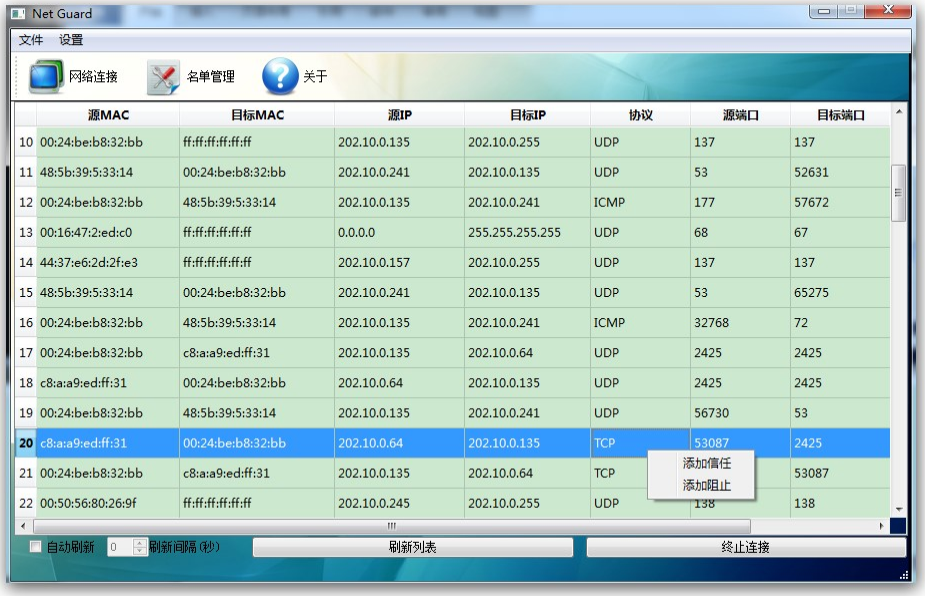
开启 Windows 防火墙和 360 防火墙，用 byshell 窃取服务端主机文件，360 防火墙没有检测到异常并提示用户。



360 未能监测到 byshell 的网络连接

由此可见，360 防火墙对于 byshell 木马并没有起到阻碍的作用，对于客户端的非法操作，如文件窃取等没有进行报警。

开启云环境下虚拟主机通信监控系统进行测试，当点击网络监控时，进入监控界面，系统列出了当前系统网络连接的列表，如下图所示：



通过客户端主机下载远程文件时，Windows 防火墙和 360 防火墙都没有任何提示或警告信息，在 360 防火墙和流量监控中也没有发现异常信息，只有在云环境下虚拟主机通信监控系统中的网络连接列表中能看到客户端主机与远程主机连接的信息。云环境下虚拟主机通信监控系统能够监测到隐藏的通信连接的功能，达到了设计的需求。

根据用户的命令，对不可信的异常连接进行强制关闭，经测试，连接已被终止。

为了对比，我们同时用 Windows 防火墙、360、瑞星等进程检测工具对不同隐蔽通信连接进行检测。

从测试结果来看，Windows 防火墙不能检测到隐蔽的网络连接，360 防火墙和瑞星对于隐蔽的网络通信连接也不能做到完全拦截，而只有云环境下虚拟主机通信监控系统能够正常监测到这些隐蔽的网络连接，并且可以根据用户的命令，对不可信的异常连接进行中断操作，实现强制关闭，从而实现对主机的保护。

经过测试可以发现，云环境下虚拟主机通信监控系统对于隐蔽的网络连接方面具有比以往安全工具更强大的功能，能够检测出 Windows 防火墙、360 防火墙、瑞星监测不出的隐蔽的网络连接，其对 Guest OS 的细粒度控制具有较好的准确性和完整性。

