Newcomers in the Derusbi family

By Fabien Perigaud on 2015/12/15, 13:30 - Reverse engineering - Permalink

Derusbi is a well-known RAT family, used in various APT attacks since at least 2008. Many papers (1,2,3) have described two known variants of this malware: a client version, acting as any other RAT by contacting its C&C server, as well as a server version, which just listens for incoming connections from a client.

This RAT seems to be continuously evolving, as enlightened by Sekoia which recently described a new way for Derusbi to bypass Windows drivers signature enforcement.

In this blog post, we'll present the analysis of two new variants we encountered: a **driver for x64 Windows**, and a **Linux** library.

Windows x64 driver

Installation

We found various samples of this driver, all signed by legitimate, stolen certificates. One has been revoked, another one is expired, and the last one is still perfectly valid. However, we didn't find any installer delivering the driver, which might mean it is installed manually, for example along with the installation of the HD Root bootkit, as multiple evidences link to Derusbi.

According to the samples we found, the driver filename is either wd.sys or udfs.sys.

The following malicious certificates are used to sign the drivers:

Name	Serial Number	Status
Fuqing Dawu Technology Co.,Ltd.	**4c0b2e9d2ef909d15270d4dd7fa5a4a5**	**Revoked**
XL Games Co.,Ltd.	**7bd55818c5971b63dc45cf57cbeb950b**	**Expired**
Wemade Entertainment co.Ltd	**476bf24a4b1e9f4bc2a61b152115e1fe**	**Valid**

Rootkit & Evasion capabilities

In the 4 different samples we found, only one was using VMProtect to obfuscate its imports. The other ones were not packed.

During the initialization phase, the driver tries to disable the kernel debugger by calling **nt!KdDisableDebugger** to prevent dynamic analysis.

As for the previous client versions (also using a kernel driver for rootkit features), the drivers filters both the filesystem and the network to hide the RAT activities..

Network rootkit

Depending on the Windows versions, hooks are performed either on **\\Device\Tcp** or **\\Driver\nsiproxy**. Whenever a specific IOCTL is performed on the device (0x120003 or 0x12001b respectively), the hook function hides network connections using **ports between 1025 and 1777**.

Filesystem rootkit

The filesystem rootkit is installed by hooking the **IRP_MJ_DIRECTORY_CONTROL** major function of **\\Filesystem\Ntfs**, and hides the presence of the driver file (which is grabbed by reading the service *ImagePath* registry value).

Embedded DLL loading

To perform all its RAT capabilities, the driver relies on a userland component, which is dynamically injected in memory from the kernel. This ensures a better stealthiness, as the userland component is never written to the disk.

This userland component is a DLL, located in the *.data* section and ciphered using the classical Derusbi ciphering (which simply consists in a 4-bytes XOR).

The DLL is injected in one of the svchost.exe processes running as *SYSTEM* or *LocalSystem*, using the following steps:

- Allocation and copy of 2 shellcodes in the process memory
- Allocation and copy of an array containing:
  - Pointers to LoadLibraryA / GetProcAddress
  - Name of the DLL export to call (“Func” in our samples)
  - Content of the DLL

Search

OK

Tags

- log
- anonymization
- welcome
- GoodFET
- Z-Wave
- side-channels
- critical infrastructures
- Hardware Trojans
- Sakula
- BlackVine
- advanced persistent threat
- ransomware
- reconnaissance
- bitcrypt
- APT kill chain
- PlugX
- Advanced Persistent Threat
- reverse engineering
- malware
- APT
- All tags

Useful Links

- Cybersecurity Job offers
- AIRBUS D&S CyberSecurity
- BitBucket CyberTools

Share

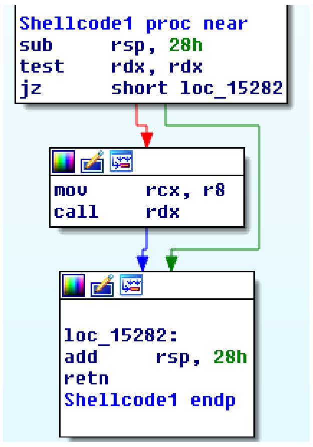
0

Subscribe

Entries feed

- Usage of **nt!KeInitializeApc** and **nt!KeInsertQueueApc** to make the process execute the first shellcode, with address of the second shellcode and of the array as arguments

