在正式讲解内核编程之前，先讲讲如何编程实现加载驱动。加载驱动的标准方法只有一种，就是利用服务管理器函数加载。也就是我今天主要讲的 **SCM 加载法**。不过在继续讲之前，我还要想吹吹牛，讲讲加载驱动的故事。

加载驱动的方法其实有很多，在 2000/XP 这些古老的 NT5 系统上，公开的至少也有两种，一种是直接使用 NtLoadDriver 函数（SCM 加载法最终也是调用了此函数，不过我这里指的是不使用 SCM 系列函数，而是自己写入注册表项并直接调用 NtLoadDriver），另外一种是使用 NtSetSystemInformation 来加载驱动。第二种加载驱动的方法非常“直接”，什么注册表项都不用写，把参数填写正确就能加载了。在第二种方法没公开之前，很多病毒都是用这种方法来对抗 HIPS（当然公开了之后很快就被和谐掉了）。除了这两种方法之外，据说还存在“地下方法”加载驱动，就是利用一些极少人关注的函数（比如 GDIXXX 这种看起来和加载驱动毫无关联的函数）来加载驱动。这些加载驱动的方法可以称为“漏洞”，在黑市上能卖很多钱，在被封杀之前，一个好用的漏洞换一套在广州市中心的房产绝对不是梦。所以大家想致富的话，就多挖掘一下驱动加载漏洞，挖到了好用漏洞就等于挖到了房子。扯淡到此结束，下面详细讲解一下 SCM 加载驱动的要领。

使用 SCM 系列函数加载卸载驱动的过程并不复杂，总体流程是：**打开 SCM 管理器（获得 SCM 句柄）->创建驱动服务（获得服务句柄，如果服务已经存在，此步则变成打开服务）->启动服务->停止服务->移除服务->关闭服务句柄->关闭 SCM 句柄。如果要与驱动通信，则用 CreateFile 打开驱动的符号链接（可以理解成获得一个“通信句柄”），然后使用 DeviceIoControl 与驱动进行信息交互。如果曾经打开过驱动的符号链接，则必须在卸载驱动前关闭“通信句柄”，否则再次加载相同的驱动时会有一些麻烦**。

DeviceIoControl 的参数也很好理解，只有 5 个重要参数：控制码，输入缓冲区，输入长度，输出缓冲区，输出长度。但这个控制码（以下简称 IOCTL）则是大有玄机的，这里则必须讲清楚。仔细看回 KrnlHW64.sys 的代码，里面关于 IOCTL 的定义是这么写的：

```
#define IOCTL_IO_TEST       CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_SAY_HELLO     CTL_CODE(FILE_DEVICE_UNKNOWN, 0x801, METHOD_BUFFERED,
FILE_ANY_ACCESS)
```

这里面有一个宏，叫做 CTL_CODE，它用函数来表示就是：

```
DWORD CTL_CODE_GEN(DWORD lngFunction)
{
    //const DWORD FILE_DEVICE_UNKNOWN = 0x22;
    //const DWORD METHOD_BUFFERED = 0;
    //const DWORD FILE_ANY_ACCESS = 0;
    return (FILE_DEVICE_UNKNOWN * 65536) | (FILE_ANY_ACCESS * 16384) | (lngFunction * 4)
| METHOD_BUFFERED;
}
```

之所以要这么计算，是因为这个 32 位的 IOCTL 的每一位都有不同的含义（IOCTL 每一位的具体含义如下图所示）。如果不遵守这个规则，随意指派控制码，那么在与驱动进行通信时将会蓝屏。

