DEFEATING ANTI-FORENSICS IN CONTEMPORARY COMPLEX THREATS

Eugene Rodionov & Aleksandr Matrosov
Eset, Slovakia

Email {rodionov, matrosov}@eset.sk

ABSTRACT

Forensic analysis plays a crucial role in cybercrime group investigation as it allows investigators to obtain such information as bot configuration data, C&C URLs, payload, stolen data and so on. Some of the modern malware falling into the class of complex threats employs various tricks to resist forensics and conceal its presence on the infected system. This paper will present technical and in-depth analysis of the most widely used anti-forensic technique, the implementation of hidden encrypted storage, as used by complex threats currently in the wild:

- Win64/Olmarik (TDL4)
- Win64/Olmasco (MaxSS)
- Win64/Rovnix/Carberp
- Win32/Sirefef (ZeroAccess)
- Win32/Hodprot.

These complex threats use hidden encrypted storage areas to conceal their data and avoid relying on the file system maintained by the operating system. In this paper we will focus on the details of hidden storage implementation as well as the ways in which it is maintained within the system by various kinds of malware. The analysis begins with the initialization procedure and the mechanisms behind it. It is shown which system mechanisms are used to store and retrieve data from the hidden container and the degree to which the malware depends on them. Close attention is paid to the self-defence mechanisms employed by the malware in order to conceal the content kept in its hidden storage areas and protect those contents against modification by the system or by security software. In addition, a detailed description of the hidden file system is presented for each threat considered, as well as a comparison of its features with the other threats analysed here.

To conclude the paper, an approach is presented on the retrieval of data from hidden storage. We will discuss the steps that should be taken to defeat self-defence mechanisms, locate hidden storage on the hard drive and read plain data.

ANTI-FORENSIC FEATURES

Nowadays there are some malware families that strongly resist forensic analysis. There are different means of counteracting malware detection and removal from systems: these include encryption and obfuscation of the C&C communication protocol, encryption of files containing payload and configuration information, and so on. In this paper we concentrate on one of the most advanced features intended to impede forensics found in modern in-the-wild complex threats – namely, implementing hidden encrypted storage.

At the heart of this relatively new technology lies the implementation of a hidden virtual storage device with transparent encryption of the data being read or written to. This allows malware employing such a technology to gain the following advantages:

- keeping its data secret and stealthy
- providing a payload with a fairly standard interface in order to store information and to retrieve it from storage
- bypassing security software.

Covert storage

The main point of maintaining hidden storage in the system is not just to provide confidentiality of the information being stored but also to conceal the very presence of the data. More often than not, malware keeps its data in encrypted files on the hard drive using the file system maintained by the operating system, and that eventually reveals its presence in the system. In the case of hidden storage as described here, however, there is usually no file available for analysis in the OS file system. None of the data related to the malware are located outside the file system and are encrypted. In such a case it is quite difficult to spot the presence of the malware based on the examination of a disk image, as often takes place during forensic analysis.

Standard interface

Access to the data stored on the hard drive is usually through the standard API using calls such as:

- CreateFile/CloseHandle
- ReadFile/WriteFile
- SetFilePointer.

Alternatively, any other system routines may be used, such as GetPrivateProfileString, WritePrivateProfileString, and so on. As a result, the development of a payload module doesn't require knowledge of any specific technologies. Data kept inside hidden storage can be accessed using standard system routines.

Hidden storage architecture

The general architecture of hidden storage implementation is presented in Figure 1. Some complex threats locate and allocate space on the hard drive – usually at the end of it, where they store the image of the hidden file system containing malicious data. Usually there is some free space – up to several MB – to be found at the end of the hard drive which isn't used by the system.

To access the data the malware performs low-level read/write operations, usually using the interface provided by the storage miniport kernel-mode driver located at the very bottom of the storage device driver stack [1]. By sending IRP_MJ_INTERNAL_DEVICE_CONTROL requests to the miniport driver, malware is able to read/write sectors of the hard drive and thus maintain its hidden file system.

