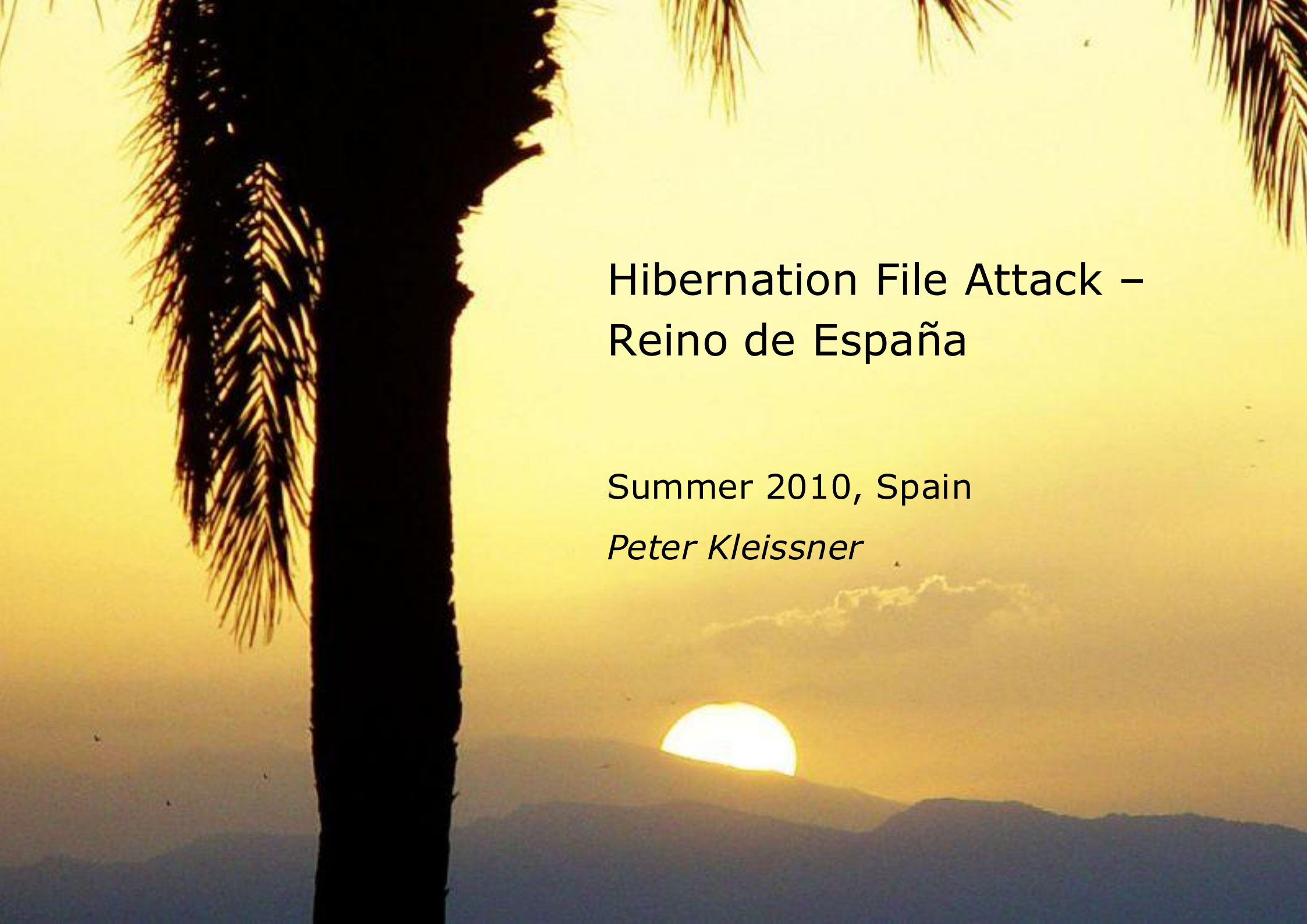


Hibernation File Attack – Reino de España

Summer 2010, Spain

Peter Kleissner

Table of Contents

Hibernation File Attack – Reino de España

In 2008 I thought about attacking the hibernation file, inspired by the page file attack. The result was that I took my code from my multi-boot solution (which had already file system drivers) and modified it to "attack" the hibernation file. Based on research of Matthieu Suiche and own research, I was able to develop the Xpress algorithm in assembler – reading and modifying the compressed images of the hibernation file attack in my own bootkit.

At that time I wrote a presentation and some smaller documents and submitted to Black Hat Europe 2009, though it was declined. Later I based on my hibernation file attack the Stoned Bootkit, which patches everything in memory and stays in memory until the Windows kernel is loaded.

It was my pleasure taking the old unfinished declined work from 2008, doing additional research and then finally writing this new paper and presentation, and holding a two-day security training in beautiful Madrid (Spain).

My schedule was very tough (only few weeks for making the entire project), though I could finish a basic proof of concept, show some very interesting things and I had very fun with the whole security training. The coolest thing is booting any machine into its previous hibernation state – you only need to restore the header. Besides that, you can use the hibernation file to get encryption keys, passwords or any other data that was loaded at runtime.

In future I am going to release the proof of concept code to this presentation – currently I am in contact with Microsoft on this issue. This paper was written and published within the United States.

Update 16.09.2010: The paper was resized to A4 and the first topic removed.

Peter Kleissner

ACPI Power States

The Advanced Configuration and Power Interface specification defines global power states [1]:

G0/S0	Working	Running system
G1	Sleeping	Simple throttling, Standby or Hibernation
G2/S5	Soft Off	Shutdown
G3	Mechanical Off	Turning power off

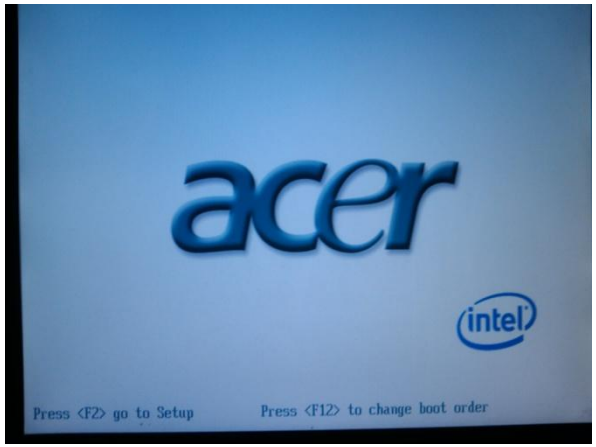
Within G1 there run the sleeping states S1 to S4:

S1	Low wake latency sleeping state
S2	System and CPU cache is lost, control restarts at processors reset vector
S3	Standby, control restarts at processors reset vector (suspend-to-memory)
S4	Hibernation, powering off all devices (suspend-to-disk)
S5	Like S4 except that OS should not save context; used to tell the OS if it has to reload anything. Same as G2.

When Windows hibernates the system, then the entire system (all devices) turns into the global power state G1, specifically into the sleep-state S4. The entire context (all running programs, drivers) is stored into the hibernation file C:\hiberfil.sys. To use the Windows hibernation feature, an ACPI conformant driver for the chipset must be installed in the system.

