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

by

Florian Rienhardt

peanut@bitnuts.de

Abstract

Since malware works fast and quiet there is demand to analyze, track and block such scrap at some central point. There is nothing as central as the kernel of an operating system. This white paper describes how to monitor and protect your Windows-based system by using a minifilter driver intercepting IRP_MJ-Functions in its PreOperation-Callback. The white paper also discusses some basic analyzing and protection drivers I have written in the past. By following Microsofts' recommendation and guidelines for multi platform compatible driver development, the resulting drivers are called kernel minifilter drivers that are reliable and compatible with all modern versions of Microsoft Windows (2000, XP, Vista, Server, 7 and 8) – including their 64 bit versions. Minifilter drivers are powerful tools to track and mitigate against many kinds of malware out there. Once you have build up your own minifilter drivers they are like a Swiss Army Knife. I highly encourage everyone in the Windows based security scene to have a deep look into the powerful stuff one can achieve with minifilter drivers.

last updated 2015/02/12

Disclaimer

powered by

excubits.com

Excubits UG (haftungsbeschränkt)

Simple. Secure. Excubits.

last updated 2015/02/12

Table of contents

1 Introduction **1**

2 Kernel-based monitoring by using a minifilter driver **2**

3 How to implement a monitoring minifilter driver **5**

4 Example drivers **7**

4.1 Monitoring IRP_MJ_CREATE 7

4.2 A minifilter that monitors executables written on your disk 8

4.3 Tracking executables on Windows 9

4.4 Tuersteher Light: A Path Based Application Whitelisting Filter Driver 10

4.4.1 Configurate and start up the driver 11

4.4.2 Tuersteher Light For Windows XP 13

4.5 Building a totally locked down Windows for POS-, ATM- and kiosk-mode-Envrioments 14

5 Catch (targeted) malware and other cyber attacks **16**

6 Drawbacks **18**

7 Conclusion **20**

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 a polymorphic 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[1] in the same fashion as used in the Google Chrome browser's sandbox or like described in Detours[2]. 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 white paper summarizes the stuff I have 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

[1] Also known as hot-patching or API re-direction.

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

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

2 Kernel-based monitoring by using a minifilter 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[3]. 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[4]. A so called kernel hacker[5] 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[6]. 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[7]. Since Microsoft introduced its 64 bit versions of Windows, including additional protection mechanisms like only loading signed drivers and its Kernel Patch Protection[8], it is even not possible to do such hacks on 64 bit Versions of Windows without running into a nasty BSOD. This

---

[3] 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; Or Mark Russinovich's monitoring tools like TokenMon etc. on http://www.sysinternals.com.

[4] 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

[5] The name kernel hacker does not name a shady hacker, I just mean enthusiasts like me "hacking the kernel".

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

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

[8] http://en.wikipedia.org/wiki/Kernel_Patch_Protection

is the most prominent example why you should NOT use such unofficial hacks to 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[9].

Question: Is it still 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 minifilter drivers instead. All modern malware scanners use minifilter drivers and do not hook the native kernel API anymore. There are also a lot of specialized security products out there that are able to block malicious files from getting written to your disk or getting executed[10].

In "Kernel Data and Filtering Support" Microsoft explains how things should be done in a clean and reliable way without hacking the kernel[11]. 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 minifilter 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

---

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

10 The following list just gives you some examples of products that support global filtering in some fashion:

- Microsoft AppLocker
  http://technet.microsoft.com/en-en/library/dd723678(v=ws.10).aspx
- TripWire
  http://www.tripwire.com/it-security-software/security-configuration-management/file-integrity-monitoring/
- Bit9 Parity Suite
  https://www.bit9.com/products/bit9-parity-suite.php
- CoreTrace Bouncer
  http://www.coretrace.com/products-2/bouncer-overview
- Lumension Application Control
  http://www.lumension.com/application-control-software.aspx
- McAfee Application Control
  http://www.mcafee.com/de/products/application-control.aspx
- Application Access Control
  http://www.tricerat.com

11 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/Kernal%20Data%20and%20Filtering%20Support%20for%20Windows%20Server%202008.doc

modern versions of Microsoft Windows (2000, XP, Vista, Server, 7 and 8) – including their 64 bit versions.

Microsoft describes[12] mini filters like:

> *[...] "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." [...]*

The minifilter support is documented at msdn.microsoft.com, especially in the part describing the Driver Develompent Kit (DDK)[13].

Microsoft published great examples on how to write file system drivers that show how an anti virus solution could be implemented[14]. 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[15]. If your are interested in analyzing drivers I highly recommend to study the drivers written for the Honey-

[12] http://msdn.microsoft.com/de-de/windows/hardware/gg462968

[13] http://download.microsoft.com/download/e/b/a/eba1050f-a31d-436b-9281-92cdfeae4b45/FilterDriverDeveloperGuide.doc

[14] PassThrough File System Minifilter Driver
http://code.msdn.microsoft.com/windowshardware/passThrough-File-System-f9975611
Scanner File System Minifilter Driver
http://code.msdn.microsoft.com/windowshardware/Scanner-File-System-426c8cbe
Windows 8 Driver Samples
http://code.msdn.microsoft.com/windowshardware/Windows-8-Driver-Samples-5e1aa62e/view/SamplePack/4?sortBy=Popularity

[15] http://msdn.microsoft.com/en-us/windows/hardware/gg487428
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=11800

net project[16]. They give you great real live code regarding detection of potential malware.