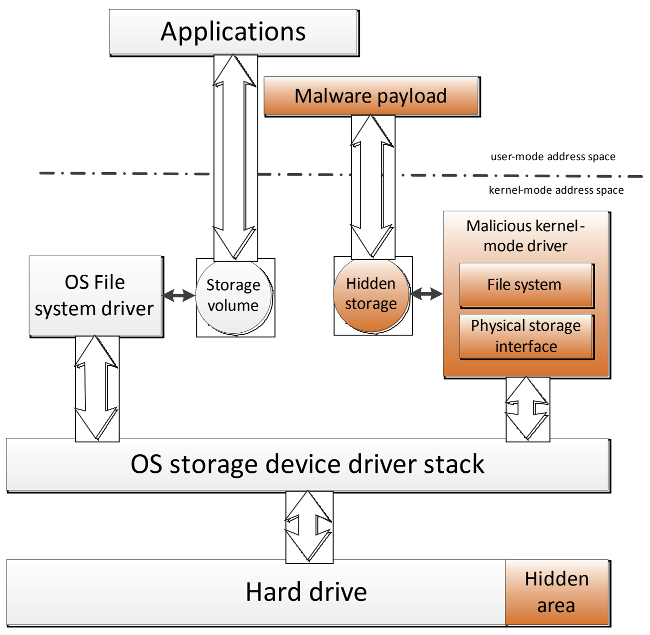


Figure 1: Hidden storage implementation architecture.

To expose the hidden storage to a payload which is executed in user-mode address space, the malicious kernel-mode driver creates a device object representing the hidden volume, through which the data may be read, or written to the hard drive. The payload operating in user-mode address space accesses this device object by its name, which in most cases is randomly generated.

Some of the complex threats considered in this paper protect the area of the hard drive corresponding to the hidden file system from being read or overwritten. In order to do this the malware hooks the IRP_MJ_INTERNAL_DEVICE_CONTROL handler of the lowest driver object in the storage driver stack. This is usually done either by hijacking the pointer to the corresponding driver object (Olmarik) or by overwriting the pointer in the MajorFunction table (Olmasco, Rovnix).

Win64/Olmarik

This family of malware (which is also often referred to as the TDL4 [2, 3, 4, 5] bootkit) is the successor to the notorious rootkit TDL3 [6, 7]. It inherits from its predecessor the ability to store both its payload and its configuration data by stealth. It relies on the hidden storage architecture depicted in Figure 1. In order to do this it reserves some space at the end of the hard drive where it establishes a hidden, encrypted partition in the layout illustrated in Figure 2.

TDL4's hidden file system starts with the root directory which is stored in the first sector (according to the direction in which the file system grows) and has the following layout:

```
typedef struct _TDL4_FS_ROOT_DIRECTORY
{
     // Signature of the block
     // DC - root directory
     WORD Signature;
```

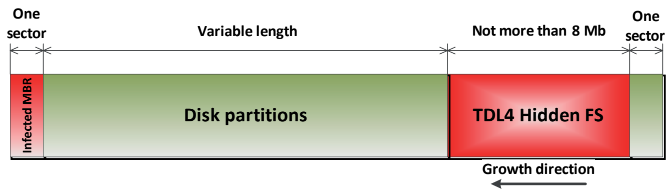


Figure 2: TDL4 hidden file system location.

```
     // Set to zero
     DWORD Reserved;
     // Array of entries corresponding to files in FS
     TDL4_FS_FILE_ENTRY FileTable[15];
}TDL4_FS_ROOT_DIRECTORY, *PTDL4_FS_ROOT_DIRECTORY;
```

Each file listed in the root directory is described by the following structure:

```
typedef struct _TDL4_FS_FILE_ENTRY
{
     // File name - null terminated string
     char FileName[16];
     // Offset from beginning of the file system to file
     DWORD FileBlockOffset;
     // Reserved
     DWORD dwFileSize;
     // Time and Date of file creation
     FILETIME CreateTime;
}TDL4_FS_FILE_ENTRY, *PTDL4_FS_FILE_ENTRY;
```

