Spam from the kernel

virusbulletin.com/virusbulletin/2007/11/spam-kernel

2007-11-01

Kimmo Kasslin

F-Secure, Malaysia

Elia Florio

Symantec, Ireland **Editor:** Helen Martin

Abstract

Trojan.Srizbi is the first example seen in the wild of a complex piece of malware that operates fully from kernel mode. Kimmo Kasslin and Elia Florio provide a detailed analysis.

Introduction

In December 2006 one of the authors of this article concluded his research paper on kernel malware with the following paragraph [1]:

‘This paper has shown the basic techniques that kernel malware is using to do their job. Their main role has been to perform some specific tasks for the main user-mode component. However, the scene is changing. There has been lots of interest in various research groups to investigate for the possibilities to do more complex tasks directly from kernel. The next big thing is going to be the network side. This year we have already seen presentations talking about how backdoors can be implemented directly from kernel mode using only the NDIS layer and custom TCP/IP stack. We have also seen a presentation about bypassing personal firewalls from kernel-mode.’

Regrettably, the prediction became true in June 2007 when the authors started to analyse a malicious kernel-mode driver consisting of large amounts of code. After deeper analysis it became evident that there were no signs of user-mode injection, meaning that all of the code was being executed at ring 0. Since then the malware family has been known as Trojan.Srizbi (or as Rootkit:W32/Agent.EA).

Origin of Trojan.Srizbi

In June 2007 several AV companies raised an alert due to a large-scale web-based attack being carried out with drive-by exploits. A malicious tag was injected into the homepages of many legitimate Italian and Russian websites. Attackers had injected an IFRAME to redirect

visitors to a website running a multi-staged exploit kit, nowadays best known as ‘MPack’ [2]. The specific MPack installation used for this attack was designed to download several trojans from the remote server 81.95.146.150, hosted by the ambiguous ‘Russian Business Network’.

One of the nasty creatures to spread from this server was Trojan.Srizbi, a sophisticated piece of malware contained within 150Kb of kernel mode code. This was the first public revelation of Srizbi, but further research tracked older samples back to April 2007. The first observed filename was ‘windbg48.sys’, followed by ‘symavc32.sys’ more recently. Other pseudo random names of this threat always take the format [CHARS][2DIGITS] (e.g. Cmjg57.sys, Wdml36.sys, Fln51.sys, etc.).

The Srizbi driver is installed by a dropper component that is packed with UPX and protected with a layer of scrambled code. This layer uses custom spaghetti code obfuscation with CALL/JMP mixed with junk opcodes. It resembles the Rustock.B polymorphic packer, but it is more advanced. The dropper code makes API calls by using PUSH/RET sequences, and parameters are pushed into the stack with MOV [ESP] and other complex indirect loading instructions, making standard static analysis almost impossible (see Figure 1).

```
mov     [esp+arg_28], offset loc_431759
pushf
push    WriteFile
pop     [esp+8+arg_20]
pushf
pushf
pushf
push    [esp+10h+arg_24]
retn    3Ch
```

```
push    eax
push    CreateFileA
pop     [esp+8+var_8]
pusha
pusha
push    [esp+44h+var_44]
push    [esp+48h+var_44]
push    [esp+4Ch+var_4]
retn    4Ch
```

```
pushf
push    GetTempPathA
pop     [esp+8+arg_34]
pushf
push    [esp+8+var_4]
mov     byte ptr [esp+0Ch+var_4], al
mov     [esp+0Ch+var_C], 0A6BDh
push    [esp+0Ch+arg_38]
retn    4Ch
```

Figure 1. Examples of scrambled API calls used by the Srizbi packer.

Reverse engineering of this piece of code is not straightforward and may require the assistance of custom tracing tools. However, it is possible to spot two principal decryption routines in the code:

1. decryption with NOT BYTE
2. decryption with BYTE XOR 0xB0

Decryption routine (1) is identical to the one inside the driver and is used to decrypt all text strings; the XOR decryption (2) is used to decrypt the embedded driver. The Srizbi installer drops a copy of the driver into the '%SystemRoot%\System32\drivers' folder and installs it as a service via OpenSCManager and CreateService. Next, the service is started and the installer self-deletes.

