

# ROOTKIT ANALYSIS

## ‘YET ANOTHER RUSTOCK ANALYSIS...’

Lukasz Kwiatek, Stanislaw Litawa  
ESET, Poland

In this article we are not going to talk about the history of this rootkit, nor will we talk about all the speculation that we have heard during the last year [1]. Instead, we will simply describe in detail a driver protector and an infector (which is also a disinfectant), and then present an overall view of system hooks and a few of the self-defence techniques used by Rustock.C.

### DRIVER PROTECTOR

The driver protector used by Rustock.C is very similar to some of the well-known ring 3 PE protectors. In this instance, we can find anti-debugging and anti-patching tricks, import table redirection, heavy code obfuscation, multiple encryption layers and so on. One ‘old’ new feature is hardware locking. Hardware locks are often used in commercial software protection schemes to avoid piracy and restrict usage to just one machine per licence. From the anti-virus industry’s point of view, we should also take note of the fact that the infected driver doesn’t have an import table.

In the Rustock.C protector, we can distinguish three protection layers:

- L0: very simple encryption (xor/sub/add-based)
- L1: initialization layer
- L2: actual protector layer

Since layer L0 consists of very simple encryption, it will not be discussed in detail.

### LAYER L1

L1 is responsible for finding ntoskrnl (ntkrnlpa, ntkrnlmp, ntkrnpmp) in memory and allocating a new memory buffer to which to copy itself. In a normal situation, analysis of this kind of protector would be very easy (even trivial), but this time we have to deal with a very advanced, multi-layer code obfuscator. During our research on Rustock.C, most of our time has been spent on the development of a deobfuscation tool to facilitate the analysis of the protector and the rootkit. So, what can we see in layer L1 after deobfuscation?

The functions for which it is responsible are:

- Searching ntoskrnl in memory – a well-known trick used by ring 3 packers.
- Obtaining the addresses of imported functions – functions are imported by 32-bit checksum value. This is also a very popular ‘ring 3’ trick (used, for example, in PESpin).

Layer L1 uses two functions from ntoskrnl: NtQuerySystemInformation with SystemModuleInformation as a parameter and ExAllocatePoolWithQuotaTag with the tag ‘Info’.

When the whole driver (less the first few bytes) is copied to a new memory buffer, the execution flow is transferred immediately to that buffer.

### LAYER L2

Layer L2 is the main layer of the protector. It handles decompression, decryption, filling of the import table, correction of the relocations and finally jumps to the original driver entry. The whole protection scheme is based on an encrypted structure that contains descriptions for each section, including the addresses and keys needed to handle imports and relocations.

Each section is compressed with aPLib and encrypted with the RC4 algorithm. The key for RC4 is constructed from three dwords, the third of which is stored in the protected driver (whose structure was mentioned earlier). The first two dwords are collected from the PCI bus. Data gathered from the PCI bus can be identified as the DeviceID and VendorID for the following two devices:

- Bridge device – ‘Host/PCI’
- Bridge device – ‘PCI/ISA’ or ‘Other’

The DeviceID and VendorID are 16-bit values – those values (in particular VendorID) can be found on a small number of lists on the Internet. The full decryption key will have the following format:

```
0xDDDDVVVV 0xDDDDVVVV 0XXXXXXXXX
DDDD – DeviceID
VVVV – VendorID
XXXXXXXX – from the protected driver
```

After RC4 initialization we can observe 111 ‘empty’ rounds. These are used to slow potential brute-force attacks and to randomize the final encryption. After these 111 rounds, there are four more rounds from which the 32-bit key is constructed. This key will be used to decrypt the import table and relocations.

The relocation table is represented in a simpler form than normal relocations from PE executables. In Rustock.C, relocations are an encrypted table of addresses that need to be fixed with the base address of the module.

