Rustock.C
When a myth comes true

Frank Boldewin

Hack.Lu 2008

Agenda

- The family's history
- How the myth started rolling
- The hunt for answers
- The loader
- The beast itself
  - Protection layers
  - Inside the rootkit
  - The botnet user mode code
- Lessons learned

FRANK BOLDEWIN'S WWW.RECONSTRUCTER.ORG

The family's history

- Rustock aka Spambot is able to send spam emails and always used top notch rootkit techniques to hide its tracks
- First version (Rustock.A) appeared in Nov 2005, followed by Rustock.B in July 2006
- Code maintained probably only by one Russian guy, who is known as "pe386" or "ntldr" in the underground
- From a reverse engineers point of view, this malware family was always a challenging task and with every evolution step also the degree of analyzing difficulty increased

How the myth started rolling

How the myth started rolling

- In Oct 2007 some people reported that a new Rustock version was seen in the wild
- Unfortunately nobody was able to prove this assertion, because of lack of a sample
- After some weeks without success in hunting, most people in the AV-industry claimed it to be myth…
- At least for 8 months. However in May 2008 the AV-company Dr. Web released a small article, giving a few details about the inner workings of Rustock.c as well as a snapshot showing a .pdb string

```
0010:85D491EB 00 02 00 00 00 50 00 00-00 FC 31 00 00 FC 09 00  .....P....1.....
0010:85D491FB 00 52 53 44 53 38 7E AB-BB 70 A5 8B 4A 91 F3 09  .RSDS8~..p..J...
0010:85D4920B 40 58 2C 6E 7E 03 00 00-00 5A 3A 5C 4E 65 77 50  @X,n~....Z:\NewP
0010:85D4921B 72 6F 6A 65 63 74 73 5C-73 70 61 6D 62 6F 74 5C  rojects\spambot\
0010:85D4922B 72 75 73 74 6F 63 6B 2E-63 5C 64 72 69 76 65 72  rustock.c\driver
0010:85D4923B 5C 61 73 6D 5F 5C 64 72-69 76 65 72 2E 70 64 62  \asm_\driver.pdb
```

The hunt for answers

```
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
cmp     eax, 7Eh
jnz     loc_672B5455
lea     ecx, [esp+110h+LibFileName]
push    104h
push    ecx
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
```

The hunt for answers

- After some further days a few samples of Rustock.C made the rounds and everyone in the industry started analyzing it
- Unfortunately these samples crashed with a BSOD on every box, right after starting the driver (We will see later why)
- Further an unanswered question was its way of infection as well as…
- Where is the dropper code?
- With help of BFK's huge malware DB it was easy to answer the question for the dropper and its infection way
- Recorded traffic revealed that Rustock.C spread through the Iframe-Cash network aka Russian Business Network

The loader

```
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
cmp     eax, 7Eh
jnz     loc_672B5455
lea     ecx, [esp+110h+LibFileName]
push    104h
push    ecx
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
```

Loader code protector properties

- Spaghetti-code with polymorphic jumps, e.g.
  - MOV EDI, offset_18030 / ADD EDI, 0F2F25958h / JMP EDI
  - MOV ECX, 0E3242A4h / JMP DWORD PTR [ECX-0E30C17Ch]
  - MOV EBX, 0Ch / XCHG EBX, [ESP+EBX] / RETN 10h
- RC4 crypted
- aPLib packed
- Unpacked code still spaghetti code structure combined with deliberately unoptimized code, e.g.
  - MOV EAX,1234 -> XOR EAX,EAX / OR EAX,1200 / ADD EAX,34
- Strings like registry paths or IP and port infos are runtime assembled to prevent easy detection
- TDI based kernel mode socket implementation is used for communication
- No extra antidebug, antidump, antivm ...