Another of Srizbi's challenges is the use of CRC values instead of basic string compares when it searches for names or resolves imported API functions dynamically. In some cases the only viable solution to find which string is being searched by Srizbi is to brute force CRCs over a set of possible strings.

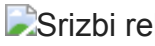
Full-kernel malware

We can say with some certainty that Trojan.Srizbi represents an important milestone in the evolution of malware utilizing kernel-mode techniques. It is the first complex full-kernel malware [3] to have features such as file and registry hiding, bypassing of memory hooks, and low-level NDIS hooks with a private TCP/IP stack. The latter is utilized to implement a fully blown spam client with an HTTP-based command and control (C&C) infrastructure – all directly from ring 0.

The fact that Trojan.Srizbi is fully implemented in kernel mode makes it very powerful. It can activate very early during the boot process, allowing it to make its system modifications before most security products even have a chance to load. Also, many security products do not consider activities initiated by kernel-mode code to be suspicious or malicious since the kernel should be trusted.

A feature that makes Srizbi unique is the fact that it disables and removes other competitor rootkits from the infected system very effectively. The trend of malware gangs fighting each other is becoming more intense nowadays since every unprotected machine is likely to be targeted by multiple pieces of malware – but in the end, there can be only one! Only the strongest and most sophisticated malware will survive.

Srizbi tries to accomplish this goal in a unique way: it rebuilds a clean copy of KiServiceTable by reading NTOSKRNL.EXE directly from disk and then it retrieves all the original (hook-free) function pointers of the required file and registry API functions. With this technique Srizbi is able to bypass most rootkits (and security products) present on the machine and can safely enumerate any files and registry keys.


Srizbi rebuilds its own Service Table from disk.
Figure 2. Srizbi rebuilds its own Service Table from disk.

We also found that Srizbi contains a generic driver removal routine based on CRC values. The routine enumerates services sub keys from following key:

```
\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\[SRV_NAME]
```

If the CRC of [SRV_NAME] matches one of the following hardcoded CRC values, then the trojan unloads the driver with ZwUnloadDriver then deletes the launch point of the service by removing the related registry key and its sub keys.

CRC value	Service name
0xe0e5a117	runtime
0x4c4f27cc	runtime2
0xbf36b345	xpdx
0x949b30b3	lzx32

Table 1. CRC values and service names targeted by Srizbi's driver removal routine.

The services in Table 1 are quite (in)famous within the AV industry, since they are used by two of today's most common rootkit malware families: Backdoor.Rustock.B (lzx32, xpdx) and Trojan.Pandex (runtime, runtime2). Srizbi also contains the text strings of some other malware, 'wincom32.sys' and 'ntio256.sys', however they are not referenced and never used in the actual code. More evidence of the ongoing war between the Srizbi gang and other malware authors was reported by researchers at Arbor Networks [4], who noticed StormWorm bots running DDoS attacks against Srizbi domains.

NDIS hijacking

From a technical point of view the most interesting part of Trojan.Srizbi is its network layer implementation. Its sole purpose is to bypass personal firewalls and other security products that monitor incoming and outgoing network packets. The implication is that the infected machine will be able to communicate with the command and control server and send thousands of spam emails even if the firewall is set to the 'block-all-traffic' mode.

Windows networking architecture is divided into different layers where the ones most commonly used by today's firewalls are TDI and NDIS. The NDIS layer abstracts the network hardware from network drivers and is the lowest layer available for third-party network drivers. This makes it the obvious choice for modern firewalls since the lower they operate the harder it is to bypass them.

