

# Detection of Rootkits in the Windows Registry

Mihir Nanavati, Bhavesh Kothari  
MIEL Labs  
MIEL e-Security Pvt. Ltd., Mumbai

## Abstract

*Windows rootkits use a number of techniques to hide their presence on a system. Most persistent rootkits share one thing in common – they have a mechanism to ensure that they are loaded when the system boots up. This usually involves the use of the registry. To prevent tools that monitor startup entries from being aware of their presence, these rootkits need to hide their registry keys from the standard APIs. The most common methodology for detection is known as cross-view detection. In this paper, we try and outline all the information necessary to perform cross-view detection of the Windows registry and some of the problems that are inherent to this approach. We also talk of the approach that we have taken with Helios for the detection of hidden registry entries.*

## 1. Introduction

In the modern Microsoft Windows family of operating systems, the registry is an important system-wide and user specific store of data. It plays a critical role in configuration of both the operating system and of various installed applications. Configuration settings such as which drivers are to be loaded and which libraries are to be linked into applications are stored in the registry and read during the boot process. This makes the registry an ideal

store for rootkits to store their startup information.

Tools, such as SysInternals' Autoruns[1], monitor the system for the modules that are loaded at boot time. If a rootkit were not to conceal its presence in the registry, such tools would detect it as it would have to place itself in the registry keys monitored by these tools. Hence, it becomes imperative for it to hide the registry keys that would reveal its presence.

The most common evasion and detection techniques for registry keys are discussed in the paper about Strider Ghostbuster by Wang, et al.[2] The most commonly used method is termed cross-view detection. In this method, a high level view of the registry is obtained using the Windows API. This gives us the data that is visible to operating system tools such as regedit.exe. A second view, termed the low level view is obtained directly from the structures on disk without using the Windows API. The two views are then compared, and the discrepancies are marked as possible hidden entries. In most cases, a hidden entry will be the result of rootkit activity. It is, of course, possible that the discrepancies may be actual discrepancies that occur because of changes in the state of the registry between the time taken by the high level and low level scans. This is usually due to the activity of background processes on the system. Such discrepancies, however, are usually easily distinguishable from rootkits.

The low level view of the registry may be obtained either in an online scanning, or an offline scanning mode. In an online scanning mode, the system that is to be scanned is booted and operational, where as in an offline mode, an alternative operating system is booted and used to acquire the data from the registry files.

In this paper we would like to focus on some of the structures that are necessary to understand to perform low-level scans of the registry. We would also like to discuss a basic detection technique similar to that we have used in Helios and the test results.

## 2. The Registry

A large part of the structure of the registry has been explained by Russinovich and Solomon[3]. The registry is a database made up of keys, values and the data associated with those values. The keys are similar to directories on the disk in that they do not have any data of their own, but contain a large number of values. The values are analogous to files and have both a name and data associated with them. The main key in the registry is the root key, while all the other keys are sub-keys of this root key.

Each of the values has data associated with it. The data can be of many different types. Some of the common data types are strings, integers, multiple string arrays and binary values.

### 2.1 Registry Hives

On disk, the registry isn't simply one large file but as a set of files known as hives. Each hive has a root key associated with it. Some of the hives are common to the entire system (such as those which are listed under HKEY\_LOCAL\_MACHINE) while

others are user specific (such as those listed under HKEY\_USERS). The system level hives are SYSTEM, SOFTWARE, SAM, SECURITY and HARDWARE.

Out of these, the SYSTEM and the SOFTWARE hives are of particular interest to rootkits and rootkit detectors. The SYSTEM hive contains a list of all the services and drivers that are present on the system. Kernel level rootkits, such as Hacker Defender[4] or BadRkDemo[5] can be detected from the SYSTEM hive, while those that use DLL Injection, such as Vanquish[6], can be detected from the SOFTWARE key.

The registry is divided into blocks that are 4096 bytes in size. The first block is called the base block and has the signature 'regf' (0x66676572) in the first 4 bytes. The base block contains an offset to the root key of that hive. The actual data is stored in containers called cells. These cells may contain keys, values, lists of sub-keys or values or data. They are distinguished by a signature present at the first two bytes of the structure. The header of each cell contains the signature and the size of that cell.