The data corresponding to files in the hidden file system is stored in sectors with the following layout:

```
typedef struct _TDL4_FS_BLOCK
{
    // Signature of the block
    // DC - root directory
    // FC - block with file data
    // NC - free bock
    WORD Signature;
// Size of data in block
    WORD SizeofDataInBlock;
    // Offset of the next block relative to file system
start
    WORD NextBlockOffset;
    // File table or file data
    BYTE Data[506];
}TDL4_FS_BLOCK, *PTDL4_FS_BLOCK;
```

Figure 3 illustrates which device object is used to access the data stored on the hard drive. The device object with the name '\Device\XXXXXXXX' is used as a volume containing all the files related to the malware. It is linked with the second device object representing hidden storage via the VPB (Volume Parameter Block) system structure.

So as to protect the data from forensic analysis, TDL4 employs transparent encryption. Each sector written to the hidden file system is encrypted with the RC4 cipher. TDL4 uses a four-byte key which is equivalent to the LBA (Logical Block Address) of the sector being written.

TDL4 protects the contents of the hidden file system by hijacking the pointer to the driver object of the lowest device object in the storage device driver stack. As a result, when

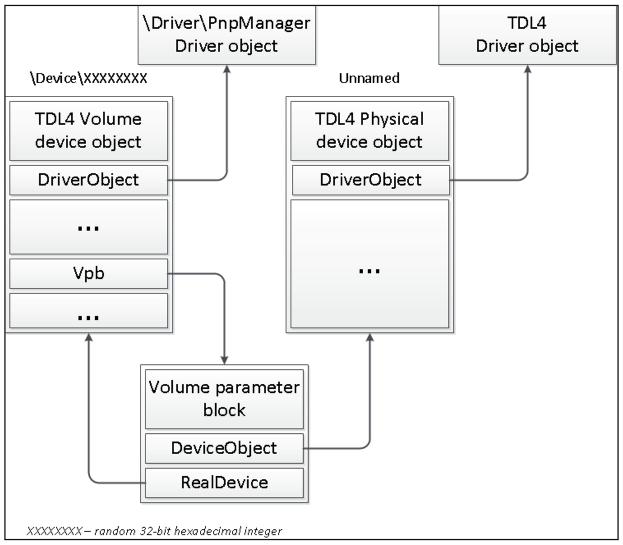

\Driver\PnpManager Driver object
TDL4 Driver object
\Device\XXXXXXXX
Unnamed
TDL4 Volume device object
DriverObject
...
Vpb
...
TDL4 Physical device object
DriverObject
...
Volume parameter block
DeviceObject
RealDevice
XXXXXXXX – random 32-bit hexadecimal integer

Figure 3: TDL4 file system device relationship.

someone/something other than the bootkit attempts to read or write sectors belonging to the hidden file system area, the malware intercepts the I/O request and zeroes the destination buffer or ignores the writing attempt.

Win64/Olmasco

Olmasco [8] is a different example of a complex threat that takes advantage of using customized hidden storage. It operates in a pretty similar way to TDL4 and locates its hidden partition at the end of the hard drive. Compared to TDL4 its file system is more mature and allows the detection of file corruption or unauthorized modification by checking its CRC32 checksum code. Here are the structures describing the Win64/Olmasco file system:

```
typedef struct _OLMASCO_FS_ROOT_DIRECTORY
{
     // Signature of the block
     // DC - root directory
     DWORD Signature;
     // Set to zero
     DWORD Reserved1;
// Set to zero
     DWORD Reserved2;
// Set to zero
     DWORD Reserved3;
// Size of the file system cluster
     DWORD ClusterSize;
// Size of file table in clusters
     DWORD SizeOfSysTableInClusters;
// Size of file table in bytes
     DWORD SizeOfSysTableInBytes;
     // Checksum of file table
DWORD SysTableCRC32;
     // Array of entries corresponding to files in FS
     OLMASCO_FS_FILE_ENTRY FileTable[];
}OLMASCO_FS_ROOT_DIRECTORY, *POLMASCO_FS_ROOT_
DIRECTORY;
```

