



Rootkits – Great fun in the sun

Disclaimer

Rootkits can be considered malware, the examples being given are for sake of example and furthered knowledge only, please do not use this knowledge for malicious purposes. I do not take any responsibility for what you do with this information. This information is furnished for you merely in the purpose of academic research so you will know how to better defend against, or remove, rootkits. Additionally you may find that a rootkit might suit your organization to increase security by inhibiting functions otherwise detrimental to your network security.

- Under no circumstance is this information to be used in situations prohibited by law.

What is a rootkit?

- **Wikipedia** – A rootkit is malware which consists of a program (or combination of several programs) designed to hide or obscure the fact that a system has been compromised.
- **My Definition** – a piece of software that abstracts itself at the kernel level in order to provide a higher level of functionality by subverting the standard kernel features

Not all rootkits are malware (legally)

- Thank you Sony and Starforce
- Rootkits have a legitimate business use
 - Preventing employees from installing software (DeepFreeze, Anti-EXE, etc)
 - Helping prevent data leaks by abstracting the networking layer
 - Preventing users from performing certain functions in the way originally intended (reducing malware portability to your system).

How do they work?

- Root layer device driver written in pure C using a largely undocumented kernel API
 - Can be written in any language you can write a driver in, E.G. Delphi (ObjectPascal)
 - Not all rootkits work on the kernel level but for sake of example, ours does
- Hides itself by removing its pointer from the driver list (a different list is used to allocate resources to the rootkit)
 - Interestingly enough, even though removing your driver from the list will hide it except from rootkit detectors, adding the driver back into the list just before the enumeration click, then removing it afterwards makes it seem as if the driver is not hiding itself to rootkit detectors (id est they bypass the detector) while effectively remaining hidden from the user

How do they work? (cont.)

- Rootkits can then implement differing functionality depending on need
 - Hiding files using the NTFS-Alternative Data Stream
 - Rewriting kernel level functions (such as ZwWriteFile and other Ki (kernel) functions) to perform alternative actions
 - Rewriting user level functions (such as DLLs)
 - Process I/O (incl. System memory)
 - Key logging
 - Etc

NTFS-ADS

- Part of an old subsystem to create interoperability with the Mac OS (Mac OS had the ability to change file icons without modifying the file itself, this is Microsoft's implementation of that ability)
- Completely hidden from the user and system
- Very simple and documented
 - Putting data in the ADS is as simple as going to DOS and typing “type testdata > visible.txt:hidden.txt”
 - Read is as simple as “more < visible.txt:hidden.txt > output.txt”
- The NTFS-ADS navigation tree is based on which file/directory the hidden data is attached to
 - Hidden data attached to directories is less detectable than those attached to files themselves

Rewriting kernel functions

- Can inject data into the System Call Table, but the level of complexity of the read-only system increases with Operating System version and exponentially increases detectability
- Better method is rewriting the kernel functions themselves in `ntoskrnl.exe`, or better yet, renaming the kernel and creating a new boot entry
- Examples include rewriting Zw functions (file/registry/process/port/etc operations), Ki functions (kernel level functions, incl. Driver stuffs), Etw functions (event tracing), Rtl (runtime library), Ldr (loader manager), etc
- Can be used to prevent users from loading drivers, or certain drivers, prevent execution, loading of DLLs, writing certain data to files, reading certain data from files, etc.

Rewriting user functions

- Also known as process injection
- Allows you to overwrite in memory DLLs, such as rundll32, or otherwise with arbitrary EXEs, DLLs, or otherwise
- Allows you to create DLLs that will satisfy the base system but will not allow the user to load certain other executables or otherwise

Process I/O and keylogging

- Allows intercept of program flow.
 - Can intercept data being passed to a function such as DeviceIoControl which is called before PGP encrypts data
- Keylogging should speak for itself

Extending

- Rootkits are not just device drivers that implement built-in functionality, they can also listen on the network for a controller to perform various actions
- Can be modularized
- Can communicate with one another

Loading

- Keep in mind that you will have to load the .sys file produced once compiled
- Various loaders do exist, just about all of them require root/Administrator access to install
 - Not always true but for our example we're using a kernel level rootkit

Further information

- Wrox Press' *Professional Rootkits* by Ric Vieler
 - Very good source of information to start on
- Various internet sources
- Windows DDK
- MSDN