For waking up from S3 (Standby) or Hibernation (S4) there are various methods, like power on button, mouse/keyboard, modem (phone call), real-time clock (alarm time) and others. The computer stores finally (probably) the system state in the CMOS RAM. If you hibernate (G1, S4) your system and turn it on again, it is still in S4.

This means that when starting your system it immediately wants to resume from hibernation and the BIOS does not let you boot from any other device. Here are 2 screenshots made with the camera, one made when resuming from hibernation (right picture), one when normally booting (left picture):



If you want to boot from another device, like when you want to inspect the hibernation file when the system is hibernated, you have to "escape" S4. In order to do archive that, you have to switch from G1 to G2 (soft off) or G3 (mechanical off). There are two ways to enter either G2 or G3:

1. Pressing the power button for more than 3 seconds -> G3 (mechanical off)
2. Restarting the system while the BIOS is active, through pressing Ctrl + Alt + Del -> G2 (Soft off)

Standby, Sleep and Hybrid Sleep States

Standby is available in XP and is power state S3. In Vista/7 it is Sleep, which also goes into standby, but before battery runs out, it goes into hibernation state [2]. Windows Vista/7 introduces a new power state called Hybrid Sleep state, where contents is written to disk, and kept in memory. If the battery goes out, nothing is lost:

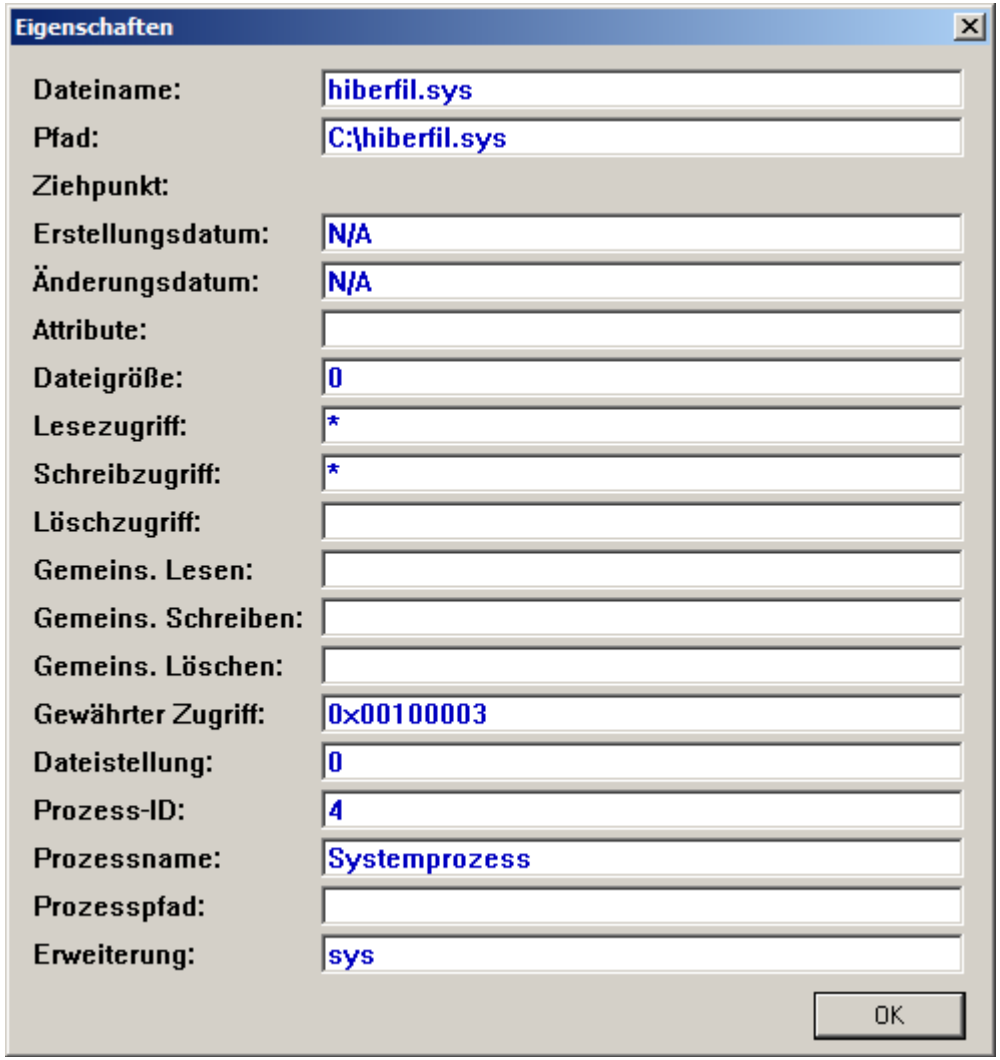
Hybrid sleep saves the OS state into RAM, but it also writes it all to the hard drive as well (sort of like hibernate does). This ensures that even if power is lost, the data will remain. This all sounds like a good idea, but in practice it's just as slow as standby was. [3]

Hibernation File

The hibernation file is C:\hiberfil.sys and has a special format. Its size must be at least 50% of the memory, according to the program powercfg.exe (German):

```
C:\Users\Peter Kleissner>powercfg.exe /?

POWERCFG <Befehlszeilenoptionen>
Beschreibung:
  Dieses Befehlszeilenprogramm ermöglicht Benutzern, die Energieeinstellungen
  in einem System zu steuern.

Parameterliste:

…

  -HIBERNATE, -H
              Aktiviert bzw. deaktiviert das Feature "Ruhezustand". Zeitüber-
              schreitung im Ruhezustand wird nicht von allen Systemen
              unterstützt.

              Syntax: POWERCFG -H <ON|OFF>
                      POWERCFG -H -Size <ProzentGröße>
              -Size  Gibt die gewünschte Größe der Ruhezustandsdatei in Prozent
                     des gesamten Speichers an. Die Standardgröße darf nicht
                     kleiner als 50 sein.
                     Mit dieser Option wird auch die Ruhezustandsdatei
                     automatisch aktiviert.
```

Hibernation can be turned on and off with this tool with parameter –h on|off and resized with the parameter –Size N. Access to the hibernation file is – by default – prohibited. The System process (PID 4) owns a handle with read and write rights (the screenshot was taken from the program OpenedFilesView):



The filename "hiberfil.sys" is missing an "e" to fit to old 8.3 file name convention in FAT. The hibernation file is never cleared; it always represents the memory state of last hibernation.