Loaders inner workings

- Grabs several OS and PCI infos from victims system
  - OS infos are queried from registry
  - PCI infos like PCI to Host Bridge and PCI to ISA Bridge are queried through low level IO port access (CF8/CFC)
- Gathered infos are encrypted with TEA and then send to a fake HTTPS server at 208.66.194.215
- Server crypts the real Rustock.C driver with the victim specific data and sends it back on the same channel
- Loader starts the crypted driver and ends

Send data illustrated

- Unencrypted

```
31 DC 84 9B 25 05 00 00-86 80 90 71 86 80 10 71   1...%.......q...q
C7 78 9B 47 05 00 00 00-01 00 00 00 28 0A 00 00   .x.G........(...
35 00 35 00 33 00 37 00-35 00 2D 00 36 00 34 00   5.5.3.7.5.-.6.4.
30 00 2D 00 33 00 32 00-32 00 38 00 35 00 30 00   0.-.3.2.2.8.5.0.
36 00 2D 00 32 00 33 00-34 00 35 00 36 00 00 00   6.-.2.3.4.5.6...
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

31 DC 84 9B 25 05 00 00: 64bit TimeStampCounter (RDTSC)

86 80 90 71: 7190 = Device, 8086 = Vendor

86 80 10 71: 7110 = Device, 8086 = Vendor

C7 78 9B 47: Install Date = 28.01.08 18:15:35

05 00 00 00-01 00 00 00 28 0A 00 00: 5.1 = CurrentVersion, 2600 = CurrentBuildNumber

ProductId

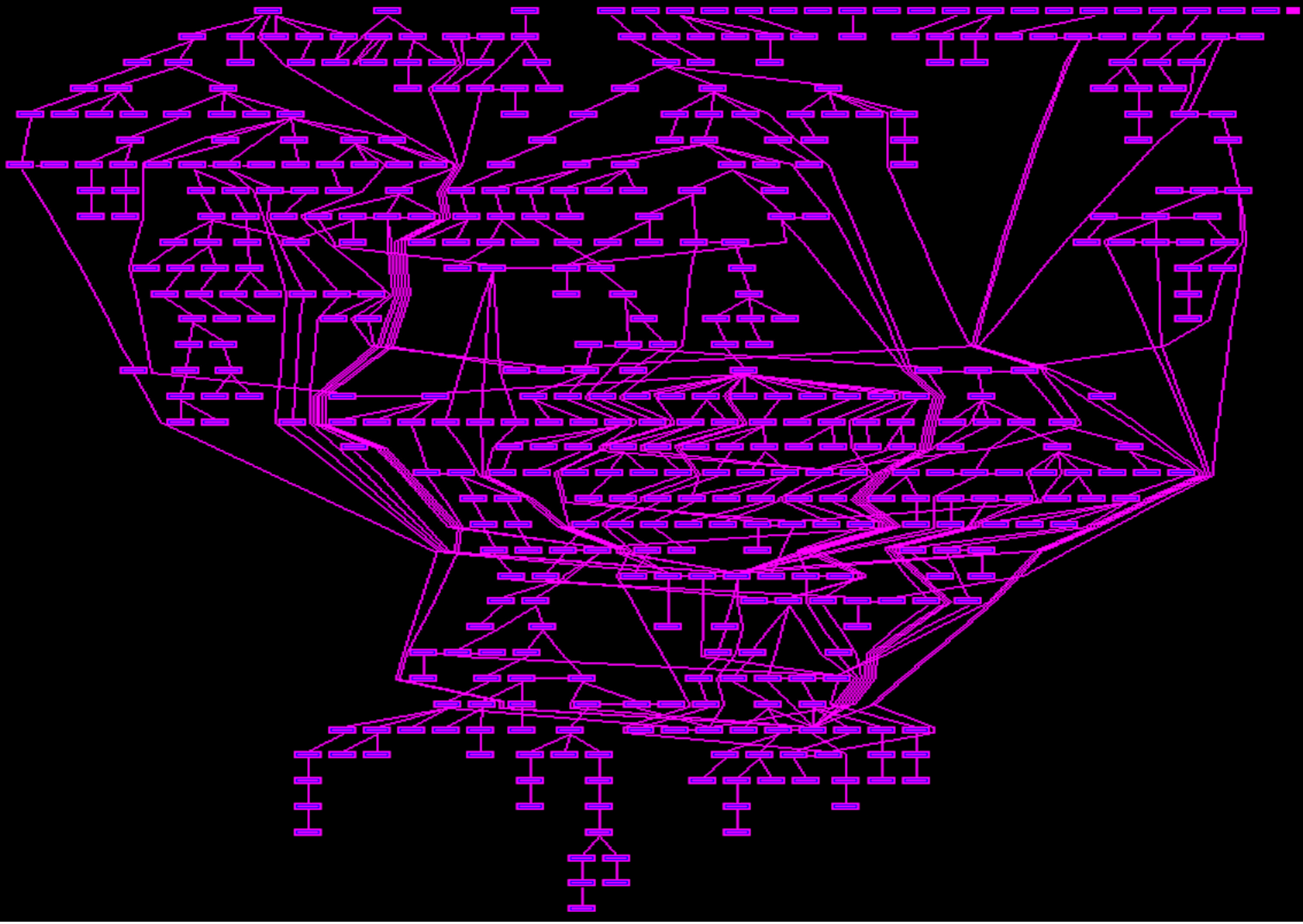
- Encrypted