经过大量测试，测试结果表明系统能够实现隐蔽的网络连接监测功能，符合设计要求。

4.3.1.2 中断网络连接功能测试

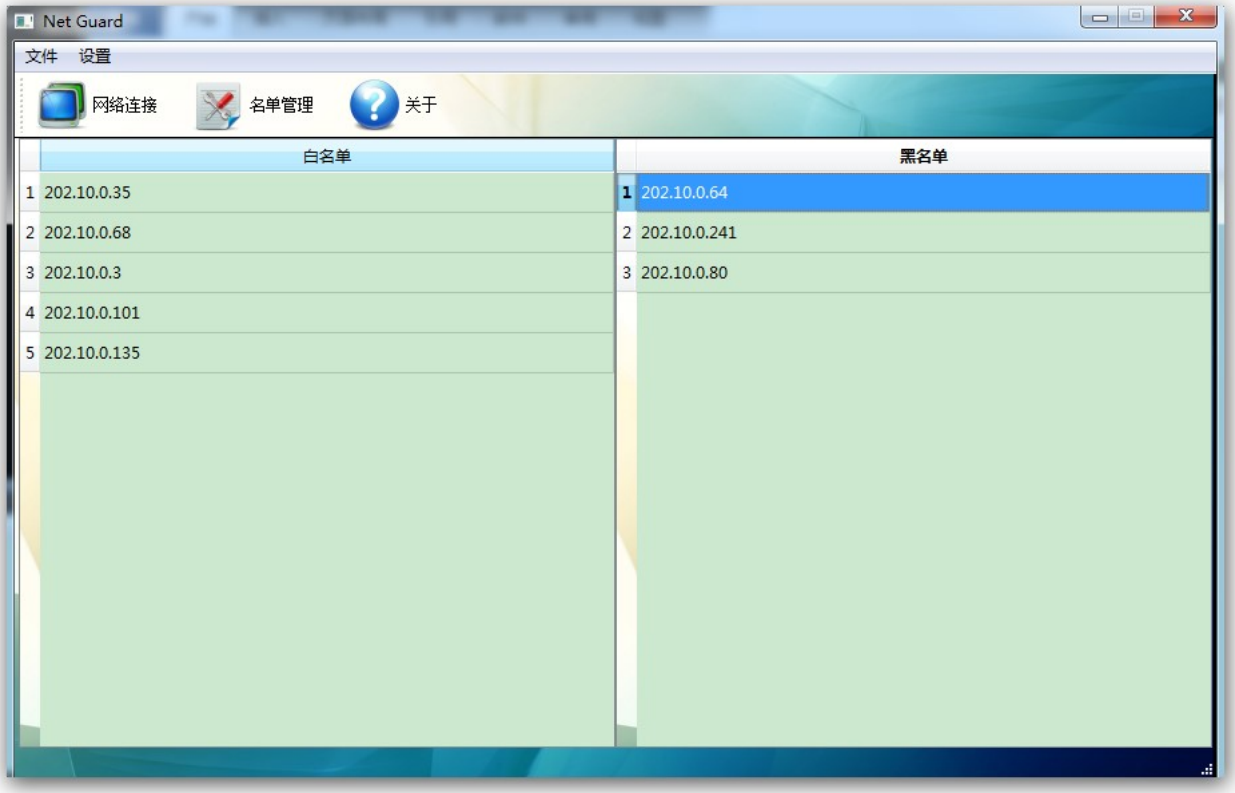
根据运行态网络连接中断终止功能的要求，从系统网络连接列表中选了一个要终止的网络连接，并对其进行终止。

测试中的其中一个测试用例：从网络连接列表中选择了飞秋的网络连接，当选择中断连接时，飞秋的网络连接即被终止。同时，为了对比，我们打开了 360 网络连接监控，也找不到飞秋的网络连接。

经过大量测试，证明该系统能够有效的中断网络连接，达到了设计的要求。

4.3.1.2 网络连接黑名单/白名单功能

根据用户的需要，可以添加网络连接黑白名单，对白名单中网络连接放行



对于测试时监控到的网络连接，根据用户指令终止连接，同时将客户端主机（IP 202.10.0.64）加入黑名单，再次进行远程连接时，无法正常连接，说明该客户端主机已被加入黑名单，查看网络连接黑名单列表，发现列表中有该主机的 IP 地址。云环境下虚拟主机通信监控系统能够建立网络连接黑名单，并进行网络连接的终止，达到了设计的需求。

（1）测试中，对于一个隐蔽的网络连接，若是用户不可信的网络连接，则根据用户指令中断网络连接，同时加入通信黑名单；

（2）再次启动该隐蔽的网络连接，发现连接不能正常进行，已经被中断，说明该连接已经被加入黑名单；

（3）若是用户可信的网络连接，则不进行中断操作，同时加入黑名单；

（4）再次启动可信的网络连接，云环境下虚拟主机通信监控系统不再提示用户判断，并且能进行正常的网络连接，说明该连接已经被加入白名单。

经过测试，测试结果表明系统能够实现网络连接黑名单/白名单功能，符合设计要求。

4.3.2 性能测试

性能测试采用上文测试环境中列出的平台，安装的是 Windows XP 操作系统，采用的软件是 PCMark 05。PCMark05 是测量个人计算机性能的一个优质工具，包括 HD 录影自动译码，数字式音乐内码，先进穿线和基于追踪的硬盘性能测试。测试情节包括 Hyper-Threading and multicore CPUs，数字式录影内码和解码，文件压缩和解压、编成密码、解密、互联网浏览，物理，3D，正文编辑，音像转换和图像加工性能测试系统等。利用该软件对云环境下虚拟主机通信监控系统启动前后系统的 CPU 及内存的各项性能指标进行了测试，结果如下表所示：