3 How to implement a monitoring minifilter driver

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

i) intercepts a IRP_MJ_-Function in its PreOperation-callback[17],

ii) analyze and check its parameter block[18] and monitor (e.g. via DbgPrint()), adjust or even block such information.

Again and in short prose we are using a minifilter driver intercepting a IRP_MJ-Function in its PreOperation-Callback[19] and check its FLT_PARAMETERS parameter block[20]. E.g. Such IRP_MJ-Functions could be

- IRP_MJ_CREATE
- IRP_MJ_CREATE_NAMED_PIPE
- IRP_MJ_READ

16 See https://projects.honeynet.org/svn/capture-hpc/capture-hpc/tags/2.5/capture-client/KernelDrivers/CaptureKernelDrivers/FileMonitor/CaptureFileMonitor.c for example. It gives you a great overview on IRP_MJ-filtering on a real world scenario.

17 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)

18 Meaning you have to check out "Data->Iopb->Parameters[...]"

19 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). And you can still cancel a successful CREATE operation in a PostOperation-Callback anyway.

20 FLT_PARAMETERS is described at http://msdn.microsoft.com/en-us/library/windows/hardware/ff544684(v=vs.85).aspx

- 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 and Christian Seifert's source code of CaptureFileMonitor[21].

If you are interested in module blocking as described in Microsoft's papers[22] 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[23].

Access to files can be monitored by setting a callback for IRP_MJ_CREATE. Use FltGetFileNameInformation() with parameter Data and FltParseFileNameInforma-

[21] Go to https://projects.honeynet.org or just google for CaptureFileMonitor.c

[22] See M. Corporation. Kernel data and filtering support for Vista SP1 / windows Server 2008. MSDN or CodeShield: Towards Personalized Application Whitelisting (http://www.cs.purdue.edu/homes/gates2/publications/acsac2012-codeshield.pdf)

[23] More details are available at http://www.winvistatips.com/modifying-targetfileobject-acquire_for_section_synch-t196674.html. To build up a powerful module filtering driver I highly recommend to check on IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION.

tion to receive the originated filename. Just have a look at Microsoft's minifilter 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 Example drivers

In the following sections I shortly discuss some basic drivers I have written in the past. I will not go into all the details, if your are interested in more, feel free and contact me. If you want to develop your own monitoring drivers, a good point to start is to study Microsoft's DDK example drivers. There are a lot of minifilter examples you can dig into. Some of the drivers, e.g. the Scanner-Demo, already implement some basic analyzing minifilter that can be expanded by you.

> *BEWARE!*
>
> *All drivers are for educational and test purposes only. They are just demonstration drivers in a pre-alpha or beta stage and might contain bugs that lead to system crashes and other damages to your system. The drivers are not digitally signed so I do not recommend to install the drivers in any production system. Use the drivers is at your own risk!*

Some of my drivers are marketed and distributed by Excubits UG (haftungsbeschränkt). If you are intereseted in aquiring a license, visit http://excubits.com.

The drivers are not digitally signed, hence you cannot run them on 64-bit version of Windows without a tweak (or work around). There you must boot into Testsigning Mode and digitally sign the drivers by yourself or you start up Windows without checking code signatures. To do the latter just press F8 while rebooting your system. On Windows 8 the classic F8-Bootmenu must be enabled by executing the following commands in an console:

```
bcdedit /set bootmenupolicy legacy
bcdedit /timeout 12
```

4.1 Monitoring IRP_MJ_CREATE

While having some time off duty winter 2011/2012 I started analyzing Microsoft's "heavy duty" passThrough DDK-example driver and tried to build a simple minifilter driver that just checks for IRP_MJ_CREATE in its preoperation callback. The driver determinates the originating/correspondig 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.

4.2 A minifilter that monitors executables written on your disk

Thinking about different approaches to monitor malware while they are installing their evil code on your machine I ended up in a monitoring minifilter driver that might help you out analyzing potential zero-days and other malicious stuff on your forensics machine.

MZWriteScanner is a simple minifilter that intercepts IRP_MJ_CREATE, IRP_MJ_CLEANUP and IRP_MJ_WRITE (and some other) to track what files should (and will) be written on your disk. The driver checks if a file contains the magic bytes for an executable, namely the string 'MZ'/'ZM' at offset (0) of the file. If this is the case MZWriteScanner outputs the filename via DbgPrint and writes the path and filename to %SystemRoot%\mzwritescanner.log. Well, the approach seems to be a bit cheesy on the first view, but should work for many malware executables that hit your face through drive-by exploit kits.

Since version 2.1 the driver needs a configuration file at %SystemRoot%\mzwritescanner.ini where you are able to enable the so called lethal mode which enables you to deny execution of newly written executable files and to whitelist paths or files that are allowed to contain newly written executables without blocking them. The latter might be helpful in automated scenarios where you might want to allow updating system executables (e.g. the executables of patches, updates etc.) or anti-virus tools. But be careful with what you gonna whitelist, because whitelisting the wrong path or file might open the doors for potential malware -- keep that in mind!

The whitelist in %SystemRoot%\mzwritescanner.ini *must* be in UNICODE file format and must contain at least one line enabling or disabling the blocking mode:

```
[LETHAL]
```

to enable it, or

```
[#LETHAL]
```

to disable it.