First shellcode is simply:



```
Shellcode1 proc near
sub     rsp, 28h
test    rdx, rdx
jz      short loc_15282

mov     rcx, r8
call    rdx

loc_15282:
add     rsp, 28h
retn
Shellcode1 endp
```

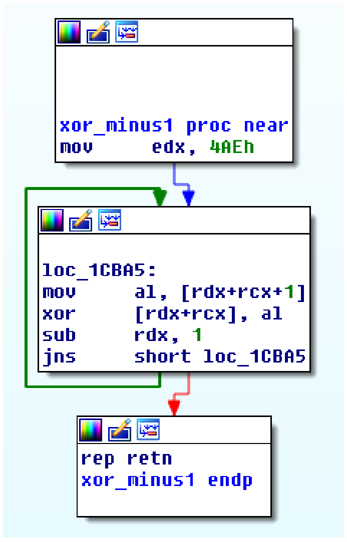
Second shellcode is a custom DLL loader, which directly loads the DLL in memory without writing it to the disk. The export is then called, with a randomly generated integer as parameter.

The randomly generated integer is used to build a named pipe called **\\.\pipe\usbpcex%d**. This pipe is then used for the communication between the driver and the userland DLL.

Once loaded, the DLL writes the machine IP address to the registry key **HKLM\SYSTEM\CurrentControlSet\Control\WMI\Ipstatus**.

Configuration

The configuration is stored just after a string made of 15 "X". It is obfuscated using a simple XOR loop:



```
xor_minus1 proc near
mov     edx, 4AEh

loc_1CBA5:
mov     al, [rdx+rcx+1]
xor     [rdx+rcx], al
sub     rdx, 1
jns     short loc_1CBA5

rep retn
xor_minus1 endp
```

The configuration has the following structure:

- Offset 0x44: Timer
- Offset 0x48-0x50: Two hour values, only authorize communication between these hours
- Offset 0x50: URL for additional C&C retrieval
- Offset 0x150: 8 C&C structures, 0x6c bytes each

The C&C structure is:

- Offset 0x0: protocol
- Offset 0x4: IP/domain address
- Offset 0x24: port

The rest of the configuration is filled with garbage.

```
00000000: 0100 0000 0048 c75f 409d 51b9 d2c8 f30c  .....H._@.Q.....
00000010: 566a 1e47 12b4 3735 638d b549 fd06 76aa  Vj.G..75c..I..v.
00000020: 6200 617f a41c 427c 5ee4 e2ad f137 6682  b.a...B|^....7f.
00000030: fcfa ea1c 1efd 5ed7 ef2b a018 3cf6 da58  ......^..+..<..X
00000040: 8301 1f71 2100 0000 0000 0000 0000 0000  ...q!...........
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
*
00000150: 2000 0000 7761 7265 2e6e 6963 6574 7279   ...ware.XXXXXXX
00000160: 2e62 697a 007c e4ee 3c09 a2a8 d427 f51e  .biz.|..<....'..
00000170: 504e 9d65 bb01 0000 0025 e96e 4d1a 9eb6  PN.e.....%.nM...
00000180: 58b9 c133 c719 8c5e 3359 d429 9911 6f0a  X..3...^3Y.)..o.
00000190: d9f9 c548 6ed8 b485 0000 0000 0028 fd5e  ...Hn........(.^
000001a0: b139 8e77 771d 9f66 6b75 f18c 00b9 2133  .9.ww..fku....!3
000001b0: 00e5 45d4 d062 c2a2 e532 715a 2000 0000  ..E..b...2qZ ...
000001c0: 7761 7265 2e6e 6963 6574 7279 2e62 697a  ware.XXXXXXX.biz
000001d0: 006a 55a0 6f17 cffa 9003 766f 3e9a 34a1  .jU.o.....vo>.4.
000001e0: 5000 0000 00be e79c 7e6d c95b 77a4 139f  P.......~m.[w...
000001f0: 0be1 2e45 c4d1 94a5 677c f161 2baf 7b6e  ...E....g|.a+.{n
00000200: 3caa 8e60 0000 0000 00fa f5f6 2b96 c211  <..`........+...
00000210: dea0 065a 4766 c8bd 00f3 1bcb 8575 7fcc  ...ZGf.......u..
```

Network Communications

The malware configuration can embed up to 8 C&C addresses. A configuration update mechanism is also available, by requesting the URL in the configuration. The resulting web page is then parsed, looking for tags **$$$--Hello** and **Wrod--$$$** surrounding a base64 string, which is the new encoded configuration blob.

