Low-level writing to NTFS file systems

Rick van Gorp[1]

[1] System and Network Engineering
Faculty of Science

July 3, 2018

Introduction

- Red teams: Unwriteable files or Endpoint Security
- Windows API allows interaction of user-mode with kernel-mode functions
- Endpoint security could monitor and block I/O activity
- Low level-writing to NTFS drives: Bypass NTFS access lists and software hooks
- Allows user to overwrite or falsify data

Research Question

- In what way can data be written to an NTFS filesystem, such that hooks in write operations in Windows are bypassed?

Related work

- Joseph Bialek (2015) - Created `Invoke-Ninjacopy`[1] that opens a read handle to an NTFS volume and parses the NTFS volume to retrieve files.
- Cloudburst Security (2016) - Shellcode in malware bypasses Anti-Virus (AV) hooks by overwriting a function prolog the AV used to hook into the function.
- Blackhat USA, Udi Yavo and Tomer Bitton (2016) - Identified security issues in the Windows hooking methods and described different hooking engines.

[1] https://github.com/clymb3r/PowerShell/tree/master/Invoke-NinjaCopy

Methods

Research environment:

- Windows 10 - Home Edition x64
- Windows 7 - Home Edition x64 (Virtualbox and VMWare instance)

Methodology:

- Desk research: Gather information regarding the Windows API and NTFS
- Static analysis with IDA Free edition: Analyse Windows API user-mode and kernel-mode
- Write experiments: Test whether we can write directly to a raw disk or NTFS volume
- Attempt to verify the found implications