In Windows XP `ntldr` and `OS Loader` handle the hibernation resume, in Vista/7 it is handled by `winresume.exe`, which is executed by `winload.exe`. Windows provides few functions for hibernation support:

```
Kernel32.dll

BOOL WINAPI SetSystemPowerState(
  __in BOOL fSuspend,
  __in BOOL fForce
);

PowrProf.dll

BOOLEAN WINAPI SetSuspendState(
  __in  BOOLEAN Hibernate,
  __in  BOOLEAN ForceCritical,
  __in  BOOLEAN DisableWakeEvent
);

BOOLEAN WINAPI IsPwrHibernateAllowed(void);

BOOLEAN WINAPI GetPwrCapabilities(
  __out  PSYSTEM_POWER_CAPABILITIES lpSystemPowerCapabilities
);

NTSTATUS WINAPI CallNtPowerInformation(
  __in   POWER_INFORMATION_LEVEL InformationLevel,
  __in   PVOID lpInputBuffer,
  __in   ULONG nInputBufferSize,
  __out  PVOID lpOutputBuffer,
  __in   ULONG nOutputBufferSize
);
```

Encryption

The issue with hibernation and encryption is the different startup when Windows is booting normally. Encryption software installs a boot-start driver [4]:

A boot-start driver is a driver for a device that must be installed to start the Microsoft Windows operating system. Most boot-start drivers are included "in-the-box" with Windows, and Windows automatically installs these boot-start drivers during the text-mode setup phase of Windows installation.

Most, if not all, aspects of hibernation are undocumented, including:

- How the hibernation file is loaded (what calls etc),
- Whether the hibernation file is loaded in Protected Mode without paging and
- What components already have been loaded into memory and are excluded from restoration

Microsofts Bitlocker always supported hibernation.

TrueCrypt supports today (2010) hibernation with Vista and 7. Back in 2008 they wrote [5]:

** Disclaimer: As Microsoft does not provide any API for handling hibernation, non-Microsoft developers of disk encryption software are forced to modify undocumented components of Windows in order to allow users to encrypt hibernation files. Therefore, no disk encryption software (except for Microsoft's BitLocker) can currently guarantee that hibernation files will always be encrypted. At anytime, Microsoft can arbitrarily modify components of Windows (using the Auto Update feature of Windows) that are not publicly documented or accessible via a public API. Any such change, or the use of an untypical or custom storage device driver, may cause any non-Microsoft disk encryption software to fail to encrypt the hibernation file. Note: We plan to file a complaint with Microsoft (and if rejected, with the European Commission) about this issue, also due to the fact that Microsoft's disk encryption software, BitLocker, is not disadvantaged by this.*

[Update 2008-04-02: Although we have not filed any complaint with Microsoft yet, we were contacted (on March 27) by Scott Field, a lead Architect in the Windows Client Operating System Division at Microsoft, who stated that he would like to investigate our requirements and look at possible solutions. We responded on March 31 providing details of the issues and suggested solutions.]

[Update 2009-05-10: Since April 2008, we have been working with Microsoft to explore possible ways to solve this issue. We have private access to a draft version of a document specifying the future API, which should allow us to solve the issue on Windows Vista and later versions of Windows. Note: We have been asked not to disclose the content of the document to any third parties, so please do not ask us to send you a copy of the document.]

Pagefile

When a system is hibernated, the pagefile (C:\pagefile.sys) is still valid. Like on the hibernation file the System (PID 4) owns a kernel handle with read/write access to it. The size is always a multiple of 4 KB and is usually about the size of the user-mode used RAM.

Kernel mode memory (where privilege level 0 software runs) can also get swapped out (it can be specifically defined in the Memory Protection Constants when using Windows API). There is an option [6] that the pagefile and hibernation file get cleared on shut down:

Computer Configuration\Windows Settings\Security Settings\Local Policies\Security Options

This ensures that sensitive information from process memory that might have made it into the pagefile is not available to an unauthorized user who has managed to directly access the page file. When this policy is enabled, it causes the system pagefile to be cleared upon clean shutdown. Enabling this security option also causes the hibernation file (hiberfil.sys) to be zeroed out when hibernation is disabled on a laptop system.

There has been the pagefile attack by Joanna Rutkowska and Alexander Tereshkin, who injected code into the kernel by overwriting pages in the pagefile using raw sector access. This works in XP and Vista RC1. With Vista RC2 Microsoft prevents all write access to a mounted partition via raw sector access. The project is open source [7].

The pagefile attack works simply by opening a handle to either \\.\C: (partition) or to \\.\PhysicalDrive0 (entire drive) using CreateFile, and writing to it using WriteFile.

Hibernation File Format

0000h	Hibernation File Header
1000h	Processor State (in Windows XP reversed)
2000h	Reserved Memory Map (in Windows XP reversed)
6000h	Memory Table 1
	Xpress Image 1
	Xpress Image 2
	…
	Memory Table 2
	Xpress Image 1
	Xpress Image 2
	…
	Memory Table 3
	…

In Windows XP the Processor State and Reserved Memory Map are reversed. The physical memory is compressed stored as Xpress images. The Hibernation File Header defines general information and starts with a dword signature:

hibr	Windows XP	Active hibernation file, system shall process resume from hibernation
wake	Windows XP	Inactive hibernation file, system shall ask the user what to do
HIBR	Windows Vista	Active hibernation file, system shall process resume from hibernation
WAKE	Windows Vista	Inactive hibernation file, system shall ask the user what to do
RSTR	Windows Vista	During restoration Also seen in NTFS restoration area, in the log file restart page header [8]
0000h		Successful restoration (entire first page is cleared)
link		ARC link to other hibernation file

The hibernation file is organized in 4 KB pages. The processor state page contains the current processors state (equal to `CONTEXT` and `KSPECIAL_REGISTERS` structures), and the Reserved Memory Map a table of all reserved and free pages in the physical memory. The first Memory Table points to the second and so on. Every Memory Table is followed by one or more Xpress Images (= compressed physical memory parts). After resume, the first page (the file header) will be cleared.

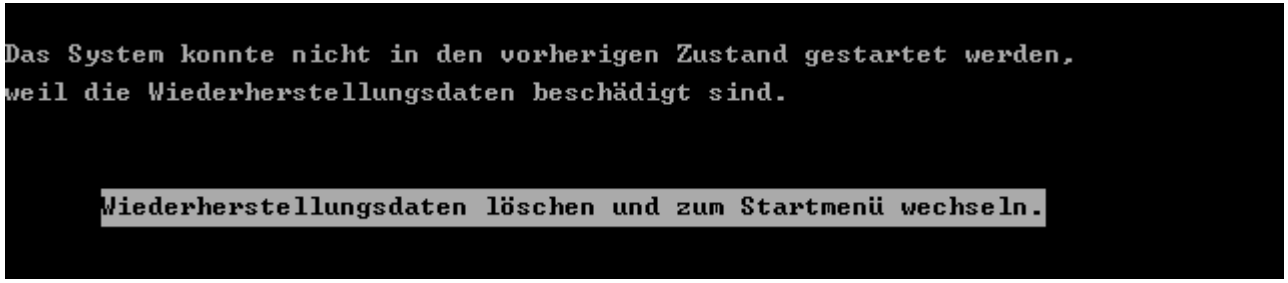
According to [9] defines the `link` signature an ARC path to another hibernation file, which is similar to the boot.ini entry (however, this is completely unproven):

```
linkmulti(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /noexecute=optin
/fastdetect
```