```
00000000  3b e9 f8 b0 a9 4f 01 d1  58 b9 55 4b 62 18 e8 f5 ;....O.. X.UKb...
00000010  ab 94 7e ec f3 0d ca 1e  21 65 ee ef 45 35 df ce ..~..... !e..E5..
00000020  c8 81 31 e0 77 68 4f d9  d8 a6 24 35 1d 30 63 65 ..1.whO. ..$5.0ce
00000030  da 04 4f 0f 18 6f ec 58  42 ab 3f c3 22 9e b6 9c ..O..o.X B.?."...
00000040  43 ce 79 73 2a b3 e1 27  75 81 11 34 b1 df f8 af C.ys*..' u..4....
00000050  bd a1 38 ac c1 b3 9e 79  56 5f e2 35 e7 12 87 a1 ..8....y V_.5....
00000060  9d cf 3a 3c 47 f9 04 b8  a7 84 46 34 22 68 52 99 ..:<G... ..F4"hR.
```

The beast itself

```
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
cmp     eax, 7Eh
jnz     loc_672B5455
lea     ecx, [esp+110h+LibFileName]
push    104h
push    ecx
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
```

Rustock.C Spaghetti-Code



Protection layer 1

- Easy polymorphic decrypter (Anti AV-signature measure)

```
00010200                  pusha
00010201                  mov     ecx, 287h
00010206                  xor     ebx, ebx
00010208                  xor     edx, edx
0001020A
0001020A loc_1020A:                              ; COD
0001020A                  add     ebx, 39F96BFDh
00010210                  adc     edx, 0
00010213                  dec     ecx
00010214                  jnz     short loc_1020A
00010216                  mov     esi, offset loc_10233
0001021B                  mov     edi, esi
0001021D                  mov     ecx, 0D6EBh
00010222
00010222 loc_10222:                              ; COD
00010222                  mov     eax, ebx
00010224                  shr     eax, 3
00010227                  add     edx, eax
00010229                  xchg    ebx, edx
0001022B                  lodsd
0001022C                  sub     eax, ebx
0001022E                  stosd
0001022F                  dec     ecx
00010230                  jnz     short loc_10222
00010232                  popa
00010233
00010233 loc_10233:                              ; DAT
00010233                  jmp     JumpToSecondLayer
```

Protection layer 2

- Searches the NTOSKRNL base and stores it
- Builds a checksum over its own buffer and encrypts NTOSKRNL image base value with this DWORD
- When trying to find NtQuerySystemInformation the checksum gets recalculated and decrypts the stored NTOSKRNL image base value. If someone changed the code in the meantime, a wrong image base value leads to BSOD
- Imports are found by using 32-bit hash values, instead of function names
- Allocates memory with ExAllocateMemoryPoolWithQuotaTag and copies the majority of its code into this area and directly jumps to layer 3

Protection layer 3

- Overwrites DRx registers
  - DR0-3 (hardware breakpoint detection)
  - DR7 (kernel debugger detection)
- 2nd code checksum trick (modified code leads to BSOD)
- Overwrites whole IDT table with fake handler, for the time of unpacking, to disturb kernel debuggers, which hook INT1 (single stepping + hardware breakpoints) and INT3 (software breakpoints))

```
FakeInterruptHandler:
                push    ebp
                mov     ebp, esp
                sub     esp, 4
                iret
```

- Software BP checks (0xCC)
- Query 8 bytes of PCI information from system (like the loader did)
- Adds 1 dword pre-stored in the buffer and uses these 12 bytes as RC4 decryption key over all 5 PE-sections
- After every PE-section decryption the buffer gets aPLib decompressed

Protection layer 3

- If the 8 bytes of PCI information are different from original ones, decryption fails and system crashes
- Brute forcing the key depends on the machine power and some luck while enumerating through the PCI vendor/device table
- To generate a more random key, 111 empty rounds after RC4init is used
- Imports rebuilding and auto section relocation are also handled in this stage
- Before jumping to the unpacked rootkit code the IDT gets restored to its original state

Inside the rootkit