If you would like to whitelist some paths or files, just add their path and/or name in a line followed by an asterisks before the new line. See the following example:

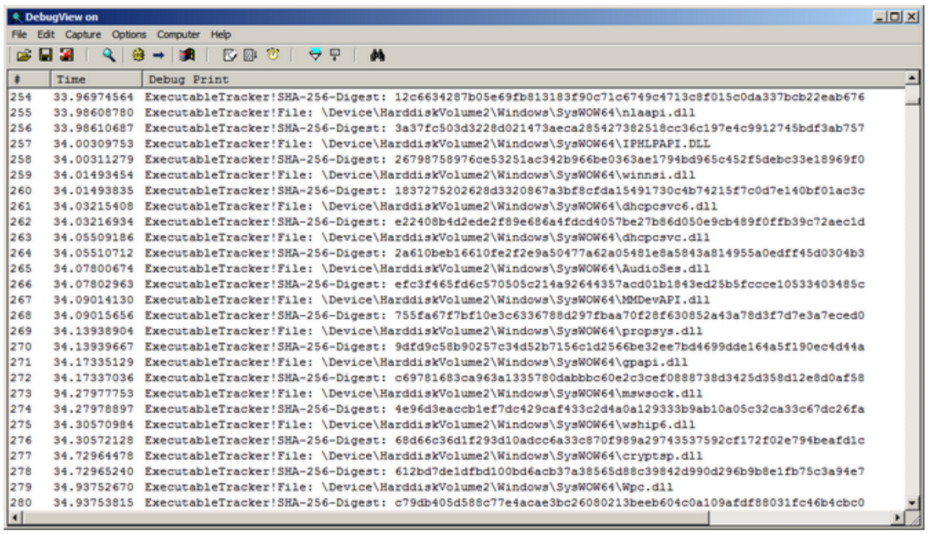
```
[LETHAL]
whitelist*
\Device\HarddiskVolume2\ProgramData\Microsoft\Windows Defender\Definition Updates\*
\Device\HarddiskVolume2\Windows\SoftwareDistribution\Download\*
\Device\HarddiskVolume2\Users\USER\Desktop\Dbgview.exe*
\Device\HarddiskVolume2\Windows\System32\Drivers\Dbgv.sys*
```

To install the driver just go into the binaries path regarding your version of Windows (Windows x86/x64 Vista, 7 and 8). Then right-select the .inf and hit "install". Then run one of the cmd-scripts to start, stop, restart and uninstall the driver. Do not forget to fire up DbgView to peek the messages the driver prints out. Make sure to disable driver signing on 64-bit versions of Windows, the driver was not signed yet.

4.3 Tracking executables on Windows

On Windows there is no way to directly monitor and track what executables (exe, dll, sys, msi, ...) are gonna be on the run, meaning: when is an executable image loaded for execution into memory. If you are a malware analyst or just interested in what executables are loaded into memory while your Windows machine was turned on, my new kernel mode driver ExecutableTracker might give you a deeper look inside.

ExecutableTracker[24] is just a simple minifilter driver that logs any executables mapped into memory for execution. You can keep track of the logged executables by using Dbgview[25] (enable 'Capture Kernel', 'Enable Verbose Kernel Output' and 'Pass-Through'). ExecutableTracker determinates the corresponding file and calculates a SHA-256 digest of the file, thus you can use the digest to fire it against a malware database for example. A typical log of Dbgview looks like:



```
#     Time          Debug Print
254   33.96974564   ExecutableTracker!SHA-256-Digest: 12c6634287b05e69fb813183f90c71c6749c4713c8f015c0da337bcb22eab676
255   33.98608780   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\nlaapi.dll
256   33.98610687   ExecutableTracker!SHA-256-Digest: 3a37fc503d3228d021473aeca285427382518cc36c197e4c9912745bdf3ab757
257   34.00309753   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\IPHLPAPI.DLL
258   34.00311279   ExecutableTracker!SHA-256-Digest: 26798758976ce53251ac342b966be0363ae1794bd965c452f5debc33e18969f0
259   34.01493454   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\winnsi.dll
260   34.01493835   ExecutableTracker!SHA-256-Digest: 1837275202628d3320867a3bf8cfda15491730c4b74215f7c0d7e140bf01ac3c
261   34.03215408   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\dhcpcsvc6.dll
262   34.03216934   ExecutableTracker!SHA-256-Digest: e22408b4d2ede2f89e686a4fdcd4057be27b86d050e9cb489f0ffb39c72aec1d
263   34.05509186   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\dhcpcsvc.dll
264   34.05510712   ExecutableTracker!SHA-256-Digest: 2a610beb16610fe2f2e9a50477a62a05481e8a5843a814955a0edff45d0304b3
265   34.07800674   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\AudioSes.dll
266   34.07802963   ExecutableTracker!SHA-256-Digest: efc3f465fd6c570505c214a92644357acd01b1843ed25b5fccce10533403485c
267   34.09014130   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\MMDevAPI.dll
268   34.09015656   ExecutableTracker!SHA-256-Digest: 755fa67f7bf10e3c6336788d297fbaa70f28f630852a43a78d3f7d7e3a7eced0
269   34.13938904   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\propsys.dll
270   34.13939667   ExecutableTracker!SHA-256-Digest: 9dfd9c58b90257c34d52b7156c1d2566be32ee7bd4699dde164a5f190ec4d44a
271   34.17335129   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\gpapi.dll
272   34.17337036   ExecutableTracker!SHA-256-Digest: c69781683ca963a1335780dabbbc60e2c3cef0888738d3425d358d12e8d0af58
273   34.27977753   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\mswsock.dll
274   34.27978897   ExecutableTracker!SHA-256-Digest: 4e96d3eaccb1ef7dc429caf433c2d4a0a129333b9ab10a05c32ca33c67dc26fa
275   34.30570984   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\wship6.dll
276   34.30572128   ExecutableTracker!SHA-256-Digest: 68d66c36d1f293d10adcc6a33c870f989a29743537592cf172f02e794beafd1c
277   34.72964478   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\cryptsp.dll
278   34.72965240   ExecutableTracker!SHA-256-Digest: 612bd7de1dfbd100bd6acb37a38565d88c39842d990d296b9b8e1fb75c3a94e7
279   34.93752670   ExecutableTracker!File: \Device\HarddiskVolume2\Windows\SysWOW64\Wpc.dll
280   34.93753815   ExecutableTracker!SHA-256-Digest: c79db405d588c77e4acae3bc26080213beeb604c0a109afdf88031fc46b4cbc0
```

