Kernel Level Anti-Spyware Using Device Stack Lock Strategy

Mohammed Gheni Alwan
Computer Science Department, University of Technology/Baghdad
EmaiL: mgaz _ mgaz @yahoo.com

Received on: 26/10/2011 & Accepted on: 1/3/2012

ABSTRACT

This paper is devoted to design and implement an Anti spyware software package. The targeted type is the kernel level spyware which is the most dangerous threat due to the capabilities granted to the spyware code injected in this level. Kernel level is the most trusted level and the code executed at this level will have accessibility to all system resources. This paper will introduce a methodology to lock device stack for any attaching of malicious filter driver, spyware is using filter driver as the main weapon to intercept data exchanged by system devices (physical, logical or virtual) and the I/O manager.

The paper interduces also, a locking methodology for the device stack is presented and all kernel level APIs are explained. The 'keyboard' is the target stack to be locked against famous attack of keyboard logger.

Keywords: device driver stack, windows kernel, IRP, spyware, computer security.

برنامج مكافحة برامجيات التجسس على مستوى النظام بأستخدام آلية غلق المكدس

الخلاصة

هذا البحث مكرس لتصميم وتنفيذ حقيبة برمجية لمكافحة البرامجيات التجسسية من النوع الذي يعمل في مستوى نظام التشغيل، ان هذا النوع يعتبر من أخطرانواع برامجيات التجسس كونه يعمل في منطقة لها الصلاحية المطلقة في استخدام جميع مصادر الحاسبة.أن جميع البرامجيات التي تعمل في مستوى نظام التشغيل تكون هي صاحبة الثقة الاعلى من بين البرامجيات الاخرى.

أن هذا البحث سوف يقوم بتقديم طريقة وآلية لأغلاق منطقة مكدس الاجهزة لمنع دخول أي شفرة مشبوهة غريبة ، أن دخول منطقة مكدس الاجهزة يعتبراحد الوسائل الفتاكة التي تعتمدها برامجيات التجسس وذلك لقدرتها على أعتراض وأستلام البيانات القادمة أو الذاهبة عبر هذه الاجهزة وعدم قدرة برامج الحماية التقليدية من الدخول الى هذه المنطقة.

خلال مراحل عرض هذا البحث سوف يتم تقديم آلية الأغلاق هذه بكل تفاصيلها وبضمنها جميع الدوال المكتبية المستخدمة لهذا الغرض.وسيكون مجال البحث هنا هو لوحة المفاتيح والمكدس الخاص بة.

الكلمات المرشدة: مكدس سواقات النظام، لبنات النوافذ،وحدات بيانات الادخال والاخراج، امنية الحاسبة.

https://doi.org/10.30684/etj.30.9.11
2412-0758/University of Technology-Iraq, Baghdad, Iraq

INTRODUCTION

Spyware refers to programs that steal the user information stored in the user's computer and transmit this information via the Internet to a designated home server without the user being aware of this transmission. Existing anti-spyware solutions are not generic and flexible. These solutions either check for the existence of known spyware or try to block the transmission of the private information at the packet level. [1,2]

The most common methods used to construct keylogging software are as follows:

- A system hook which intercepts notification that a key has been pressed (installed using WinAPI SetWindowsHook for messages sent by the window procedure. It is most often written in C);
- A cyclical information keyboard request from the keyboard (using WinAPI GetAsyncKeyState or GetKeyboardState – most often written in Visual Basic, sometimes in Borland Delphi);
- Using a filter driver (requires specialized knowledge and is written in C).[2] The most effected one is that which uses filter driver due to most anti-spyware dose not reaches that area without a special privileged authorization, this is in a hand and it is not easy to tell if the filter driver is malicious or normal in a second hand.