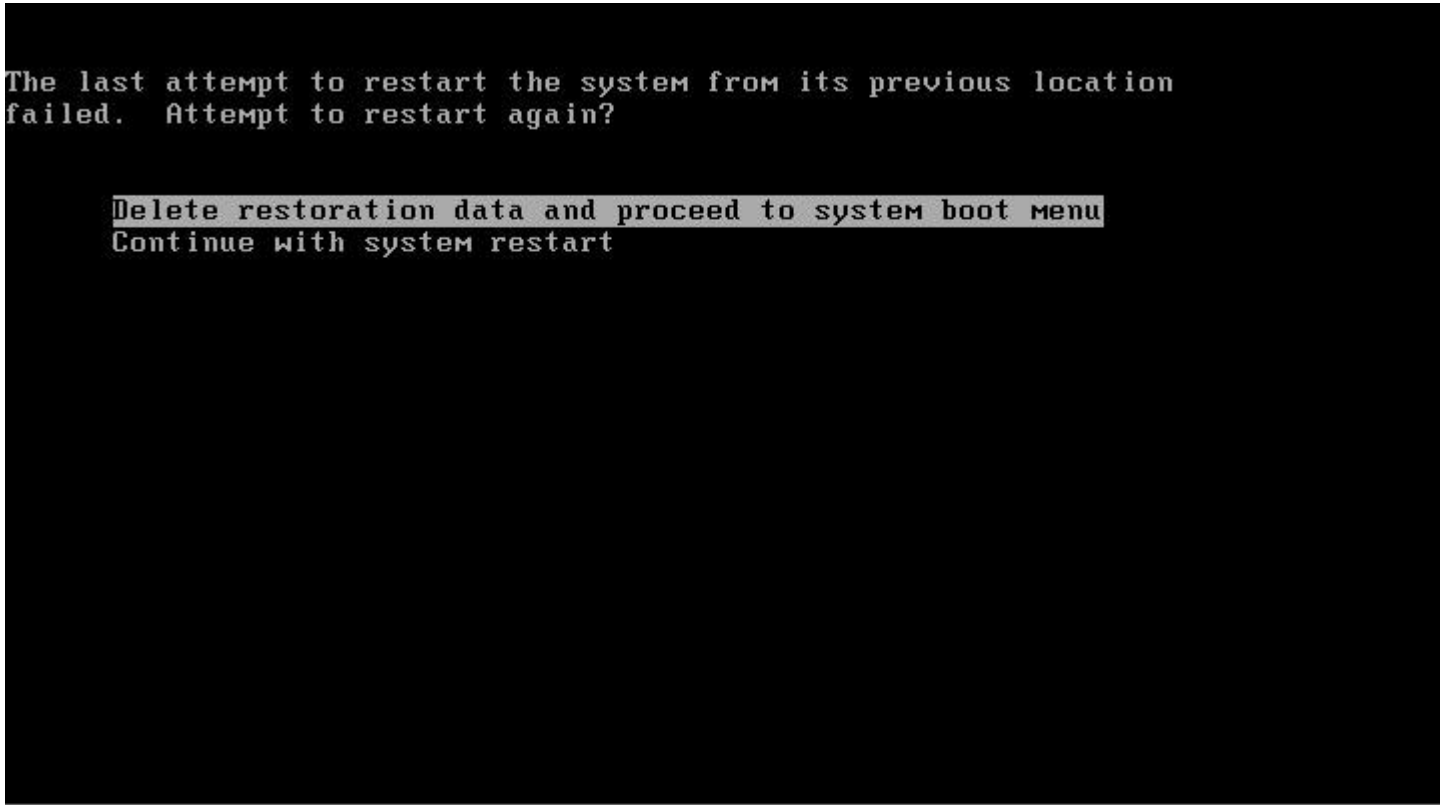
Windows can theoretically boot from the same hibernation file twice. This was a special feature in Windows XP Embedded called Hibernate Once/Resume Many (HORM) [10]. This is not only theoretical possible, it was shown during the live security training that (when restoring the header) Windows just resume from hibernation. It was just restoring following bytes:

```
00000000  68 69 62 72 00 00 00 00 05 E7 00 00 A8 00 00 00  hibr.....ç..¨...
00000010  29 29 01 00 00 10 00 00 00 00 00 00 00 00 00 00  ))..............
00000020  86 F2 6E A0 7E 13 CB 01 48 93 20 2B 03 00 00 00  †òn ~.Ë.H" +....
00000030  FF 3F 03 A0 02 00 00 00 10 00 00 00 00 00 BE FF  ÿ?. ..........¾ÿ
00000040  00 BF 06 01 00 00 00 00 00 03 00 00 0D E7 00 00  .¿............ç..
00000050  4F C5 00 00 3D 93 00 00 03 00 00 00 6E 93 00 00  OÅ..="......n"..
00000060  B2 80 6D 74 04 00 00 00 24 76 B2 68 02 00 00 00  ²€mt....$v²h....
00000070  04 D6 50 41 00 00 00 00 09 F3 1A 63 F9 02 00 00  .ÖPA.....ó.cù...
00000080  4C 26 00 00 38 1F 00 00 C9 01 00 00 E1 10 00 00  L&..8...É...á...
00000090  08 34 00 00 3D 93 00 00 F8 7D 09 09 EF 09 00 00  .4..="..ø}..ï...
000000A0  03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Windows checks the checksum in the header, and if not matching it will output a message that the hibernation file is corrupt (german):



The `wake` signature is used to show an inactive file when resume fails. On the next reboot Windows asks the uses to either try again or delete the restoration data and boot normally [9]:



Hibernation File Header

```
typedef struct
{
     ULONG Signature;
     ULONG ImageType;
     ULONG CheckSum;
     ULONG LengthSelf;
     ULONG PageSelf;
     ULONG PageSize;
//   ULONG ImageType;                    // removed with Vista
     LARGE_INTEGER SystemTime;
     UINT64 InterruptTime;
     ULONG FeatureFlags;
     UCHAR HiberFlags;
     UCHAR spare[3];
     ULONG NoHiberPtes;
     ULONG HiberVa;
     LARGE_INTEGER HiberPte;
     ULONG NoFreePages;
     ULONG FreeMapCheck;
     ULONG WakeCheck;
     ULONG TotalPages;
     ULONG FirstTablePage;
     ULONG LastFilePage;
     PO_HIBER_PERF PerfInfo;             // with Windows XP
     ULONG NoBootLoaderLogPages;         // with Windows Vista
     ULONG BootLoaderLogPages[8];
     ULONG TotalPhysicalMemoryCount;
} PO_MEMORY_IMAGE, *PPO_MEMORY_IMAGE;

typedef struct _PO_HIBER_PERF
{
     UINT64 IoTicks;
     UINT64 InitTicks;
     UINT64 CopyTicks;
```