Up to 3 different protocols are supported:

- Raw TCP
- Raw UDP
- HTTP

Once a C&C has been reached, its information is stored in the following registry key, xored with 0x51: **HKLM\SYSTEM\CurrentControlSet\Control\WMI\Level10**.

The DNS Server to use for DNS requests is stored in value **Level01**, xored with 0x51. Source port for the DNS requests is randomly chosen between 1025 and 1777.

Proxy settings are stored in registry values **Level02** to **Level05**, xored with 0x51.

DNS server IP address and proxy settings are retrieved by the userland DLL, by setting up a raw socket to sniff all outcoming traffic and parsing:

- DNS requests
- HTTP requests to look for a *Proxy-Authorization: Basic* header and the proxy address

These two settings are then written to the registry.

All network communications are performed by the kernel driver. Each received packet is decoded and sent to the userland in the pipe, and each data received through the pipe is encoded and sent back on the network.

Packets have a 0x1c bytes clear header, followed by encrypted/compressed data. The header has the following structure:

```
struct packetHeader {
    DWORD random;
    DWORD moduleID;
    DWORD rawPacketSize;
    DWORD checkSum;
    DWORD xorKey;
    DWORD bCompressed;
    DWORD uncompressedSize;
};
```

The ciphering is a simple XOR with the 4-bytes key, packet might be compressed with LZO and checkSum is a CRC32.

Userland component

The userland component is in charge of the network communications decoding and commands interpretation. It receives, from the kernel module, the moduleID and the deciphered/uncompressed data from the network packet.

Modules

This part of the malware seems modular, and each module is associated to a module ID. Previous papers described some of those modules, but our samples contained two new modules. Here is the list of all the modules compiled in the DLL:

Module ID	Class Name	Description
0x80	**PCC_SYS**	**Various commands related to processes and services**

Module ID	Class Name	Description
0x81	**PCC_CMD**	**Execute commands**
0x82	**PCC_PROXY**	**Network proxy**
0x83	**PCC_GUI**	**Remote Desktop**
0x84	**PCC_FILE**	**Files manipulation**
0xC0	**PCC_VPN**	**VPN feature**
0xF0	**-**	**Uninstall, Disconnect, GetLastError, etc.**

The two new modules are **PCC_GUI** and **PCC_VPN**.

Embedded PE files

The userland DLL embeds 5 PE files, all stored in *.data* section and ciphered with a 4-bytes XOR key.

Embedded PE #1

This one is a 64-bits DLL (export "Func") for the **PCC_GUI** module. It is injected in "winlogon.exe" directly in memory, with the custom DLL loader. Communication with the main DLL is performed through a named pipe called **\\.\pipe\usbpcg%d**.

Embedded PE #2

This is a 64-bits DLL (export "R32") used for commands execution (**PCC_CMD** module). It is dropped in **%Systemroot%\web\safemedo.html**, and executed with rundll32.exe.

Communication with the main DLL is performed through named pipes **\\.\pipe\usb%si** and **\\.\pipe\usb%so**, where “%s” is GetTickCount output in decimal.

Process is created with its IO redirected to the pipes. It can be run under a logged-in user, or by specifying user credentials.

Embedded PE #3

This is a 64-bits DLL (export "Func") for keylogging capabilities via the *GetAsyncKeyState* API. It is directly injected in "explorer.exe" process, and a mutex *Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.3790.4770-ww_05FDF087* is created.

Data is saved to *%tmp%\Ziptmp$$__.1*.

