MALWARE ANALYSIS 1

THE CURSE OF NECURS, PART 2

Peter Ferrie
Microsoft, USA

In the first part of this series on the Necurs rootkit [1], we looked at what it does during start-up and when it is not loaded as a boot-time driver. This time, we will look at what Necurs does when it is loaded as a boot-time driver.

BOOT-TIME DRIVER

When Necurs is loaded as a boot-time driver, it remains resident in memory (unlike when it is loaded as a standard driver). It sets every entry in its IRP table to point to a single routine (described below). It attempts to create a new '\Device\NtSecureSys' device and a '\??\NtSecureSys' symbolic link to the device. The symbolic link allows the user-mode component to communicate with the kernel-mode component, and to send I/O control requests to it.

LOW-FLYING CODE

The rootkit attempts to retrieve the address of the ObRegisterCallbacks() function. This API was introduced in *Windows Vista*. If the rootkit is running on a platform that supports the API, then it registers callbacks for process and thread objects, intending to intercept process and thread creation events before they occur. The rootkit registers itself using an altitude of '20101'. The altitude describes how low in the stack the callback should be placed. The rootkit uses a value in the reserved region of 'FSFilter System', corresponding to a level that is even lower than the lowest documented level.

If the rootkit is running on a platform that does not support the ObRegisterCallbacks() API, then it queries the build number of the currently running version of *Windows*. The rootkit is specifically interested in builds 2600 (*Windows XP*), 3790 (*Windows 2003*) and 6000 (*Windows Vista SP0*). The rootkit uses the build number to determine the function indexes that correspond to the NtOpenProcess() and NtOpenThread() functions in the Service Descriptor Table. The rootkit allocates memory for the entire service table, then maps and locks the pages so that they can be read without issue. It saves the pointers to the original NtOpenProcess() and NtOpenThread() functions, and replaces them with rootkit-specific versions.

DATABASE FILES

The rootkit attempts to access the 'DB1' registry value under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\<random numbers>' key that it created previously [1]. If the value doesn't exist, the rootkit creates it later. If the value does exist, the rootkit requires the data – an array of zero-terminated Unicode strings – to be at least four bytes long and even in length. The rootkit uses this array when determining whether a registry access request should be allowed.

The rootkit registers a callback for registry operations, but does so using the CmRegisterCallback() function, which is documented as being obsolete for *Windows Vista* and later. It adds the current thread handle to a thread array that it carries, and sets the reference count to one. The array is used for access control for the rootkit functionality. Any thread handles which appear in the array are allowed to request that the rootkit performs certain actions or queries certain information.

The rootkit creates a file system filter device for the device that hosts the rootkit file, and attempts to attach the filter to the top of the file system stack so that it is the first device to receive all requests. If that request fails (which can occur if the subsystem has not yet been initialized), the rootkit creates a thread that runs once every 100ms to attempt to register the device. The thread runs until it succeeds.

The rootkit attempts to access the 'DB0' registry value under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\<random numbers>' key. If the value doesn't exist, the rootkit creates it later. If the value does exist, the rootkit requires the data to be a multiple of 16 bytes in length. The data is an array of MD5 hash values that form a whitelist of MD5 hashes of memory images. The rootkit uses this array when determining whether an already-loaded driver should be allowed to remain loaded.

The rootkit attempts to access the 'DB2' registry value under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\<random numbers>' key. If the value doesn't exist, the rootkit creates it later. If the value does exist, then the rootkit requires the data – an array of FNV-1 hash values that form a whitelist of driver names – to be a multiple of eight bytes in length. The rootkit uses this array when determining whether a driver should be allowed to load.

The rootkit requests the list of loaded modules, then examines each entry in the list. It is interested in two key entries: win32k.sys and itself. The rootkit also pays attention to the order in which they have been loaded. If the 'win32k.sys' module is in the list, the rootkit sets a flag which is checked later. If the rootkit module is seen, then the blacklist and whitelist behaviour is enabled, if the 'DB0' and 'DB2' registry values exist.