```
    UINT64 StartCount;
    ULONG ElapsedTime;
    ULONG IoTime;
    ULONG CopyTime;
    ULONG InitTime;
    ULONG PagesWritten;
    ULONG PagesProcessed;
    ULONG BytesCopied;
    ULONG DumpCount;
    ULONG FileRuns;
    UINT64 ResumeAppStartTime;
    UINT64 ResumeAppEndTime;
    UINT64 HiberFileResumeTime;
} PO_HIBER_PERF, *PPO_HIBER_PERF;
```

The header differs slightly with XP, Vista and 7. The variable `FirstTablePage` points to the first Memory Table. Here is a comparison table with the differences in the structure. The one shown above is only valid for Vista.

	XP	Vista	7
00h	Signature	Signature	Signature
04h		ImageType = 0	ImageType = 0
04h	Version = 0		
08h	CheckSum	CheckSum	CheckSum
0Ch	LengthSelf = 168	LengthSelf = 240	LengthSelf = 224
10h	PageSelf	PageSelf	PageSelf
14h	PageSize = 4096	PageSize = 4096	PageSize = 4096
	ImageType = 0		
	SystemTime	SystemTime	SystemTime
	InterruptTime	InterruptTime	InterruptTime
	FeatureFlags	FeatureFlags	FeatureFlags
	HiberFlags = 00000010b	HiberFlags = 00010000b	HiberFlags = 00000001b
	Spare = 0	Spare = 0	Spare = 0
	NoHiberPtes = 16	NoHiberPtes = 32	NoHiberPtes = 32
	HiberVa	HiberVa	HiberVa

	HiberPte	HiberPte	HiberPte
	NoFreePages	NoFreePages	NoFreePages
	FreeMapCheck	FreeMapCheck	FreeMapCheck
	WakeCheck	WakeCheck	WakeCheck
	TotalPages	TotalPages	
	FirstTablePage = 3	FirstTablePage = 6	FirstTablePage = 6
	LastFilePage	LastFilePage	
	PerfInfo	PerfInfo	PerfInfo
			FirmwareRuntimeInformationPages
			FirmwareRuntimeInformation
		NoBootloaderPages	
		BootLoaderLogPages	
		TotalPhysicalMemoryCount	
			Not Used
			ResumeContextCheck
			ResumeContextPages

A reliable way to determine the operating system is by using the LengthSelf field, which is 168 for XP, 240 for Vista and 224 for 7. It is always present at offset 12. Like already mentioned, if you restore the header (which is cleared on successful restoration) you can resume again from the hibernation state. If you do not know the header (for example because it is some computer you stole out of the government), then you can create a working header based on heuristics.

To parse the memory only 2 things are important: If the hibernation file is from XP/Vista or 7, and the FirstTablePage variable. You can get the FirstTablePage variable with a heuristic algorithm, to scan

Memory Table

The physical memory is organized as multiple memory tables. Each memory table has entries, and each entry defines a range of physical pages. The physical memory is compressed and stored as Xpress Images following the memory table.

```
struct MEMORY_TABLE
{
     DWORD PointerSystemTable;
     UINT32 NextTablePage;
     DWORD CheckSum;
     UINT32 EntryCount;
     MEMORY_TABLE_ENTRY MemoryTableEntries[EntryCount];
};

struct MEMORY_TABLE_ENTRY
{
     UINT32 PageCompressedData;
     UINT32 PhysicalStartPage;
     UINT32 PhysicalEndPage;
     DWORD CheckSum;
};

