Any application-defined hook procedure on my machine?

By Posted by zairon on December 6, 2006

Published: 2006-12-05 · Archived: 2026-04-05 15:57:30 UTC

Some times ago my antivirus didn't recognize a malware on my machine; the malware installed a keylogger and it did run silently in background for some hours. In general, you can discover the presence of a keylogger looking at the net traffic log or looking for some suspicious file name inside the task manager list but sometimes an application able to find out hooks will come in handy. At first I tried to find some informations browsing the net without luck. Hmm, no one has written such a program before? Is it possible or am I unable to use google??? I only got one usefull hint about the problem, nothing else.
Well, I decided to take a look at the problem. All my investigations are done under Windows XP/SP1.

Before starting with the analysis I want to show you the declaration of the function used to set a windows hook:
HHOOK SetWindowsHookEx(
int idHook, // Specifies the type of hook procedure to be installed.
HOOKPROC lpfn, // Pointer to the hook procedure
HINSTANCE hMod, // Handle to the DLL containing the hook procedure pointed to by the lpfn parameter
DWORD dwThreadId // Specifies the identifier of the thread with which the hook procedure is to be associated
);
It's always good to know which parameters are passed to the function because you'll have to deal with them later. Ok, time to start…

A possible starting point is represented by win32k.sys file, it's everything inside it. Looking at the implementation of SetWindowsHookEx I've seen a call to HMAllocObject. This function is not really known but if you have ever read 'Using Softice' help file you surely read the phrase: "The routine HMAllocObject creates USER object types…". Interesting, setting a bpx over the function I got the following informations:

.text:BF853AAB push 34h ; nBytes
.text:BF853AAD push 5 ; TYPE_HOOK
.text:BF853AAF push dword ptr [edi+3Ch] ; PTHREADINFO.rpdesk
.text:BF853AB2 push edi ; PTHREADINFO
.text:BF853AB3 call _HMAllocObject@16 ; HMAllocObject(x,x,x,x)

The function takes four parameters and the 3° parameter is related with the type of object that is created. In this specific case I'm dealing with hook so the type is TYPE_HOOK and HMAllocObject returns a pointer to a structure named HOOK, it contains general informations about the new object and it looks like:
typedef struct _HOOK {
ULONG hHook;
ULONG cLockObj;
PTHREADINFO pti;

```
ULONG rpdesk;
ULONG pSelf;
struct _HOOK *phkNext;
int iHook;
ULONG offPfn;
unsigned int flags;
int ihmod;
PTHREADINFO ptiHooked;
PDESKTOP rpdesk;
} HOOK, *PHOOK;
```

Fields:
– *hHook*
Handle to the hook procedure, it's the value returned by SetWindowsHookEx and it comes from HMAllocObject
– *clockObj*
!?!
– *pti*
Pointer to THREADINFO structure of the process which sets the hook
– *rpdesk*
!?!
– *pSelf*
Pointer to this HOOK structure
– *phkNext*
Next structure in the hook chain
– *iHook*
Hook type (i.e. WH_MOUSE or WH_KEYBOARD). This is the first parameter passed to SetWindowsHookEx
– *offPfn*
Offset of the filter procedure, it is obtained by a simple substration between the address of the hook procedure and the initial address of the dll
– *flags*
HF_xxx flags (HF_GLOBALS, HF_LOCAL, HF_DESTROYED…)
– *ihmod*
Number of hooks set into the module
– *ptiHooked*
Pointer to THREADINFO structure of the hooked thread. If HF_GLOBAL is setted the pointer is setted to NULL because the hook works for every running thread
– *rpdesk*
!?!

As you can see the fields are not all explained, I'll try to gain more informations.

Some of the fields are filled by HMAllocObject itself and they are non TYPE_HOOK related field; I mean, the HOOK structure contains informations about the hook you want to install (i.e. the hook type, the offset of the filter

procedure) and informations not so strictly related with the hook (i.e. phkNext, pSelf). Since of all the work is done trying to understand if there are global hooks installed on the system I'm only interested in TYPE_HOOK related fields and these are:

1. **pti**

pti is really usefull because it gives me the possibility to access the process that has installed the hook. The first dword of THREADINFO structure is the pointer to ETHREAD structure which contains a pointer to EPROCESS structure. Once you have EPROCESS you can read the image file name (Imagfilename field) and the pid (UniqueProcessId) of the process that has setted the hook. At this point you can also read which dlls are loaded by the process or get any other usefull informations about it, especially which dll has the filter procedure inside. If you are not totally confident with undocumented structures you can even retrieve the informations using functions from Psapi library (i.e. EnumProcessModules/GetModuleFileNameEx to obtain a loaded dll…).

2. **iHook**

without it you won't be able to know which hook was installed.

3. **offPfn**

only necessary if you want to know where the filter procedure is located at.

Starting from the HOOK structure I can retrieve all the needed informations about the HOOK but I need a fundamental information, maybe the most important one. Where can I find the HOOK structure? As I said before I only got an hint about the problem, it comes from an old thread at sysinternals forum (htt://forum.sysinternals.com/); I don't remember the name of the user, thanks to him btw. The hint sounds like: "look at aphkStart", nothing else. aphkStart is defined as:

PHOOK aphkStart[CWINHOOKS];

It's an array of pointers to HOOK structure, one for each possible WH_xxx hook; the pointer at index 'i' is NULL if you are not under the hook. As far as I have seen there are two aphkStart arrays, one for local hooks (inside THREADINFO structure) and one for global hooks (inside DESKTOPINFO structure). To find out if a global hook has been installed you only have to scan the entire aphkStart array (the one inside DESKTOPINFO) looking for a not NULL entry, if you find it you are under a global hook.

The picture below represents a little sum-up. You start from the TEB of the current process, the process that check the installed hooks. When you have the TEB you can pass through some structures:

1. From TEB.Win32ThreadInfo you get THREADINFO structure
2. From THREADINFO.pDeskInfo you get DESKTOPINFO structure
3. If DESKTOPINFO.aphkStart[i] is not NULL you get HOOK structure of WH_i hook otherwise WH_i hook is not installed and you can check the next one, WH_i+1
4. From HOOK.pti you get THREADINFO structure of the process that has setted the hook
5. From THREADINFO.pEthread you get ETHREAD structure
6. From ETHREAD.ThreadsProcess you get EPROCESS structure

Inside EPROCESS there are many informations about the process, just read the necessary ones.


Way to installed WH_xxx hook