Each file listed in the root directory is described by the following structure:

```
typedef struct _OLMASCO_FS_FILE_ENTRY
{
     // File name - null terminated string
     char FileName[16];
     // Offset from beginning of the file system to file
     DWORD OffsetInClusters;
     // Size of the file in clusters
     DWORD SizeInClusters;
     // Size of the file in bytes
     DWORD SizeInBytes;
     // Checksum
     DWORD Crc32;
}OLMASCO_FS_FILE_ENTRY, *POLMASCO_FS_FILE_ENTRY;
```

The contents of the file system are protected by the RC4 cipher as well as by the hooking of the IRP_MJ_INTERNAL_DEVICE_CONTROL handler. In contrast to the Win64/Olmarik bootkit, this kind of malware overwrites the pointer to the handler in the MajorFunction table of the corresponding driver object.

Win64/Rovnix/Carberp

The Win64/Rovnix bootkit [4] family uses a VBR (Volume Boot Record) modification technique [9] to infect the system and get itself loaded ahead of the operating system. There are three modifications of the bootkit, one of which was used in the Carberp banking trojan. Table 1 summarizes the differences and similarities between these modifications.

Functionality	**Rovnix.A**	**Carberp with bootkit**	**Rovnix.B**
VBR modification	☑	☑	☑
Polymorphic VBR	☒	☒	☑
Kernel-mode driver encryption algorithm	Custom (ROR + XOR)	Custom (ROR + XOR)	Custom (ROR + XOR)
Hidden file system type	☒	FAT16 modification	FAT16 modification
Hidden file system encryption algorithm	☒	RC6 modification	RC6 modification

Table 1: Rovnix bootkit family comparison.

The payload injected into user-mode processes is stored in the kernel-mode driver binary on the hard drive and is described with a structure of the following type:

```
typedef struct _PAYLOAD_CONFIGURATION_BLOCK
{
      DWORD Signature;         // “JFA\0”
      DWORD PayloadRva;        // RVA of payload
      DWORD PayloadSize;       // Payload start
DWORD NumberOfProcessNames;  // Number of NULL-
terminated strings in ProcessNames
      char ProcessNames[0];   // Array of NULL-
terminated strings describing target
```

```
                          // processes to inject the
payload
}PAYLOAD_CONFIGURATION_BLOCK, *PPAYLOAD_CONFIGURATION_
BLOCK;
```

The latest modification of the bootkit, Win64/Rovnix.B [10], employs a hidden file system to store configuration data and payload. The malware occupies some space either at the beginning or at the end of the hard drive. If there are 0x7D0 (2000 in decimal) free sectors or more before the partition with the lowest starting LBA, then Win64/Rovnix.B locates the hidden partition right after the MBR (Master Boot Record) sector and it extends for 0x7D0 sectors (almost 1MB). If there is not enough space at the beginning of the hard drive the malware tries to locate the hidden partition at its end. The Win64/Rovnix.B bootkit employs a modification of the FAT16 file system as the layout of its hidden partition.

The malware implements on-the-fly encryption with a modification of the RC6 block cipher in ECB (Electronic Code Book) mode and a key length of 128 bits. The key is stored in the last 16 bytes of the very first sector of the hidden partition. Win64/Rovnix.B also hooks the IRP_MJ_INTERNAL_DEVICE_CONTROL handler to protect its hidden file system from being read or modified by other software in a similar way to Win64/Olmasco.

Win32/Sirefef (ZeroAccess)

The ZeroAccess rootkit is also known to be strongly resistant to forensic analysis due to its implementation of a hidden encrypted file system. There are two modifications of ZeroAccess, each of which employs rather a different approach to storing malicious components. Both approaches are described in [11, 12].

The latest modification of malware intended for running on 32-bit systems creates a subdirectory 'C:\windows\system32\$NtUninstallKBXXXXXXXX' where XXXXXXXX is a randomly generated 32-bit integer. The directory created is used to store the rootkit's payload and configuration information. To restrict access to the directory and the files contained within it, the malware creates a symbolic link to the folder and deletes all entries from its ACL (Access Control List). As a result of these manipulations, the folder can still be accessed using its symbolic link name and by the System account which owns the created folder. In addition, Win32/Sirefef implements transparent encryption of all the files kept in hidden storage.