struct IMAGE_XPRESS_HEADER
{
     CHAR Signature[8] = 81h, 81h, "xpress";
     BYTE UncompressedPages = 15;
     UINT32 CompressedSize;
     BYTE Reserved[19] = 0;
};
```

A Memory Table (except the last) is exactly 4 KB big, 16 bytes header + 255 entries * 16 bytes per entry. The `NextTablePage` variable points to the next table and is the page number within the hibernation file. Each entry describes a physical memory range in an Xpress Image.

Following the memory table are the Xpress Images (one following directly the other), which are not page aligned. Thus when searching for physical memory, every Xpress Image needs to be parsed. The `PageCompressedData` variable is supposed to point to the xpress image, but is (because of missing alignment of Xpress Images) only valid for the first (which always follows immediately the Memory Table).

To get the real size of the compressed xpress data calculate CompressedSize / 4 + 1 and round it up to 8. To get the real amount of uncompressed pages calculate UncompressedPages + 1. Please note that this calculation and fields differ from Matthieus Suiche research. Directly after the xpress header there is the compressed data following.

The maximum one Xpress Image can hold is 16 pages (64 KB), which is also the maximum that can be access in real mode using a segment. The compression algorithm used is LZ77 + DIRECT2. MS-OXCRPC [11] explains the details:

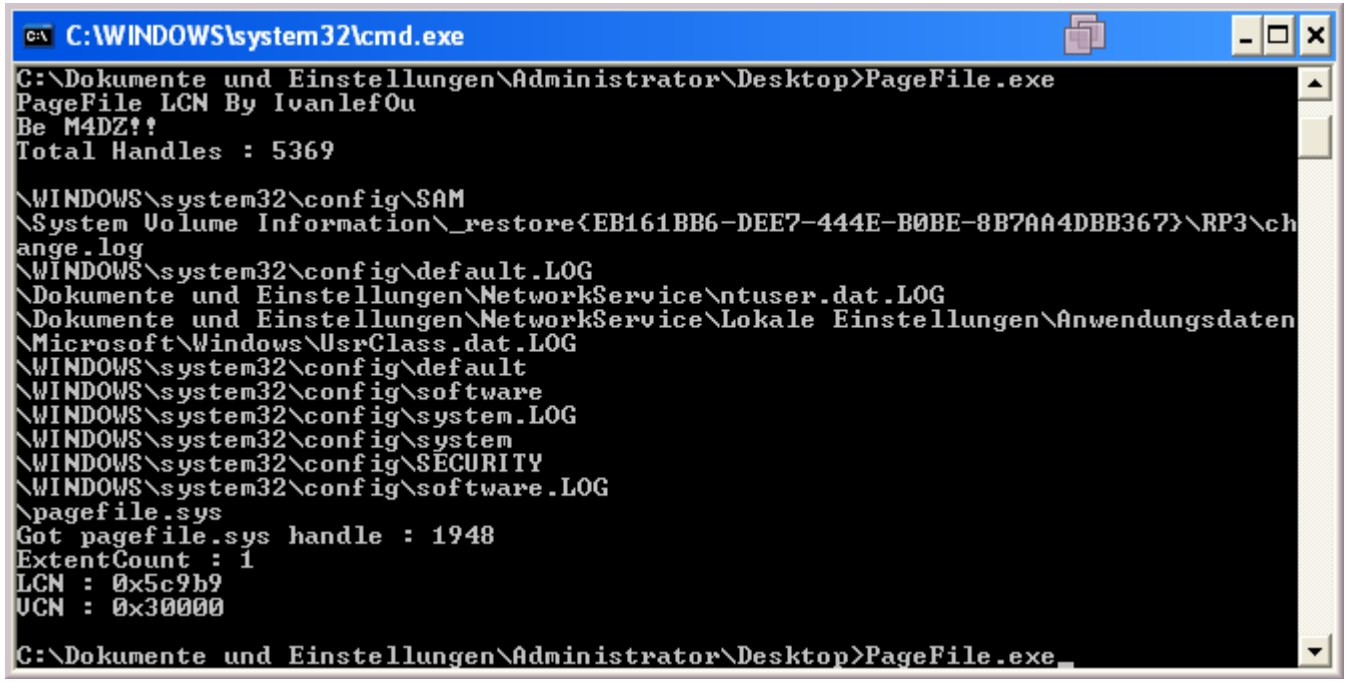
Position	1	2	3	4	5	6	7	8	9
Input	A	A	B	C	B	B	A	B	C

Compression Step	Position	Match	Data	Final Output
1	1	-	A	0, 0, A
2	2	A	B	1, 1, B
3	4	-	C	0, 0, C
4	5	B	B	2, 1, B
5	7	A, B	C	5, 2, C

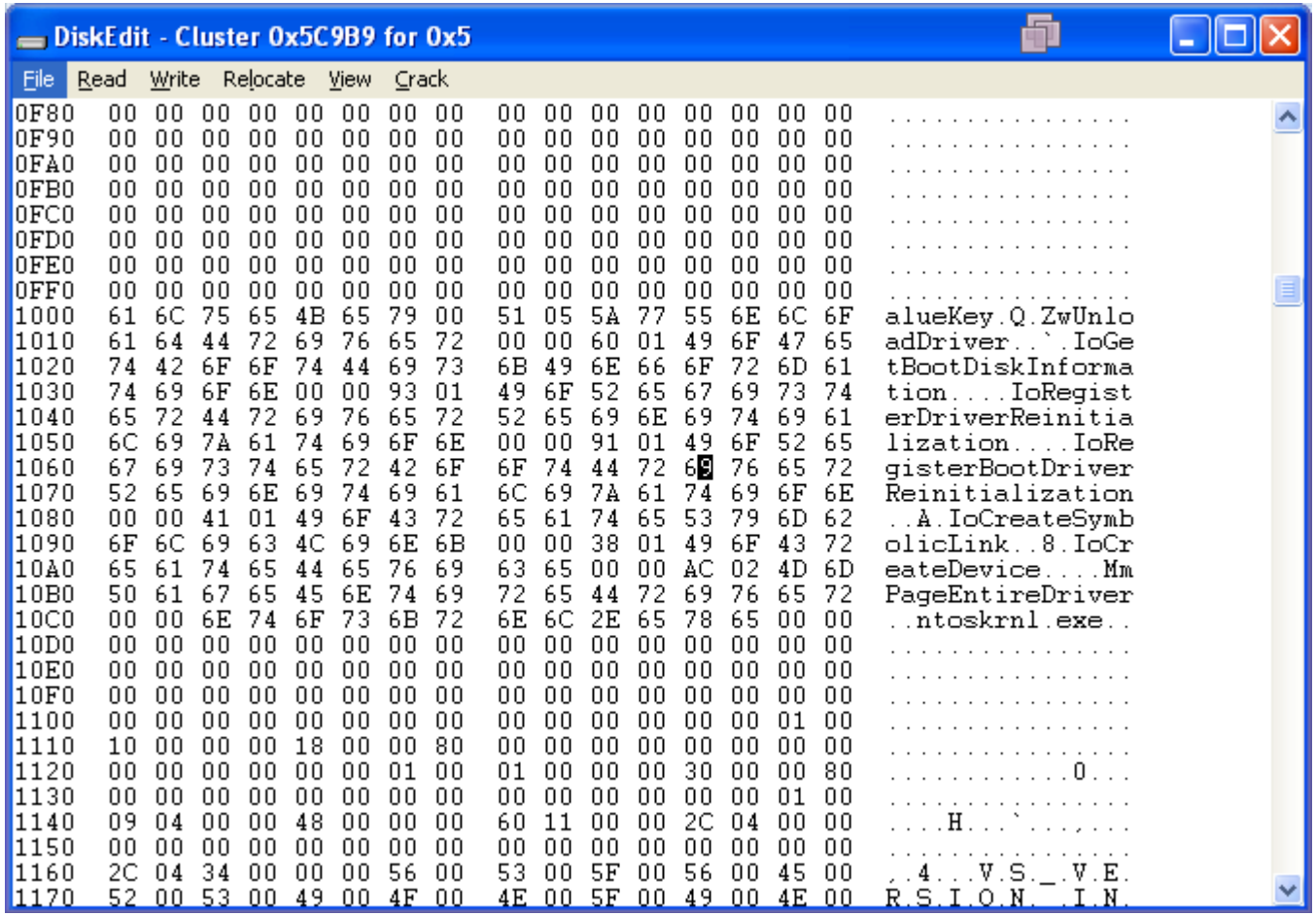
This algorithm is used in the Microsoft Exchange Protocol. The final output is human readable here, in real it is encoded then using DIRECT2, which defines bit-wise indicators for defining a sequence as metadata or data.

Pagefile Attack – The Training

We can use the original open source version [7] of the pagefile attack. PageFile.exe displays only NTFS information about the page file, PageFileAttack.exe attacks it also. The displayed information of the pagefile:



The LCN (Logical Cluster Number) is the important information and tells us where the pagefile is stored at. Using DiskEdit [12] we can directly open that cluster number and display it as hex editor view (we can also modify it there):



The file we are looking here is the NT Disk Manager (dmload.sys) in the pagefile. This can also be verified by shuting down VMware, mounting the drive (C:) and looking manually into the pagefile.sys in an hex editor. The software above (PageFile.exe) enumerates all handles, catches the one of the pagefile and uses then ZwFsControlFile API (exported undocumented in ntdll) with the IOCTL FSCTL_GET_RETRIEVAL_POINTERS. Another possible way would be opening a handle directly to the file system driver (using CreateFile and C:\) and using then DeviceIoControl and sending the IOCTL then.

Null Device Driver

An easy attack is to patch the Null Device Driver (null.sys) in the pagefile or hibernation file. That driver is present on every system, including Linux and Unix and is very small. It only exists to provide the null device `\Device\Null` and `NUL` in Windows:

The Null Device Driver component provides the functional equivalent of \dev\null in the Unix environment by accepting I/O request packets and returning them to the caller. [13]

The file itself is located at C:\Windows\System32\drivers and is between 2 and 5 KB of size. It has following imports:

```
ntoskrnl.exe

IoDeleteDevice
IoDeleteSymbolicLink
RtlInitUnicodeString
IofCompleteRequest
IoCreateDevice
MmPageEntireDriver
KeTickCount (only 7)
```

Here is the reversed source of Windows XP null.sys (the entry point). It was "reversed" using PE explorer and manually edited. The interesting first call is MmPageEntireDriver, which makes the entire null device driver pageable. Then the code creates the device name and registers its handler for IRP_MJ_CREATE, IRP_MJ_CREATE_NAMED_PIPE, IRP_MJ_CLOSE, IRP_MJ_READ, IRP_MJ_LOCK_CONTROL, IRP_MJ_WRITE and DriverUnload.

