

## 二、 HVM 介绍

### 1) HVM 概述

在讨论 HVM 之前首先谈谈虚拟，虚拟（virtualization）指对计算机资源的抽象，一种常用的定义是“虚拟就是这样的一种技术，它隐藏掉了系统，应用和终端用户赖以交互的计算机资源的物理性的一面，最常做的方法就是把单一的物理资源转化为多个逻辑资源，当然也可以把多个物理资源转化为一个逻辑资源（这在存储设备和服务器上很常见）”

实际上，虚拟技术早在 20 世纪 60 年代就已出现，最早由 IBM 提出，并且应用于计算技术的许多领域，模拟的对象也多种多样，从整台主机到一个组件，其实打印机就可以看成是一直在使用虚拟化技术的，总是有一个打印机守护进程运行在系统中，在操作系统看来，它就是一个虚拟的打印机，任何打印任务都是与它交互，而只有这个进程才知道如何与真正的物理打印机正确通信，并进行正确的打印管理，保证每个 job 按序完成。

有了虚拟技术的基本概念，下面我们来谈谈 HVM。HVM, Hypervisor Virtual Machine 的缩写，是一种基于硬件的虚拟技术（Hardware Enabled virtualization），也就是在硬件层面上，更确切的说是在 CPU 里，对虚拟技术提供了支持，而且虚拟层要比操作系统权限要高，这就使得宿主机（Host Computer）被启动后，在引导操作系统前先初始化每个虚拟机（Virtual Machine），每个虚拟机就像有了自己的硬件一样，因而可以完全隔离的运行自己的寄宿操作系统（Guest OS）。在 2005 年和 2006 年，Intel 和 AMD 都开发出了支持虚拟技术的 CPU。当然也可以用虚拟技术来做一些其他的事情，当前 HVM 已经在虚拟机，安全，加密等领域上有所应用。例如 VMware Fusion, Parallels Desktop for Mac, Parallels Workstation 和 DNGuard HVM

### 2) HVM 的架构

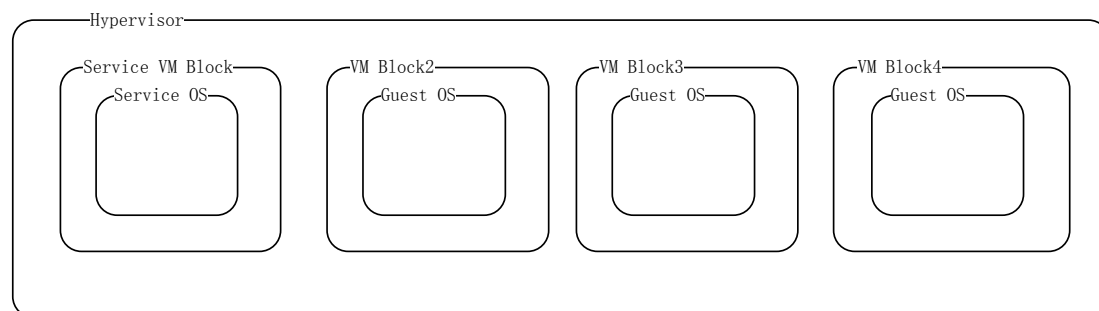


图 1: Hvm 架构图

从图中可以看到，VM 层的权限要高于操作系统的权限，我们知道，操作系统的内核态已经处在了 Ring0 特权级上，因此这也就要求 VM 层实际上要运行在“Ring -1”特权级上。于是我们也就需要新的指令，寄存器以及标志位去实现这个新的特权级的功能。

### 三、 HVM 特定平台介绍

#### 1) AMD-V

##### 概述

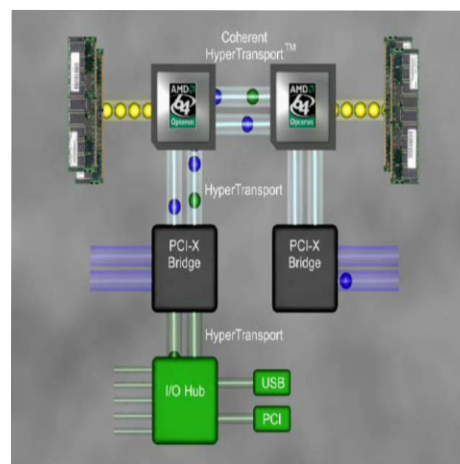
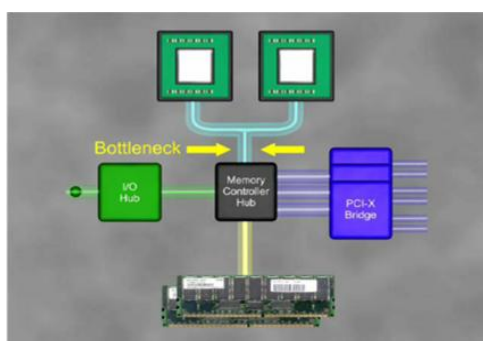
AMD 芯片支持虚拟化的技术被称作 AMD-V(在技术文档中也被称为 SVM,其全称是 AMD Secure Virtual Machine)。其主要是通过一组能够影响到 VM 管理层 (hypervisor) 和寄宿层的中断实现的。同时 AMD-V 技术也对下面的要求提供了支持:

