

# Kernel-based monitoring on Windows (32/64 bit)

by

Florian Rienhardt

peanut@bitnuts.de

2011/11/18 (updated 2012/02/03)

## Abstract

*Since malware works fast and quiet there is demand to analyze such programs at some central point. There is nothing as central as the kernel of an operating system. This whitepaper describes how to monitor your Windows-based system by using a mini-filter driver intercepting IRP\_MJ-Functions in its PreOperation-Callback. By following Microsofts' recommendation and guidelines for multi platform (e.a. Microsoft Windows versions) compatible driver development, the resulting drivers are called kernel mini-filter drivers that are reliable and compatible with all modern versions of Microsoft Windows (2000, XP, Server, 7, 8) – including their 64 bit versions.*

**Table of contents**

**1 Introduction..... 1**

**2 Monitoring by using a mini-filter driver..... 2**

**3 What to implement?..... 4**

**4 Drawbacks of such a solution..... 6**

**5 Example (driver demo) ..... 6**

**6 Conclusion..... 6**

## 1 Introduction

These days malware often uses a zero-day exploit within commonly used applications like browsers, multimedia- or portable document viewers to bootstrap the process of infecting your system with a trojan or bot for example. To avoid DEP and all the other security stuff modern operating systems like Microsoft Windows come along, most malware uses ROP-like (return oriented programming) approaches to execute their evil code even if common protection mechanisms are enabled. In most cases the exploit starts or acts as a dropper that downloads and executes the intended malware. There is not much transient code executed in first instance. Authors of malware often use some general checks against virtual machines, anti virus software and debuggers to avoid getting discovered – bad news if you want to quickly analyze such smuck, because if you run such detection software, modern malware just destroys themselves and you will never see what such malware would write or execute on your system. You only get the exploit or a dropper that must be analyzed (decrypted, disassembled and debugged) to find out what such a beast really copies to your system's hard disk drive and executes then.

In most cases these droppers are not very interesting, because a lot of web sites are currently hit by already known exploit kits that just download and execute the malicious code using an also well known (or polymorphed version of such a) dropper – thus what we are really interested in, is the malicious code and its behavior on your system; meaning what will be downloaded and written to your machine. Since malware often works fast and quiet there is demand to analyze such programs at some central point. There is nothing as central as the kernel of an operating system.

In the past years I tried different approaches to analyze malware fast and without the need to fully debug or disassemble such programs. Some of my implemented solutions used user mode hooking<sup>1</sup> in the same fashion as used in the Google Chrome browser's sandbox or like described in Detours<sup>2</sup>. Other approaches used kernel mode hacks like the well known SSDT-hooks. There are pros and cons to use one or the other, but at the end I was not satisfied with all these solutions, because most of them are a bit cracky-hacky as I am calling it.

Well, I started thinking about what and how Microsoft would (and/or will) implement central monitoring stuff in their operating systems. This whitepaper summarizes the stuff I found and gives you a possible solution. What is outlined here is no secret, nor is it something totally new, but surfing around the web I did not find a

---

<sup>1</sup> Also known as hot-patching or API re-direction.

<sup>2</sup> Galen Hunt and Doug Brubacher, "Detours: Binary Interception of Win32 Function", <http://research.microsoft.com/en-us/projects/detours/>

general whitepaper or tutorial so I decided to write one to make life a bit easier if you want to start writing some kernel-based monitoring stuff for Windows.

## 2 Monitoring by using a mini-filter driver

Back in the days where Windows 2000 and XP were wide spread there were commonly used techniques to monitor and detect file creation, the creation of registry settings and even the execution of processes, drivers, etc<sup>3</sup>. A lot of monitoring tools and experts intercepted kernel functions to monitor on a system wide basis.