Embedded PE #4

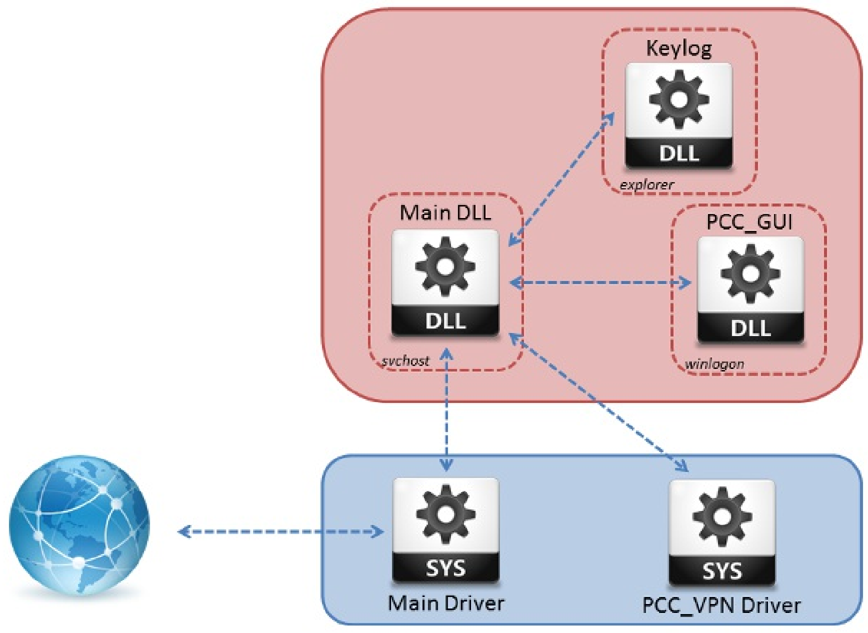
This one is another 64-bits DLL (export "Func") for keylogging capabilities, this time via the *SetWindowsHook* API. It is dropped in **%Systemroot%\web\safemode.html**, executed with rundll32.exe, and a mutex *Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.3790.4770-x_05FDF087* is created.

Data is saved to *%tmp%\Ziptmp$$__.2*.

Embedded PE #5

This is a 64-bits NDIS driver for the VPN feature. It is dropped in **%SystemRoot%\Drivers\{1D24B7E2-869D-49D8-B4EB-1424B36C42B6}.sys**, and is signed with the XL Games Co.,Ltd. (7bd55818c5971b63dc45cf57cbeb950b) certificate.

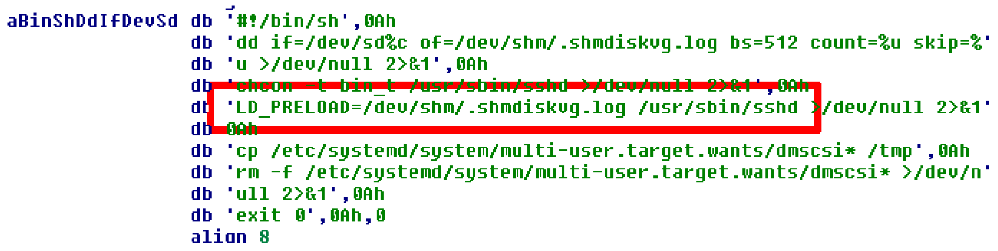
Overall Architecture



The architecture of this new Derusbi variant is distributed amongst various drivers and processes, each one being responsible of a specific task. This avoids having a single process performing all the malicious tasks, and prevents security software from raising alerts.

Linux Library

We recently noticed that the group behind HD Root bootkit was interested in compromising Linux systems as well. A very interesting Chinese article describes the features of the Linux version of the bookit. Focusing on the payload loading part, we can see that a userland library is loaded into a */usr/bin/sshd* process using *LD_PRELOAD*.



```
aBinShDdIfDevSd db '#!/bin/sh',0Ah
                db 'dd if=/dev/sd%c of=/dev/shm/.shmdiskvg.log bs=512 count=%u skip=%'
                db 'u >/dev/null 2>&1',0Ah
                db 'chcon -t bin_t /usr/sbin/sshd >/dev/null 2>&1',0Ah
                db 'LD_PRELOAD=/dev/shm/.shmdiskvg.log /usr/sbin/sshd >/dev/null 2>&1'
                db 0Ah
                db 'cp /etc/systemd/system/multi-user.target.wants/dmscsi* /tmp',0Ah
                db 'rm -f /etc/systemd/system/multi-user.target.wants/dmscsi* >/dev/n'
                db 'ull 2>&1',0Ah
                db 'exit 0',0Ah,0
                align 8
```

Interestingly, we discovered a Linux library exporting all the classical PAM exports, with an added *CTOR* leading to a **PccMain()** function exposing the Derusbi server behaviour. This library could be the perfect payload for the Linux HD Root, comforting us in the assumption of a strong link between the bootkit and the RAT.

