When I'm x64:
Bootkit Threat Evolution in 2011

David Harley, ESET North America
Aleksandr Matrosov, ESET Russia
Eugene Rodionov, ESET Russia

This article was originally published in the 50th issue of Hakin9 Magazine in February 2012, and this pre-print version is made available here by permission of Software Press.

Introduction

It's traditional in security (almost considered compulsory in PR circles) at the end of each year to offer a retrospective view of security-related events in the past 12 months and predictions of likely trends in the threat/anti-threat landscape for the upcoming year. We suspect that by the time this article appears, you'll be sick and tired of crystal balls, but by the end of 2011 we had noted and documented some particularly interesting growth trends in complex threats, especially those targeting the Microsoft Windows 64-bit platform and bootkits in particular (Matrosov).

Figure 1 is a (pretty much self-explanatory) diagram depicting the evolution of bootkit threats over time. The left-hand column represents Proof-of-Concept bootkits that have played an important part in the development of this type of threat but haven't had the same impact "in the wild" as widespread malware like Olmarik (TDL4).

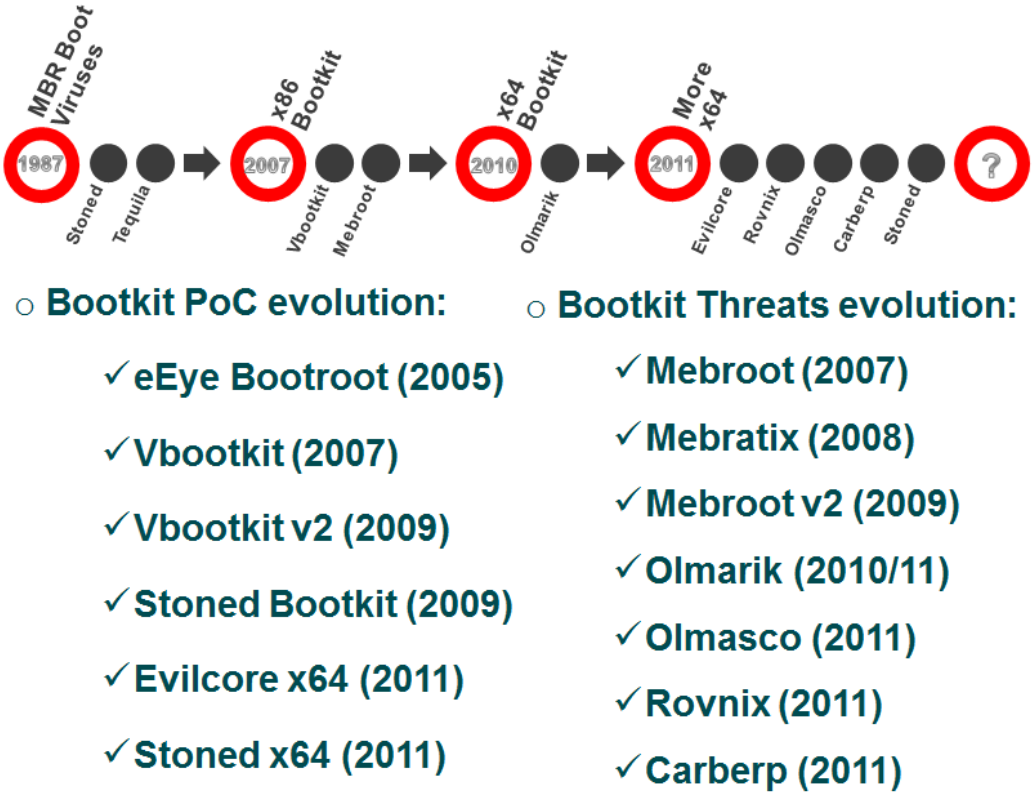

Figure 1: Bootkit threat evolution.
(Modified from TDSS part 1: The x64 Dollar Question)

eEye's Bootroot was an NDIS backdoor that used customized boot sector code to compromise the kernel during loading.