云环境下虚拟主机通信监控系统启动前性能测试结果统计表

测试项	综合得分	测试子项		测试结果	
CPU	6602	File Compression	File Compression	11.67086124	B/s
		File Decompression	File Decompression	172.7537537	B/s
		File Encryption	File Encryption	73.30141449	B/s
		File Decryption	File Decryption	73.45169067	B/s
		Multithreaded Test 1	File Compression	11.75443935	B/s
		Multithreaded Test 1	File Encryption	72.42146301	B/s
		Multithreaded Test 2	File Decompression	86.77573395	B/s
		Multithreaded Test 2	File Decryption	36.44037247	B/s
Memory	4748	Memory Read - 16 MB	Memory Read - 16 MB	3435.395	B/s
		Memory Read - 192 KB	Memory Read - 192 KB	17181.02	B/s

		Memory Write - 16 MB	Memory Write - 16 MB	3893.196	B/s
		Memory Write - 192 KB	Memory Write - 192 KB	15525.5	B/s
		Memory Copy - 16 MB	Memory Copy - 16 MB	3383.068	B/s
		Memory Copy - 192 KB	Memory Copy - 192 KB	15637.51	B/s
		Memory Latency - Random 16 MB	Memory Latency - Random 16 MB	10.13143	Accesses/s
		Memory Latency - Random 192 KB	Memory Latency - Random 192 KB	161.7777	Accesses/s

云环境下虚拟主机通信监控系统启动后性能测试结果统计表

测试项	综合得分	测试子项		测试结果	
CPU	6177	File Compression	File Compression	11.51100922	B/s
		File Decompression	File Decompression	168.0036011	B/s
		File Encryption	File Encryption	72.05750275	B/s
		File Decryption	File Decryption	72.14125824	B/s
		Multithreaded Test 1	File Compression	8.728037834	B/s
		Multithreaded Test 1	File Encryption	71.929245	B/s
		Multithreaded Test 2	File Decompression	72.74863434	B/s
		Multithreaded Test 2	File Decryption	35.02264786	B/s
Memory	4700	Memory Read - 16 MB	Memory Read - 16 MB	3311.16284	B/s
		Memory Read - 192 KB	Memory Read - 192 KB	17051.5879	B/s
		Memory Write - 16 MB	Memory Write - 16 MB	3836.31665	B/s
		Memory Write - 192 KB	Memory Write - 192 KB	15472.6436	B/s
		Memory Copy - 16 MB	Memory Copy - 16 MB	3361.69873	B/s
		Memory Copy - 192 KB	Memory Copy - 192 KB	15592.6523	B/s
		Memory Latency - Random 16 MB	Memory Latency - Random 16 MB	9.9603157	Accesses/s
		Memory Latency - Random 192 KB	Memory Latency - Random 192 KB	161.767899	Accesses/s

通过对上述两表中的测试结果进行分析，分析结果如下表所示：

性能测试分析结果统计表

	本系统启动前	本系统启动后	比值
CPU	6602	6177	93.56%
Memory	4748	4700	98.98%

从测试的结果来看，CPU 性能下降了 6.44%，内存下降了 1.02%。CPU 和内存的性能下降都比较低，说明该系统对 CPU 和内存几乎没有影响，总体来讲云环境下虚拟主机通信监控系统达到了系统设计的目标。CPU 性能略有下降主要是因为 Guest OS 每次进程切换会产生 VM-exit，陷入到 VMM 中，VMM 在每次陷入后作相应的处理，占用了一定的时间片。

4.3.3 稳定性测试

通过在不同的环境下进行测试,测试结果表明,该系统支持的操作系统有 Windows Xp sp1 、Windows Xp sp2、Windows Xp sp3。

经过在不同环境下进行多次测试，系统能够准确的隐蔽的网络连接监控、网络连接黑名单/白名单功能、中断网络连接等功能。测试结果表明系统运行稳定，功能实现准确，达到了预期设计的要求。

4.3.4 与同类产品的比较

在隐蔽的网络连接监控、中断网络连接方面，本系统超越了目前主流的安全检测工具。同时，本系统检测软件运行于 VMM，降低了检测程序本身被攻击的可能性。此外，目前安全检测工具对操作系统版本要求比较高，与其相比本系统具有更强的平台移植性。但是，本系统在系统性能消耗方面较大。

随着支持虚拟化的CPU广泛应用，支持该系统的平台成为主流，本系统将大大优于目前的安全检测软件，在隐蔽的网络连接监控、网络连接黑名单/白名单和中断网络连接方面具有较强的实用价值。