BLACKLIST

If the blacklist behaviour is enabled, the rootkit performs a case-insensitive comparison of the module name with each entry in the following list (sorted for easier reading – the original unsorted list was likely created by adding the names as they were found):

a2acc.sys
a2acc64.sys
a2gffi64.sys
a2gffx64.sys
a2gffx86.sys
ahnflt2k.sys
AhnRec2k.sys
AhnRghLh.sys
amfsm.sys
amm6460.sys
amm8660.sys
AntiLeakFilter.sys
antispyfilter.sys
AntiyFW.sys
ArfMonNt.sys
AshAvScan.sys
aswmonflt.sys
AszFltNt.sys
ATamptNt.sys
AVC3.SYS
AVCKF.SYS
avgmfi64.sys
avgmfrs.sys
avgmfx64.sys
avgmfx86.sys
avgntflt.sys
avmf.sys
BdFileSpy.sys
bdfm.sys
bdfsfltr.sys
caavFltr.sys
catflt.sys
cmdguard.sys
csaav.sys
cwdriver.sys
drivesentryfilterdriver2lite.sys
dwprot.sys
eamonm.sys
eeCtrl.sys
eeyehv.sys
eeyehv64.sys
eraser.sys
EstRkmon.sys
EstRkr.sys
fildds.sys
fortimon2.sys
fortirmon.sys
fortishield.sys
fpav_rtp.sys
fsfilter.sys
fsgk.sys
ggc.sys
HookCentre.sys
HookSys.sys
ikfilesec.sys
ino_fltr.sys
issfltr.sys
issregistry.sys
K7Sentry.sys
klbg.sys
kldback.sys
kldlinf.sys
kldtool.sys
klif.sys
kmkuflt.sys
KmxAgent.sys
KmxAMRT.sys
KmxAMVet.sys
KmxStart.sys
kprocesshacker.sys
lbd.sys
MaxProtector.sys
mbam.sys
mfehidk.sys
mfencoas.sys
MiniIcpt.sys
mpFilter.sys
NanoAVMF.sys
NovaShield.sys
nprosec.sys
nregsec.sys
nvcmflt.sys
NxFsMon.sys
OADevice.sys
OMFltLh.sys
PCTCore.sys
PCTCore64.sys
pervac.sys
PktIcpt.sys
PLGFltr.sys
PSINFILE.SYS
PSINPROC.SYS
pwipf6.sys
PZDrvXP.sys
Rtw.sys
rvsmon.sys
sascan.sys
savant.sys
savonaccess.sys
SCFltr.sys
SDActMon.sys
SegF.sys
shldflt.sys
SMDrvNt.sys
snscore.sys
Spiderg3.sys
SRTSP.sys
SRTSP64.SYS
SRTSPIT.sys
ssfmonm.sys
ssvhook.sys
STKrnl64.sys
strapvista.sys
strapvista64.sys
THFilter.sys
tkfsavxp.sys
tkfsavxp64.sys
tkfsft.sys
tkfsft64.sys
tmevtmgr.sys
tmpreflt.sys
UFDFilter.sys
v3engine.sys
V3Flt2k.sys
V3Flu2k.sys
V3Ift2k.sys
V3IftmNt.sys
V3MifiNt.sys
Vba32dNT.sys
vcdriv.sys
vchle.sys
vcMFilter.sys
vcreg.sys
vradfil2.sys
ZxFsFilt.sys

If a match is found, the rootkit writes some code at the module's entrypoint, which causes it to return immediately with a STATUS_UNSUCCESSFUL result, in turn causing the driver to be unloaded by *Windows*, if the code is executed. It does not stop the driver from running if it was already active. If the module's name is not on the blacklist, then the rootkit will check the flags field for the undocumented 'VP' device status. If the flag is set, then the rootkit always allows it. Otherwise, it checks the whitelist.

WHITELIST

The check for a whitelist entry is complicated. It begins with the rootkit allocating a block of memory that is equal in size to the module being checked. The entire contents of the module are then copied to the block of memory, and the copied image is relocated as though it were loaded to a fixed base of 0x10000. The rootkit supports two kinds of relocation items: IMAGE_REL_BASED_HIGHLOW and IMAGE_REL_BASED_DIR64. The imports table is parsed, but all entries are zeroed out. The rootkit calculates the MD5 hash of the headers and each of the sections, and then searches for a match in the MD5 whitelist.

There is a vulnerability in the way in which the rootkit calculates the hash of the sections, which means that a knowledgeable person could alter an allowed driver in such a way that the original MD5 hash would be retained, but entirely different code could be executed. This technique could be used to bypass the protections of the rootkit and then uninstall it. However, we will not go into the details here.

If the MD5 hash matches one of the entries in the MD5 whitelist, the rootkit allows the driver to remain in memory. Otherwise, it performs the same code alteration as for the blacklisted drivers. This creates a race condition whereby a just-loaded driver might be caught by the code change and then exit, but a driver that loaded just a little earlier might complete its entry routine and thus escape the effect of the alteration. However, it is clear that once the rootkit has loaded, no unrecognized drivers can be loaded, and no updated drivers can be installed.