- 快速的 VM 监视层 (VMM, 和 hypervisor 是一个概念, 下文均用 VMM 表示) 和寄宿机 (有时也指寄宿操作系统, 若无特别说明均用 guest 表示) 之间的切换
- 中断 guest 中特定的指令或事件(events)
- DMA 访问保护
- 中断处理上的辅助并对虚拟中断 (virtual interrupt) 提供支持
- 一个新的 TLB (其实就是一个 Cache) 来减少虚拟化造成的性能下降。

AMD 修改了架构以更好的支持虚拟化技术, 如图:

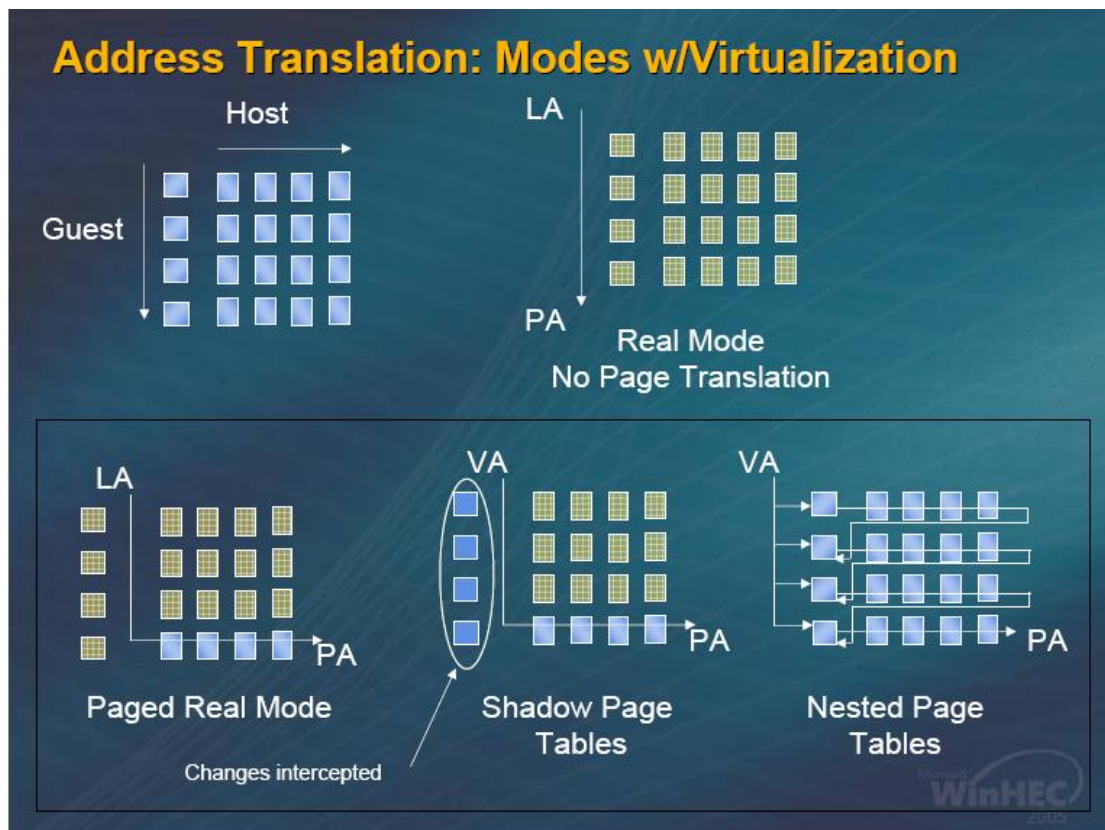
传统的前端总线架构造成瓶颈

AMD采用直连架构消除了CPU和北桥以及内存间的瓶颈



大多数的 server 还采用像左面那样的 SMP (对称多处理器结构)。

至于地址翻译也发生了很大改变, 在虚拟技术下的地址翻译流程如图所示: (该图摘自 WinHEC2005, 版权归原作者所有)



下面对这张图解释一下，从左上角可以看出，这些页表是用来处理一个 Guest 虚拟机的情况的。每个小方格代表一个页表。总的思想是由于允许多个 VMCB 的存在和多个 Guest OS 的同时运行，所以我们必须让 Guest OS 发出的访问物理地址的操作并不能真正访问到内存的物理地址，因此我们引入了另一套页表放在 VM 层中，Guest OS 的物理地址操作在 VM 层看来也是虚拟地址，只有通过处在 VM 层的这套页表的地址转化才能生成真正的物理地址，当然这也带来了内存保护。