Win32/Hodprot

Win32/Hodprot [13] is a specialized downloader designed to distribute banking trojans in the Russian region. In particular, it was used to distribute one of the most dangerous banking trojans, Win32/Carberp [14, 15]. It was especially designed to resist forensic analysis and to withstand or evade security and anti-virus software. Win32/Hodprot has a complex architecture and consists of several modules including the kernel-mode driver used to inject the payload into user-mode address space.

The payload downloaded from the C&C server is stored in the registry, as is the bot main module and configuration information, but they are not found as files on disk. As a result, the bot's main module, responsible for communicating with C&C servers, never appears as a file anywhere in the OS file system. This makes investigation of cybercrimes committed using this bot quite challenging. Table 2 contains the registry values of the HKLM\SOFTWARE\Settings registry key used by Win32/Hodprot to store both its components and the downloaded payload.

Value name	Description
CoreSettings	The main module of the bot communicating with C&C servers, downloading and executing payload
ErrorControl	Loader code responsible for initializing IAT, relocations etc. of main module during injection by kernel-mode driver
HashSeed	List of C&C URLs

Table 2: Registry key values used by Win32/Hodprot.

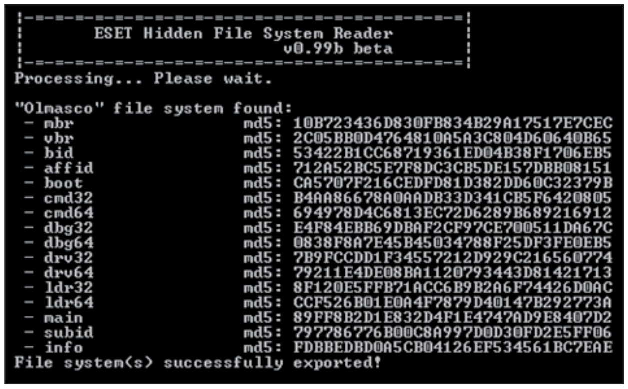
All these registry values are encrypted with a custom encryption algorithm. This consists of sequential XOR-ing and ROR-ing of each byte with the corresponding byte of the key. The key is generated based on the information obtained from the ProductId value of the registry key HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion.

HIDDEN FS READER TOOL

In the course of our research into complex threats, we have developed a tool intended to recover the contents of hidden storage used by such complex threats as:

- Win32/Win64/Olmarik (TDL3/TDL3+/TDL4)
- Win64/Olmasco (MaxSS)
- Win64/Rovnix/Carberp)
- Win32/Sirefef (ZeroAccess).

The tool can be obtained at [16]. Figure 4 presents a screenshot of the tool's output.



```
|-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=|
|       ESET Hidden File System Reader       |
|                       v0.99b beta          |
|-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=|
Processing... Please wait.

"Olmasco" file system found:
 - mbr                  md5: 10B723436D830FB834B29A17517E7CEC
 - vbr                  md5: 2C05BB0D4764810A5A3C804D60640B65
 - bid                  md5: 53422B1CC68719361ED04B38F1706EB5
 - affid                md5: 712A52BC5E7F8DC3CB5DE157DBB08151
 - boot                 md5: CA5707F216CEDFD81D382DD60C32379B
 - cmd32                md5: B4AA86678A0AADB33D341CB5F6420805
 - cmd64                md5: 694978D4C6813EC72D6289B689216912
 - dbg32                md5: E4F84EBB69DBAF2CF97CE700511DA67C
 - dbg64                md5: 0838F8A7E45B45034788F25DF3FE0EB5
 - drv32                md5: 7B9FCCDD1F34557212D929C216560774
 - drv64                md5: 79211E4DE08BA1120793443D81421713
 - ldr32                md5: 8F120E5FFB71ACC6B9B2A6F74426D0AC
 - ldr64                md5: CCF526B01E0A4F7879D40147B292773A
 - main                 md5: 89FF8B2D1E832D4F1E4747AD9E8407D2
 - subid                md5: 797786776B00C8A997D0D30FD2E5FF06
 - info                 md5: FDBBEDBD0A5CB04126EF534561BC7EAE
File system(s) successfully exported!
```

Figure 4: Output of hidden FS reader tool.