The space for the registry is not allocated in multiples of cells, but in those of bins. The bins contain the cells and have a header signature of 'hbin' (0x6E696268). It is possible that particularly large data fields may be split over more than one bin.

The different cells are linked to one another to give a structure to the registry hive. Some cells may contain a link to child cells, while others may contain a link to the parent cell. All the links are maintained via offsets into the same data structure, i.e. the same registry hive file. This offset is an absolute offset from the base block, not a

relative offset inside the bin or from the current cell. Thus, any cell can easily be accessed and referenced via the offset even without knowing the current position from which it is linked.

## 2.2 Types of Cells

A detailed description of the cell types are illustrated in 'Beginning to See the Light'[7], while the structures have been adapted from the 'Offline NT Password and Registry Editor'[8].

**Key Cell (NK):** A cell that contains a registry key. It can be distinguished by the signature 'nk' (0x6B6E) in the header. A field also signifies whether it is the root key, or a sub-key. The most important fields it contains are the name of the key and the length of the name of the key. It also contains fields containing the number of sub-keys and the number of values present in this key. Two other fields give offsets to the sub-key list cell that contains links to the sub-keys and the value list cell that contains links to all the values.

**Sub-key List Cell (RI/LH/LF):** A cell that contains the links or offsets to all the sub-keys of a particular key. It is distinguished by the signatures 'lf' (0x686C), 'lh' (0x666C) or 'ri' (0x6972). The LH and LF entries are similar and contain a list of offsets to the key cells that form the sub-keys.

The RI key is used for keys that have a large number of sub-keys. Rather than pointing directly to the LH or LF structures, the key cell will point to a set of RI structures. The RI structures contain an offset to several of the LH/LF structures. The offsets are stored in a list, similar to the list used for storing key cell offsets in the LH/LF structures.

**Value-list Cell:** A cell that contains a list of offsets to the value cells of a particular key. There is no header to distinguish this entry. The entries of this list are similar to the earlier lists.

**Value Cell (VK):** A cell that contains information about a key's value. It can be distinguished by the signature 'vk' (0x6B76) in the header. The fields give information such as the name of the value, the length of the name of the value and the data associated with the value.

The values may be named or unnamed. Unnamed values are displayed by the name (Default) by the Windows Registry Editor. They have a name length of 0 and no name. The data for the value may be resident or non-resident. If the value links to the data, the offset to the data is stored in the key. If, however, the value contains the data, it is stored in lieu of the offset to the data cell in the key.

**Data Cell:** A cell that contains the data associated with the value. There is no header to distinguish this entry.

## 2.3 Traversing the Registry

The base block occupies the first 4kb of the hive file and contains an offset to the first hbin structure which contains an offset to the root key of the registry hive.

Every key or subkey in the registry hive is represented with a key cell or NK structure. It contains an offset to the value-list member containing the values present under the keys and an offset to the relevant sub-key structure, which may point to an LH, an LF or an RI structure.

The LH or LF entry contains a member that contains a list of offsets to all the other

sub-keys of the key. Let us assume this element is called `offsetToSubkeys`. It is a pointer to the first entry in a list of offsets to sub-keys. Hence, `offsetToSubkeys [0]` will be the offset to key cell for the first sub-key, `offsetToSubkeys [1]` to the key cell of the second sub-key and so on.

In case of RI structures, there is a list of offsets to the relevant LH or LF structure. They contain the offset to the LH in a similar way to the list of offsets in the LH or LF entry.

The value-list contains pointers to value keys. The entries in the list can be treated similarly to the above lists, i.e. the first offset points to the value key structure for the first value, the second to the 2nd value and so on.

The value entry contains information related to the value present under a key and the associated data. The data may or may not be inline. If the data size contains a 1 in its MSB (`0x80`), the key contains inline data, ie – the offset to the data is the actual data rather than an offset to the data.

In case of non-inline data, the offset to the data is the offset of the data cell. In case the first two bytes of the data cell contain the signature 'db' (`0x6264`), it is a data cell with non-contiguous data. This may be because the size of the data is very large, and cannot be accommodated in a single bin. The data is then spread over a number of bins. The next two bytes then tell us how many bins the data is spread over. Following these two bytes is a list of offsets to the data cells that make up this data entry. The data from all these cells can be concatenated to form the actual data of that particular value.