可以看到一共有四种地址翻译模式：

**Real Mode:**直接把虚拟地址（Virtual Address, Linear Address）转化为物理地址。不经过任何翻译。

**Paged Real Mode:**在 Guest 操作系统中

**Shadow Page Tables:**

**Nested Page Tables:**

## 关于 VM 层及指令

在 SVM 中，VM 控制块被称为 VMCB(Virtual Machine Control Blocks)，其信息主要分为两块，第一块是控制信息存储部分，同时也包含是否允许拦截某特定异常的遮罩（interception enable mask），Guest 中不同的指令和事件都能以修改 VMCB 中相应控制位的方法拦截，SVM 支持的两类主要的拦截是异常拦截和指令拦截，第二部分则是 guest 的状态信息保存，这里会保存段寄存器以及大部分的虚拟内存的入口控制寄存器，不过浮点寄存器信息不会被保存。需要注意的是 VMCB 在不同的处理器间不共享，并且 VMCB 一定要是页对齐的连续内存空间。

SVM 中主要的指令有以下这些：

**VMLoad:** 从VMCB加载guest的状态, VMCB与guest是有对应关系的。

**VMMCALL:** 通过该方法guest可以与VMM显式的交流, 方法是利用生成#VMEXIT从guest层退到VM层。

**VMRUN:** 加载VMCB, 并开始执行guest层的指令, VMCB的物理地址将通过rAX获得, 这个VMCB对应于要执行的guest

**VMSAVE:** 存储处理器状态的子集到VMCB中, 这个VMCB的物理地址由rAX寄存器给出。

**STGI:** 用于设置全局中断标志(Global Interruption Flag)为1, 这个指令属于Secure Virtual Machine。

**CLGI:** 用于设置全局中断标志(Global Interruption Flag)为0, 同样这个指令属于Secure Virtual Machine。

**INVLPGA:** 使得TLB上一个ASID和一个虚拟页(Virtual Page)之间的映射关系无效, 这个指令属于Secure Virtual Machine。

**SKINIT:** 安全的重新初始化CPU, 使得CPU可以开始执行一段受信任的程序(trusted software)其方法是该代码进行安全的哈希比较(secure hash comparison)。这也就是开发者可以开发一个更安全的VMM loader。

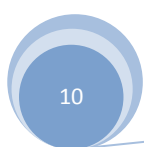
**改进的MOV指令:** 现在的MOV指令可以直接读写CR8寄存器(任务优先级寄存器Task Priority Register), 因此可以用来提高SVM应用的性能。

## AMD-V 的开发逻辑

其实由上文的描述可以看出, 开发基于AMD-V的程序最主要是编写一个循环, 这个循环要包含VMRUN命令以便从VM层启动一个Guest虚拟机, 也要包含一段程序用于处理当#VMEXIT发生后的异常情况, 这其中可能要手动做一些必要的保存现场和恢复现场的工作, 具体造成异常的起因等均可通过读取VMCB中的数据获得。

### 2) Intel-VTX





## 四、 体验 NewBluePill

首先介绍下我的平台，在整个项目中我用了两台计算机

PC1（调试机）：Intel Core 2 6300, 1G RAM, XP SP2(X32)+windbg+WDK6001.18001

PC2（运行机）：Intel Core 2 6300, 1G RAM, Windows Server 2008 Beta 1(X64),NewBluePill  
（以下简称 nbp）只能运行于这台机器上。