Vbootkit targeted Vista, the security of which was weakened by two inherent assumptions: that there was no possibility that malware could take hold before the Vista loader kicked in, and that once an executable's integrity has been checked on loading, its image in memory will not change before it's actually loaded (Kumar). Vbootkit version 2 extends the hooking of Int 13h and subsequent patching in memory to Windows 7 (Softpedia).

The Stoned bootkit, of course, has no direct connection with 1987's boot sector/partition sector infector Stoned (a.k.a. New Zealand), arguably one of the most successful viruses (in terms of longevity) of all time. Its name is, however, clearly quite deliberate: Stoned Bootkit author Peter Kleissner describes Stoned as "probably the first bootkit?" and used a variation on the famous Stoned message "Your PC is now Stoned" in a BlackHat presentation describing the bootkit (Kleissner 2009). Development of a 64-bit version is described in a subsequent document (Kleissner 2011).

The right-hand column shows bootkits that have had more impact "in the wild". Mebroot has been used by a number of botnets including Torpig (Sinowal) . Mebratix writes itself to the MBR, displacing the original MBR code to another sector (the sector varies according to the variant) in something very close to classic BSI (boot sector infector) style: Brain, the great-grandaddy of PC viruses, did something very similar (Harley, Slade, Gattiker, 2001).

As we've focused quite a lot of research attention on other entries in that column, they get sections to themselves.

TDL4 bootkit

As we predicted in 2010, TDL4 (Win32/Olmarik) has been evolving over 2011. Its developers attempted to bypass the *KB2506014* security update, which addressed a vulnerability allowing abuse of WinPE mode.

TDL4 could be referred to as the first widely spread bootkit targeting 64-bit systems. In order to get control before the OS loader does, it overwrites MBR code while leaving the partition table untouched. This can be seen in figure 2:

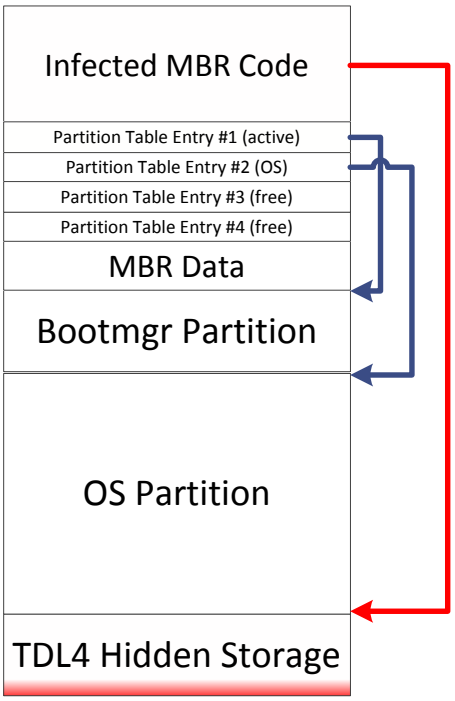

Figure 2: TDL4-infected partition schema

When the malicious boot code receives control it locates TDL4's hidden storage and continues the boot process using the malware's bootkit components.

A particularly striking feature of TDL4 is that it has implemented various techniques to load its kernel-mode driver on x64 systems (64-bit versions of Microsoft Windows Vista and Windows 7) despite their enforced kernel-mode code signing policy and implement kernel-mode hooks even with kernel-mode patch protection policy enabled.

The dropper is unable to load the kernel-mode driver on x64 operating systems it is unable to load the kernel-mode driver, as the driver isn't signed. To get round this limitation the dropper wrote all its components directly to the hard disk, sending IOCTL_SCSI_PASS_THROUGH_DIRECT requests to a disk class driver. Having obtained the disk's parameters, it created an image of its hidden file system in the memory buffer and then wrote it to the hard drive. Subsequently, it modified the MBR so that its malicious components were loaded at boot time. Then it rebooted the system, calling the **ZwRaiseHardError** routine and passing **OptionShutdownSystem** as its fifth parameter**.** This routine resulted in the system's displaying a Blue Screen of Death and rebooting.