WINDOWS ARCHITECTURE AND DEVICE STACK

Windows NT allows several driver layers to exist between an application and a piece of hardware. Thus drivers are grouped together in stacks that work together to completely process a request targeted at a particular device object.[3,4,5]

Windows NT uses a layered driver model to process I/O requests. In this model, drivers are organized into stacks. Each driver in a stack is responsible for processing the part of the request that it can handle, if any. If the request cannot be completed, information for the lower driver in the stack is set up and the request is passed along to that driver.[3,4]

This layered driver model allows functionality to be dynamically added to a driver stack. It also allows each driver to specialize in a particular type of function and decouples it from having to know about other drivers.[3,4,5]

In the windows Driver Model, each hardware device has at least two device drivers. One of these drivers, which is the *function driver*, is which it appears to be thought as the device driver; it understands all the details about how to make the hardware work. It's responsible for initiating I/O operations, for handling the interrupts what occur when those operations finish, and for providing a way for the end user to exercise any control over the device that might be appropriate. [1,3,5]

FUNCTIONAL DRIVER

In the windows Driver Model, each hardware device has at least two device drivers. One of these drivers, which is the *function driver*, is which it appears to be thought as the device driver ; it understands all the details about how to make the hardware work. It's responsible for initiating I/O operations, for handling the interrupts what occur when those operations finish, and for providing a way for the user End to exercise any control over the device that might be appropriate. [1,5]

FILTER DRIVERS

A Filter Driver is a special type of layered driver. What sets a filter driver apart from the layered driver is that it is invisible. They attach themselves to any other driver and intercept requests directed at the lower driver's Device objects. It is developed primarily to allow the addition of new functionality beyond what is currently available. The filter driver may either use the services of the original target of the I/O request, or use the services of other kernel-mode drivers to provide value-added functionality.[3,4,5]

Filter drivers are divided into two classes, upper and lower class filters, what is above the function driver is upper driver while what is below the function driver is lower driver. Upper filter drivers see IRPs before the function driver, and they have the chance to support additional features that the function driver doesn't know about. Lower filter drivers see IRPs that the function driver is trying to send to the bus driver. [3,5]

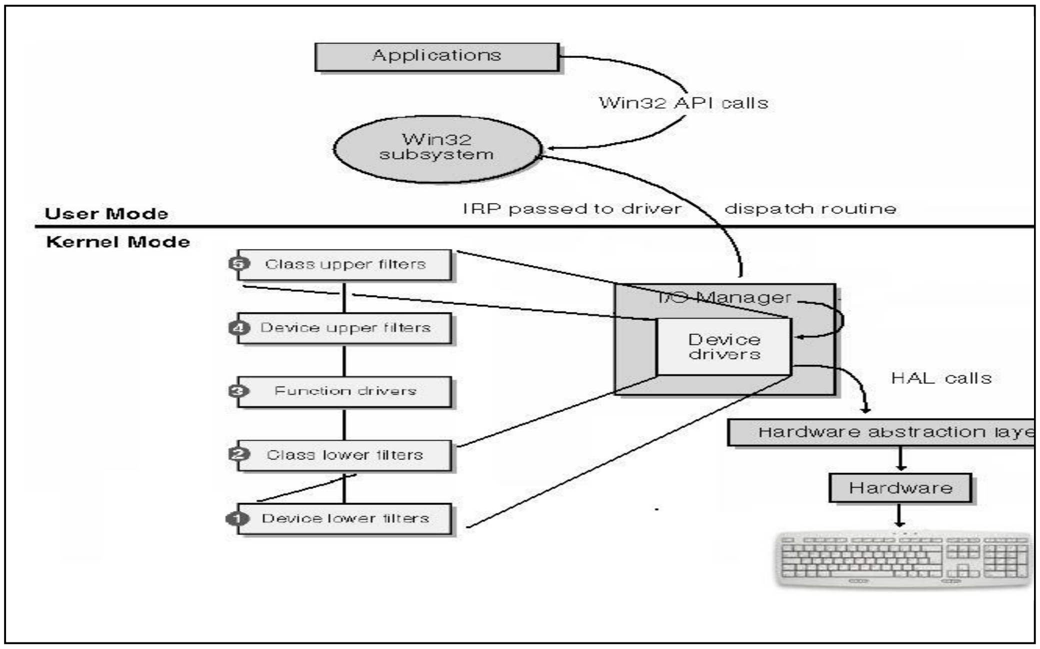


Figure (1): Communication Scheme Between the Application and the Hardware[3]