```
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
cmp     eax, 7Eh
jnz     loc_672B5455
lea     ecx, [esp+110h+LibFileName]
push    104h
push    ecx
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
```

Inside the rootkit

- Unpacked code still spaghetti code structure combined with deliberately unoptimized code
- Checks the presents of kernel debuggers
  - WinDbg (KdDebuggerEnabled)
  - String-scans in memory for NTICE + Syser traces
- Registers a callback routine with KeRegisterBugCheckCallback, which cleans its memory when KeBugCheck happens
- Code checksum routine
- Software breakpoint checks (0xCC)

Inside the rootkit

- Botnet usermode code, stored in the last PE section, gets injected into winlogon.exe or services.exe under VISTA
- Driver infector
  - Infects a random Microsoft driver listed in HKLM\SYSTEM\CurrentControlSet\Control\Safeboot\Minimal registry path
  - Rustock looks for version information strings inside the binaries before infection (scans for “Microsoft Windows”)
- Disinfection is time based, before it infects another MS driver, but can be forced when trying to change an infected binary

Inside the rootkit

- NTOSKRNL hook at _KiFastCallEntry, a very smart way to control all Nt/Zw variants of native functions
- The hook is protecting usermode botnet component to hide its threads and from being read, written, erased or terminated and to have a communication channel through INT 2Eh, between both rings
- The following native functions are being hooked:
  - ZwQuerySystemInformation
  - ZwReadVirtualMemory
  - ZwWriteVirtualMemory
  - ZwProtectVirtualMemory
  - ZwCreateThread
  - ZwTerminateThread
  - ZwOpenThread
  - ZwDuplicateObject
  - ZwDelayExecution
  - ZwSetEvent
  - ZwSetInformationThread
  - ZwResumeThread
  - ZwTerminateProcess
  - ZwCreateUserProcess (only on VISTA)
  - ZwCreateThreadEx (only on VISTA)

Inside the rootkit

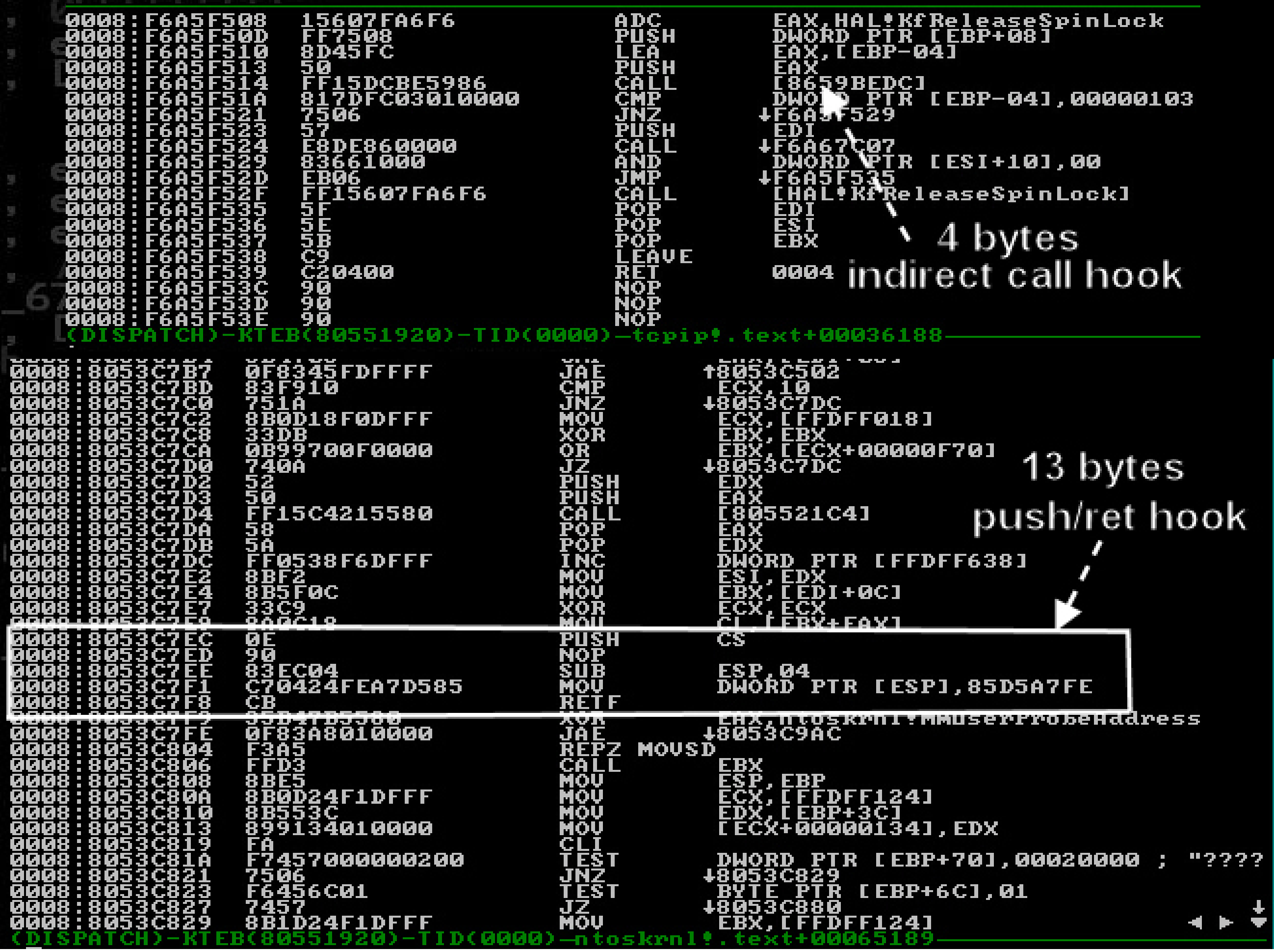
- NTFS.SYS hooks to fake file size and to notice read/writes on infected driver
  - _NtfsFsdWrite
  - _NtfsFsdRead
  - _NtfsFsdSetInformation
  - _NtfsFastQueryFSDInfo
  - _NtfsFsdClose
  - _NtfsFsdCreate
  - _NtfsFsdDispatchWait
  - _NtfsFsdDirectoryControl
- In case of FAT32 the hooks are placed on FASTFAT.SYS

Inside the rootkit

- To prevent local sniffing, also some hooks are placed on IP-based drivers
- TCPIP.SYS
  - _ARPSendData
  - _TCPDispatch
  - _TCPDispatchInternalDeviceControl
  - _ARPClose
  - _FreeARPInterface
  - _ARPRegister
- WANARP.SYS
  - _WANSendPackets

Inside the rootkit

- Two different types of hooks are used (indirect call + push/ret)



The botnet user mode code