In our report The Evolution of TDL4: Conquering x64 we described a method used by the TDL4 bootkit to load its malicious unsigned driver on 64-bit systems. Subsequently, Microsoft released a patch addressing the vulnerability)in x64 OS's (Windows Vista and later) exploited by TDL4, which centred on the way in which the OS checked the integrity of loaded modules.

On unpatched systems there were three BCD (Boot Configuration Data) options that determined how the OS checked integrity of the kernel-mode modules:

On a patched system only two of these are left: BcdLibraryBoolean_DisableIntegrityCheck and BcdLibraryBoolean_AllowPrereleaseSignatures. BcdOSLoaderBoolean_WinPEMode BCD option is no longer used in the initialization of code integrity policy. The routine BlImgQueryCodeIntegrityBootOptions in winload.exe returns the value that determines code integrity policy, and the function was probably added in order to increase the size of the export directory so that the TDL4 bootkit was unable to replace it.

Subsequently, however, an updated version of the TDL4 bootkit worked around this patch by introducing modifications in the ldr16 component in order to reintroduce the ability to successfully infect x64 architecture.

The intention was to modify the ld32 or ldr64 components of kdcom.dll, according to the system targeted. Rather than switching into WinPE mode, this version of TDL4 patched *I_CheckImageHashInCatalog*, a routine used to validate the integrity of the modules being loaded by *winload.exe*.

When the *I_CheckImageHashInCatalog* routine fails to verify the integrity of a module, the value *0xC0000428* (*STATUS_INVALID_IMAGE_HASH*) is returned, preventing the system from booting However, the bootkit patched this routine so as to make it return *0x0000C428* instead of *0xC0000428*. This latest value is not an error code *per se* (error codes in kernel-mode normally have the most significant bit set to 1), so the replacement of kdcom.dll was not detected by the operating system.

Olmasco bootkit

At the beginning of 2011 a brand new bootkit, Win32/Olmasco (also known as MaxSS), appeared in the wild. This new malware family is based on enhanced techniques developed and evolved within the TDL4 family. Unlike TDL4, Win32/Olmasco modifies the partition table of the disk rather than patching MBR code. It looks for an empty entry in the partition table and free space at the end of the hard drive in order to create a new hidden partition containing payload and configuration information. Figure 3 illustrates the way in which partitions are laid out on the infected hard drive:

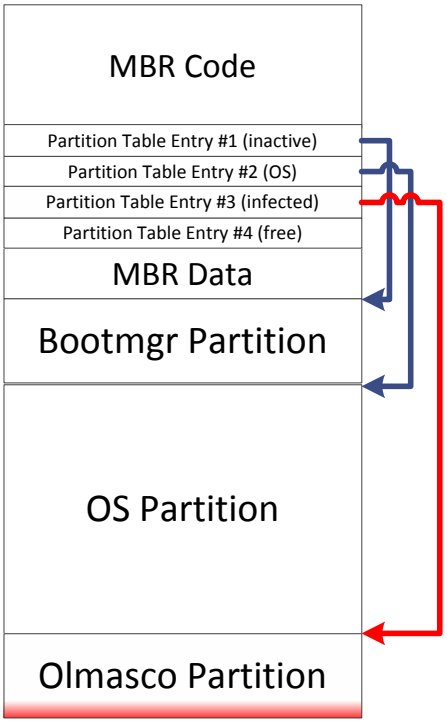

Figure 3: Olmasco-infected partition schema

Here is the beginning of the Win32/Olmasco VBR (Volume Boot Record):