Usually NDIS hooking firewalls install their triggers to the following locations:

- NDIS library functions
- NDIS_PROTOCOL_BLOCK structure handler function pointers
- NDIS_OPEN_BLOCK structure handler function pointers

The actual triggers controlling inbound and outbound traffic are implemented by replacing several function pointers from inside the two internal NDIS structures that the NDIS layer uses for sending packets to and receiving packets from the driver controlling the hardware. The following are some commonly hooked handler functions:

- NDIS_OPEN_BLOCK->SendHandler
- NDIS_OPEN_BLOCK->SendPacketsHandler
- NDIS_OPEN_BLOCK->ReceiveHandler
- NDIS_OPEN_BLOCK->ReceivePacketHandler

The firewall driver normally installs inline hooks into four functions exported by ndis.sys that allow it to hook the required NDIS handler functions for any newly registered protocol driver. Otherwise the new protocol driver would be able to send and receive traffic without the firewall seeing it. The hooked functions are:

- NdisRegisterProtocol
- NdisDeregisterProtocol
- NdisOpenAdapter
- NdisCloseAdapter

Now that we have had a brief introduction to how modern firewalls are able to filter the traffic, let's continue by looking at what makes it possible for Trojan.Srizbi to bypass them completely.

First, Trojan.Srizbi finds a network adapter that it can use to communicate with the Internet. It does this by finding a suitable interface from the TCP/IP driver's registry settings. It calls an undocumented LookupRoute function exported by tcpip.sys which will give it the IP address of the default gateway. Then it enumerates all sub keys under the key:

```
\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces
```

It uses the default gateway address to find the matching interface, or if there are no matches it will take the one that has proper IP settings defined. Next, it enumerates all sub keys under the key:

```
\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Adapters
```

Finally, it selects the adapter whose IpConfig value matches the previously found interface.

Trojan.Srizbi gets access to the NDIS structures by installing a dummy protocol temporarily. It first registers a new protocol with a random name by calling NdisRegisterProtocol. Then it binds the protocol to the previously selected adapter by calling NdisOpenAdapter. As a result, the temporary protocol's NDIS_OPEN_BLOCK handler functions will be set up by the underlying system.

As we have already mentioned these two functions are commonly hooked by the firewall. Trojan.Srizbi solves this problem by using its private and hook-free version of ndis.sys. During initialization it loads the ndis.sys file into memory, resolves its imports and finally relocates the new module into the base address of the original ndis.sys module. This means that the private module will still be using e.g. the same global variables as the original ndis.sys module.

The following is the disassembly of the private and original NdisRegisterProtocol functions:

```
kd> u poi(Yol33!NdisRegisterProtocol)
816b917d 8bff         mov    edi,edi
816b917f 55           push   ebp
816b9180 8bec         mov    ebp,esp
816b9182 51           push   ecx
816b9183 53           push   ebx
816b9184 56           push   esi
816b9185 57           push   edi
816b9186 b938d16ff9   mov    ecx,offset NDIS!ndisPkgs+0x20 (f96fd138)

kd> u ndis!NdisRegisterProtocol
NDIS!NdisRegisterProtocol:
f96ff17d e9b2d75200   jmp    fwdrv+0x934 (f9c2c934)
f96ff182 51           push   ecx
f96ff183 53           push   ebx
f96ff184 56           push   esi
f96ff185 57           push   edi
f96ff186 b938d16ff9   mov    ecx,offset NDIS!ndisPkgs+0x20 (f96fd138)
```

With the help of the dummy protocol the driver now has a handle to the registered protocol that is bound to the underlying adapter. The handle is returned by NdisRegisterProtocol's second argument called NdisProtocolHandle, which is defined as PNDIS_HANDLE. This is actually a pointer to the dummy protocol's NDIS_PROTOCOL_BLOCK. By using the information stored in this structure the malware is able to find the adapter's NDIS_MINIPORT_BLOCK structure that is part of the lowest layers of NDIS. It then searches for other protocols that are bound to the same adapter, and takes the first one to match any of the following protocols:

1. TCPIP
2. PSCHED
3. TCPIP_WANARP

The way this whole process works is illustrated in Figure 3.

