Virtual File Systems for Beginners

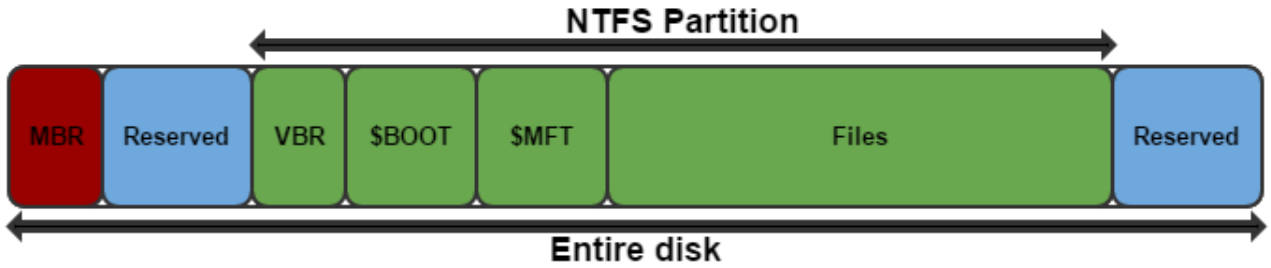
By Marcus Hutchins

Published: 2014-11-28 · Archived: 2026-04-05 13:33:08 UTC

A virtual File System (VFS), sometimes referred to as a Hidden File System, is a storage technique most commonly used by kernel mode malware, usually to store components outside of the existing filesystem. By using a virtual filesystem, malware developers can both bypass antivirus scanners as well as complicating work for forensic experts.

Filesystem Basics

If you're running Windows and not using hardware from the 90s, or have your OS installed on a flash drive; chances are, you're using the New Technology File System (NTFS). In order to understand how a VFS benefits malware developers, first we need to dive into a bit of filesystem basics.



In this example we have a disk containing only one partition (which runs Windows).

- The Master Boot Record (MBR) gives the system information about the partition, such as its start sector and size.
- The Volume Boot Record (VBR) is the primary boot code and will load and Windows bootloader and execute it; The VBR is the first sector within the NTFS partition.
- $BOOT is the boot area and contains the Windows boot loader.
- $MFT is the Master File Table and tells the system where to find files within the filesystem.

Antivirus Scans A full system scan will go through every file in the master file table and scan it, additionally the antivirus can hook the filesystem driver and scan files on creation / write. If somebody didn't want a file to be scanned, not adding an entry to the MFT would be a good start. Unfortunately, if sectors within the partition are not referenced by the MFT, they are assumed unused and likely to be overwritten as more files are written to the disk.

Malware Forensics There are lots of techniques used when analyzing an infected system; however, looking for new/modified files is a common starting point for an analyst. To speed up file deletion, the system simply deletes the file's record in the MFT but leaves the actual file intact, this way the sectors can be overwritten by an new file and the system doesn't have to waste time zeroing out the old one. Due to the fact there's going to be random data

left by deleted files all over the disk, it's very easy for an encrypted virtual filesystem to hide, further complicating analysis.

Obviously if we can't write directly to free sectors within the partition for fear of them being overwritten, then we're going to have to write our VFS outside of the partition; What makes this possible is the fact that there is unused reserves space on both ends of the disk.

Disk Basics

Space after the MBR A disk platter is divided into tracks which are divided into sectors; a single sector is 512 bytes in size and there are a fixed number of sectors per a track. As technology advanced the physical size of sectors got smaller so more sectors could be fit onto a single track; however, the MBR field that describes the number of sectors is 6 bits in size, thus can only support numbers 0 – 63, limiting the sectors per track to 63.

Eventually, someone figured out that the the closer to the edge of the disk you get, the longer the tracks are and the more sectors the can hold. Nowadays the number of sectors per a track varies depending on how far away from the spindle the track is, making the sectors per a track field of the MBR totally meaningless; For compatibility reason, disks with more than 63 sectors per a track will just leave the value set at 63, the same goes for SSDs or other media that doesn't have tracks.