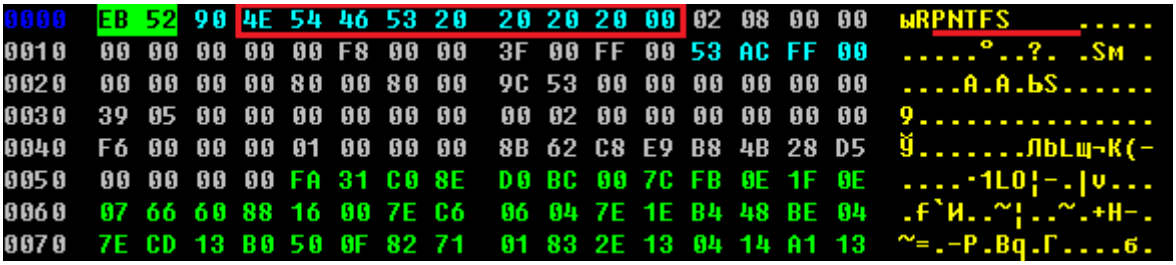

Figure 4: Olmasco VBR

The VBR of the malicious partition mimics the VBR of the legal NTFS partition: this approach makes Win32/Olmasco stealthier and therefore more difficult to detect.

ZeroAccess rootkit

Early in 2011 a new 64-bit ZeroAccess (Win32/Sirefef)modification appeared in the wild. Unlike TDL4 and Win32/Olmasco, Sirefef doesn't implement bootkit functionality. Although there is a version of the malware targeting 64-bit systems, it doesn't contain a kernel-mode driver. The only ZeroAccess version which *does* include a driver targets x86 systems only. For this reason, the machine-infection algorithm is different for 32- and 64-bit Operating System versions.

On x86 systems ZeroAccess behaves like the TDL3 rootkit. It infects a kernel-mode boot start driver by completely overwriting the driver with its own code. As a result, at boot time the system loads the malicious driver, not the original, legitimate code. In order to protect itself against security software and to conceal the fact that the driver was overwritten it sets up low-level hooks in the storage device driver stack.

On a 64-bit OS, however, since there is no kernel-mode 64-bit driver, the malware drops *consrv.dll* library into the "*systemroot**system32*" directory and registers it as part of the Windows Subsystem, which has to be loaded by the Session Manager Subsystem (*smss.exe*) process (trusted system process) during system startup . The list of subsystems which need to be loaded is stored under the *Required* value of the "*HKLM**SYSTEM**CurrentControlSet**Control**Session Manager**SubSystems*" registry key. If one of the components of "required" subsystems is missing the system is rendered unbootable. Thus, removing the threat by deleting *consrv.dll* without applying corresponding changes to the registry key will break the system.

Rovnix and Carberp bootkits

In 2011 a new bootkit – Win32/Rovnix – established a new trend: modification of the VBR and Bootstrap Code (Win32/Rovnix, Win32/Carberp). Using such a technique allowed malware to bypass many security and antivirus programs since the feature makes detecting and removing these threats more difficult.

Interestingly enough, the bootkit builder for VBR bootkits like Win32/Rovnix was offered for sale. For instance, Win32/Carberp – one of the most dangerous banking trojans – was upgraded to include bootkit functionality. Its developers started testing the for-sale bootkit in the end of the summer.

The bootkit component of Carberp is almost identical to that of the Rovnix bootkit, which we discussed in more technical detail in the slide deck "Defeating x64: Modern Trends of Kernel-Mode Rootkits" accompanying a talk given at the Ekoparty 2011 Security conference.

However, the installer had changed significantly. In addition to installing the bootkit it now tried to exploit several vulnerabilities in order to escalate its privileges. This was necessary as Carberp requires administrative privileges in order to install the bootkit. Primarily, Carberp targets corporate users using RBS (Remote Banking Systems) software which often lacks administrative privileges, so that an attack relying purely on social engineering doesn't cut the mustard. The installer exploited the following vulnerabilities in the system software in order to escalate privilege:

- MS10-073: a win32k.sys KeyboardLayout vulnerability in Windows 2000 and XP, originally exploited by Stuxnet. It worked by loading a specially crafted keyboard layout file, making it possible to execute arbitrary code with SYSTEM privileges. Privilege escalation occured while dispatching input from the keyboard using the NtUserSendInput system service in the Win32k.sys module.
- MS10-092: a Task Scheduler vulnerability also exploited by Stuxnet, which actually worked on 64-bit systems as well. It worked because in order to protect the integrity of the job configuration files Task Scheduler calculated a CRC32 checksum, fine for detecting unintentional errors but making it easy to create another message with the same checksum.
- MS11-011 (win32k.sys SystemDefaultEUDCFont vulnerability described at http://support.microsoft.com/kb/2393802).
- .NET Runtime Optimization vulnerability triggered by insecure permissions in the service's .EXE file (seehttp://osvdb.org/show/osvdb/71013).

Carberp and its relationship with the Black Hole bootkit has been discussed at some length in the ESET Threatblog and a white paper.

Rootkit Hidden Storage

We are now seeing that more and more complex threats have started using their own hidden storage and avoid relying on services provided by OS. This approach allows malware to keep its payload and configuration data secret where antivirus and security software is less likely to find it, as well as evading security measures integral to the operating system. It is interesting and instructive to compare (briefly, on this occasion) the hidden file systems of the most sophisticated of these threats: Win32/Olmaco, TDL4 and ZeroAccess.

TDL4 hidden storage

As the successor of TDL3 and TDL3+ this family of malware inherits almost all the features of its predecessors regarding the storage of payload modules. It reserves some space at the end of the hard drive for housing its file system, the contents of which are protected by low-level hooks and an RC4 stream cipher. TDL4 uses the same technique for allocating space on a hard drive for its file system as its predecessor; namely, it starts at the last but one sector of the hard drive and grows towards start of the disk space. However, there are changes in the layout of the file system compared to that used in TDL3. ESET's white paper "Evolution of TDL: Conquering x64". includes in-depth analysis of TDL4. The bootkit protects the contents of its file system by encrypting its blocks. Like TDL3 it uses the RC4 encryption algorithm, a stream cipher with varying key length. However, TDL4 uses as a key the 32-bit integer LBA of the sector block being encrypted.

Win32/Olmasco hidden storage

The Win32/Olmasco developers went even further in the design and implementation details of its hidden file system. Generally, Olmasco's system resembles a schema which is used by TDL4 but with additional enhancements:

- A supporting hierarchy with files and folders;
- Verification of file integrity to check if its components are corrupted
- Better management of internal file system structures.

Unlike the TDL4 hidden file system, which is only capable of storing files, the system implemented in Win32/Olmasco could store both files and directories.

For instance, the VBR of Win32/Olmasco's hidden partition includes code to load a file with "boot" name from the root directory '\':

```
seg000:01EA halt:                                   ; CODE XREF: seg000:0075↑j
seg000:01EA                                         ; sub_B0+35↑j ...
seg000:01EA                 hlt
seg000:01EA sub_191         endp
seg000:01EA
seg000:01EB ; ---------------------------------------------------------------------------
seg000:01EB                 jmp     short halt
seg000:01EB ; ---------------------------------------------------------------------------
seg000:01ED aBoot           db '\boot',0            ; DATA XREF: seg000:008E↑r
seg000:01F3                 db    0
```

Figure 5: Loading the file "\boot"