The **driver**s form the chain with IRP as the data medium. Correspondingly the simplest way to hook data from the device **driver** (and keyboard **driver** in particular) is to attach own specially developed **driver** to the stack with the existing ones.[4,5]

Several *Stack Locations* might exist within an IRP. One for each device belonging to the chain of layers that follow until the IRP reaches the target driver. In

other words, there is a Stack Location for the target driver and an additional one for each filter driver installed on it.[3,4]

```
typedef struct _IRP
        {
                    SHORT Type;
                    WORD Size;
                    PMDL MdlAddress;
                    ULONG Flags;
                    ULONG AssociatedIrp;
                    LIST_ENTRY ThreadListEntry;
                    IO_STATUS_BLOCK IoStatus;
                    CHAR RequestorMode;
                    UCHAR PendingReturned;
                    CHAR StackCount;
                    CHAR CurrentLocation;
                    UCHAR Cancel;
                    UCHAR CancelIrql;
                    CHAR ApcEnvironment;
                    UCHAR AllocationFlags;
                    PIO_STATUS_BLOCK UserIosb;
                    PKEVENT UserEvent;
                    UINT64 Overlay;
                    PVOID CancelRoutine;
                    PVOID UserBuffer;
                    ULONG Tail;
        } IRP, *PIRP;
```

Figure (2) shows IRP (Input/Output Request Packet) structure

Building the Device Stack

Each filter and function driver has the responsibility of building up the stack of device objects, starting from the PDO(Physical Device Object) and working upward.

This can be accomplished with a call to IoAttachDeviceToDeviceStack:[1,4,5]

```
NTSTATUS AddDevice(..., PDEVICE_OBJECT pdo)
                                {
                        PDEVICE_OBJECT fdo;
                        IoCreateDevice(..., &fdo);
                        pdx->LowerDeviceObject                              =
                        IoAttachDeviceToDeviceStack(fdo, pdo);
                                }
```

The first argument to IoAttachDeviceToDeviceStack (fdo-filter driver object) is the address of the created device object. The second argument is the address of the PDO. The second parameter to AddDevice is this address. The return value is the address of whatever device object is immediately underneath fdo, which can be the PDO or the address of some lower filter device object. Figure-3 illustrates the situation when there are three lower filter drivers for your device. By the time AddDevice function executes, all three of their AddDevice functions have already been called. They have created their respective FiDOs (Filter Driver Object) and linked them into the stack rooted at the PDO. When a call IoAttachDeviceToDeviceStack, the address of the topmost FiDO will be returned. [1,4]

Being a topmost FiDO is an easy task, all we need to do is to call IoAttachedDeviceReference which it responsibility is to find the previous top most device object (i.e., FiDO) and return a pointer to it. The pointer will be used to mount new FiDO on it. [1, 4]

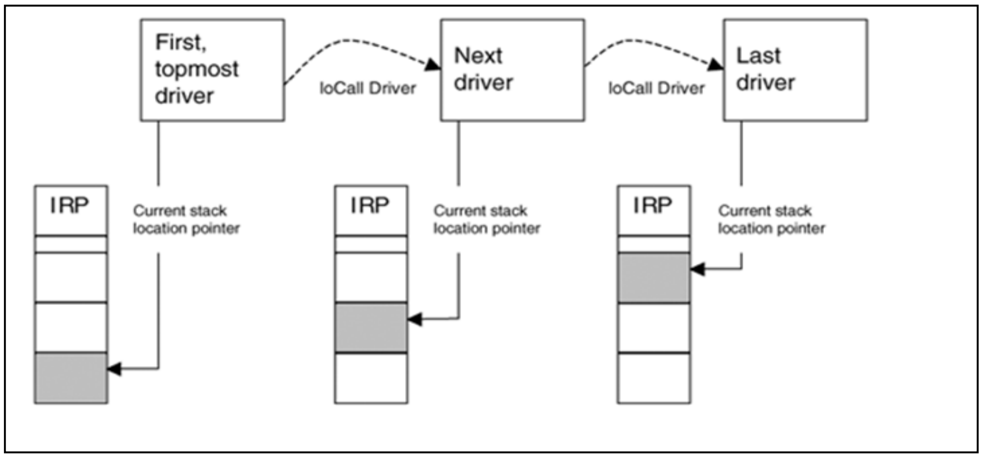

First, topmost driver
IoCall Driver
Next driver
IoCall Driver
Last driver
IRP
Current stack location pointer
IRP
Current stack location pointer
IRP
Current stack location pointer