If the whitelist does not exist, the rootkit will create it by initiating a new thread to gather the information. The thread waits until the ntdll.dll file can be opened, meaning that the file system driver has become active. The thread makes an attempt once every 200ms until it succeeds. At that point, all of the critical system drivers will have been loaded, which the rootkit considers sufficient time to allow before creating the whitelist of allowed drivers.

The rootkit enumerates each of the entries in the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services' registry hive. The driver is not added to the whitelist if it has no 'Type' registry value, or if the driver type is not a kernel driver, a file system driver, or a 'recogniser' driver. If the driver's path is 'system32\<driver name>', then the rootkit will reformat the path to '\systemroot\system32\<driver name>'. If the driver has no 'ImagePath' registry value, then the rootkit will supply '\SystemRoot\System32\Drivers\<driver name>.sys'. Otherwise, the rootkit will accept the 'ImagePath' value, regardless of what it contains.

The rootkit checks whether the driver name is among the blacklisted names, and will not add it to the whitelist if it is. Otherwise, the rootkit opens the file, reads the entire file into memory, relocates it to a fixed base of 0x10000, and calculates the MD5 hash, as described above. The rootkit then attempts to find the resource section in the image. Interestingly, it supports 64-bit files in this routine, even though such files are excluded explicitly during the MD5 calculation, so the code-path is never executed. The rootkit parses the resource section to find the version information item, and the digital certificate. If either target is found, the rootkit searches the version information and/or the digital certificate for references to any entry in the following list (which is sorted for easier reading):

Agnitum Ltd

Anti-Virus

antimalware

Avira GmbH

Beijing Jiangmin

Beijing Rising

BITDEFENDER LLC

BitDefender SRL

BullGuard Ltd

Check Point Software Technologies Ltd

CJSC Returnil Software

Comodo Inc

Comodo Security Solutions

Doctor Web Ltd

ESET, spol. s r.o.

FRISK Software International Ltd

G DATA Software

GRISOFT, s.r.o.

Immunet Corporation

K7 Computing

Kaspersky Lab

KProcessHacker

NovaShield Inc

Panda Software International

PC Tools

Quick Heal Technologies

Sophos Plc

Sunbelt Software

SUNBELT SOFTWARE

Symantec Corporation

VirusBuster Ltd

Any driver that references any of the names on the list will not be added to the whitelist, but if the driver has not been excluded, the rootkit will add the MD5 hash to the MD5 whitelist. The rootkit also calculates the FMV-1 hash of the driver path, and adds that to the FMV-1 whitelist.

After examining each of the services in the registry, the rootkit performs the same checks for each of the files in the '\SystemRoot\System32\Drivers' directory, and each of the DLLs in the '\SystemRoot\System32' directory. After examining each of the DLLs, the rootkit waits until the 'win32k.sys' module appears in the loaded module list. At that point, it queries the list of loaded modules again, and adds all of the entries that are not on the blacklist, as described above. There is some duplicated code here, whereby the rootkit calculates the FMV-1 hash of the driver path, and adds that to the FMV-1 whitelist again. This is harmless though, since the duplicated entries will be removed later.

If the rootkit is running on a version of *Windows* prior to *Windows Vista*, the rootkit adds the 'ntldr' and 'boot.ini' files manually to the FMV-1 whitelist. Otherwise, it adds the 'bootmgr' and '\SystemRoot\System32\winload.exe' files manually to the FMV-1 whitelist. The rootkit sorts the MD5 and FMV-1 whitelists, and removes any duplicated entries. It then writes the 'DB0' and 'DB2' registry values with the contents of the MD5 and FMV-1 whitelists, respectively. The rootkit also registers a callback which receives control when an image is loaded, before the image gains execution control. The callback watches for 'win32k.sys' being loaded, and sets the flag that the whitelisting thread checks (if it is not set already). If the loaded file can be opened, the rootkit reads the entire file into memory and then performs the whitelist check, as described above. Otherwise, the rootkit performs only the MD5 hash check on the in-memory image. If the image fails the verification, the rootkit performs the same code alteration as for the blacklisted drivers.

Next time, we will look at the different IRP functions, and the details of the rootkit's stealthing abilities.

REFERENCE

[1] Ferrie, P. The curse of Necurs, part 1. Virus Bulletin. April 2014, p.4. http://www.virusbtn.com/pdf/magazine/2014/201404.pdf.