In addition, upon reading a file from the file system Win32/Olmasco performs some checks in order to detect corruption of file contents. An additional field was introduced into the data structure with CRC32 checksumming of the file contents. If Win32/Olmasco detects that a file is corrupted it removes the corresponding entry from the file system and frees the occupied sectors. This is shown in figure 6:

```
unsigned int __stdcall RkFsLoadFile(FS_DATA_STRUCT *a1, PDEVICE_OBJECT DeviceObject, const char *FileName,
{
  unsigned int result; // eax@1

  result = RkFsLocateFileInDir(&a1->root_dir, FileName, FileEntry);// locate file in the root dir
  if ( (result & 0xC0000000) != 0xC0000000 )
  {
    result = RkFsReadFile(a1, DeviceObject, FileEntry);// read the file from the hard drive
    if ( (result & 0xC0000000) != 0xC0000000 )
    {
      result = RkFsCheckFileCRC32(FileEntry);   // verify file integrity
      if ( result == 0xC000003F )
      {
        MarkBadSectorsAsFree(a1, FileEntry->pFileEntry);// free occupied sectors
        RkFsRemoveFile(a1, &a1->root_dir, FileEntry->pFileEntry->FileName);// remove corresponding entry
        RkFsFreeFileBuffer(FileEntry);
        RkFsStoreFile(a1, DeviceObject, &a1->root_dir);// update directory
        RkFsStoreFile(a1, DeviceObject, &a1->bad_file);
        RkFsStoreFile(a1, DeviceObject, &a1->bitmap_file);// update bitmap of occupied sectors
        RkFsStoreFile(a1, DeviceObject, &a1->root);// update root directory
        result = 0xC000003Fu;
      }
    }
  }
  return result;
}
```

Figure 6: Removing a corrupted file

Since the file system implemented in Win32/Olmasco is more mature than that implemented in TDL4, it therefore requires more efficient management in terms of free space usage and manipulation of data structures. Therefore some special files were introduced to help support file system integrity :

- $bad
- $bitmap

Figure 7 shows the code for Win32/Olmasco's file system initialization routine:

```
unsigned int __stdcall RkFsLoadSystemFiles(FS_DATA_STRUCT *FsData, char *Data)
{
  FS_DATA_STRUCT *_FsData; // esi@1
  unsigned int result; // eax@3
  FS_LIST_ENTRY_STRUCT *v4; // ecx@6
  FS_LIST_ENTRY_STRUCT *FileEntry; // [sp+14h] [bp+8h]@3

  _FsData = FsData;
  if ( FsData && Data )
  {
    FsData->field0 = (int)Data;
    FsData->root.pFileEntry = (FS_FILE_ENTRY_STRUCT *)Data;
    FsData->root.FileBuffer = (int)Data;
    FileEntry = &FsData->bad_file;
    result = RkFsLocateFileInDir(&_FsData->root, "$bad", &_FsData->bad_file);
    if ( (result & 0xC0000000) == 0xC0000000 )
    {
      _FsData->field0 = 0;
      _FsData->root.pFileEntry = 0;
      _FsData->root.FileBuffer = 0;
      return result;
    }
    result = RkFsLocateFileInDir(&_FsData->root, "$bitmap", &_FsData->bitmap_file);
    if ( (result & 0xC0000000) == 0xC0000000 )
    {
      v4 = FileEntry;
LABEL_7:
      _FsData->bad_file.pFileEntry = 0;
      _FsData->root.FileBuffer = 0;
      _FsData->root.pFileEntry = 0;
      _FsData->field0 = 0;
      v4->pDirEntry = 0;
      return result;
    }
    result = RkFsLocateFileInDir(&_FsData->root, "\\", &_FsData->root_dir);
    if ( (result & 0xC0000000) == 0xC0000000 )
    {
      FileEntry->pDirEntry = 0;
      v4 = &_FsData->bitmap_file;
      _FsData->bitmap_file.pFileEntry = 0;
      goto LABEL_7;
    }
    result = 0;
  }
  else
  {
    result = 0xC000000Du;
  }
  return result;
}
```

Figure 7: Olmasco file system initialization

Both of these files are at the same hierarchical level in the root directory and are not accessible for payload (i.e., they're for system use only). The purpose of these files is to store information about unused space and sectors containing corrupted data (just as files with the same names are used in NTFS).

Another reason for introducing these files is to make the file system resemble NTFS more closely, so as to confuse security software.

ZeroAccess hidden storage

Yet another malware family that implements its own hidden storage is the ZeroAccess rootkit. Like TDL4 and Win32/Olmasco ZeroAccess stores its payload and configuration information in a custom

hidden file system. In this case, however, the contents of the hidden storage are saved into a file in the OS file system rather than directly into hard drive sectors.