Figure (3) shows Device stack that contains three drivers[3]

CHALLENGES IN PROTECTING AGAINST SPYWARE

Windows operating has a layered architecture that enable developers to add more functionality to the system or the application, for example windows grants the developer the opportunity to add compression layer to a network interface or add coding to a modem connected to the internet.[6]

This feature is very attractive from developer point of view but it is a night mare for the security developers due to the vulnerability occurred, windows can't prevent software from installing layered software due to the privileges the operating system got from and in the other hand it is not possible to expose the system is this way.[5,6]

Many security software were trying to verify and validate software products before allowing its kernel to be embedded within the system, many methodologies were created to make this task accomplished in a way to promote the security of the system by validating the authenticity of the third party software before grant it the ability to install its kernel there.[7]

Spyware uses different methodologies to embed its sensors within the system (i.e., embedding key logger filter driver within device stack), all methodologies rely on compromising the protection of the system. Anti-spy software tries to expose these methodologies and reject the embedding, previous efforts count on understanding the methodology and design software module to fight back this methodology.[7,8]

This paper is presenting a new style of preventing spyware in a way that it does not matter the type of the spyware or the methodologies used to accomplish the attack. The proposal in this paper introduces locking mechanism for the device driver stack which is the most crucial part of the operating system. The locking is done by a maneuver technique around the sequence that the operating system is designed to install drivers accordingly.

LOCKING DEVICE DRIVER STACK

This paper is presenting a technique to lock the device driver stack according to the following scenario; figure-4 shows the standard architecture of device driver stack.

1- Installing uppermost filter driver using ‘IoGetAttachedDeviceReference’, make sure no other filter driver is above by inserting code inside upper filter driver to drop I/O operation if there is other filter above. To be the upper most the IRP should hold a stack size exactly the same as the location of the trusted upper most filter in the device driver stack.

2- Installing lowestmost filter driver using the pointer to PDO (physical Device Object) , make sure no other filter driver is below , this can be done by using the registry keys and values.

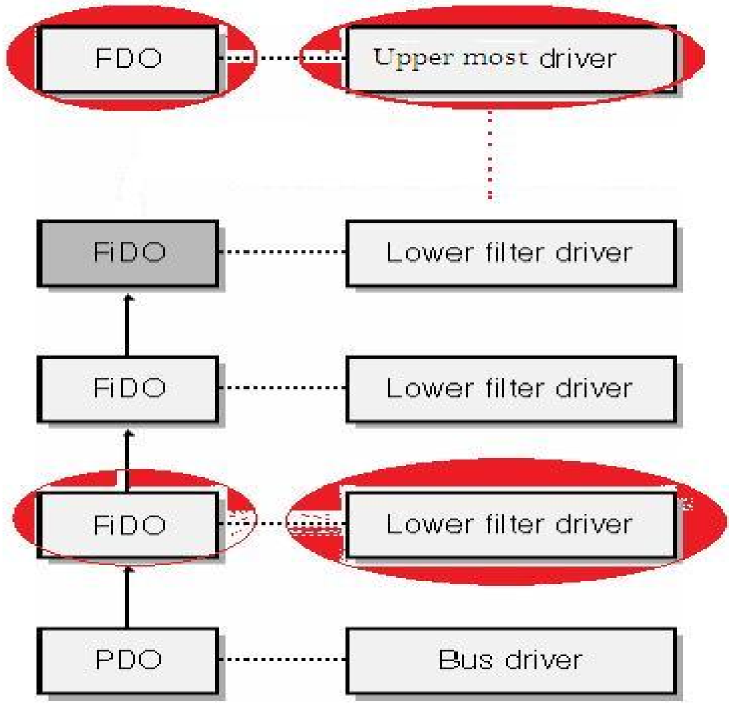


Figure (4) shows the device driver stack after implementing the locking scenario [5]

Driver and send through the device driver stack. this will insure that even if in a way or another a malicious filter driver managed to hook itself with the device driver stack, it will get an encrypted traffic.