For optimization reasons when partitioning the disk, the Windows partition manager will read the sectors per track value and align the partition on the track boundary (63 sectors per track vmeans that the MBR will be sector 0 track 0, while the start of the partition will be sector 0 track 1, leaving 62 sectors of unused space between the MBR and first partition).

The only problem with aligning the partition to 63 virtual (512kb) sectors is if the disk internally used 4kb sectors, then there's going to be a huge performance penalty because 63 * 512 is not a multiple of 4kb, so the OS will constantly be writing across sector boundaries and wasting time with unnecessary Read-Modify-Write cycles. In Windows Vista and onward Microsoft addresses this issue by starting the partition on the 2048th sector (leaving 1 MB of reserved space and 4kb aligning the partition), nobody is exactly sure why they chose to leave so much space, but when it comes to malware, 1 MB is a lot of storage.

Space at then end of the disk Because the space at the start of the disk can be pretty small and isn't guaranteed on GPT systems, the space at the end may be a better bet. When allocating a partition, the Windows partition manager will end the partition before the end of the disk to leave space for dynamic disk information. As it happens, dynamic disks are incredibly rare on most computers because they're only used for software RAID and other black magic, which leave between 1 mb and 100 mb of space at the end of the disk.

Virtual File System

The location of the Virtual File System depends on the space needed and the system specifications, here's a quick overview of the reserved space.