### 1) 编译 NewBluePill

了解了以上那么多，是不是很想亲自动手尝试下呢？不过先别急，还是先把工具准备好再说。

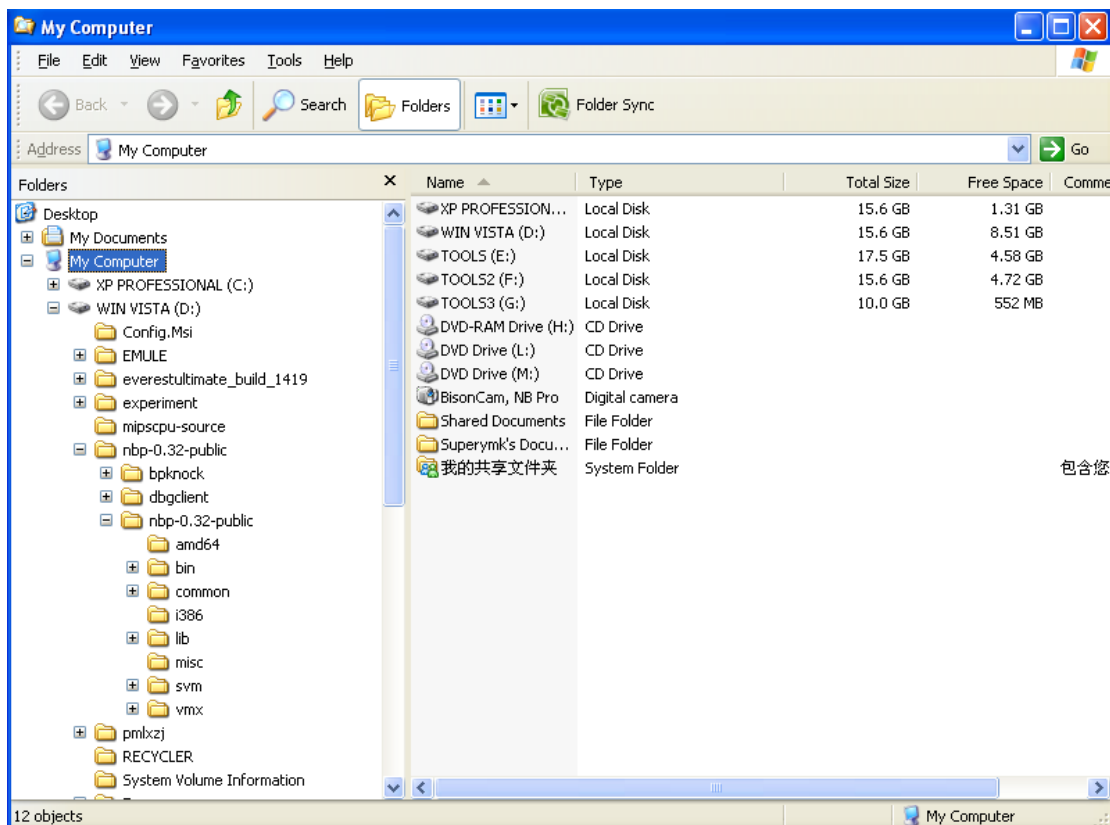
工具一共有下面几个：

1. Windbg
2. DebugView 到 <http://download.sysinternals.com/Files/DebugView.zip> 下载
3. InstDrv 到  
<http://dl2.csdn.net/fd.php?i=23314208212665&s=0affa2ecb56fc0dcc14cff07345a388e>  
下载

4. Windows Driver Kits (WDK 6001.18001)

总体来说编译 NewBluePill 的过程很简单。

步骤 1. 首先确保手上了 nbp-0.32-public.zip 这个代码。没有的可以去 <http://www.bluepillproject.org/> 上去下载，然后解压缩到一个根目录，在这里我们假设是 D 盘。目录结构应该是这样的：



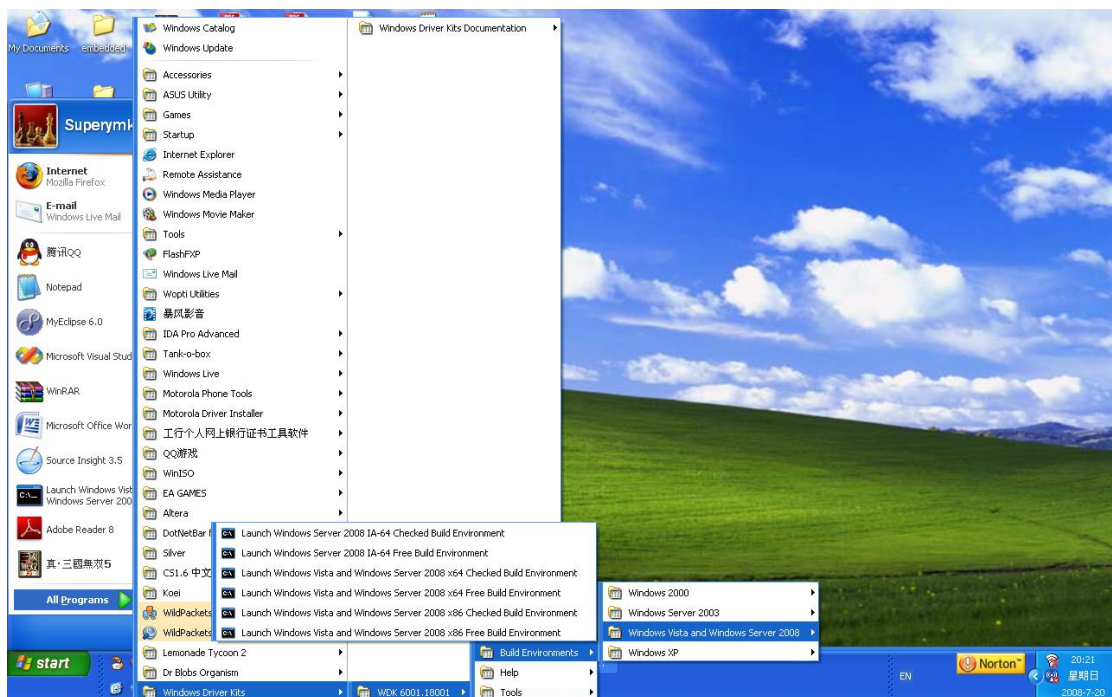
步骤 2. 然后为了后面的调试过程中可以下断点，修改下 common 目录下的 newbp.c 文



件，在 DriverEntry 方法的一开始添加 CmDebugBreak()方法调用，修改后的代码如下：

```
NTSTATUS DriverEntry (
    PDRIVER_OBJECT DriverObject,
    PUNICODE_STRING RegistryPath
)
{
    NTSTATUS Status;
    CmDebugBreak();
#ifdef USE_COM_PRINTS
    PIoInit ((PUCHAR) COM_PORT_ADDRESS);
#endif
    ComInit ();
    .....
}
```