After implementing this scenario, the device driver stack should be like figure 6, where lower most and upper most filter drivers are surrounding the whole stack and make sure that I/O traffic are fully encrypted. The following sections will present how to install upper and lower most filter drivers.

INSTALL UPPERMOST FILTER

The key technique used by this proposal is to install two filter drivers; the uppermost and lowermost filter drivers. Microsoft does not provide direct way to do that, thus this proposal introduces someway around this limitation.

This section is introducing the technique used by this paper to install uppermost filter driver:

The following flow chat is presenting the methodology of installing upper most filter driver, all structures and kernel APIs are declared in ntddk.h within the DDK package from Microsoft.

1- **[Initialization]** Define place holder of type PDEVICE_OBJECT and let it be topmost and another one of type PDEVICE_EXTENSION and let it be devExt.
2- **[Move Device Pointer to top most]**Get current topmost device driver by using IoGetAttachedDeviceReference(Physical Device Object) method as the following
   topmost = IoGetAttachedDeviceReference(kbpdo);
   Now topmost points to the head of device stack.
3- **[Create device object]** Create Device Object to be use by driver, use IoCreateDevice() API to create that device. Let created object named mUpperFilter
4- [Get pointer to mUpperFilter extension]
   PDEVICE_EXTENSION pDevExt = mUpperFilter →DeviceExtension;
5- [Chain all device stack created object]
   attach topmost device object to created object in step-3
   pDevExt->TopOfStack = IoAttachDeviceToDeviceStack(mUpperMost, topmost);
6- [end]

INSTALLING LOWER MOST FILTER DRIVER

Installing a lower filter driver is a little bit different than installing upper filter driver, where , some registry values have to be added to some specific registry keys and a re-boot is needed to finish the installation. The adding of these registry values can be done in two ways, either to write a setup program if the device is not of PNP type or to install it using inf file.

The key point in installing lower most filter driver is the following registry keys *device* and *lowerfilter*, where windows installer will first search for all devices registered in the registry, and then looks for other keys within that device such as lowerfilter key and load drivers in the order it appears in that registry key, figure-5 shows the procedure of installing filter drivers