```
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
cmp     eax, 7Eh
jnz     loc_672B5455
lea     ecx, [esp+110h+LibFileName]
push    104h
push    ecx
push    2
call    sub_672B3730
add     esp, 0Ch
test    eax, eax
jnz     short loc_672B5428
lea     edx, [esp+110h+LibFileName]
push    edx
call    sub_672B35F0
mov     edi, off_672CA058
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+114h+LibFileName]
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
```

The botnet user mode code

- The first variants had the name botdll.dll and send spam the classic way using port 25 (SMTP)
- But as more and more SMTP gateways successfully detect such spam bots, a new user mode payload was distributed in march 2008 and changed to HTTP-mode spamming over hotmail with stolen accounts (hotsend.dll)
- Spam templates are downloaded from the C&C server, which are temporarily stored as tmpcode.bin
- Currently it is unknown what malware steals the hotmail accounts involved in spamming
- To communicate with the kernel INT 2Eh is used, to inform about new tasks, e.g. self-disinfection or a new C&C

Lessons learned

- Kernel mode driver could easily host other user mode payload, e.g. banking trojans, DDoS client …
- Without automated deobfuscation scripts, it would be nearly impossible to analyze the code
- Brute forcing would have been impossible, if a stronger encryption had been applied
- Disinfection wouldn't be that easy, if the original driver in the last PE-section would have been better crypted

Questions?

Thanks for good discussions and review fly to:

UG North

Elia Florio

Sergei Shevchenko

Lukasz Kwiatek