[24] ExecutableTracker is some sort of cleanroom-design implementation of an executable tracker that bases on information gained from the CodeShield architecture paper (http://www.cs.purdue.edu/homes/gates2/publications/acsac2012-codeshield.pdf).

[25] Download Dbgview at http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx

ExecutableTracker does not block any executable image nor is it some kind of protection driver, so beware of what you gonna execute on your machine. The driver just monitors what executables are loaded and will be executed on your machine. To detect (and block) malware there is more to do, ExecutableTracker is just a stripped down version of a driver that ships with the ExploitBuster framework and Türsteher, that we currently use to detect malware on our forensic machines. We will not publish such drivers for free right now and think you have sympathy for this decision.

4.4 Tuersteher Light: A Path Based Application Whitelisting Filter Driver

AppLocker's capatilities to whitelist and block executables, libraries and scripts with the comfort of group policies are great but it is pain if you need to use AppLocker as a helping hand to monitor, track and block potential malicious code in forensic scenarios. On the other hand AppLocker is only available in Enterprise versions of Microsoft Windows, thus not within reach to the majority of Windows users.

Having developed several minifilter drivers I was able to build up a light and easy to use filter driver acting like AppLocker helping you to monitor and block executables (exe, dll, sys, ocx...) that were not started from a trusted path. Components of this driver are part of our malware detection framework ExploitBuster and Tuersteher, but Tuersteher Light does not contain all the sticky icky features I have build into our heavy weight versions.

As known from AppLocker, in Tuersteher Light you also simply specify a whitelist of trusted paths and fire up the driver. The driver then checks the corresponding path and filename against a list before allowing it to be read into memory for execution. Thus the driver is able to block malicious code started from external USB drives, e-mail attachments, your Internet browser's cache and many more. It is no silver bullet against all attacks with regards to 0-days leading to privilege escalation, in-memory transient malware that only resists in the exploit’s allocated memory portion but most exploits we were investigating in 2012/2013 initially stored their malicious executable modules somewhere into the user’s folder space or Windows’s system paths and hence could effectively being blocked by the driver’s approach using a carefully defined white- and blacklist of paths (or files). There are limits, but as far as I know there does not exist any endpoint security solution out there, that faces all possible attacks[26].

In a typical forensics scenario where you run a test machine against potential toxic web contents I heavily encourage you to only whitelist the folders \Windows\ and

[26] For high security mitigation I suggest to combine Tuersteher Light with MZWriteScanner, an AntiVirus-Solution and EMET (http://www.microsoft.com/en-us/download/details.aspx?id=41138). Although this will not be the silver bullet it is really close to it, especially if you are subject to special crafted targeted attacks that focus on draining your intellectual property and know how.

\Program Files\. Then fire up DbgView, open a toxic web site (e. g. running an exploit kit) and watch out what my driver blocks and logs.

4.4.1 Configurate and start up the driver

First at all define the list of paths or files you want to white- or blacklist, save this list into an unicode file named tuersteher_light.ini and copy it to your Windows folder (in most cases it is c:\Windows\). The file must be an unicode text file to prevent deadlocking your machine. Tuersteher Light supports unicode making it possible to use the driver all over the world, from the USA to Europe, Russia, Africa and also Asia and the Middle East. Thus you are able to define filenames and paths like:

☻♫♪♥♠слу́шаю!אָדער ... אָדערNǐhǎo!مَرْحَبًا

The list of names is case sensitive, beware of that.

You are not allowed to specify paths and files directly by their DOS filename, e.g. c:\Windows\ etc. Instead you must use the path's device and volume descriptor. To make things a bit more clear I included an example tuersteher_light.ini file into the driver's package, so check out this file for more details and on how to specify the names there. I also included a tool that prints out your mount points in the correct format (see MountPointFunctions.exe).

As a first step I recommend to specify all paths or filenames by their path and filename you want to whitelist into the file tuersteher_light.ini. You start the whitelist with the section identifier **whitelist***. After each line of a path or filename in the section of the whitelist you must set an asterisk (*). If you forget the asterisk you might crash or deadlock your system after starting up the driver.

Files for the blacklist are defined in the section **blacklist|**. Files or paths you want to blacklist must be marked with a vertical dash (|). After each line of a blacklisted file or path you must set the dash. If you forget the dash you might crash or deadlock your system after starting up the driver.