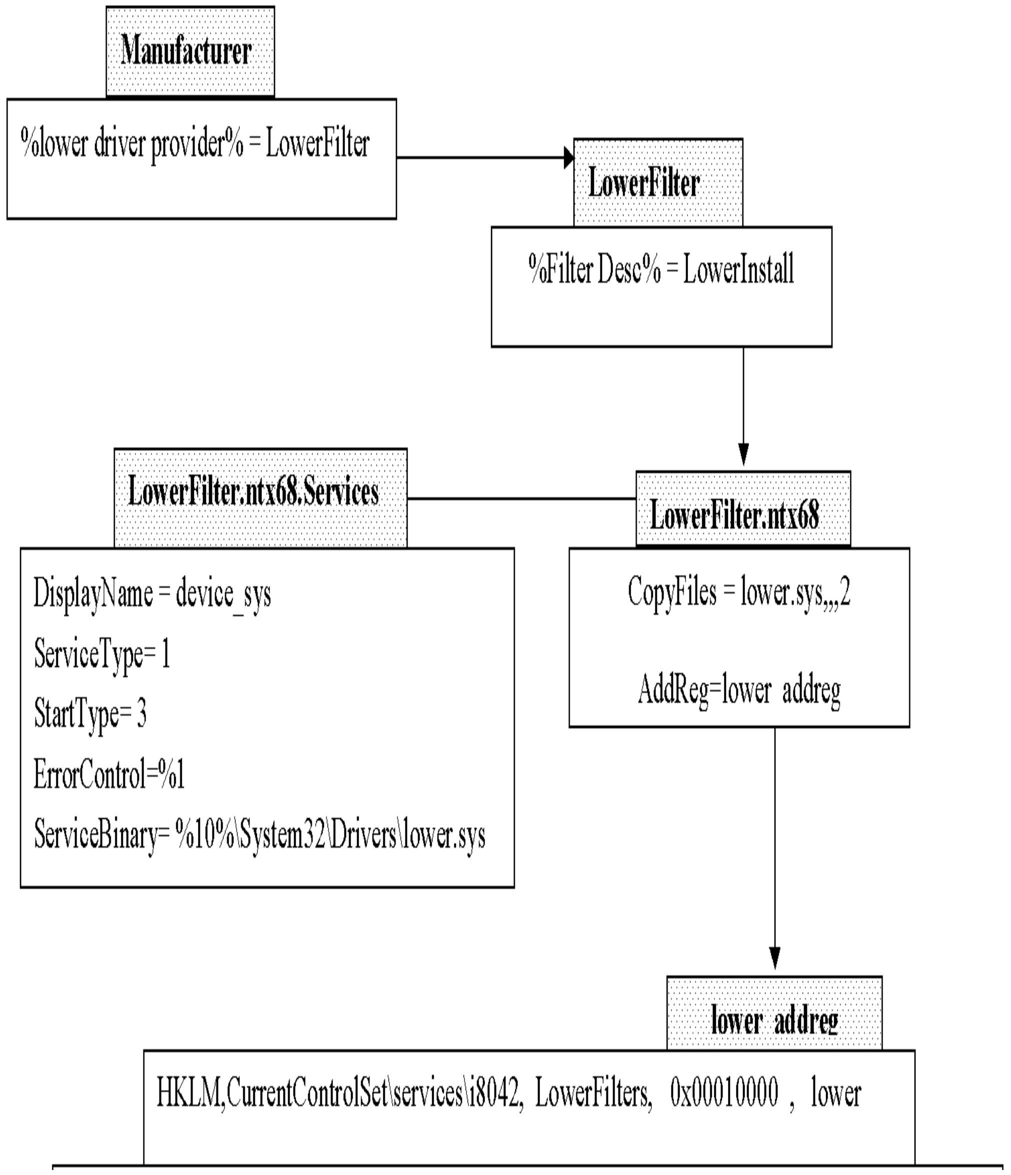
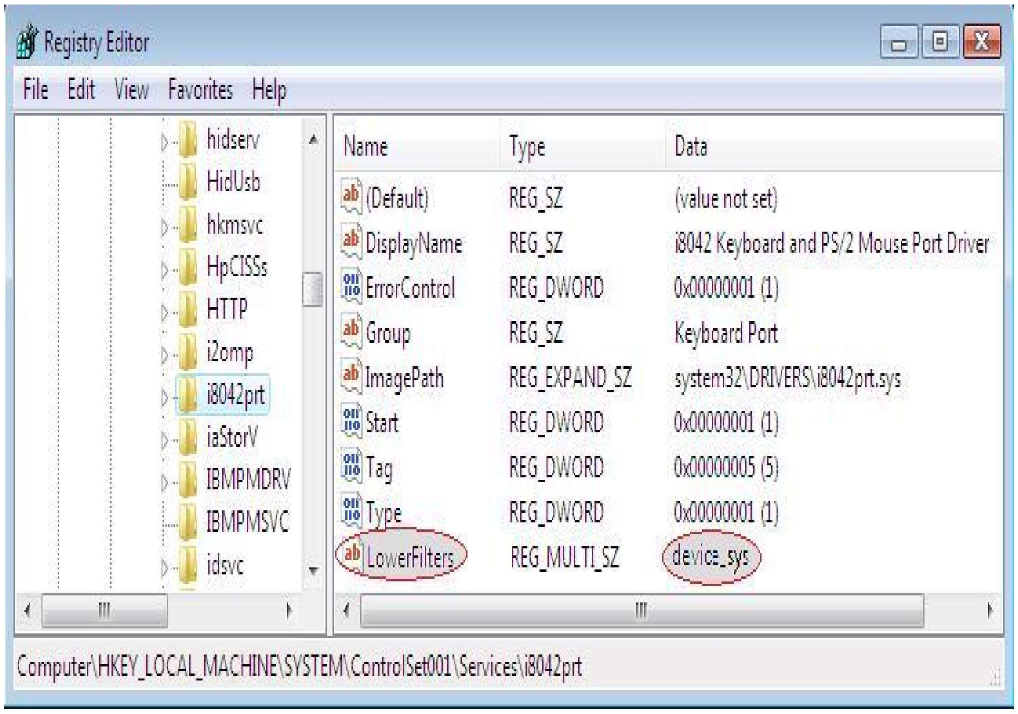