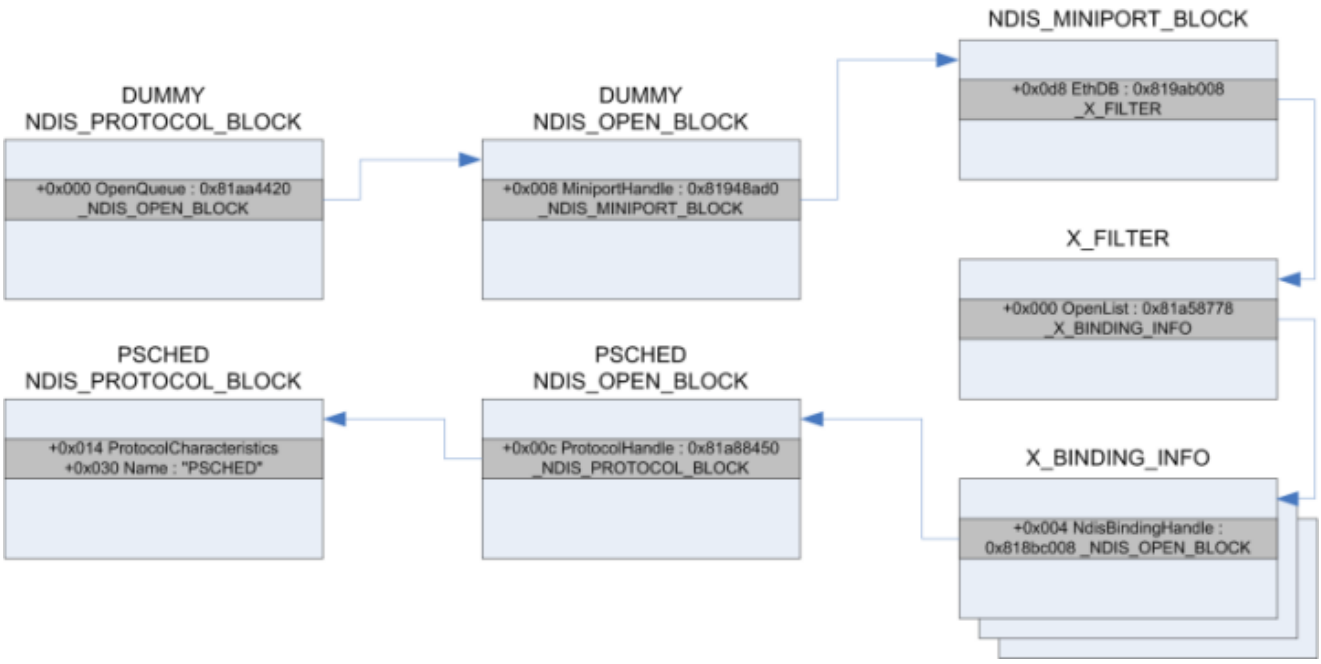


Figure 3. Trojan.Srizbi uses the 'dummy protocol' approach to find the NDIS structures that it hooks.

Now that the driver has access to the NDIS structures of an existing protocol it will replace a bunch of handler functions with its own. In addition, it fetches the addresses of certain handler functions that it will use itself. After it has finished it will just uninstall the dummy protocol.

Trojan.Srizbi uses its NDIS hooks and the original handler functions together with its own TCP/IP stack to send and receive packets.

To send packets it uses the following handler function:

NDIS_MINIPORT_BLOCK->SendPacketsHandler

To get a notification after the send operation has completed it uses the following hook:

NDIS_OPEN_BLOCK->SendCompleteHandler

To receive packets it uses the following hooks:

- NDIS_OPEN_BLOCK->ReceiveHandler
- NDIS_OPEN_BLOCK->ReceivePacketHandler

Since the malware is using its own TCP/IP stack it has somehow to identify which of the received packets should be passed to its private stack instead of the stack used by the system. One solution to this problem would be to use its own IP and MAC addresses since their combination should be unique on most physical machines [5].