If you enable to block the execution of binary modules by defining the **[LETHAL]** tag, Tuersteher Light blocks such modules like in AppLocker. You can also enable a logfile by defining **[FORENSICS_PATH]**, telling Tuersteher Light to log the filenames and/or paths into the file

```
tuersteher_light.log
```

located in your Windows installation path (in most cases c:\Windows). Make sure that you copy an empty Unicode Textfile named tuersteher_light.log into your Windows installation path (in most cases c:\Windows).

Example of a tuersteher_light.ini file:

```
[LETHAL]
[FORENSICS_PATH]
whitelist*
\Device\HarddiskVolume2\Windows\*
\Device\HarddiskVolume2\Program Files\Common Files\*
\Device\HarddiskVolume2\Users\Magnum\AppData\Local\Google\Chrome\User Data\PepperFlash\*
\Device\HarddiskVolume2\Users\Magnum\AppData\Local\Google\Chrome\User Data\SwiftShader\*
\Device\HarddiskVolume2\Program Files\MSBuild\*
\Device\HarddiskVolume2\Program Files\Reference Assemblies\*
\Device\HarddiskVolume2\Program Files\Windows Defender\*
\Device\HarddiskVolume2\Program Files\Windows Journal\*
\Device\HarddiskVolume2\Program Files\Windows Media Player\*
\Device\HarddiskVolume2\Program Files\Windows NT\*
\Device\HarddiskVolume2\Program Files\Windows Photo Viewer\*
\Device\HarddiskVolume2\Program Files\Windows Portable Devices\*
\Device\HarddiskVolume2\Program Files\Internet Explorer\*
\Device\HarddiskVolume2\Program Files (x86)\Internet Explorer\*
\Device\HarddiskVolume2\ProgramData\Microsoft\Windows Defender\*
\Device\HarddiskVolume2\Program Files (x86)\Google\*
\Device\HarddiskVolume2\Program Files (x86)\Microsoft Silverlight\*
\Device\HarddiskVolume2\Program Files\7-Zip\*
\Device\HarddiskVolume2\Program Files (x86)\Common Files\*
\Device\HarddiskVolume2\ProgramData\Microsoft\IdentityCRL\*
\Device\HarddiskVolume2\Users\Magnum\Desktop\Dbgview.exe*
\Device\HarddiskVolume2\Users\Magnum\Desktop\SoftMaker Office 2010\*
\Device\HarddiskVolume2\Users\Magnum\Desktop\LibreOfficePortable\*
\Device\HarddiskVolume2\Users\Magnum\Desktop\☻♫♪♥♠слуш́аю!דער אַ … דער אַNĭhǎo!مَرْحَبًا\*
\Device\HarddiskVolume3\downloads\TrueCrypt\*
blacklist|
\Device\HarddiskVolume2\Windows\notepad.exe|
\Device\HarddiskVolume2\Program Files\Internet Explorer\iexplore.exe|
\Device\HarddiskVolume2\Program Files (x86)\Internet Explorer\iexplore.exe|
```

Make sure that you divide the file into the two parts whitelist* and blacklist|. It is important not to mix filenames and paths from one section into another section. Meaning, **please do not** something like:

```
whitelist*
\Device\HarddiskVolume2\Windows\*
\Device\HarddiskVolume2\Windows\notepad.exe|
blacklist|
\Device\HarddiskVolume2\Program Files\Internet Explorer\iexplore.exe|
\Device\HarddiskVolume2\Program Files\Common Files\*
\Device\HarddiskVolume2\Users\Magnum\AppData\Local\Google\Chrome\User Data\PepperFlash\*
```

The (bad) example above will not work properly and might end up in a halting or crashing system. So take care while defining the tuersteher_light.ini file!

The driver was compiled for Microsoft Windows Vista, 7, 8 and 8.1 (32/x86 and 64/x64 bit versions). To start it up go into the driver binary's path regarding your version of Windows and execute the corresponding *.inf file in order to install the driver.

If you use a 32bit Version of Windows, driver signing is not required and you should be able to run Tuersteher Light just out of the box. In Windows 7, 8 and 8.1 x64 you need to digitally sign any driver. This is Microsoft policy for all kernel drivers in recent versions of Windows, for more details see Driver Signing Requirements for Windows.

As a temporary work around you can also disable the signature check in Window's boot options. An alternative way is to digitally sign the driver by yourself using a test certificate and booting up Windows into the TESTSIGNING mode:

Download, install the Windows Driver Kit, then open a WDK Build Environment console as Administrator.

Run the MakeCert.exe tool to create a test certificate, e.g. with:

```
MakeCert -r -pe -ss TestCertStoreName -n "CN=TestCertName" CertFileName.cer
Install the test certificate with CertMgr.exe, e.g. with
CertMgr /add CertFileName.cer /s /r localMachine root
Sign Tuersteher.sys with the test certificate, e.g. with
SignTool sign /v /s TestCertStoreName /n TestCertName Tuersteher.sys
Enable Windows TESTSIGNING mode, to do this, run the command
Bcdedit.exe -set TESTSIGNING ON
```

After these steps you should be able to run the driver without disabling driver signature check every time.

Since Q4/2014 Türsteher is commercial product, available at Excubits UG (haftungsbeschränkt). The English version is also known as Excubits Bouncer.