步骤 3. 然后打开 Launch Windows Vista and Windows Server 2008 x64 Checked Build Environment 编译环境：



步骤 4. 在该编译环境中执行 nbp-0.32-public\nbp-0.32-public\build\_code.cmd，如果编译成功则会出现以下窗口：

```
1>Compiling - vmx\vmxdebug.c
2>Building Library - lib\amd64\svm.lib
1>Building Library - lib\amd64\vmx.lib
1>BUILD: Compiling and Linking d:\nbp-0.32-public\nbp-0.32-public\common directory
1>Assembling - amd64\msr.asm
1>Assembling - amd64\svm-asm.asm
1>Assembling - amd64\vmx-asm.asm
1>Assembling - amd64\common-asm.asm
1>Assembling - amd64\regs.asm
1>Assembling - amd64\cpuid.asm
1>Assembling - amd64\intstubs.asm
1>Compiling - common\newbp.c
1>Compiling - common\hvm.c
1>Compiling - common\portio.c
1>Compiling - common\comprint.c
1>Compiling - common\hypercalls.c
1>Compiling - common\traps.c
1>warnings in directory d:\nbp-0.32-public\nbp-0.32-public\common
1>d:\nbp-0.32-public\nbp-0.32-public\common\traps.c : warning C4819: The file contains a character that cannot be represented in the current code page (936). Save the file in Unicode format to prevent data loss
1>Compiling - common\interrupts.c
1>Compiling - common\common.c
1>Compiling - common\paging.c
1>Compiling - common\snprintf.c
1>Compiling - common\chicken.c
1>Compiling - common\dbgclient.c
1>Linking Executable - bin\amd64\newbp.sys
BUILD: Finish time: Sun Jul 20 20:22:39 2008
BUILD: Done

30 files compiled - 2 Warnings
2 libraries built
1 executable built

D:\nbp-0.32-public\nbp-0.32-public>ctags -R
'ctags' is not recognized as an internal or external command,
operable program or batch file.

D:\nbp-0.32-public\nbp-0.32-public>
```

如果看到这个提示，恭喜你，编译成功了！

## 2) 演示 NewBluePill

运行 nbp 就有一定要求了，首先要求必须运行在支持虚拟技术（HVM）的 CPU 上，并且推荐在 64 位或者支持虚拟 64 位技术的 CPU 上运行，原因是虽然 nbp 程序中附带了支持 32 位 CPU 的代码，但是有几个函数在编译时（Vista x86 Checked Mode）会出问题，而且有几个函数是未实现的，所以还是在 x64 上去跑吧。

下面是详细步骤：

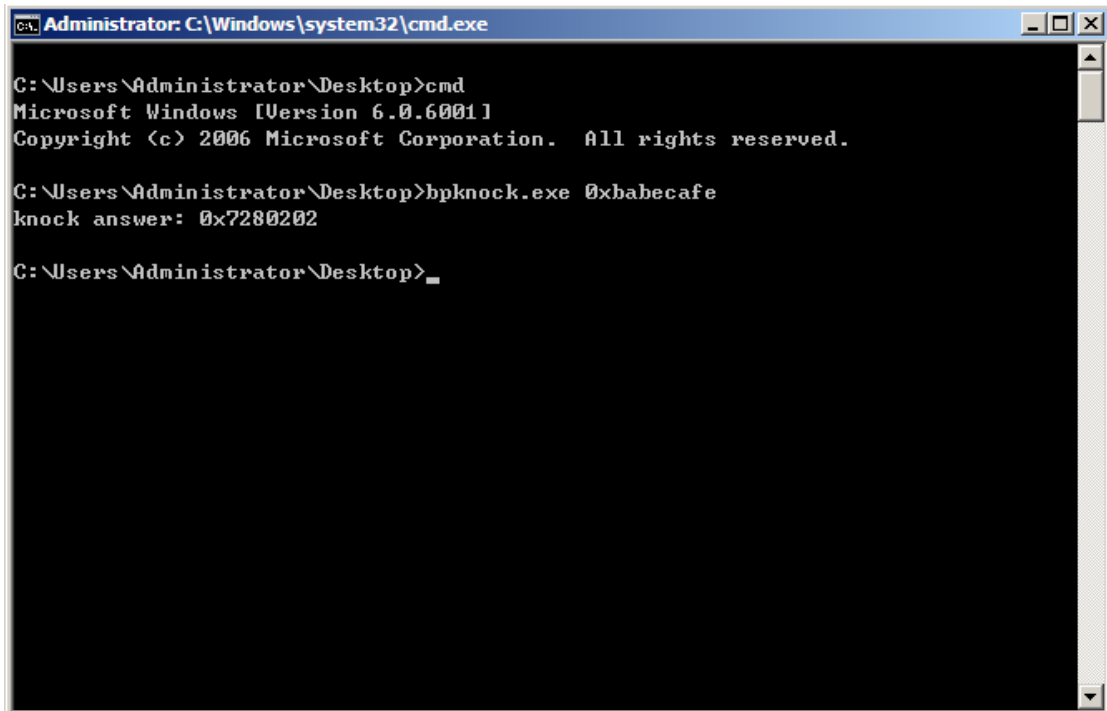
步骤 1：参考 Debugging Windows Vista（文章来源：[http://www.microsoft.com/whdc/driver/tips/debug\\_vista.mspx](http://www.microsoft.com/whdc/driver/tips/debug_vista.mspx)）修改启动项和调试项（这一步只需做这一次就可以）