For this reason, ZeroAccess creates the directory "*WindowsDir**$NtUninstallKB_BotId$**BotId*" into which payload and configuration information is stored. In this case, the malware behaves like a file system filter driver: it redirects payload read/write operations to corresponding files in the system, as well as performing transparent file encryption.

Eugene Rodionov, Malware Researcher
Aleksandr Matrosov, Senior Malware Researcher
David Harley, Senior Research Fellow

References:

http://www.slideshare.net/matrosov/defeating-x64-modern-trends-of-kernelmode-rootkits

http://resources.infosecinstitute.com/tdss4-part-1/

http://www.eeye.com/eEyeDigitalSecurity/media/ResearchPapers/eEyeDigitalSecurity_Pixie-Presentation.pdf?ext=.pdf

http://www.blackhat.com/presentations/bh-europe-07/Kumar/Presentation/bh-eu-07-kumar-apr19.pdf

http://news.softpedia.com/news/Windows-7-Vbootkit-2-0-Attack-Tool-Goes-Open-Source-111063.shtml

http://www.blackhat.com/presentations/bh-usa-09/KLEISSNER/BHUSA09-Kleissner-StonedBootkit-SLIDES.pdf

http://stoned-vienna.com/downloads/The%20Art%20of%20Bootkit%20Development.pdf

http://www.cs.ucsb.edu/~seclab/projects/torpig/index.html

Viruses Revealed: David Harley, Robert Slade, Urs Gattiker; Osborne 2001

http://go.eset.com/us/resources/white-papers/The_Evolution_of_TDL.pdf

http://blog.eset.com/2011/04/15/kb2506014-kills-tdl4-on-x64

http://blog.eset.com/2011/05/10/the-co-evolution-of-tdl4-to-bypass-the-windows-os-loader-patch-kb2506014

http://blog.eset.com/2011/10/18/tdl4-rebootedhttp://www.eset.eu/encyclopaedia/win32-sirefef-a-trojan-dropper-pmax-a-horse-trojandropper

http://go.eset.com/us/resources/white-papers/Stuxnet_Under_the_Microscope.pdf

http://blog.eset.com/2010/10/15/win32k-sys-about-the-patched-stuxnet-exploit

http://go.eset.com/us/resources/white-papers/carberp.pdf

Resources

TDSS part 1: The x64 Dollar Question
By Aleksandr Matrosov, Eugene Rodionov & David Harley, April 2011
Considers and contrasts the distribution and installation of the TDL3 and TDL4 bootkits.

TDSS part 2: Ifs and Bots
By Aleksandr Matrosov, Eugene Rodionov & David Harley, April 2011
Looks in more depth at the internals of the TDSS malware.

TDSS part 3: Bootkit on the other foot
By Aleksandr Matrosov, Eugene Rodionov & David Harley, April 2011
The last part of the series describes the TDSS loading process.

Rooting about in TDSS
By Aleksandr Matrosov & Eugene Rodionov, October 2010
This article for Virus Bulletin describes a utility for dumping the TDSS rootkit's file system.
Originally published in Virus Bulletin, October 2010.*

Win32/Carberp: When You're in a Black Hole, Stop Digging
By Aleksandr Matrosov, Eugene Rodionov, Dmitry Volkov and David Harley, December 2011
This paper consolidates information published by ESET and Group-IB researchers on Russian malware that attacks Russian RBS (Remote Banking Systems) transactions: now updated to version 1.1 to include additional material.

Modern Bootkit Trends: Bypassing Kernel-Mode Signing Policy
By Aleksandr Matrosov and Eugene Rodionov, October 2011
This presentation continues the authors' consideration of modern bootkit techniques for evading kernel mode code signing policy as applied to currently In-the-Wild malware.

Defeating x64: The Evolution of the TDL Rootkit
By Aleksandr Matrosov and Eugene Rodionov, May 2011
A presentation for Confidence 2011, held in May 2011 in Krakow, on the analysis and implications of the latest generation of the TDL rootkit (TDL4).