4.4.2 Tuersteher Light For Windows XP

I got a lot of feedback on Tuersteher in the last few months, most regarding a special edition of Tuersteher Light for Windows XP as some kind of endpoint protection system. Since Microsoft discontinues support for Windows XP, this operating system will no longer receive any (security) updates by Microsoft. Operating Windows XP is risky, because security issues are not fixed, users must expect attacks on their beloved XP machines. Something like Tuersteher for Windows XP sounds great to mitigate against the risk. I think that's why I received so many comments and questions on a special edition for XP. Tuersteher Light was initially not available for Windows XP, but I am now proud to say: Tuersteher Light runs under Windows XP.

Tuersteher prevents your system from executing untrusted executables in a so-called whitelisting approach. Just specify what files/paths are okay and what files/paths are not, and the magical pixie dust behind Tuersteher manages to block any attempt to pass by. Whitelisting is a well known technique and is already built into the business versions of Microsoft Windows. Bad news for the ordinary user, because such techniques are only available in the supreme ultimate editions of Windows (e.a. AppLocker), most users do not buy and use even worse: Windows XP never ever supported whitelisting out of the box.

I wrote lots of kernel drivers back in the days to detect, analyze, protect and mitigate against the ordinary malware out there. Once upon time I have decided to program a simple to use driver for the ordinary Windows user. One of many solutions was Tuersteher Light. It is a AppLocker like, path-based minifilter driver that enables you to specify from which path Windows is allowed to start executables. You are also able to define bad paths, you do not want any executable started from and thus blacklist them. From a blacklisted path even Windows itself is not able to start up an executable file from.

For example, you can whitelist "c:\WINDOWS" and "c:\Program files" and can blacklist paths like your browser's cache "..\Temporary Internet Files\" etc. By simply disallowing everything except "c:\WINDOWS\" and your trusted program file paths you will mitigate against many attacks out there without a too complicated configuration.

Tuersteher Light is a helping hand, mitigating against ordinary attack vectors and supporting you to operate Windows XP a bit more secure these days. But keep in mind: Tuersteher Light will not protect you from all possible attacks, especially transient in-memory attacks will not be caught - but these attacks are fairly not detectable by the vast majority of protection tools out there, and most of them are very expensive. If you need additional information on what Tuersteher Light is able to mitigate against and what not, contact me.

In general: do not get reckless when running Tuersteher Light on your XP box! You are still using an old and outdated operating system, so no time to get cocky. As I said, use it as a helping hand and get yourself updated soon.

Since Q4/2014 Türsteher is commercial product, available at Excubits UG (haftungsbeschränkt). The English version is also known as Excubits Bouncer.

4.5 Building a totally locked down Windows for POS-, ATM- and kiosk-mode-Envrioments

During my last winter holidays in 2013/2014 I was working on Türsteher and brought its drivers to full Windows 8.1 support (incl. x64-support). I also tweaked and optimized the driver and did a lot of stress testing on a Windows 8.1 box. While performing boring driver testing the following idea came through my mind:

> *"On a kernel-level: Is it achievable to limit the access to executable code to an absolute minimum, so that there is only a limited set of executables loaded and allowed to run?"*

Meaning: You are able to boot up Windows and run, for example, only notepad and a browser - nothing else. And "nothing else" means absolutely nothing -- not even a driver, or any additional module after you've started up the machine.

If this is possible, one is able to protect a kiosk-mode Windows envrioment besides guest accounts, special lock down GPOs or other third party applications. Hence such an approach provides additional security and mitigation against attacks on such envrioments. Currently most of these kiosk-mode envrioments are protected by simple mitigations that just check for running only allowed executables, or limit access to the Explorer and its "execute as..."-Dialog, hide the TaskManager, Startmenue or Taskbar etc. Indeed, blocking all executables except the white listed ones is some sort of protection at application/desktop level, but this is not an in depth approach. Most available kiosk-mode envrioments do not block all the background binaries (DLLs, drivers, etc...) that e. g. get fired up if an user plugs in an USB-device, manages to init additional DLLs for plug-ins etc or even is able to detect that some sort of exploit is able to execute a library.

Well, I began to adjust Türsteher's kernel mode driver a little bit, so that it logs and measures out a minimum set of executables needed to successfully boot up and execute my pre-configured Windows 8.1. That proof of concept kiosk-mode Windows should only allow to run Notepad, the Calculator and Google's Chrome. And by saying "only allow" I mean, that only the vivit system binaries can be executed and only application modules that are needed to run the named applications.

At the end I had a list of about 800 executable files (drivers, exe, ocx, dll, ...) that must at least be available to Windows to boot up, log on and run the intended applications noted above. I managed Türsteher to real-time-check this set of executable binaries and to ensure, that those files are loaded correctly into system memory.

Impossible?! Noop! My proof of concept is able to boot up Windows with a minimum set of executables and take care that only Notepad, Calculator and Chorme are running on the system -- and even more: that all executable code loaded in is trustworthy. Even if ~800 files sounds like a huge number it is still little in contrast to all the executables that are stored on a typical installation of Windows (also if your subject matter is a lovely vanilla installation of Windows).

Well, and what is it good for? Security! As I said some lines above: if you set up a locked down Windows in an kiosk-mode envrioment it is still possible for attackers to start up libraries or executables through a bug in the permitted applications.

How many of the sheer mass of executables of your actual running Windows box are really needed by a POS, ATM or kiosk-mode envrioment, where you already limit access to executables/applications through GPOs and others?

You will come to the conclusion: Not so much. So why should you allow to execute such files even by accident (or through an exploit that utilizes them to start). It is better to lock out everything that is not needed and this is what I did with Türsteher: Limit things to the absolute minimum.