为了方便大家，我已经把这套东西整合成了一个类。

```
#include <advapi32.h>
#pragma comment(lib,”advapi32.lib”)

class cDrvCtrl
{
public:
	cDrvCtrl()//初始化各个变量
	{
		m_pSysPath = NULL;
		m_pServiceName = NULL;
		m_pDisplayName = NULL;
		m_hSCManager = NULL;
		m_hService = NULL;
		m_hDriver = INVALID_HANDLE_VALUE;
	}
	~cDrvCtrl()//清除垃圾
	{
		CloseServiceHandle(m_hService);
		CloseServiceHandle(m_hSCManager);
		CloseHandle(m_hDriver);
	}
public:
	DWORD m_dwLastError; //最后的错误
	PCHAR m_pSysPath; //驱动路径
	PCHAR m_pServiceName; //服务名
	PCHAR m_pDisplayName; //显示名
	HANDLE m_hDriver; //驱动句柄
	SC_HANDLE m_hSCManager; //SCM句柄
	SC_HANDLE m_hService; //服务句柄
	BOOL Install(PCHAR pSysPath,PCHAR pServiceName,PCHAR pDisplayName); //安装驱动服务
	BOOL Start();//启动驱动服务
	BOOL Stop();//停止驱动服务
	BOOL Remove();//移除驱动服务
	BOOL Open(PCHAR pLinkName); //打开驱动句柄
	BOOL IoControl(DWORD dwIoCode, PVOID InBuff, DWORD InBuffLen, PVOID OutBuff, DWORD
OutBuffLen); //IO控制
};
```

```
//打开已经存在的服务
BOOL cDrvCtrl::GetSvcHandle(PCHAR pServiceName)
{
	m_pServiceName = pServiceName;
	m_hSCManager = OpenSCManagerA(NULL,NULL,SC_MANAGER_ALL_ACCESS);
	if (NULL == m_hSCManager)
	{
		m_dwLastError = GetLastError();
		return FALSE;
	}
	m_hService = OpenServiceA(m_hSCManager,m_pServiceName,SERVICE_ALL_ACCESS);
	if (NULL == m_hService)
	{
		CloseServiceHandle(m_hSCManager);
		return FALSE;
	}
	else
	{
		return TRUE;
	}
}

//安装服务服务
BOOL cDrvCtrl::Install(PCHAR pSysPath,PCHAR pServiceName,PCHAR pDisplayName)
{
	m_pSysPath = pSysPath;
	m_pServiceName = pServiceName;
	m_pDisplayName = pDisplayName;
	m_hSCManager = OpenSCManagerA(NULL,NULL,SC_MANAGER_ALL_ACCESS);
	if (NULL == m_hSCManager)
	{
		m_dwLastError = GetLastError();
		return FALSE;
	}
	m_hService = CreateServiceA(m_hSCManager,m_pServiceName,m_pDisplayName,

SERVICE_ALL_ACCESS,SERVICE_KERNEL_DRIVER,SERVICE_DEMAND_START,SERVICE_ERROR_NORMAL,
				m_pSysPath,NULL,NULL,NULL,NULL,NULL);
	if (NULL == m_hService)
	{
		m_dwLastError = GetLastError();
		if (ERROR_SERVICE_EXISTS == m_dwLastError)
		{
			m_hService = OpenServiceA(m_hSCManager,m_pServiceName,SERVICE_ALL_ACCESS);
```

```
            if (NULL == m_hService)
            {
                CloseServiceHandle(m_hSCManager);
                return FALSE;
            }
        }
        else
        {
            CloseServiceHandle(m_hSCManager);
            return FALSE;
        }
    }
    return TRUE;
}

//启动服务
BOOL cDrvCtrl::Start()
{
    if (!StartServiceA(m_hService,NULL,NULL))
    {
        m_dwLastError = GetLastError();
        return FALSE;
    }
    return TRUE;
}

//停止服务
BOOL cDrvCtrl::Stop()
{
    SERVICE_STATUS ss;
    if (!ControlService(m_hService,SERVICE_CONTROL_STOP,&ss))
    {
        m_dwLastError = GetLastError();
        return FALSE;
    }
    return TRUE;

}

//移除服务
BOOL cDrvCtrl::Remove()
{
    if (!DeleteService(m_hService))
    {
```

```
        m_dwLastError = GetLastError();
        return FALSE;
    }
    return TRUE;
}

//打开驱动的符号链接
BOOL cDrvCtrl::Open(PCHAR pLinkName)//example: \\\\.\\xxoo
{
    if (m_hDriver != INVALID_HANDLE_VALUE)
        return TRUE;
    m_hDriver = CreateFileA(pLinkName, GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0);
    if(m_hDriver != INVALID_HANDLE_VALUE)
        return TRUE;
    else
        return FALSE;
}

DWORD cDrvCtrl::CTL_CODE_GEN(DWORD lngFunction)
{
    return (FILE_DEVICE_UNKNOWN * 65536) | (FILE_ANY_ACCESS * 16384) | (lngFunction * 4) |
METHOD_BUFFERED;
}

//和驱动实现通信的核心函数
BOOL cDrvCtrl::IoControl(DWORD dwIoCode, PVOID InBuff, DWORD InBuffLen, PVOID OutBuff,
DWORD OutBuffLen)
{
    DWORD dw;
    return DeviceIoControl(m_hDriver, CTL_CODE_GEN
(dwIoCode),InBuff,InBuffLen,OutBuff,OutBuffLen,&dw,NULL);
}
```

测试的代码如下：

```
int main()
{
    BOOL b;
    cDrvCtrl dc;
    //设置驱动名称
    char szSysFile[MAX_PATH]={0};
    char szSvcLnkName[]="KrnlHW64";;
    GetAppPath(szSysFile);
    strcat(szSysFile,"KrnlHW64.sys");
```

```
	//安装并启动驱动
	b=dc.Install(szSysFile,szSvcLnkName,szSvcLnkName);
	b=dc.Start();
	printf("LoadDriver=%d\n",b);
	//“打开”驱动的符号链接
	dc.Open("\\\\.\\KrnlHW64");
	//使用控制码控制驱动（0x800：传入一个数字并返回一个数字）
	DWORD x=100,y=0;
	dc.IoControl(0x800,&x,sizeof(x),&y,sizeof(y));
	printf("INPUT=%ld\nOUTPUT=%ld\n",x,y);
	//使用控制码控制驱动（0x801：在DBGVIEW里显示HELLOWORLD）
	dc.IoControl(0x801,0,0,0,0);
	//关闭符号链接句柄
	CloseHandle(dc.m_hDriver);
	//停止并卸载驱动
	b=dc.Stop();
	b=dc.Remove();
	printf("UnloadDriver=%d\n",b);
	getchar();
	return 0;
}
```

接下来是测试。**首先进入虚拟机的双机调试环境（必须是双机调试环境，不能仅仅是测试模式或者测试签名模式）**。然后双击 ScmDrvLoader.exe。如果出现以下结果，那就证明成功了。如果不是这个结果，那么多测试几次。如果全都不行，那就说明你的环境配置有问题了。请温习上几节课。