Imports are encrypted in a similar way to relocations. Each imported function is represented as a nine-byte structure:

```
DWORD relativeAddress;
DWORD checksum;
BYTE unknown;
```

To rebuild the import table, we need to match checksum values with the names of functions. Imported functions are

called through another function that checks for standard software breakpoints (0xCC) and debug breakpoint registers (dr0, dr1, dr2, dr3) at the beginning of the imported function.

The Rustock.C protector also incorporates a few anti-debugging tricks:

- the clearing of debug registers
- the setting of empty functions for all IDT entries
- memory checksums

Technical details on the protector, including keys and code snippets have been published elsewhere [2].

## DRIVER INFECTOR

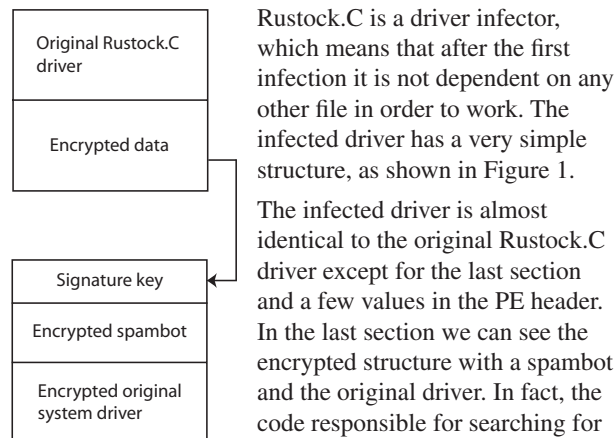


Figure 1: Infected driver structure.

The infected driver is almost identical to the original Rustock.C driver except for the last section and a few values in the PE header. In the last section we can see the encrypted structure with a spambot and the original driver. In fact, the code responsible for searching for that structure in Rustock.C allows the structure to be placed anywhere in the driver. The signature and key structure is rather easy to verify: it is 16 bytes (four dwords) long. The first dword is used as a decryption key and the next three values are used to validate the signature:

```
dword01 = decryption key
dword02 = dword01 - 0x747517C7
dword03 = dword01 ^ 0x945133B7
dword04 = dword01 - 0x0FCFD0AC
```

Data is decrypted with a 'xor-based' algorithm:

```
DWORD* data = addr_of_sig + 0x94;
for (i = 0; i < size; i++)
{
    data[i] ^= dword01;
    dword01 += 0x945133B7;
}
```

At the beginning of the decrypted buffer we have three variables:

```
DWORD offset; //
DWORD size; // x3
BYTE key; //
```

The offset is relative to the beginning of the signature. The key is a one-byte value used for XOR encryption. The first structure describes botdll.dll (the spambot module injected

into winlogon or services). Botdll.dll is encrypted with a one-byte XOR and compressed with aPLib. The second structure describes the original driver in a similar way to the first one, with the exception of compression. The original driver is just XORed with 'key', and after decryption mapped into memory at the base address of the infected driver. This is the reason why the rootkit body is copied to a new memory buffer during the unpacking stage.

## SELF DEFENCE

Rustock.C uses several techniques to protect itself:

- Timer1 checks KdDebuggerEnabled
- Timer2 searches the memory space of all loaded drivers for the following strings:
  - 'NTICE'
  - 'Syser'
  - 'Bpload'
  - 'BPLoad'
  - 'ISO\_S\_'
- The rootkit memory is cleared in case of bugcheck (KeRegisterBugCheckCallback)
- Inline hooks are set on the functions following the functions from the file system driver IRP table. In Ntfs.sys:
  - NtfsFsdCreate
  - NtfsFastQueryStdInfo
  - NtfsFsdClose
  - NtfsFsdDirectoryControl
  - NtfsFsdDispatchWait
  - NtfsFsdRead
  - NtfsFsdSetInformation
  - NtfsFsdWrite

File system hooks are responsible for hiding the rootkit: any attempt to read the infected driver causes on-the-fly disinfection and returns data from the original driver, while the driver remains infected. Also, the size of the driver can be seen from the original file.