Do not get me wrong: GPOs and lock-down applications for POS, ATMs and kiosk-mode envrioments are great mitigation techniques to protect them, but together with a whitelisting approach like Türsteher that limits the attack vector to an absolute minimum of really needed executable code, it is possible to protect your POS, ATM or other kiosk-mod envrioment even better.

I tested Türsteher on several POS and kiosk-envrioments, but my resources are limited. So, if you are interested in protecting or locking-down your Windows-based POS-, ATM- or kiosk-mode-envrioment feel free and contact me to set up Türsteher for your needs (I have more than 20 years of experience in the field of Microsoft Windows, and about 8 years of experience in Microsoft Windows Kernel Development, I know how to set things up and give you a helping hand). The basic driver runs under Windows 32bit/64bit-versions of 7, 8 and 8.1. The driver is also able to run under Microsoft Windows Server Editions as well. If you are interested, do not hesitate and fire an e-mail to me.

5 Catch (targeted) malware and other cyber attacks

More and more sophisticated and targeted zero-day attacks rise in our internet and computer driven world. Traditional security defenses, such as anti-virus, IDS or next-generation firewalls are not able to keep up with the amount of new attacks flooding computer systems and networks day after day. The impact to organizations is significant: Denial of Service (DoS), help desk calls, network downtime, information and intellectual property loss, etc. That all summarizes up in lost productivity and lost money.

Traditional tools like firewalls, anti-virus tools, behavior-analysis were designed only for already known patterns of malicious code and attacks. But today we see personalized attacks against cooperate IT-infrastructures, their users and the intelligence property. These attacks survive most classical detection systems like firewalls, IDS, AVs and filtering internet gateways. Such solutions offer only little protection against such attackers, because new attacks (zero-days) have a really good chance getting not detected and nailed by traditional solutions.

In the past years we saw lots of (targeted) attacks making use of zero-days in PDF, well known office suits, scripting hosts and widely used web browsers or their plug-ins. All these attacks trigger some kind of executable to place their malicious code on an user's machine. Except the zero-days in most cases there is nothing new and special there. Most targets getting hit because they lack an anti-virus solution or their anti-viruses do not know the second or third stage malware served by the exploit. Again, nothing new here - we all know that targeted attacks are prone being not detected by the ordinary anti-virus.

A well known approach to chase executable code, especially second stage malware that should be installed on your machine after an exploit successfully hit your system is to use some kind of trust- and real-time-based proactive application control. Meaning that code[27] will only be executed if such code was identified as trustworthy in first instance. Unknown or already known untrusted code should be blocked[28].

This is known as whitelisting as defined in Wikipedia[29]:

> *[...] A whitelist is a list or register of entities that, for one reason or another, are being provided a particular privilege, service, mobility, access or recognition. Only entities on the list will be accepted, approved, and/or recognized. Whitelisting is the converse of blacklisting, the practice of identifying entities that are denied, unrecognised, or ostracised, and the term "white-list[ing]" was formed by back-formation from "blacklist[ing]". [...]*

The NSA notes[30]

> *[...] Application Whitelisting is a proactive security technique where only a limited set of approved programs are allowed to run, while all other programs (including most malware) are blocked from running by default. [...]*

And there come the minifilter drivers in place: Setting up the right minifilter driver enables you to catch many exploits and their second or third stage malware that will be installed on an ordinary Windows based computer system. You can combine filtering

- IRP_MJ_CREATE
- IRP_MJ_CREATE_NAMED_PIPE
- IRP_MJ_READ
- IRP_MJ_WRITE

[27] For example an executable, library, driver, script etc.

[28] There are plenty products out there that help you to protect against cyber attacks. For example check out Bit9, Lumension or FireEye.

[29] http://en.wikipedia.org/wiki/Whitelist

[30] http://www.nsa.gov/ia/_files/factsheets/Application_Whitelisting_Trifold.pdf

to track what files will be written or accessed. One solution might be checking (executable) content against a whitelist as defined by Wikipedia or NSA.

As part of Exploitbuster[31] a friend of mine and me started to write and test many proactive application control and content drivers (e.g. a more advanced version of MZWriteScanner). By using minifilter drivers we were (and are) able to compile the drivers in no time for Windows XP (32 bit), Vista, 7 and 8 (32bit/64bit) and Windows Server without changing any line of code. The drivers run stable, fast with a small binary footprint and we are able to adjust them at any time for upcoming needs and requests.

I see minifilter drivers as powerful tools to detect many kinds of malware out there. In contrast to other heavy weight tools that are closed source, require central managing servers etc., once you have build up your own minifilter drivers they are as powerful as a Swiss Army Knife. For (us) forensics guys they are awesomely perfect because there must be no GUI, nor a console application to run it from etc. For example our application control driver Türsteher[32] is a stand alone driver that just needs a configuration file and that's it. It does not need a service and fires up immediately after kernel init. Logs are instantly written through DbgPrint() or into a simple log file making it working smooth in automated forensic scenarios. Everything can be done in the driver itself, there is no communication into real-mode, there is no GUI and other code making stuff complicated.

We successfully ran Türsteher against well known exploits and it is really nice to know that Türsteher was able to detect and block **all** recently reported cyber attacks reported since November 2012 until now (July 2013). To us this proves that combining minifilter drivers leads in advanced protection or analyzing systems that are able to defend or to detect state of the art malware and cyber attacks without any additional definition or heuristic data base[33]. This makes you independent and it is an advantage over the classical anti-virus solutions - especially if you are subject to targeted attacks where we all know anti-viruses frequently fail.