步骤 2：重启计算机，可以看到启动项中多了一个 DebugEntry [debugger enabled]项，选中它按 F8，然后选择 Disable Driver Signature Enforcement(切记一定要用这个模式启动，否则不能加载未签名的驱动程序)

步骤 3：去 nbp-0.32-public 主目录及其子目录内找到下面几个编译生成的二进制文件：bpknock.exe, dbgclient.sys, newbp.sys



步骤 4: 运行下 bpknock 0xbabecafe 看下没运行 nbp 的输出结果。



```
Administrator: C:\Windows\system32\cmd.exe

C:\Users\Administrator\Desktop>cmd
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop>bpknock.exe 0xbabecafe
knock answer: 0x7280202

C:\Users\Administrator\Desktop>
```

步骤 5: 打开 DebugView, 在 DebugView 中的 Capture 菜单中选中下列项:

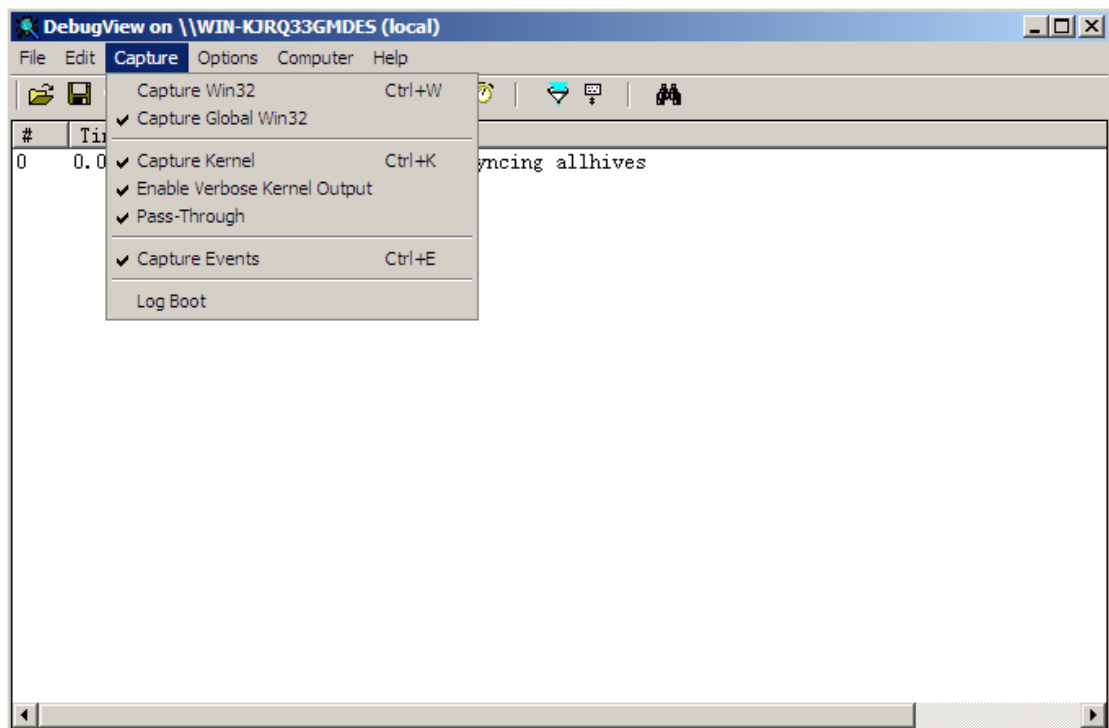
Capture Global Win32

Capture Kernel

Enable Verbose Kernel Output(这个一定要选中)