Manufacturer
%lower driver provider% = LowerFilter
LowerFilter
%Filter Desc% = LowerInstall
LowerFilter.ntx68.Services
DisplayName = device_sys
ServiceType= 1
StartType= 3
ErrorControl=%1
ServiceBinary= %10%\System32\Drivers\lower.sys
LowerFilter.ntx68
CopyFiles = lower.sys,,,2
AddReg=lower addreg
lower addreg
HKLM,CurrentControlSet\services\i8042, LowerFilters, 0x00010000 , lower

Figure (5) shows the main components of INF file used to install lower most filter driver

After installing the filter driver, a new value will be added to direct the system to load lower drivers in the sequence appeared in LowerFilters value. The following screen shot shows how registry keys have been added to the registry.



Name	Type	Data
(Default)	REG_SZ	(value not set)
DisplayName	REG_SZ	i8042 Keyboard and PS/2 Mouse Port Driver
ErrorControl	REG_DWORD	0x00000001 (1)
Group	REG_SZ	Keyboard Port
ImagePath	REG_EXPAND_SZ	system32\DRIVERS\i8042prt.sys
Start	REG_DWORD	0x00000001 (1)
Tag	REG_DWORD	0x00000005 (5)
Type	REG_DWORD	0x00000001 (1)
LowerFilters	REG_MULTI_SZ	device_sys

Computer\HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\i8042prt

Figure (6) shows a screen shot for the registry after installing lower most filter driver

When the installation is completed the upper filter driver can talk to the lower filter driver and an enumeration to stack size and layers can be accomplished, this way any third party software tries to install a spyware within this stack will be detected and prevented, this will lock the device stack against any spyware or malicious third party modules.

CONCLUSIONS

As a conclusion from the work presented in this paper, it has been proved that multi-layer operating system is hard to be secured against embedding malicious software module, where the nature of the environment is prohibited rejecting automation. The real assistance can come by embedding software and hardware modules. The hardware modules will play the role the security gate guard before accepting new software module that want to inject itself into the system, it is not possible to rely totally on the security presented by the operating system due to the fact, it adopts the same layering architecture. USB dongles can serve well in this issue as it could be combined with kernel level drivers.

The stack driver is a very crucial element in the security schema imposed by windows operating system, where authorization can be compromised easily if a malicious software managed to inject a filter driver into a device stack of any type, this will not just provide the access to the data exchanged as I/O traffic but to compromise operating system security. User accounts can easily accessed and system resources can be deployed.

REFERENCES

[1]Art Baker and Jerry Lozano, "The Windows 2000 Device Driver Book", second edition , prentice-Hall, 2001
[2]Sherman S.M. Chow and Lucas C.K. Hui , "A generic anti-spyware solution by access control list at kernel level ", 2004
[3]Walter Oney, "Programming the Microsoft Windows Driver Moderl", second edition ,2003
[4]Microsoft Division, "Microsoft Windows 2000 Driver Design Guide", Microsoft press,2000
[5]ark E. Russinovich and David A. Solomon, "Windows internals, Fifth Edition", Microsoft, 2009
[6]Andrew S. Tanenbaum, "Modern Operating Systems", second edition , prentice-Hall 2001.
[7]Nikolay Grebennikov, "Keyloggers: How they work and how to detect them ", 2007
[8]Charis Cant, "writing Windows WDM device drivers", Berkeley, 1999