The sample we found included all debug symbols, helping the understanding and the naming of each feature.

Lock

When started, the malware creates a lock in **/dev/shm/.shmfs.lock**.

Network Communications

The sample listens on port **40101** for incoming connections. The communication protocol is the same than the Windows version. All the network I/O are handled by the **BD_SOCK** and **PCC_SOCK** classes.

Modules

As for the Windows versions, various modules are embedded in the library, each one dedicated to a specific task:

Module ID	Class Name	Description
0x80	PCC_SYS	Various commands related to processes and services
0x81	PCC_CMD	Execute commands
0x82	PCC_PROXY	Network proxy
0x84	PCC_FILE	Files manipulation
0xF0	-	Uninstall, Deconnect, GetLastError, etc.

The **PCC_CMD** module spawns a new process, which argv[0] is replaced by **[diskio]**. The following *PS1* variable is set:

```
RK# \u@\h:\w \$
```

Development framework

Regarding the debugging information still present in the library, it seems that the authors reused code from the Windows version of their RAT, and implemented wrappers for the original Windows API. Source file name is *Win32APIWarp.cpp*. We found various functions corresponding to Windows API, such as:

- CreateThread(void *, unsigned int, void *(__cdecl *start_routine)(void *), void *, unsigned int, unsigned int *)
- GetFileAttributesW(wchar_t *)
- GetTickCount(void)
- WSAGetLastError(void)

Dropped kernel module

A kernel module is embedded in the *.data* section, ciphered using the classical 4-bytes XOR algorithm. Its goal is to perform rootkit features, by accepting all packets to port 40101 and hiding network communications corresponding to the RAT.

Technically, it sets hooks in the netfilter stack by using the *nf_register_hook* function, and accept all packets matching the RAT communication. For the hiding part, a hook is set on special file **/proc/net/tcp** which hides connections using ports between 40101 and 40500.

The driver is first patched to insert proper *.modinfo* and dropped in **/tmp/.secure**. It is then loaded by simply calling *insmod*. The correct version information inserted in *.modinfo* is retrieved by trying to read various hardcoded kernel modules which should be present on Linux machines. After loading, the module file is deleted.

Conclusion

Derusbi seems to be a trendy malware, evolving on several fields. The interest of APT actors in the Linux systems should encourage the community to improve and develop forensics and malware investigation tools for these platforms.

Annex

Hashes

- 1b449121300b0188ff9f6a8c399fb818d0cf53fd36cf012e6908a2665a27f016
- 50174311e524b97ea5cb4f3ea571dd477d1f0eee06cd3ed73af39a15f3e6484a
- 6cdb65dbfb2c236b6d149fd9836cb484d0608ea082cf5bd88edde31ad11a0d58
- e27fb16dce7fff714f4b05f2cef53e1919a34d7ec0e595f2eaa155861a213e59
- 75c3b22899e39333c0313e80c4e6958d6612381c535d70b691f5f42afc8c214f

Yara rules

```
rule derusbi_kernel
{
    meta:
        description = "Derusbi Driver version"
        date = "2015-12-09"
        author = "Airbus Defence and Space Cybersecurity CSIRT - Fabien Peri
    strings:
 $token1 = "$$$--Hello"
 $token2 = "Wrod--$$$"
 $cfg = "XXXXXXXXXXXXXXX"
 $class = ".?AVPCC_BASEMOD@@"
 $MZ = "MZ"

    condition:
        $MZ at 0 and $token1 and $token2 and $cfg and $class
}

rule derusbi_linux
{
    meta:
        description = "Derusbi Server Linux version"
        date = "2015-12-09"
        author = "Airbus Defence and Space Cybersecurity CSIRT - Fabien Peri
    strings:
 $PS1 = "PS1=RK# \\u@\\h:\\w \\$"
 $cmd = "unset LS_OPTIONS;uname -a"
 $pname = "[diskio]"
 $rkfile = "/tmp/.secure"
 $ELF = "\x7fELF"

    condition:
        $ELF at 0 and $PS1 and $cmd and $pname and $rkfile
}
```