### **3. Obtaining the Low Level Data of the Registry**

The low level data of the registry can be obtained in a number of different ways. In case of an offline scan, the files can simply be copied from their locations and parsed. During an online scan, one can use API functions such as `RegSaveKeyEx()` to dump a copy of the registry hive to disk, from where it can be parsed. This is the approach used by `RootkitRevealer[9]`. Another approach is to parse the entire file system to obtain the physical locations on the drive where the registry hives are stored and making a copy using low level disk copy routines.

### **4. Cross-View Detection in Helios**

Helios performs an online scan of the registry of the system. The "low-level" data is obtained by reading the registry hives by directly reading the sectors from the hard drive after the NTFS structures are parsed. Thus, low level operations are used to obtain the registry hive files, bypassing the Windows API altogether for acquisition of data.

The data is then parsed according to the structures that have been discussed previously. The keys, values and the data in all of them are stored.

The higher level view is obtained using the `RegEnumKeyEx()`, `RegEnumValueEx()` API functions that are used by `Regedit` to display registry. Registry keys and values that have been hidden will be apparent as they will be absent in the higher level view. Helios will mark them as discrepancies accordingly.

This technique is successful in detecting most of the current rootkits that hide their

presence in the registry. It can detect Hacker Defender, Vanquish and BadRkDemo and display the keys that have been hidden.

## 5. Limitations

The greatest difficulty in an online scan is obtaining a reliable source of information since the system cannot be trusted to accurately report its state. However, the convenience of being able to perform a scan under operational conditions often makes this an acceptable risk. Production environments may not be accepting of downtime, and in these environments, the online scan provides a level of defense despite its limitations.

It has been observed that attackers may use non-persistent rootkits on systems that are unlikely to be rebooted (such as production servers). As they do not have a disk or registry footprint, the aforementioned cross-view detection approach will be unable to detect them.

In case of the registry, detection engines that rely on RegSaveKeyEx() such as RootkitRevealer are vulnerable to attacks that hook the registry functions and filter out the relevant data before dumping the file. Helios uses disk read operations to obtain the registry files. However, it is possible to filter out the data at this level as well. This could be by hooking the read functions, filtering data from ntfs.sys or disk.sys or with the use of filter drivers.

## 6. Conclusion

In this paper we discussed the structures that make up the Windows registry and how they can be used by rootkit detection engines to look for traces of a rootkit on the system. We hope that the information can

be used by people to gain a further understanding about how Windows actually stores data in the registry.

We also discussed some of the common methods to use this data and some of the possible measures that rootkits may use to counter them. Most of the commonly seen rootkits are detected by the cross-view detection method discussed. However, the world of evasion is constantly evolving and it is a given that newer and more sophisticated rootkits will use different techniques to hide their presence.

## References

- [1] M. Russinovich and B. Cogswell, Autoruns. <http://www.sysinternals.com>
- [2] Y.M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski. Detecting Stealth Software with Strider GhostBuster. Microsoft Research Technical Report MSR-TR-2005-25, February 2005.
- [3] M. Russinovich and D. Solomon. Microsoft Windows Internals (4<sup>th</sup> Edition): Microsoft Windows Server 2003, Windows XP, and Windows 2000. Microsoft Press, January 2005.
- [4] Holy Father. Hacker Defender. <http://www.hxdef.org/>
- [5] CardMagic. BadRkDemo. <https://www.rootkit.com/vault/cardmagic/badrkdemo.rar>
- [6] XShadow. Vanquish. <http://www.rootkit.com/vault/xshadow/vanquish-0.2.1.zip>
- [7] P. Clark. Beginning to See the Light. <http://www.beginningtoseethelight.org/ntsecurity/>
- [8] P. Nordahl. Offline NT Password and Registry Editor. <http://home.eunet.no/pnordahl/ntpasswd>
- [9] B. Cogswell and M. Russinovich. RootkitRevealer <http://www.sysinternals.com/>