Most of these drivers hooked the System Service Table<sup>4</sup>. A so called kernel hacker<sup>5</sup> had just to launch such a SSDT hooking driver that controls vital parts of the kernel and was likely able to monitor the system using a central anchor (the kernel API represented through the SSDT). This technique seems to be some kind of "silver bullet" because it runs in privileged kernel mode having access to nearly all objects of the operating system and cannot be fooled by classic API-redirection/hooking techniques that take place in user mode. Unfortunately hooking vital kernel functions is not the appropriate (and officially documented) way implementing reliable software today. It was and is not only used by hackers but also by authors of rootkits and other malicious programs; obviously, redirecting vital parts of the kernel perfectly suits the need of malware authors trying to hide their actions or to control your operating system<sup>6</sup>. Understandably Microsoft does not recommend ISVs using this technique, because the functionality of (undocumented) kernel functions may change during the life cycle of a product like Microsoft Windows (think about a service pack for example, kernel updates, etc.) and patching vital kernel functions is not only critical to system stability it is also critical to your system's security<sup>7</sup>. Since Microsoft introduced its 64 bit versions of Windows, including additional protection mechanisms like only loading signed drivers and its Kernel Patch Protection<sup>8</sup>, it is even not possible to do such hacks on 64 bit Versions of Windows without running into a nasty BSOD. This is the most prominent example why you should NOT use such unofficial hacks to

---

<sup>3</sup> E.g. see Bassov A., (2005): "Hooking the native API and controlling process creation on a system-wide basis", The Code Project [http://www.codeproject.com/system/soviet\\_protector.asp](http://www.codeproject.com/system/soviet_protector.asp); Or Mark Russinovich's monitoring tools like TokenMon etc. on <http://www.sysinternals.com>.

<sup>4</sup> For more details on SSDT-hooking a good reference is Hoglund G., Butler J., (2005): "Rootkits - Subverting the Windows Kernel", Addison-Wesley/Pearson Education, New Jersey

<sup>5</sup> The name kernel hacker does not name an evil hacker, I just mean enthusiasts like me "hacking the kernel".

<sup>6</sup> For more details I highly recommend the Book of Hoglund G., Butler J., (2005): "Rootkits - Subverting the Windows Kernel".

<sup>7</sup> E.g. see <http://www.matousec.com/info/articles/plague-in-security-software-drivers.php> for more details on this topic.

<sup>8</sup> [http://en.wikipedia.org/wiki/Kernel\\_Patch\\_Protection](http://en.wikipedia.org/wiki/Kernel_Patch_Protection)

build reliable software on. If Microsoft decides to change parts of its kernel architecture (like in 64bit versions of Microsoft Windows) your driver crashes the whole system. Well, there are ways to bypass Kernel Patch Protection, but it seems to be kind of difficult and hacky<sup>9</sup>.

Question: Is it possible to monitor on a system wide basis without hooking native kernel API functions? Yes indeed. Microsoft highly recommends not hooking the native API in software and suggests ISVs to use so called mini-filter drivers instead.

In "Kernel Data and Filtering Support" Microsoft explains how things should be done in a clean and reliable way without hacking the kernel<sup>10</sup>. As Microsoft notes, several ISVs have requested the ability to monitor (or block) several objects like processes, drivers or files without hooking native API functions. Microsoft concludes that there still exist (or existed) well known and supported functionality that could be used instead, to achieve the desired behavior. Microsoft states that in particular, a file system mini-filter can be utilized such, that it can monitor the creation or change of files, loading of modules etc. in both user mode and kernel mode. Following these basic guidelines a kernel hacker ends up in a driver that is reliable and compatible with all modern versions of Microsoft Windows (2000, XP, Server, 7, 8) – including their 64 bit versions.

Microsoft describes<sup>11</sup> mini filters like: Quotation [...] *“A file system filter driver intercepts requests targeted at a file system or another file system filter driver. By intercepting the request before it reaches its intended target, the filter driver can extend or replace functionality provided by the original target of the request. Examples of file system filter drivers include anti-virus filters, backup agents, and encryption products. To develop file systems and file system filter drivers, use the IFS (Installable File System) Kit, which is provided with the Windows Driver Kit (WDK).”* [...] *“The Filter Manager is a file system filter driver provided by Microsoft that simplifies the development of third-party filter drivers and solves many of the problems with the existing legacy filter driver model, such as the ability to control load order through an assigned altitude. A filter driver developed to the Filter Manager model is called a minifilter. Every minifilter driver has an assigned altitude, which is a unique identifier that determines where the minifilter is loaded relative to other minifilters in the I/O stack. Altitudes are allocated and managed by Microsoft.”* [...]

