Thoughts about Cross-View based Rootkit Detection

Joanna Rutkowska
http://invisiblethings.org
June 2005

Recently, cross-view based approach to rootkit detection, especially in regards to hidden files and registry keys, became very popular. This is mostly because of the recent release of the tools like Rootkit Revealer and Black Light as well as Microsoft research project, with a friendly name GhostBuster. Many people started to think that it is going to be the ultimate way for detecting all rootkits and system compromises in general...

Cross-view based detectors, like Rootkit Revealer, compare a "low level" system view with a "high level" view. Let's focus here on hidden files detection on Windows systems. How to obtain a low level view of the file system? Of course by reading a raw disk sectors and parsing them according to NTFS layout. But how the detectors read disk sectors? Well they use `CreateFile("\\.\C")` to get the handle to a volume and then use well-known `ReadFile()` function to read the sectors. Alternatively, detectors may try to open "`\\.\PHYSICALDRIVE0`" pseudo-file, and then use the same `ReadFile()` function to get the sectors of the physical drive.

Both of these methods can be easily cheated by the rootkit. It is only necessary to hook `ReadFile()` API and cheat about the contents of the disk sectors. It is not really true that such scenario "would require a level of sophistication not seen in rootkits to date". Officially undocumented, NTFS structure is in fact known well enough (as some open source projects show) to allow for implementing of such behavior.

To achieve better true estimation, hidden files detectors need to go deeper. Next level would require having an agent in the kernel, which would bypass the Windows userland API and may use native `ZwCreateFile()/ZwReadFile()` from within kernel mode. This can be, of course, bypassed by very old technique of System Service Table hooking or IAT hooking.

Detector's agent can go deeper and manually build appropriate I/O Request Packet (IRP) and use `IoCallDriver()` to send it to the disk driver directly (bypassing all the API functions) asking it to read some sectors from the disk. This technique can vary as we may choose to speak with a class disk driver, port driver or even miniport driver. The latter provides, of course, the deepest level among the above methods.

Not surprisingly this can also by relatively easy intercepted by the technique known as IRP hooking (which *can* be seen in the, even publicly available, open source rootkits today).

Can the detector go even deeper then? Yes, but it seems that the deepest possible level it can achieve is the use of `in` and `out` machine instructions to actually speak with the HDD controller. Can this be cheated? Probably not. But this might be very hard to implement, so that all the hardware HDD controller were supported. In fact such detector would just double the standard operating system disk drivers' stack code. Wouldn't it be simpler to just copy the `atapi.sys` file to `myatapi.sys` and use it instead (together with the whole stack above those

miniport drivers)? Yes, we would have to also change the naming of the kernel objects too.

We may start to think that, although the agent would be getting the real sectors, the rootkit might decide to hook the communication between the rootkit detector and the agent in a very similar way as it did when the detector was using the system disk driver… We face here the very inelegant subject of implementation specific attacks. In fact every rootkit detector can be beaten with this approach (unless we exploit some hardware support). It is not fair then to count this attack against Rootkit Revealer or any other detector, since it applies to all others detectors as well.

Let's assume then, that we have such sophisticated rootkit detector, which doubles OS disk drivers' stack functionality. Can this be cheated in a more general way? It seems that the only way for achieving this would be to first detect that the detector is running a file system scan and temporarily unhide all its hidden files against this process. This way the hidden files will not be reported. This approach can also be used to cheat so called outside-the-box analysis, which gets the necessary information when the system is booted from the clean CDROM.

The above idea is actually widely exploited by the present rootkits which decided to hide from Rootkit Reveler or Black Light detectors.

One may ask a simple question now: why bother to hide files at all? Isn't the idea of "hide in the crowed" equally stealth? The answer, fortunately, is no (in other case it would turn out that the whole effort to create stealth technology was just a waste of time and we will get back to the less interesting world of traditional malware).

The answer is no, because the current antivirus technology is able to find all (unhidden) executable files and then perform some kind of analysis if the given PE file looks like a potential rootkit/malware installer (for e.g. check if it uses functions like `OpenProcess()`, `OpenSCManager()`, `ZwSetSystemInformation()` and similar). When designing such scanner we need to remember that rootkit executable can comprises of two parts, one being an actual malware loader and the other being a (polymorphic) decoder. But this is (hopefully) not beyond the advanced AV scanners today.

So, to get back to the main thread, we concluded that:

- Cross-view based detectors, to be effective, need to implement extremely deep method for getting system information. The disadvantage here is the complexity to support all hardware. Besides it is not very elegant, since we actually duplicate parts of the operating system.
- Even if the detector implements such deep method, it is still possible (and even simpler) to cheat it by temporarily un-hiding hidden files to the detector process.
- It is bad idea not to hide files at all, since some kind of heuristics, similar to used by advanced AV scanners, can be used to find malware executables.

Of course we can now connect our heuristic based scanner with a cross-view based hidden file detector. Similar approach was described by the Microsoft researchers, were GhostBuster DLL was injected into traditional AV program and signature based solution was exploited to catch the rootkits which decided to unhide their files to cheat hidden files detectors.

And how this can be cheated (i.e. heuristic scanner cooperating together with cross-view based hidden files detector)? This will be left as an exercise for the reader and

we will get back to this problem when appropriate detectors appear.

And last but not least, we should stop thinking that every rootkit will always be interested in actually surviving the system restart. When considering rootkits installed on a corporate servers by an reliable 0day exploit, the attacker might decide the she doesn't want to leave any traces on disk for the forensic investigators. This is understandable when we take into account fact, that such servers are not restarted every day, but much more rarely and the attacker may, rightly, think the if she succeeded once with her 0day exploit, she will probably succeed next time, that is after the system reboot.

When we focus on Windows desktop machines, we should also consider, somewhat similar, idea of worm-based rootkit. The worm component takes care about infecting desktop computers. When it succeeds to exploit one of the 0day bugs it has in its database, it downloads its rootkit component, which then takes care about "protecting" (read: hide) other potential malicious modules installed in such compromised desktop system, like password sniffers, backdoors, etc… Now, because thousands of computers are infected, the rootkit may not care about surviving the reboot on a single desktop, because it will be automatically infected next time it will become online (this assumes very aggressive warm propagation algorithm, in practice it would probably be infected after some number of reboots). We have then a number of compromised computers, collaboratively taking care of infecting each other. All of them takes also care about being as stealthy as possible, thus not creating any hidden files on disk (nor registry keys).

The last two scenarios requires, of course, very stealthy and reliable way of exploiting bugs, which is capable of delivering and installing rootkit/backdoor/sniffer/etc... to the host without giveaway that something wrong is happening. Such technology is accessible today. Obviously different methods of detection are need to fight with such malware.

Is having a cross-view based rootkit detector a bad idea then? Definitely not. Author only wanted to remind, that system compromise detection is a complex field and we should not expect a single idea to revolutionize it. It is always good if we raise a bar a little bit higher which will result, hopefully, in more interesting rootkits to appear in the future, making our world more interesting. I wrote 'hopefully', because, the real plague these days are implementation specific attacks, which include intercepting communication between detector client program and their agent in kernel, or even, in the very extreme forms, intercepting the GUI functions responsible for presenting the results. These attacks are not elegant not even to say ugly. How to deal with them is another story though...