4.4 小结

按照软件测试要求，对云环境下虚拟主机通信监控系统的功能、性能指标和稳定

性进行测试，通过对测试数据的分析，本系统实现了在基于虚拟化的平台下隐蔽的网络连接监控、中断网络连接和网络连接黑名单/白名单的功能，符合设计要求。通过与市场上同类软件的比较，结果显示本系统在隐蔽的网络连接监控方面功能更加强大、能更有效的中断网络连接、并对网络连接起到了良好的监控功能。

第五章 创新性

5.1 思想创新

该系统不挂钩操作系统的任何系统函数，不更改操作系统的数据，保证系统的完整性。通过对云计算、Windows 操作系统中断机制、TCP/IP 协议栈和网卡原理的深入研究，在云环境下的虚拟主机上进行通信监控，成功完成了隐蔽通信的检测及终止功能。

5.2 技术创新

5.2.1 云环境下虚拟主机的隐蔽通信监控技术

本系统提出了一种云环境下虚拟主机的隐蔽通信监控技术，它不依赖于挂钩系统中任何函数，不破坏操作系统中的任何数据结构。相对于目前主流的通信监控工具通过挂钩函数或者依赖于未公开的数据结构查找隐蔽通信具有很大的优越性，能够有效的检测出系统中的隐蔽通信。

5.2.2 基于中断窗口机制的中断注入技术

在 VMM 中传统的处理中断的基本方法是“中断注入”，即把陷入的中断请求原封不动地注入回客户机处理（根据需要再对某些中断作特殊处理）。注入的代码为：WriteVMCS(0x00004016, ExitInterruptionInformation); 但是这样的处理会直接死机或蓝屏。通过采用中断窗口机制，即当客户机的 IF 位为 0 时，状态不满足注入要求，我们打开 VM 执行控制域中的“中断窗口”，将此时到来的中断挂起。这样，一旦 IF 位变为 1，即引发“中断窗口陷入”，在这个陷入的处理中实现对挂起中断的正常注入。

5.2.3 在VMM中直接操纵网卡获得原始数据

传统的通信监控方式，主要通过调用系统函数，即使是直接操作网卡也必须依靠系统函数挂靠到某个进程的地址空间。恶意软件容易采取挂钩系统函数的方式隐藏网

络连接。本系统利用VMM的最高系统权限，对系统的各种资源具有绝对的控制权，无需调用系统函数挂靠到进程地址空间，而是直接操作网卡获得原始数据帧，大大提高了监控的可信度。

5.2.4 VMM、内核模块和应用程序间通信机制

VMM、内核模块和应用程序间需要互相通信，才能使系统正常的运行，完成系统的功能。

本系统通过共享缓冲区来实现网络数据交付，这就需要解决VMM和应用程序间的同步问题。我们通过引入“信号位”来实现同步问题，这样应用程序根据信号便可知道何时读取新的数据。

本系统采用VMCALL指令和规则约定方式，实现应用程序到VMM的单向通信，VMM可以根据事先约定的规则（比如EAX=35353535）来执行相应动作。

第六章 总结

由于思想上和技术上的创新，本系统能够有效的检测出云环境下虚拟主机中的隐藏通信，对通信进行有效的监控和终止，并达到了自身的安全防护。经过验证，本系统在很多方面超过了目前流行的通信监控工具。

随着云计算技术的发展，在互联网上云服务将会越来越普及。而一些不法分子若攻击云服务提供的虚拟机实例将会给云服务提供商和用户造成了巨大的损失。云环境下的虚拟主机是一种新的环境，在隐蔽通信的检测和终止上的难度比较大，传统的网络安全工具已经不能有效的完成相应的工作，虚拟主机的安全得不到应有的保护。本系统利用当前流行的硬件虚拟化技术，完成了隐蔽通信的检测、通信的创建监控和终止，并有效的保护了自身的安全。具有检测效果准确、终止通信有效、自身安全性高等优点，在当前环境下，具有很好的实用和市场价值。

参考文献

[1] 张帆，史彩成。 Windows 驱动开发技术详解[M]。 北京：电子工业出版社，2008。

[2] 谭文，邵坚磊。 天书夜读:从汇编语言到Windows内核编程。 北京：子工业出版社，2008年10月。

[3] 谭文，杨潇，邵坚磊等。 寒江独钓：Windows内核安全编程。北京：电子工业出版社,2009年8月.

[4] 毛德操。 Windows内核情景分析——采用开源代码ReactOS。 北京：电子工业出版社。

[5] Intel 64 and IA一32 Architectures Software Developerg Manual Volume 3B:System Programming Guide，Part2[EB/OL]。

http://download.inte1.corn/design/processor/manuals/253669.pdf。

[6] 张晓峰，刘飞，张秀珍等。RTL8139网卡直接IO编程技术.pdf。科技创新导报 2008 NO.25