---

<sup>9</sup> See <http://uninformed.org/index.cgi?v=3&a=3&p=3> for more details.

<sup>10</sup> For more details search for “Kernel Data and Filtering Support” available at <http://download.microsoft.com/download/4/4/b/44bb7147-f058-4002-9ab2-ed22870e3fe9/Kernel%20Data%20and%20Filtering%20Support%20for%20Windows%20Server%202008.doc>

<sup>11</sup> <http://msdn.microsoft.com/de-de/windows/hardware/gg462968>

The mini-filter support is documented at [msdn.microsoft.com](http://msdn.microsoft.com), especially in the part describing the Driver Development Kit (DDK)<sup>12</sup>. Microsoft published great examples on how to write file system drivers that show how an anti virus solution could be implemented. I highly recommend reading the DDK documentation, the MSDN about filter driver development and do not miss to check out Microsoft's great examples shipped with the WDK<sup>13</sup>.

### 3 What to implement?

So what to do folks? Well, for me Microsofts' suggestions in "Kernel Data and Filtering Support" sound something like: "Write a mini-filter driver that..."

- i) intercepts a IRP\_MJ\_-Function in its PreOperation-callback<sup>14</sup>,
- ii) analyze and check its parameter block<sup>15</sup> and monitor (e.g. via DbgPrint()), adjust or even block such information.

Again and in short prose we are using a mini-filter driver intercepting a IRP\_MJ-Function in its PreOperation-Callback<sup>16</sup> and check its FLT\_PARAMETERS parameter block<sup>17</sup>. E.g. Such IRP\_MJ-Functions could be

---

<sup>12</sup> <http://download.microsoft.com/download/e/b/a/eba1050f-a31d-436b-9281-92cdfeae4b45/Filter-DriverDeveloperGuide.doc>

<sup>13</sup> <http://msdn.microsoft.com/en-us/windows/hardware/gg487428>  
<http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=11800>

<sup>14</sup> Microsoft states "A file system minifilter driver uses one or more preoperation callback routines to filter I/O operations. A minifilter driver registers a preoperation callback routine for a particular type of I/O operation by storing the callback routine's entry point in the OperationRegistration member of the FLT\_REGISTRATION structure. The minifilter driver passes this member as a parameter to FltRegisterFilter in its DriverEntry routine. Minifilter drivers receive only those types of I/O operations for which they have registered a preoperation or postoperation callback routine." (for more details about PreOperation-Callbacks and the whole topic navigate to: [http://msdn.microsoft.com/en-us/library/windows/hardware/ff557336\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff557336(v=vs.85).aspx))

<sup>15</sup> Meaning you have to check out "Data->Iopb->Parameters[...]"

<sup>16</sup> Additional note: You can also use its PostOperation-Callback, whether kind of information of the desired object (file, process, section or what ever) you are interested in. In some situations you get much more information in the PostOperation-Callback because generic I/O related stuff was done at this point to give you more details on the operation. As Microsoft states "When the filter manager calls a minifilter driver's postoperation callback routine for a given I/O operation, the minifilter driver temporarily controls the I/O operation." (see: [http://msdn.microsoft.com/en-us/library/windows/hardware/ff557325\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff557325(v=VS.85).aspx)). And you can still cancel a successful CREATE operation in a PostOperation-Callback anyway.

