MALWARE ANALYSIS 2

THE CURSE OF NECURS, PART 3

Peter Ferrie
Microsoft, USA

In the previous two parts of this series on the Necurs rootkit [1, 2], we looked at what it does to hook the system. This time, we will look at what those hooks actually do.

TERMINATE WITH PREJUDICE

An early version of the rootkit created a TCP filter device, and attempted to attach it to the top of the network stack so that it would be the first device to receive all requests. If that attempt failed – which could happen if the subsystem had not been initialized yet – the rootkit created a thread that ran once every 100ms to attempt to register the device. The thread ran until it succeeded. However, newer versions of the rootkit do not create a TCP filter device, and the associated code has been deleted – it is unclear as to why this functionality has been removed.

The rootkit retrieves the *Windows* version information. It checks for versions 5.1 (*Windows XP*) SP0-3, 5.2 (*Windows Server 2003*) SP0-2, 6.0 (*Windows Vista*) SP0-2 and 6.1 (*Windows 7*) SP0-1. *Windows 8* (6.2) and later are not supported, which might prevent the rootkit from being able to elevate the privileges of the calling process (see below).

The rootkit uses the NtQuerySystemInformation() function to find the base address of ntoskrnl.exe. It searches the ntoskrnl.exe section table for the 'PAGE' section, and then searches the entire section for a platform-specific sequence of code. There is a bug in the search, which is that if the first *n* bytes of the search sequence happen to be the last *n* bytes in the section, and if *n* is less than the length of the sequence, then the search will access memory beyond the end of the section and possibly cause a crash. If the search sequence is found, the rootkit remembers the offset of the sequence within ntoskrnl.exe. The search sequence corresponds to the kernel-mode routine that terminates a process. If the search sequence is found on *Windows Vista* or *Windows 7*, the rootkit assigns itself a platform-specific value for the offset within the token structure, which might be used later to elevate the privileges of the calling process.

The rootkit searches for the current thread handle in the thread array that it carries, and then deletes the entry. This action revokes the thread object's access rights to the rootkit functionality. At this point, the rootkit is fully installed but it remains dormant until the user-mode component registers itself with the driver.

IRP

The rootkit supports multiple I/O control codes that the user-mode component can supply. To communicate with the rootkit, the user-mode component opens the '\Device\NtSecureSys' device, then sends the device an I/O control request with a specific I/O control code and particular parameters.

I/O control code 0x220000 is the 'on' switch. An earlier version of the rootkit contained a date check which restricted use of the interface to any date prior to 2011/11/01 – presumably to enforce the use of I/O control code 0x220020 instead. However, this check has since been removed. The I/O control code uses only simple authentication: it is used with a buffer that is 12 bytes long, where the first DWORD is a key whose value is chosen randomly, the second DWORD is the key XORed with 0xDEADC0DE, and the third DWORD is the key XORed with the process ID. When this I/O control code is used, the rootkit queries the process handle and saves it for later use. This action 'unlocks' the rootkit and enables its full functionality. Once the calling process object has been registered, the method cannot be used again.

I/O control code 0x220020 is used with a buffer containing a special sequence of data. The rootkit calculates the MD5 hash of the data, and requires that the result is 0x377E10EFF125EF3D68DCEFD20EBAACAF. It is currently not known what form the data takes, since at the time of writing this article, no sample using the API has been seen. If the hash matches the expected value, the rootkit queries the process handle and saves it for later use. This action also 'unlocks' the rootkit and enables its full functionality. The method could be used as an 'override' access, since it can be used even after the registered process object has been assigned, and causes the new caller to become the registered process object. This is likely the reason why no sample has been found that makes use of it – as soon as one sample has been found that carries the correct data, all versions of the rootkit become accessible in the same way, and are thus vulnerable to being uninstalled.

The following I/O control codes can be used only by the registered process object:

- I/O control code 0x220004 is used to grant the current thread object access rights to the rootkit functionality.
- I/O control code 0x220008 is used to revoke the current thread object's access rights to the rootkit functionality.
- I/O control code 0x220014 is used with a buffer that is four bytes long. It receives a hard-coded value, which might be the version number (currently 0x11).
- I/O control code 0x22000c is used to request the path name of the driver file ('\SystemRoot\System32\Drivers\<DriverName>.sys').
- I/O control code 0x220010 is used to request the registry path of the driver file ('\Registry\Machine\System\CurrentControlSet\Services\<DriverName>').
- I/O control code 0x220018 is used to update the rootkit driver file, by replacing it with the contents of the supplied buffer.
- I/O control code 0x22001c is used to uninstall the rootkit, by deleting the driver file and its associated registry key.
- I/O control code 0x220024 is used with a buffer that is two bytes long. It is used to assign the port on which the rootkit listens for incoming network connections.
- I/O control code 0x220028 is used with a buffer that is four bytes long. It is used to terminate a process using the supplied process ID. If the termination routine is found, the rootkit will call it directly. Otherwise, the rootkit will use the documented APIs to request termination, which might be disallowed.
- I/O control code 0x22002c is used to terminate a process using the supplied process name.
- I/O control code 0x220030 is used to acquire system-level privileges for the registered process. The rootkit duplicates the access token of the system process and attempts to assign it to the registered process. If that fails – which can happen, for example, if other security-related software refuses the request – then the rootkit retrieves a pointer to the current process object and a pointer to the primary access token for the current process. The rootkit verifies that the offset within the token structure (which the rootkit saved previously) matches the pointer to the primary access token. If the token structure is at the expected location, the rootkit increases the reference count to the maximum value, then copies the token pointer directly into the process token.
- I/O control code 0x220034 is used to construct the list of registry values that the rootkit will check in the registry callback. The list contains comma-separated Unicode strings, which are converted to a multi-SZ list and then written to the 'DB1' registry value. The entries in the list are also converted to individual Unicode structures, which are then sorted according to the value of the code points. The rootkit supports up to 128 entries in the list.
- I/O control code 0x22003c is used to construct the list of registry keys that the rootkit will check in the

registry callback. The list contains comma-separated Unicode strings, which are converted to individual Unicode structures then sorted according to the value of the code points. The rootkit supports up to 128 entries in the list.