CONCLUSION

Nowadays, complex threats employ a range of sophisticated mechanisms to counteract forensic analysis. In this paper we have considered one of the most advanced anti-forensic techniques used by complex threats in the wild – hidden storage. Such malware families as Win64/Olmarik, Win64/Olmasco and so on take advantage of the technique to secretly store their payload and configuration information. The paper contains details of the use of this approach by specific malware. Descriptions of hidden file system layouts have been presented, along with their protection mechanisms. As a countermeasure to the hidden storage technique the authors have developed a tool intended for the retrieval of hidden storage content as built into the most widespread complex threats. This utility is freely available for downloading.

REFERENCES

[1] Storage Miniport Drivers. MSDN Library. http://msdn.microsoft.com/en-us/library/windows/hardware/ff566993(v=vs.85).aspx.

[2] Rodionov, E.; Matrosov A. The Evolution of TDL: Conquering x64. http://go.eset.com/us/resources/white-papers/The_Evolution_of_TDL.pdf.

[3] Rodionov, E.; Matrosov, A.; Harley, D. Bootkit Threat Evolution in 2011. http://blog.eset.com/2012/01/03/bootkit-threat-evolution-in-2011-2.

[4] Matrosov, A.; Rodionov, E.; Harley, D. TDL4 reloaded: Purple Haze all in my brain. http://blog.eset.com/2012/02/02/tdl4-reloaded-purple-haze-all-in-my-brain.

[5] Matrosov, A.; Rodionov, E. Defeating x64: Modern Trends of Kernel-Mode Rootkits. http://go.eset.com/us/resources/white-papers/Ekoparty2011_preso.pdf.

[6] Matrosov, A.; Rodionov, E. Rooting about in TDSS. Virus Bulletin, October 2010. http://go.eset.com/us/resources/white-papers/Rooting-about-in-TDSS.pdf.

[7] Matrosov, A.; Rodionov, E. TDL3: The Rootkit of All Evil? http://go.eset.com/us/resources/white-papers/TDL3-Analysis.pdf.

[8] Matrosov, A.; Rodionov, E.; Harley, D. TDL4 rebooted. http://blog.eset.com/2011/10/18/tdl4-rebooted.

[9] Matrosov, A.; Rodionov, E.; Harley, D. Hasta La Vista, Bootkit: Exploiting the VBR. http://blog.eset.com/2011/08/23/hasta-la-vista-bootkit-exploiting-the-vbr.

[10] Rodionov, E.; Matrosov, A.; Harley, D. Rovnix Reloaded: new step of evolution. http://blog.eset.com/2012/02/22/rovnix-reloaded-new-step-of-evolution.

[11] Wyke, J. The ZeroAccess rootkit under the microscope. http://sophosnews.files.wordpress.com/2012/04/zeroaccess2.pdf.

[12] Giuliani, M. ZeroAccess – an advanced kernel mode rootkit. http://pxnow.prevx.com/content/blog/zeroaccess_analysis.pdf.

[13] Rodionov, E.; Matrosov, A.; Volkov, D. Hodprot: Hot to Bot. http://go.eset.com/us/resources/white-papers/Hodprot-Report.pdf.

[14] Matrosov, A.; Rodionov, E.; Volkov, D.; Harley, D. Win32/Carberp When You're in a Black Hole, Stop Digging. http://go.eset.com/us/resources/white-papers/carberp.pdf.

[15] Matrosov, A.; Rodionov, E.; Volkov, D.; Kropotov, V. Carberp Evolution and BlackHole: Investigation Beyond the Event Horizon. http://www.eset.com/fileadmin/Images/US/Docs/conference_papers/carberp_evolution_and_black-hole.pdf.

[16] Hidden FS reader tool. http://eset.ru/tools/TdlFsReader.exe.