6 Drawbacks

There are not just pros, hence I will give you an overview on the drawbacks. The following does not mean that minifilters are useless or can be attacked by every script kid, it is still hard to bypass them but it is also good to know, that nothing is 100%

[31] For more details see http://www.exploitbuster.com

[32] Türsteher is the German term for doorman or bouncer.

[33] Remark: Minifilter drivers enhance security and as such one might use them in combination with a firewall and anti-virus. But well, we think that in controlled scenarios one might drop an anti-virus and just use a firewall and a high class minifilter driver and will not lose any security.

bullet proof and there are vectors that could be attacked. It does not mean that they will be attacked (on your use case) but still keep the following paragraphs in mind when developing your minifilter drivers in the context of security solutions.

Monitoring minifilter drivers might help; as the word *might* implies, there exist some drawbacks. If you're getting infected by some sticky-icky beast[34] that makes use of 0-days enlarging its privileges to ring-0 (kernel mode) and using rootkit technology to hide its objects you are **done** – it is just that simple. This is also 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. an exploit that does not use LoadLibrary, CreateFile or CreateProcess etc. to start up the second stage code, meaning malware that just resists in memory through its exploit without an executable image loaded. Hence it will be loaded and initialized by the exploit itself, not by any operating system function[35].

Reflective DLL injection[36] is a really nice library injection technique in which the library is responsible for loading itself by implementing a minimal Portable Executable (PE) file loader. The main advantage of self made library loading is that an attacker does not use any of the operating system featured functions to load executable code into memory (like LoadLibrary, Exec, CreateProcess, etc.), hence as such, the "loaded" code is not registered in any way with the host system and as a result is largely undetectable at both a system and process level. A problem is, that an executable injected with this method is almost transient and will be gone after a reboot if the attacker cannot ensure, that his exploit will take place after a reboot to re-inject the library again.

Another drawback comes in place if you are using so called scripting-hosts on your machine: E.g. Java, Python, Ruby or Perl scripts might also be malicious but, but it is likely that you will not see the script itself as an originator; in most cases you will log just see the scripting host's executable[37] which does not seem to be suspicious in first instance. To protect against such attacks you must expand filtering on an accesed file's content and block such file, if it contains unknown or untrusted lines of code. This is not an easy task to do. Thus if possible I recommend not to use any scripting host on targeted machines.

[34] I am talking about a targeted attack, remember Stuxnet for example.

[35] Such malware is very rare and hard to implement. Currently we saw such exploits as proof of concepts - never heard about a widely spread exploit.

[36] As introduced by Stephen Fewer in „Reflective DLL Injection", see http://www.harmonysecurity.com/ReflectiveDllInjection.html

[37] E.g. the just in time compiler or interpreter's executable.

I think that transient malware in combination with scripting hosts can be a show stopper for minifilter drivers if you do not add additional logic into the filter mechanism itself - meaning: a simple and “stupid” minifilter driver is not enough. Especially care should be taken with regard to always on systems like Windows 8, because new operating systems are build to run for weeks and months without a “cold” start. This makes them perfectly suitable for transient malware, that could resist for a long time, logging the keyboard, screen, scanning for interesting files etc. As shown by Samrat Ashok's kautilya[38] framework it is really simple to program PowerShell based hacking tools that run behind the scenes without executing suspicious executables or even injecting a library into another process. Well, we will see what's coming up next. I expect some fancy and tricky transient malware approaches for the upcoming versions of Windows in the next years. Thus new filter mechanisms are heavily needed to track such upcoming malware.

7 Conclusion

Using a kernel minifilter driver makes it easy to **screen and protect** your Windows based operating system on a system wide basis. Such drivers might also 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 its dropper in first instance[39]. Minifilter drivers will not avoid further analysis, but they give you fast and initial results on what is going on. This helps to protect a critical environments and to deploy mitigations before someone hit the whole infrastructure.

I see minifilter drivers as powerful tools against many kinds of malware out there. In contrast to other heavy weight tools requiring central managing servers etc., once you have build up your own minifilter drivers they are powerful and flexible like a Swiss Army Knife. So I highly encourage everyone in the Windows based security scene to have a deep look into the heavy-duty stuff one can achieve with minifilter drivers.

Although minifilter drivers are cool with regards to transient malware I highly recommend

- never lean back,
- keep up to date,
- reanalyze your minifilter approach once a year (at minimum) and

38 For more details see http://code.google.com/p/kautilya

39 It is also possible to monitor the load of modules as described in "Kernel Data and Filtering Support".

- always check on what the bad guys currently do.

Are you subject to targeted and special crafted (cyber) attacks?

Your Windows based information technology is attacked to drain your intellectual property?

An Anti-Virus and other protection systems are too generic forcing against these attacks?

Well, you have visited the right place.

During the last years we carefully developed technology against attacks on Windows based operating systems. Benefit from our experience, we are able to

- build up fully customized Microsoft Windows Kernel drivers helping you to identify, protect & mitigate against threats.
- certify and audit our sources if needed. The main parts of our solutions are less than 1800 lines of code, so it is easy to peek through.

If you are interested in our solutions just contact us via e-mail:

info@excubits.com

We can build up fully customized demos with respect to your needs.

Do not hesitate. We look forward hearing from you.