```
EntryPoint:

; NTSTATUS DriverEntry(
;    __in  struct _DRIVER_OBJECT *DriverObject,
;    __in  PUNICODE_STRING RegistryPath
; )

push ebp                                        ; create the stack frame
mov ebp,esp
```

```
sub esp,3*4                                     ; allocate variables (3 dwords)
push esi

; make the entire driver pageable
;   The MmPageEntireDriver routine causes all of a driver's code and data to be made pageable,
;   overriding the attributes of the various sections that make up the driver's image.
push EntryPoint                                 ;   AddressWithinSection
call [ntoskrnl.exe!MmPageEntireDriver]

; create the unicode string of the device name
push SWC00010580__Device_Null                   ;   __in_opt  PCWSTR SourceString = L"\Device\Null"
lea eax,[ebp-0Ch]                               ; address of first variable (allocated on the stack)
push eax                                        ;   __inout   PUNICODE_STRING DestinationString = on stack
call [ntoskrnl.exe!RtlInitUnicodeString]

; create the device \Device\Null
mov esi,[ebp+08h]                               ; address of driver object (parameter)
lea eax,[ebp-04h]                               ; address of another stack allocated variable
push eax                                        ;   __out     PDEVICE_OBJECT *DeviceObject
push 00000000h                                  ;   __in      BOOLEAN Exclusive = false
push 00000100h                                  ;   __in      ULONG DeviceCharacteristics =
FILE_DEVICE_SECURE_OPEN
push 00000015h                                  ;   __in      DEVICE_TYPE DeviceType = FILE_DEVICE_NULL
lea eax,[ebp-0Ch]
push eax                                        ;   __in_opt  PUNICODE_STRING DeviceName = created one
push 00000000h                                  ;   __in      ULONG DeviceExtensionSize = 0 (no device
extension)
push esi                                        ;   __in      PDRIVER_OBJECT DriverObject = driver object
from entry point
call [ntoskrnl.exe!IoCreateDevice]

test eax,eax                                    ; check if an error was returned
jl Exit_Null_Driver

; fill DRIVER_OBJECT structure
mov eax,DriverDispatch                          ; IRP dispatcher
```

```
mov [esi+38h],eax                                   ;   MajorFunction, install IRP_MJ_CREATE handler
mov [esi+40h],eax                                   ;   MajorFunction, install IRP_MJ_CREATE_NAMED_PIPE
handler
mov [esi+44h],eax                                   ;   MajorFunction, install IRP_MJ_CLOSE handler
mov [esi+48h],eax                                   ;   MajorFunction, install IRP_MJ_READ handler
mov [esi+7Ch],eax                                   ;   MajorFunction, install IRP_MJ_LOCK_CONTROL handler
mov [esi+4Ch],eax                                   ;   MajorFunction, install IRP_MJ_WRITE handler
mov eax,[ebp-04h]                                   ; DeviceObject
mov dword ptr [esi+34h],DriverUnload                ;   DriverUnload function
mov dword ptr [esi+28h],FastIoDispatch              ;   FastIoDispatch structure
mov [DeviceObject],eax

xor eax,eax                                         ; return without any errors

Exit_Null_Driver:

pop esi
leave

ret 8


SWC00010580__Device_Null:
  unicode '\Device\Null',0000h

DeviceObject:
L000103F0:
  dd  00000000h     ; stores the pointer to the DEVICE_OBJECT structure
```

The simplest way is to patch the first instruction of the IRP dispatcher (DriverDispatch) and append the own code after the code of null.sys in memory. The null.sys driver may be in the pagefile or the hibernation file, thus both files must be checked when injecting the code. The proof of concept code for the live security training contains two parsers for the two files and call both the same ScanMemory function which takes a block of memory and scans it for a null device driver signature. That signature consists of 2 parts:

1. L"\Device\Null", within the data section of null.sys
2. "IoDeleteDevice", within IAT of null.sys, and must be in the same memory block as signature part 1

If both strings are found, the base address of null.sys can be easily retrieved by scanning back for the MZ signature. Writing back the modified memory to the pagefile is easy, it can be written directly to disk. For the hibernation file the procedure is much trickier, because the memory is compressed stored in an Xpress Image. Nevertheless we can still replace the xpress image, because Microsofts xpress compression algorithm is not as good as own developed here.

The algorithm published by Matthieu Suiche written in C++ cannot be used, because it does not have the same limitations as the Microsofts one. For example, Matthieus algorithm scans a match of zeroes going over the current position, which is illegal in Microsofts decompression algorithm:

```
        /* search for the longest match in the window for the lookahead buffer */
        for (Offset = 1; (ULONG)Offset < MaxOffset; Offset++) {
            str2 = &str1[-Offset];

            /* maximum Len we can encode into Metadata */
            MaxLen = min((255 + 15 + 7 + 3), ByteLeft);
//            MaxLen = InputIndex - Offset;

            for (Len = 0; (Len < MaxLen) && (str1[Len] == str2[Len]); Len++);

            /*
            * We check if Len is better than the value found before, including the
            * sequence of identical bytes
            */
            if (Len > BestLen) {
                found = TRUE;
                BestLen = Len;
                BestOffset = Offset;
            }
        }
```

MaxLen is set here to 255 + 15 + 7 + 3 or bytes to compress left (whichever is smaller) – but Offset may be for example -1 from current position and MaxLen would be then 280 and could and in fact would capture "future" zeroes. The solution to fix this is shown with the commented MaxLen = InputIndex – Offset, which ensures that a match is only searched in "present"

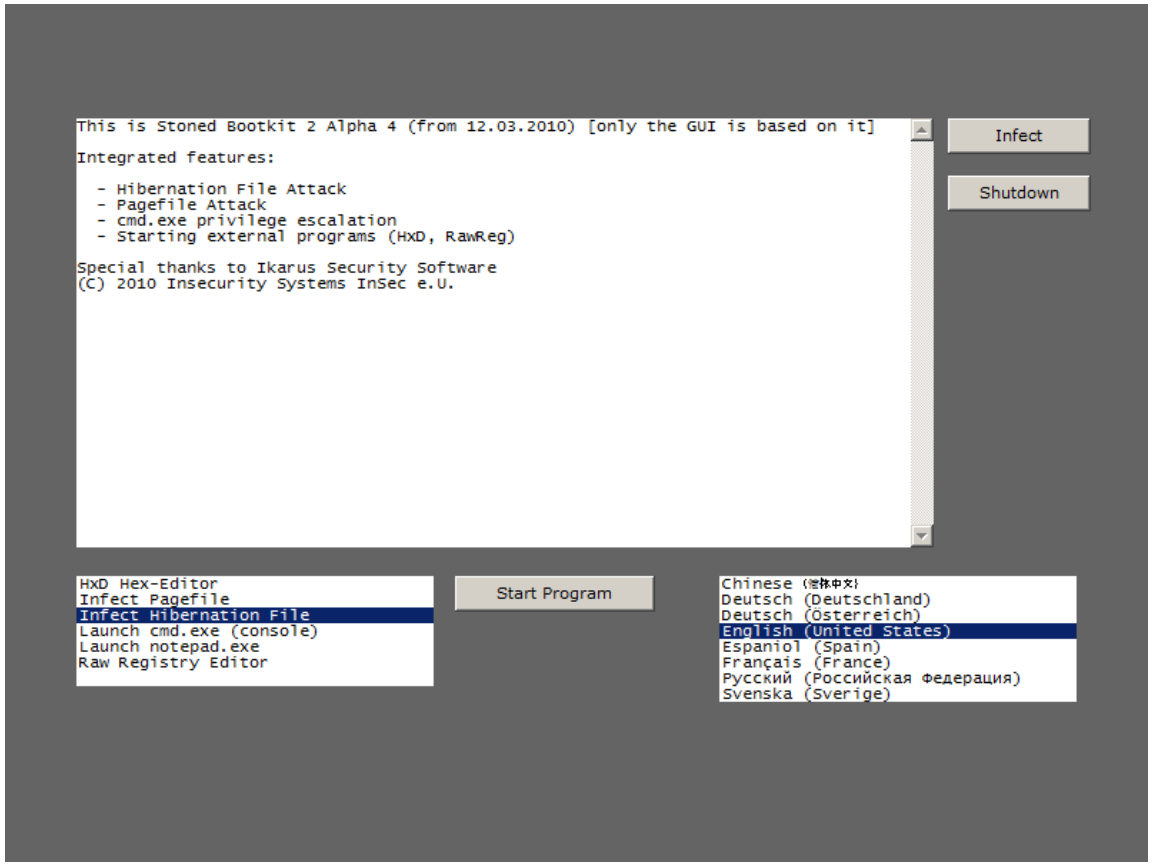
memory (which still not makes Matthieus algorithm working due to other problems, also lack of support for > 280 byte matches).

The maximum match offset can be 1FFFh bytes away and the storage for the maximum length is theoretically 3 bits + 4 bits + 1 byte + 1 word for length + 3.

Hibernation File Attack – The Training

To test the proof of concept code coming with this security training, do following steps:

1. Compile the project Injector with Visual Studio 2008 or 2010
2. Hibernate your VMware, map the drive
3. Start the injector, choose "Infect Hibernation File" and click on "Start Program"
4. In the open-file dialog choose the hibernation file



The output will look like:

```
Opened hibernation file Z:\hiberfil.sys successfully, handling 511 MB
  Windows XP hibernation file
  Found null.sys in Memory Table 4041, entry 244
  In physical memory range 0x0700b000 to 0x0700d000
  Hibernation file successfully patched :)
```

That executable can also be placed on a Windows PE Live CD:

1. Download the Windows AIK and install it
2. In the "Deployment Tools Command Prompt" execute "copype.cmd x86 c:\hibernation" to create the Windows PE Build Environment
3. Mount the image using "Dism /Mount-Wim /WimFile:C:\hibernation\winpe.wim /index:1 /MountDir:C:\hibernation\mount"
4. Copy the executable (Injector.exe) and all other utils to C:\hibernation\mount\Hibernation\ (create that directory)

To execute that automatically when Windows PE starts, create a `Winpeshl.ini` file in the System32 directory with following file contents:

```
[LaunchApp]
AppPath = "%SYSTEMDRIVE%\Hibernation\Injector.exe"
```

a. Insert VMware SCSI driver

Download http://www.vmware.com/download1/software/support/VMware-BusLogic-SCSIDriver-1.2.0.0.flp and extract it with WinImage.
Store it to C:\hibernation\Files\vmscsi

Dism /image:C:\hibernation\mount /Add-Driver /Driver:C:\hibernation\Files\vmscsi

b. Insert the files from "Modification Files", which may be started from the user interface

Those are just some optional tools that may be useful in the Live CD interface:

\Program Files\HxD
\Program Files\RawReg

c. Add Language Pack

Dism /image:C:\hibernation\mount /Add-Package /PackagePath:"C:\Program Files\Windows AIK\Tools\PETools\x86\WinPE_FPs\winpe-fontsupport-zh-cn.cab"

Set language and packages:
Dism /image:c:\hibernation\mount /Get-Intl
Dism /image:c:\hibernation\mount /Get-Packages

d. Windows Management Instrumentation support

Dism /image:C:\hibernation\mount /Add-Package /PackagePath:"C:\Program Files\Windows AIK\Tools\PETools\x86\WinPE_FPs\winpe-wmi.cab"

5. Commit the changes "Dism /Unmount-Wim /MountDir:C:\hibernation\mount /Commit"
6. Use the Windows Image (.wim) for the Live CD "copy c:\hibernation\winpe.wim c:\hibernation\ISO\sources\boot.wim /Y"
7. No "Press any key to boot from CD" message: "del C:\hibernation\ISO\boot\bootfix.bin"
8. Create the iso "oscdimg -n -bC:\hibernation\etfsboot.com C:\hibernation\ISO "C:\hibernation\hiber.iso""
9. That's it! Burn the iso to a CD or DVD.

All steps above can also be done for a 64-bit image (replace x86 with amd64) and with the Vista AIK instead of the 7 AIK (use the imagex tool instead of the dism program).

With Vista the hibernation file cannot be opened, because the access rights prevent it from being read or modified by “anyone”. The Live CD with Windows PE also checks the NTFS file system rights.

More information TBA

Related Work

Matthieu Suiche has presented “Windows hibernation file for fun 'n' profit” at Black Hat USA 2008. There is a hibernation file and other software converting hibernation files to crash dumps from him available [14]. Matthieu was the first guy documenting the hibernation file format publicly.

References

[1] ACPI specification 4.0a
http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf

[2] Vista Notes, Episode #2: Sleep vs. Hybrid Sleep
http://blogs.msdn.com/b/justsean/archive/2006/12/08/vista-notes-episode-2-sleep-vs-hybrid-sleep.aspx

[3] Disable Hybrid Sleep Mode in Windows Vista
http://www.howtogeek.com/howto/windows-vista/disable-hybrid-sleep-mode/

[4] Installing a Boot-Start Driver
http://msdn.microsoft.com/en-us/library/ff547570(VS.85).aspx

[5] TrueCrypt Documentation: Hibernation File
http://www.truecrypt.org/docs/?s=hibernation-file

[6] Clear virtual memory pagefile when system shuts down
http://msdn.microsoft.com/en-us/library/ms814147.aspx

[7] Pagefile attack
http://www.ivanlef0u.tuxfamily.org/?p=77

[8] NTFS Documentation, page 47 "RSTR"
http://data.linux-ntfs.org/ntfsdoc.pdf

[9] Hibernation File Signatures
http://jessekornblum.livejournal.com/254105.html

[10] Enabling a Hibernate Once/Resume Many Environment with EWF
http://msdn.microsoft.com/en-us/library/ms940851(WinEmbedded.5).aspx

[11] [MS-OXCRPC]: Wire Format Protocol Specification

http://msdn.microsoft.com/en-us/library/cc425493.aspx
[12] DiskEdit
http://web17.webbpro.de/index.php?page=diskedit

[13] Null Device Driver
http://msdn.microsoft.com/en-us/library/aa939249(WinEmbedded.5).aspx

[14] Your hibernation file in a nutshell – Part II
http://www.msuiche.net/2008/12/13/your-hibernation-file-in-a-nutshell-part-ii/