Windows API - Write function

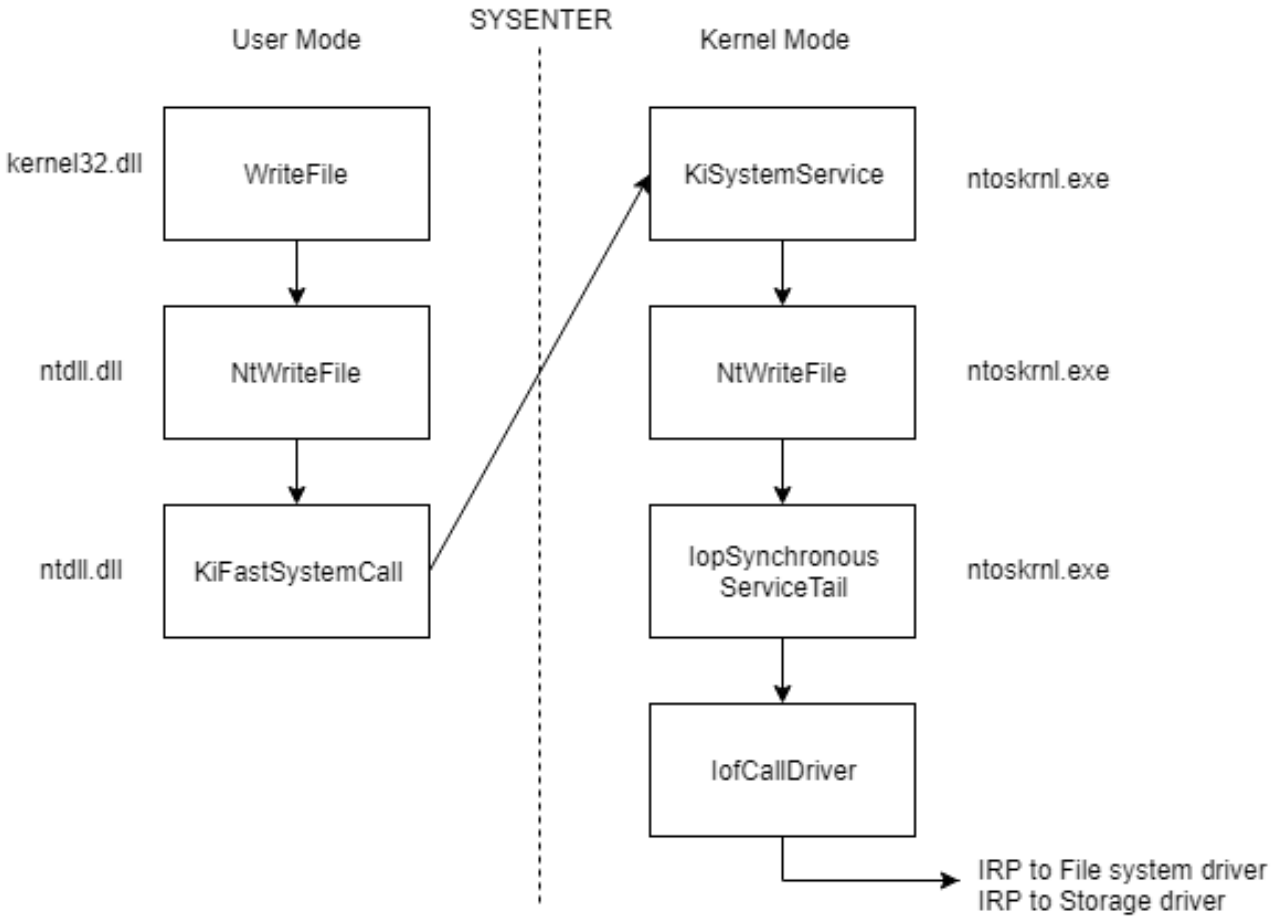

Figure 1: Windows API call follow-up scheme: user-mode and kernel-mode

[1] IRP = Input Output Request Packet

Windows API - Storage driver stack

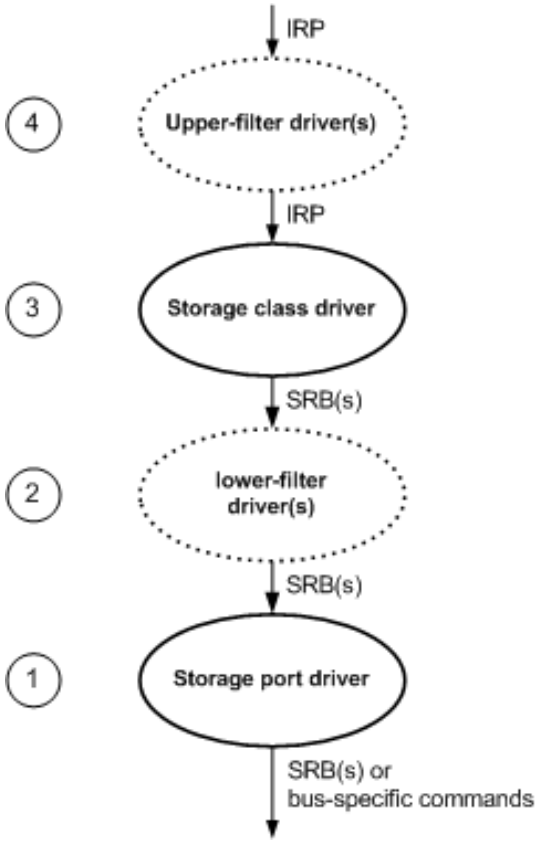

Figure 2: Processing of IRP from the filter driver by storage class driver (Microsoft 2017[2])

[2] https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

[2] SRB = SCSi Request Block

Hooking - User-mode

- Listen to or modify the behavior of a process by intercepting an instruction of a program

- DLL-injection: Inject code into another process
- Import Address Table: Change memory address of target function
- Inline hooking:

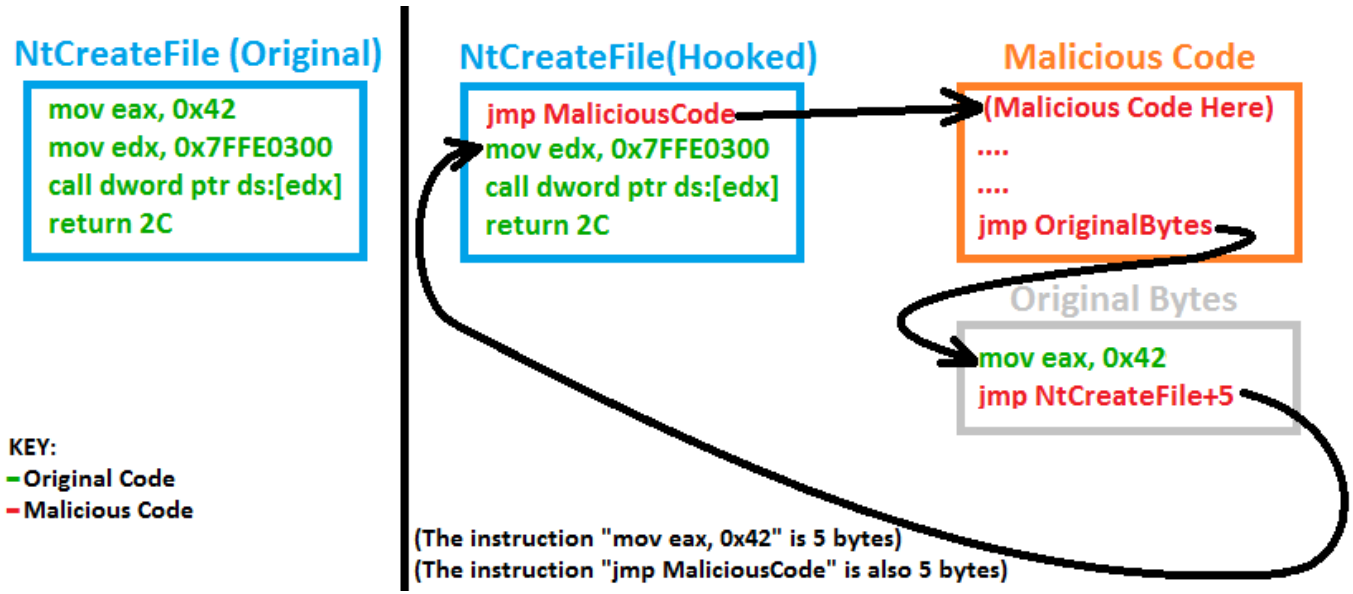

Figure 3: Inline hooking example of NtCreateFile (UserPC.net 2017[3])

[3] https://userpc.net/wp-content/uploads/2017/12/InlineHook.png

Hooking - Kernel-mode

- System Service Dispatch Table: Replace pointers for Nt-functions with pointers to own code.
- `SYSENTER_EIP`: Replace register address with address of detour function.
- Interrupt Service Routines: Map interrupt with response. Replace address of response with hooking function.
- IRP Major Function: Driver object contains function pointers, that are called from other drivers through `IoCallDriver`. Other drivers could replace those pointers to its own functions.

File system filter drivers

- Log, monitor, modify or prevent I/O operations related to the file system.
- Priority controlled by altitudes: Multiple Endpoint security companies are registered with Microsoft [4].