- I/O control code 0x220038 existed in a previous version of the rootkit. It was used with a buffer that was 36 bytes in length and was used to query the TDI connection information for the specified ID.

TCP FILTER DEVICE

The TCP device that was created in the early version of the rootkit watched for TDI_CONNECT requests that were not initiated by the registered process. It was interested in connections on port 80 to IP addresses other than 127.0.0.1. The rootkit saved the name of the requesting module and created a connection to 127.0.0.1 on the listening port that was assigned earlier, before allowing the original request to proceed. Since the original outgoing connection was made by a process other than the rootkit, it did not trigger unexpected firewall events. Thereafter, the user-mode component of the rootkit could listen for data received on the requested port. As mentioned previously, the TCP device is not present in the more recent versions of the rootkit.

FILESYSTEM DEVICE

The filesystem device watches for requests for open or create, close, and write or set information, for a given file. When a request to open/create a file is seen, the rootkit checks if the thread handle is present in the thread array that it carries. If the handle is present in the array, the rootkit allows the request to proceed without interference. Otherwise, if the filename refers to a file that has been opened from within a subdirectory, the rootkit requests the name of the subdirectory and prepends that to the filename. If the file has not been opened from within a subdirectory, the filename is used without modification.

The rootkit searches the filename for the last slash. If the filename includes a stream name, it discards the stream name and looks only at the filename. If the filename matches either the name of the rootkit driver file or the name of the registered process, the rootkit denies the access request. If the filename matches the name of the first driver in the rootkit's loader group, the rootkit denies requests to replace the file, because it might also be the rootkit filename, if a different version of the rootkit is run.

If the request is to open or create a file whose name matches the name of the first driver in the rootkit's loader group, the rootkit saves a copy of the file object in an array that it carries. The rootkit makes use of the array to deny all write and set information requests for matching file objects. If the filename is not restricted, then the rootkit searches for a match among the entries from the 'DB2' file. If a match is found, the rootkit denies requests to replace the file. Otherwise, it saves a copy of the file object in an array that it carries.

When a request to close a file is seen, the rootkit searches for a match in its file object array. If no match is found, the rootkit allows the request to proceed without interference. If a match is found, the rootkit removes the entry from the file object array. Interestingly, the search is allowed to continue at that point, until the end of the array is reached. It appears that the rootkit's author forgot to add a break from the loop.

PROCESS/THREAD CALLBACK

The process and thread callback begins by checking if the callback was triggered by a process or a thread. If the object is a thread, the rootkit determines the process that owns it. If no process has been registered, if the target process object is not referring to the registered process, or if the calling process is the registered process, then the call is allowed to proceed without interference. Otherwise, the rootkit checks the requested operation.

If the request is to open the registered process, then the rootkit disallows the following operations: suspend/resume, set information, set quota, dup handle, VM read (a previous version of the rootkit omitted this flag), VM write, VM operation, create thread and terminate. If the request is to open a thread within the registered process, then the rootkit disallows the following operations: set information, set context, suspend/resume and terminate.

There are two exceptions for the adjustment to the process access rights: the rootkit determines the name of the process that is making the request. If the name is 'svchost.exe', the rootkit allows the 'dup handle' operation; if the name is 'lsass.exe', the rootkit allows the 'VM write' and 'VM operation' operations. There is one exception for the adjustment to the thread access rights: if the requesting process is duplicating a handle that it already owns, all requested access is granted.

OPENPROCESS HOOK

The OpenProcess hook works in a similar way to the process-specific portion of the process and thread callback.

The rootkit calls the original function and returns immediately if an error occurs. The rootkit also returns immediately if no process has been registered, if the calling process is the registered process, or if the handle does not refer to the registered process. If the handle does refer to the registered process, the rootkit closes it and then reopens it with the same process-specific operations disallowed as for the callback, and with the same exceptions as for ‘svchost.exe’ and ‘lsass.exe’. A previous version of the rootkit contained a bug in this code, which would open the process only if the requesting process was either ‘svchost.exe’ or ‘lsass.exe’.

OPENTHREAD HOOK

The OpenThread hook works in a similar way to the thread-specific portion of the process and thread callback. The rootkit calls the original function and returns immediately if an error occurs. The rootkit also returns immediately if no process has been registered, if the calling process is the registered process, or if the handle does not refer to the registered process. If the handle does refer to the registered process, the rootkit closes it and then reopens it with the same thread-specific operations disallowed as for the callback, but without any exceptions.

REGISTRY CALLBACK

The registry callback checks if the current thread handle is among the thread handles in the array that the rootkit maintains. If the handle is not a rootkit thread, the rootkit watches for attempts to set or delete registry values, and checks against the entries in the registry value list. It denies the access request if there is a match. The rootkit watches for attempts to create or open registry keys, and checks against the rootkit driver path. It denies the access request if there is a match. The rootkit checks for ‘wuauserv’ and ‘BITS’, and denies the access request to anything other than ‘services.exe’. The rootkit checks against the entries in the registry key list and denies the access request if there is a match.

Next time, we will look at what the user-mode component does.

REFERENCES

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

[2] Ferrie, P. The curse of Necurs, part 2. Virus Bulletin, May 2014, p.18. http://www.virusbtn.com/pdf/magazine/2014/201405.pdf.