However, Trojan.Srizbi uses a different approach. When it sends packets it always uses a source port that is higher than 32,000. For every packet received it checks whether it is a TCP packet, whether its destination IP equals the client's address, and whether the destination port is larger than 32,000. If all of these are true then it forwards the packet to its private stack. To make sure that no other application (using ephemeral ports) sends packets using its reserved source port range it sets the MaxUserPort registry value to 31,999, which is defined under the following key:

```
\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

Stealth spam

Srizbi (like Rustock) is a spam bot. A sophisticated, stealthy and powerful spam machine. Once the trojan takes over NDIS networking, it contacts C&C servers on TCP port 4099 and retrieves spam instructions and configurations. The spam backdoor has a large set of commands that allow the botmaster to define a lot of parameters. A deep analysis of the C&C protocol is beyond the scope of this document, but we can briefly summarize how it works. Each spam session starts with the download of a ZIP package containing the files shown in Table 2.

File	Content
config	Configuration file with all spam parameters (e.g. task_owner, max_mails, max_sim conn, pipeline, subj, etc.)
message	Text/HTML body of the spam message
mlist	List of recipients
000_data22	List of mail domains
001_ncommall	List of names/surnames
002_senderna	List of names/surnames
003_sendersu	List of names/surnames
mxdata	MX records of the domains

Table 2. Configuration files included in the package provided by the Srizbi C&C server.

Srizbi also has an advanced feature that allows the trojan eventually to bypass some badly configured honeypot machines. The trojan does not trust the locally configured DNS server of the infected machine and instead receives all the necessary DNS information as part of the ZIP package. For example, if Srizbi needs to resolve the ‘smtp.acme.org’ server to send spam, it will receive in the package the necessary MX record info for the ‘acme.org’ domain. Any honeypot that simply blocks/redirects DNS resolutions to prevent threats from spamming

will be bypassed by Srizbi because it has a kind of private DNS server over the C&C channel. Srizbi can send image spam in HTML format using English and also Cyrillic Unicode character sets.

Conclusion

Despite all the advanced features implemented for spam and networking, the major weakness of Srizbi is its hiding technique. The rootkit attempts to hide itself by placing the following kernel hooks:

- Inline hook: NtOpenKey, NtEnumerateKey
- \FileSystem\Ntfs driver: IRP_MJ_CREATE, IRP_MJ_DIRECTORY_CONTROL;

These hooks effectively hide the malware's driver file and registry keys, but currently they can easily be detected by common rootkit detectors and can effectively be bypassed to remove the threat from the infected system.

Bibliography

[1] Kasslin, K. Kernel malware: the attack from within. 2006. http://www.f-secure.com/weblog/archives/kasslin_AVAR2006_KernelMalware_paper.pdf.

[2] Symantec. MPack – The Movie. http://www.symantec.com/enterprise/security_response/weblog/2007/06/mpack_the_movie.html.

[3] Malicious code that executes all its code in ring 0. After it is installed into the system by a dropper component (user-mode code or kernel-mode exploit) it will be able to operate by itself.

[4] McPherson, D. When spambots attack – each other! http://asert.arbornetworks.com/2007/07/when-spambots-attack-each-other/.

[5] Hoglund, G.; Butler, J. Rootkits: subverting the Windows kernel. 2005. Upper Saddle River, NJ. Addison-Wesley Professional. 324 pages. ISBN 0-321-29431-9.

Latest articles:

Cryptojacking on the fly: TeamTNT using NVIDIA drivers to mine cryptocurrency

TeamTNT is known for attacking insecure and vulnerable Kubernetes deployments in order to infiltrate organizations' dedicated environments and transform them into attack launchpads. In this article Aditya Sood presents a new module introduced by…

Collector-stealer: a Russian origin credential and information extractor

Collector-stealer, a piece of malware of Russian origin, is heavily used on the Internet to exfiltrate sensitive data from end-user systems and store it in its C&C panels. In this article, researchers Aditya K Sood and Rohit Chaturvedi present a 360…

Fighting Fire with Fire

In 1989, Joe Wells encountered his first virus: Jerusalem. He disassembled the virus, and from that moment onward, was intrigued by the properties of these small pieces of self-replicating code. Joe Wells was an expert on computer viruses, was partly…

Run your malicious VBA macros anywhere!

Kurt Natvig wanted to understand whether it's possible to recompile VBA macros to another language, which could then easily be 'run' on any gateway, thus revealing a sample's true nature in a safe manner. In this article he explains how he recompiled…

Dissecting the design and vulnerabilities in AZORult C&C panels

Aditya K Sood looks at the command-and-control (C&C) design of the AZORult malware, discussing his team's findings related to the C&C design and some security issues they identified during the research.

Bulletin Archive