- The inline hook on KiFastCallEntry is used to hook some functions from the ServiceDescriptorTable:
  - ZwQuerySystemInformation
  - ZwCreateThread
  - ZwTerminateThread
  - ZwResumeThread
  - ZwOpenThread
  - ZwReadVirtualMemory
  - ZwWriteVirtualMemory
  - ZwProtectVirtualMemory
  - ZwDuplicateObject

- ZwDelayExecution
- ZwTerminateProcess
- ZwCreateUserProcess (*Vista* only)
- ZwCreateThreadEx (*Vista* only)

These hooks are used to protect and hide botdll.dll in winlogon.exe (or services.exe on *Windows Vista*). ZwQuerySystemInformation is called with special parameters and used to access functions from the rootkit (ring 3 to ring 0 communication).

- Infection can easily migrate to another driver and disinfect the current infected file. Infected drivers must be in the following registry path: ‘Registry\Machine\System\CurrentControlSet\Control\SafeBoot\Minimal’.
- Firewall bypassing techniques are employed (a few hooks on tcpip.sys, ndis.sys, wanarp.sys).

## CONCLUSION

Analysis of Rustock.C would be much easier without the advanced code obfuscation (218 KB of obfuscated code versus 70 KB of clear, optimized code). In the future, we will probably see rootkits with private kernel-mode code virtualizers (similar to commercial products like *VMPProtect* or *Code Virtualizer*) becoming more popular. This version of Rustock.C was used as a part of a spam botnet, but the architecture of the rootkit allows it to do anything (password stealing, phishing attacks, DDoS and so on). The botdll.dll file is appended like a plug-in that can be easily changed to another spam-sending module [3] or anything you want. Who knows, maybe there is another variant of Rustock.C in the wild...

## REFERENCES & FURTHER READING

- [1] <http://www.rootkit.com/newsread.php?newsid=879>.
- [2] Kwiatek, L. Rustock.C – kernel mode protector. <http://www.eset.com/threat-center/blog/?p=127>.
- [3] Shevchenko, S. <http://blog.threatexpert.com/2008/06/new-rustock-switches-to-hotmail.html>.
- [4] Shevchenko, S. Rustock.C – unpacking a nested doll. <http://blog.threatexpert.com/2008/05/rustockc-unpacking-nested-doll.html>.
- [5] Florio, E.; Pathak, P. Raising the bar: Rustock and advances in rootkits. *Virus Bulletin*, September 2006.
- [6] Molenkamp, S.; O’Dea, H. Have you got anything without spam in it? Proceedings of the Virus Bulletin Conference, September 2007.
- [7] Rusakoff, V. [http://www.drweb.com/upload/6c5e138f917290cb99224a8f8226354f\\_1210062403\\_DDOCUMENTSArticales\\_PRDrWEB\\_RustockC\\_eng.pdf](http://www.drweb.com/upload/6c5e138f917290cb99224a8f8226354f_1210062403_DDOCUMENTSArticales_PRDrWEB_RustockC_eng.pdf).



## VB2008 OTTAWA 1–3 OCTOBER 2008

Join the VB team in Ottawa, Canada for *the* anti-virus event of the year.

- What:**
- Three full days of presentations by world-leading experts
  - Automated analysis
  - Rootkits
  - Spam & botnet tracking
  - Sample sharing
  - Anti-malware testing
  - Corporate policy
  - Business risk
  - Last-minute technical presentations
  - Networking opportunities
  - Full programme at [www.virusbtn.com](http://www.virusbtn.com)

**Where:** The Westin Ottawa, Canada

**When:** 1–3 October 2008

**Price:** Special VB subscriber price \$1795

**BOOK ONLINE AT  
[WWW.VIRUSBTN.COM](http://WWW.VIRUSBTN.COM)**