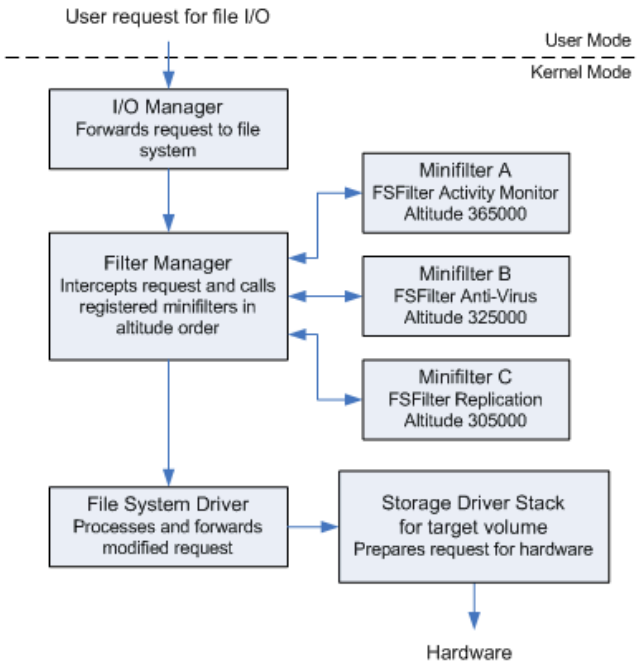

Figure 4: Simplified I/O Stack with filter manager en three filter drivers (Microsoft, 2017 [5])

[4] https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/allocated-altitudes

[5] https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

Write experiments - Setup

- Attempt to write to NTFS boot sector, Master File Table and file system space

Experiment 1

- User-mode application[6]: Opens write-handle to volume/physical disk with `CreateFile` and `WriteFile`.

Experiment 2

- Kernel-mode driver[6]: Open write-handle to volume/physical disk with `NtCreateFile` and `NtWriteFile`.

Experiment 3

- Kernel-mode driver[6]: Write directly to storage class driver with `IRP_MJ_WRITE` and flag `SL_FORCE_DIRECT_WRITE`.

[6] https://github.com/rickvg/low-level-ntfs

Write experiments - Results 1 & 2

	Handle to volume	Handle to harddisk
Write to file system	✗	✗
Write to NTFS boot sector	✓	✗
Write to Master File Table	✗	✗
Alert by Endpoint security	✗	✓

Table 1: Results of write experiments sorted by writing to volume and harddisk

	Handle to volume	Handle to harddisk
Write to file system	✗	✗
Write to NTFS boot sector	✓	✗
Write to Master File Table	✗	✗
Alert by Endpoint security	✗	✗

Table 2: Results of write experiments sorted by writing to volume and harddisk

Write experiments - Results 3

	IDE	SATA	SCSI	SAS
Write to file system	✗	✗	✓	✓
Write to NTFS boot sector	✗	✗	✓	✓
Write to Master File Table	✗	✗	✓	✓
Alert by Endpoint security	✗	✗	✗	✗

Table 3: Results of write experiments while directly communicating with the storage class driver, sorted by storage technology

- SATA & IDE: Invalid SCSI block request: No proper translation between IRP and SRB.
- Possible solution: Communicate with storage port drivers directly using own SRBs.

Writing to files on an NTFS volume

- Experiments: Raw disk access possible
- Objective: Locate the data linked to files

- Open a read handle to the raw disk and read its contents
- Locate the NTFS volume: Identified by hex-string `EB 52 90 4E 54 46 53`
- Parse the boot sector to identify the location of the Master File Table
- Parse the Master File Table (MFT) to locate the data in the volume using data runs
- Overwrite the file data at the location specified

Writing to files on an NTFS volume - Parsing the boot sector

- Data in table 4 resides in the BPB-section[7] of the boot sector, starting at offset 0xB

Offset (hex)	Length	Value
0x0B	2 bytes	Bytes per sector (S_{bytes})
0x0D	1 byte	Sectors per cluster ($C_{sectors}$)
0x28	8 bytes	Total amount of sectors
0x30	8 bytes	Logical cluster number of MFT ($MFT_{clusterloc}$)
0x38	8 bytes	Logical cluster number copy MFT
0x40	1 byte	Clusters per MFT record
0x44	1 byte	Clusters per index buffer

Table 4: Sector and Cluster information and MFT location information offsets within BPB of NTFS bootsector

- Calculate MFT position in bytes: $ByteLoc_{MFT} = S_{bytes} * C_{sectors} * MFT_{clusterloc}$

[7] Bios Parameter Block

Writing to files on an NTFS volume - Parsing the MFT (1)

- Check whether file is deleted or still available
- Get location of attributes and find filename and data