<sup>17</sup> FLT\_PARAMETERS is described at [http://msdn.microsoft.com/en-us/library/windows/hardware/ff544684\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff544684(v=vs.85).aspx)

- IRP\_MJ\_CREATE
- IRP\_MJ\_CREATE\_NAMED\_PIPE
- IRP\_MJ\_READ
- IRP\_MJ\_WRITE
- IRP\_MJ\_SET\_VOLUME\_INFORMATION
- IRP\_MJ\_DIRECTORY\_CONTROL
- IRP\_MJ\_FILE\_SYSTEM\_CONTROL
- IRP\_MJ\_NETWORK\_QUERY\_OPEN
- IRP\_MJ\_MDL\_READ
- IRP\_MJ\_MDL\_READ\_COMPLETE
- IRP\_MJ\_PREPARE\_MDL\_WRITE
- IRP\_MJ\_MDL\_WRITE\_COMPLETE
- IRP\_MJ\_VOLUME\_MOUNT
- IRP\_MJ\_VOLUME\_DISMOUNT

for more see Microsoft's WDK driver examples.

To fire up your own project I recommend to start up with one of Microsoft's filter driver samples like the scanner demo that demonstrates how to implement a basic virus- or content scanner.

Another good resource is the New Zealand Honeynet Alliance. I highly recommend to check out Ramon Steenson's Christian Seifert's Capture (especially the source code of CaptureFileMonitor)<sup>18</sup>.

If you are interested in module blocking as described in Microsoft's papers you should intercept IRP\_MJ\_ACQUIRE\_FOR\_SECTION\_SYNCHRONIZATION in order to block images from loading. You could modify the TargetFileObject in the FLT\_CALLBACK\_DATA structure<sup>19</sup>.

---

<sup>18</sup> Go to <https://projects.honeynet.org> or just google for CaptureFileMonitor.c

<sup>19</sup> More details are available at [http://www.winvistatips.com/modifying-targetfileobject-acquire\\_for\\_section\\_synch-t196674.html](http://www.winvistatips.com/modifying-targetfileobject-acquire_for_section_synch-t196674.html)

Access to files can be monitored by setting a callback for IRP\_MJ\_CREATE. Use FltGetFileNameInformation() with parameter Data and FltParseFileNameInformation to receive the originated filename. Just have a look at Microsoft's mini-filter driver passThrough bundled with the WDK. This driver sets callbacks for all IRP\_MJ-Calls, so you can simply adjust this driver for your needs.

## 4 Drawbacks of such a solution

Well, such monitoring drivers might help during some initial analysis; as the word *might* implies, there exist some drawbacks. If you're getting infected by some sticky-beast<sup>20</sup> that makes use of 0-days enlarging its privileges to ring-0 (kernel mode) and using rootkit technology to hide its objects by using techniques you are **done** – it is just that simple. This is true for session based malware that will be executed on the fly without using well known API-functions to load their malicious code e.g. like LoadLibrary, CreateFile or CreateProcess etc.

Another drawback comes in place if you are using any scripting-host on your machine: E.g. Java-Executables, Python, Ruby or Perl scripts might also be malicious but, but it is likely that you will not see the script as an originator; in most cases I assume that you will just see the scripting host which does not seem to be suspicious in first instance (maybe).

## 5 Example (driver demo)

While having some time off duty this winter I started analyzing Microsoft's "heavy duty" passThrough DDK-example driver and tried to build a simple mini-filter driver that just checks for IRP\_MJ\_CREATE in its preoperation callback. The driver determines the originating/corresponding filename for that IRP-call by using FltGetFileNameInformation. The resulting string will be printed via DbgPrint. You can use DebugView (download at <http://technet.microsoft.com/en-us/sysinternals/bb896647>) to trace what this demonstration driver is doing. Download my demo driver at <http://www.bitnuts.de/IrpMjCrtMon.zip>

## 6 Conclusion

Using a kernel mini-filter driver makes it easy to screen your Windows based operating system on a system wide basis. Such drivers might act as a first step in analyzing the behavior of malicious code trying create or modify files on your system without the painful process of debugging or disassembling an exploit or dropper in first in-

---

<sup>20</sup> I am talking about a targeted attack, remember Stuxnet for example.

stance<sup>21</sup>. Such drivers will not avoid further analysis, they just give you some fast and initial results on what is going on.

If you have any questions, suggestions or what ever feel free and contact me. Feedback is appreciated. Last but not least I would like to thank Pai Wu for his feedback and his demonstration driver.

---

<sup>21</sup> It is also possible to monitor the load of modules as described in "Kernel Data and Filtering Support".