Pass-Through

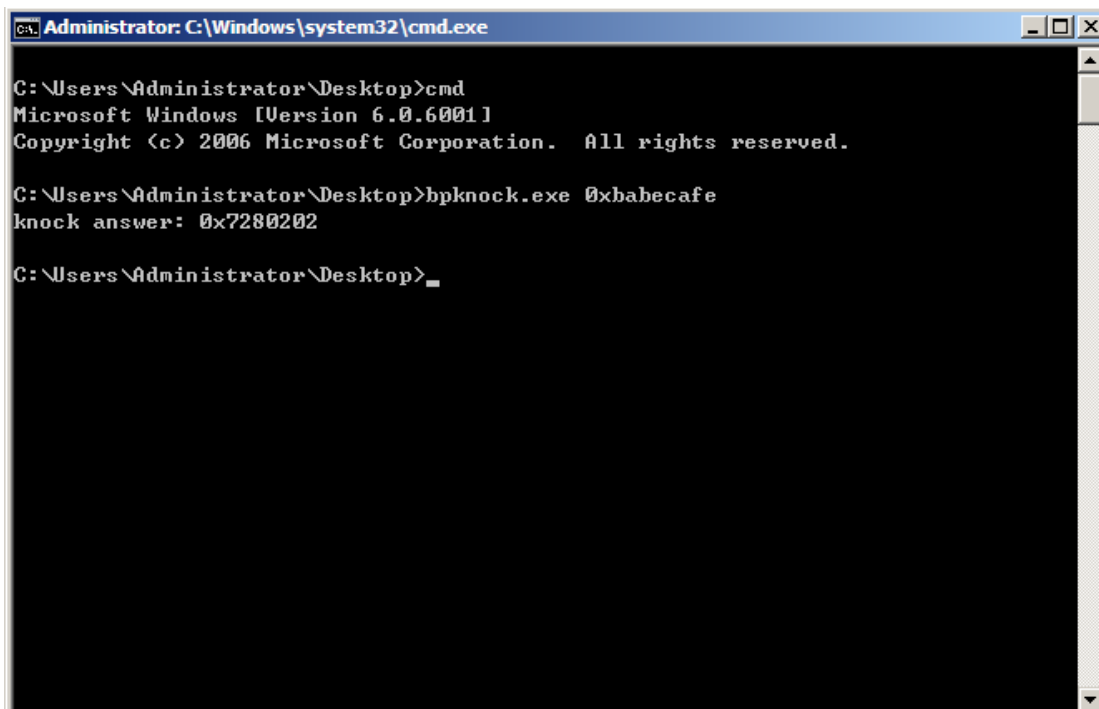
Capture Events



然后打开 InstDrv, 先安装并启动 dbgclient.sys 驱动, 再安装并启动 newbp.sys 驱动

步骤 6: 再运行下 `bpknock 0xbabecafe` 看下运行了 `nbp` 的输出结果。

注: 如果死机了那么可以把编译过程中的第 2 步去掉, 不过这样就不能调试 `nbp` 了。

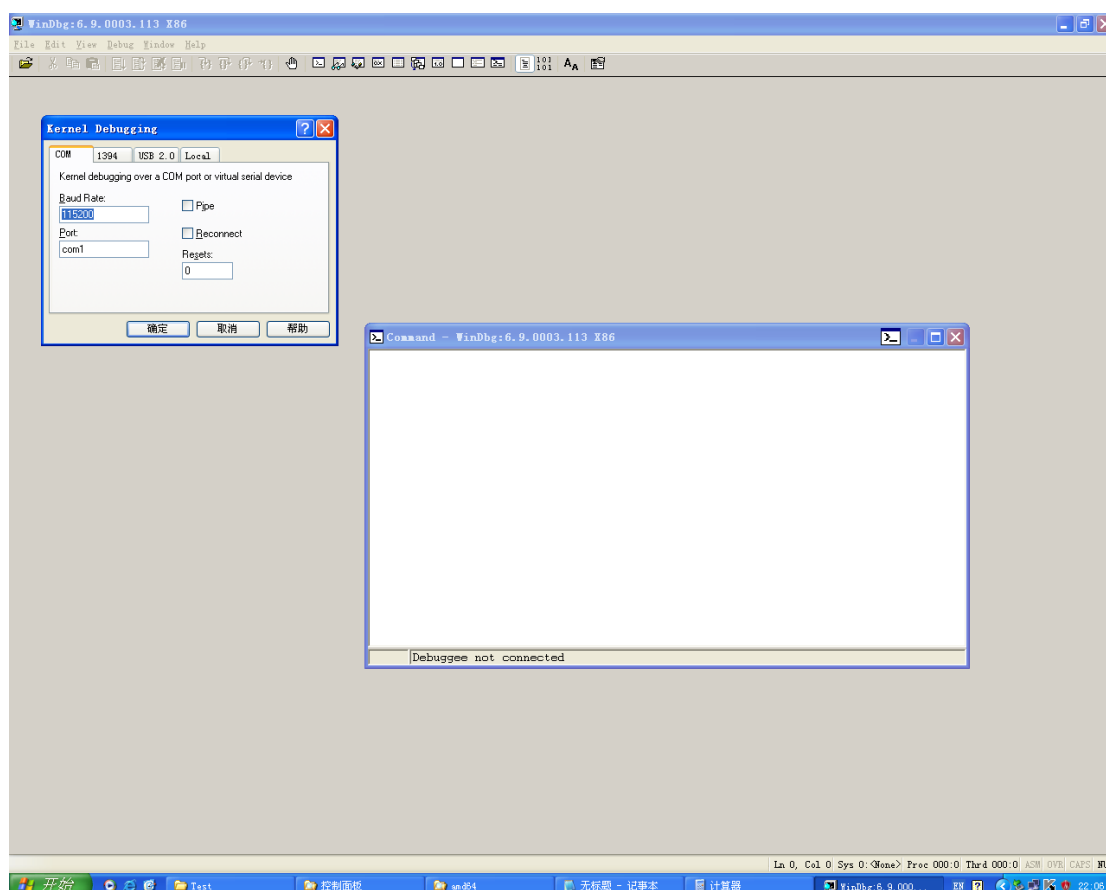


### 3) 调试 NewBluePill

调试 `nbp` 需要用到 WinDbg, 主要过程如下:

步骤 1: 设置 `_NT_SYMBOL_PATH` 环境变量, 指向 `newbp.pdb` 所在的目录, 用于链接符号表。

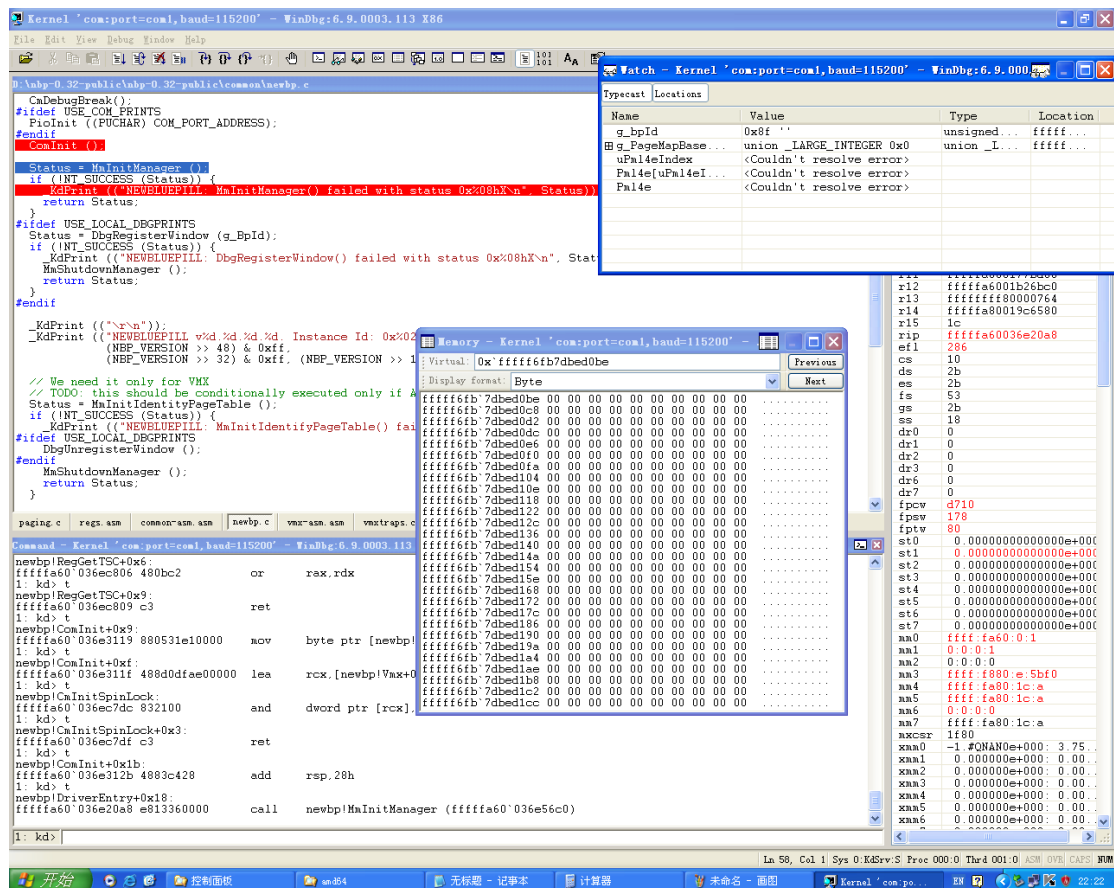
步骤 2: 启动 WinDbg, 单击 File 菜单选择 Kernel Debugging, 在弹出的对话框输入 Baud Rate 为 115200, Port 用 com1。这是由于刚才在演示过程的第一步我们用的是默认配置, 如果调试端口发生相应改变, 这里也要改。



步骤 3: 调试机上加载 `dbgclient.sys` 和 `newbp.sys` 两个驱动，开始调试。

如果出现 `symbol` 不能被加载的情况可以试试 WinDbg 中的 `.reload` 命令，如果不行可以试试用 `.sympath` 在 WinDbg 运行时设定 `symbol` 路径，然后 `.reload` 重新加载符号表。

成功情况下的截图：



Ok, 有了调试平台，我们可以更快的揭开 nbp 的层层面纱了。

注：有个问题就是如果我们用了 `CmDebugBreak()` 方法，则基本上可以发现每次调试机在 `bpknock` 时会死机，这一点很可能是由于因为代码会在此时陷入 VM 中造成，不过具体原因还在探索中。