Offset (hex)	Length	Value
0x0	4 bytes	FILE, if invalid BAAD
0x14	2 bytes	Attribute offset
0x16	2 bytes	00 00 = Deleted 01 00 = Allocated 02 00 = Directory deleted 03 00 = Directory allocated
0x18	4 bytes	Actual record size
0x1C	4 bytes	Physical record size

Table 5: Relevant entries of an MFT-record that point to the filename and data of file and contain length data

Writing to files on an NTFS volume - Parsing the MFT (2)

- First four bytes: Attribute type (30 00 00 00 for filename) (80 00 00 00 for data)
- Attribute contains resident or non-resident header
- If attribute does not match: Get attribute length at offset 0x4 and skip attribute

Offset (hex)	Length	Value
0x0	1 byte	Header
0x1	Defined by first 4 bits of header	Cluster count
Unknown	Defined by last 4 bits of header	LCN[8] Offset

Table 6: NTFS Data run structure repeated x times and terminated by 00, where the actual location of the data is shown at LCN offset

- Data at location of LCN offset(s) can be overwritten, which results in overwriting file data

[8] Logical Cluster Number

Discussion

- Access to raw disk, without communicating to the file system driver, results in bypass of NTFS permissions: Write data to any file on the system.
- Endpoint security software that blocks write operations at minifilter driver level can be bypassed in that function, since the IRP does not pass the minifilter driver. Malicious code can be written to any location on the disk.
- Possible to intercept `IRP_MJ_WRITE` with an IRP hook, attached to the storage class or storage port driver.
- Kernel mode drivers unsigned: On x64-systems load after disabling driver signature enforcement (DSEFix[9]).
- Endpoint security might detect loading of unsigned kernel mode drivers.

[9] https://github.com/hfiref0x/DSEFix

Conclusion

- From user-mode: the location pointer to the Master File table can be changed using write handle to volume
- From kernel-mode: Raw write access to the disk is possible by building an IRP.
  - Bypass NTFS Access lists
  - Bypass Endpoint security that operates at minifilter driver level
  - Bypass software hooks on write operations higher than the storage class layer
  - `SL_FORCE_DIRECT_WRITE` flag must be set.
  - User-mode application could send data to the driver to write

Future work

- Research lower level methods that directly communicate with the storage port drivers. Requires specific commands for ATA and IDE based harddisks.
- Verify whether the writing methods bypass Endpoint security solutions that might be hooked to write-related functions on a lower level than storage class.
- Research new techniques for loading unsigned kernel mode drivers or methods that use vulnerabilities in already signed drivers to communicate with the storage drivers.

Bonus - Detection rates of Internet Security Solutions

- Results: Detection of several Internet Security solutions
- Default settings: Writing detection only happens when an on-access feature is enabled
- Worm: Loveletter.vbs

	1	2	3	4	5	6
Regular write	✓	✓	✓	✓	✓	✓
Loading driver	✗	✗	✗	✗	✗	✗
Communication with driver	✗	✗	✗	✗	✗	✗
DSEFix activity	✗	✗	✗	✗	✗	✓
Write from kernel driver to disk	✗	✗	✗	✗	✗	✗

Table 7: Detection of writing malicious code to the disk by Internet Security solutions from user-mode and kernel-mode

Bonus 2 - Use cases

Write malicious content:

- Situation: Defense has Endpoint security running that uses minifilter driver as lowest level protection and has real-time protection enabled.
- Load kernel mode driver and write malicious content to disk, without the real-time protection blocking the write operation.

Become domain administrator:

- Situation: Attacker is in position of performing a Remote Code Execution attack on a Domain Controller.
- Access Active Directory Database (ntds.dit), which is always locked.
- Change password of Domain Administrator or create new Domain Administrator and operate from that user.
- Future research: Since we write directly to the file and not through Active Directory functions, would performing this action appear in the Windows logging and alert system administrators?

Bonus 2 - Use cases 2

To be verified:

- Bypass Windows audit logs on important change or addition events?
- Manipulate evidence by unnoticed writes?