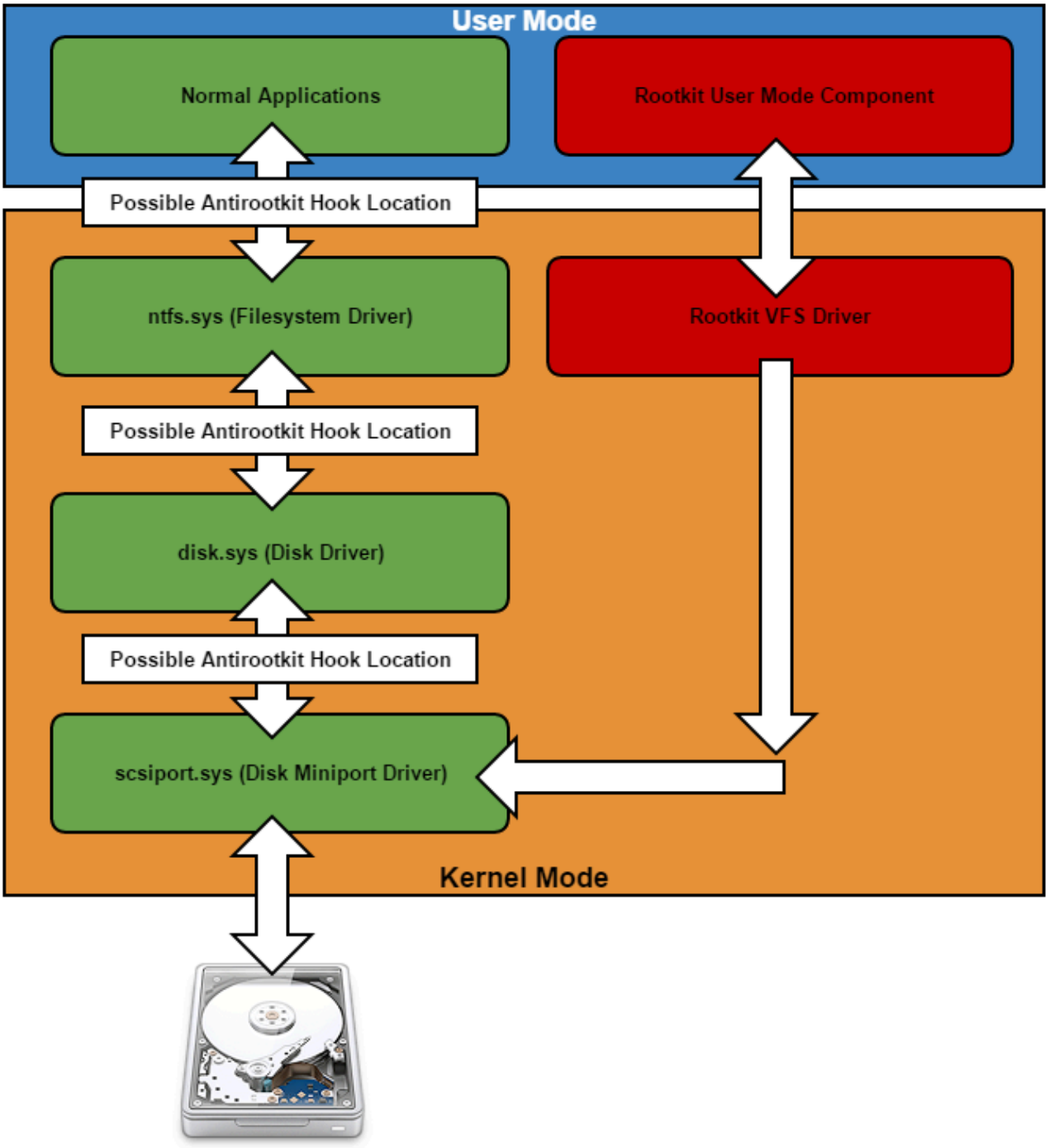
Start Of Disk

- On XP systems using the MBR partition format you are guaranteed 62 sectors (31.7 KB) of space between the MBR and the first partition.
- On Vista+ systems using the MBR partition format you are guaranteed 2047 sectors (1 MB) of space between the MBR and the first partition.
- Because the GUID Partition Table (GPT) is of variable size and not restricted to 1 sector like the MBR, it is uncertain how much space will be available on systems using the GPT.
- Other than by the GPT, this space is never used by windows.

End Of Disk

- Between 1 MB and 100 MB, there doesn't appear to be any OS specifications for the exact size so the variation is likely to do with disk geometry (Ex: 1 disk track is reserved).
- Some of the space can be used for dynamic disk information (most system do not use dynamic disks unless using software RAID).

Contrary to popular belief, a VFS can be created and accessed by a user mode application, as long as it is running as administrator. To prevent malware from bypassing kernel code signing, raw disk access was "disabled" in vista and onward; however, there is an exception for boot sectors and sectors residing outside of the filesystem (both reserved areas reside outside the filesystem), enabling user mode access to the VFS. Although, direct user mode access is possible, most malware tends to manage the VFS from a kernel driver and expose an API to user mode components for reading/writing via the driver; This allows the VFS to be hidden from normal applications and other drivers.



It’s quite common for a VFS driver to send requests directly to the lowest level disk driver (the disk miniport), as a result the disk read/write requests cannot be intercepted by the antivirus or any standard disk monitors, providing better stealth. Although you could write standard files using this method, ntfs.sys handles the NTFS specification, so you’d have to create your own ntfs driver which would be a lot of work especially as NTFS is not fully documented by Microsoft.

The actual format of the VFS is entirely dependent on the developer, some have chosen to use FAT32 with RC4 encryption, whilst others use custom file systems with modified encryption algorithms. Almost always the VFS is encrypted in an attempt to make the data look like random leftover bytes and not executables or log files.

Bootkits most commonly use a VFS because it reduces the attack surface to a single point of attack: The infected bootloader reads the rootkit driver from the VFS and loads it into the kernel long before the antivirus, leaving the kernel driver time to install hooks and cover its tracks before the OS even initializes. A bootkit using a VFS driver has only one weakness: The infected boot record; this can be easily resolved by using the bootkit's driver to hook the disk miniport and spoof read/write requests to the boot sector, tricking the AV into thinking the boot sector contains the original Windows boot code, the same method can also be used to just display empty sectors if something other than the rootkit tries to read the VFS.
