Several samples of the Torpig/Anserin/Sinowal family of Trojans detect virtual machine monitors and abort infection when present. This technique involving the interrupt descriptor table location has been discussed in many articles, including one of my own called Using IDT for VMM Detection. Please read this before moving on, as information required for understanding where the following code came from is in that document and will not be repeated here.

The samples of this malware that we have gathered in the past are typically packed with UPX. Below is a hex-view/disassembly combination of the instructions issued by these Trojans to detect the VMM. This code is only available (in this form) after unpacking the original file.

```
call codebegin
test eax,eax
jnz vmdetected
...
getsidt:
      51                // push ecx
      51                // push ecx
      0F 01 4C 24 00    // sidt qword [esp]
      8B 44 24 02       // mov eax,dword ptr [esp+2]
      59                // pop ecx
      59                // pop ecx
      C3                // retn
codebegin:
      E8 ED FF FF FF    // call getsidt
      25 00 00 00 FF    // xor eax,0x0FF000000
      33 C9             // xor ecx,ecx
      3D 00 00 00 80    // cmp eax,0x80000000
      0F 95 C1          // setnz cl
      8B C1             // mov eax,eax
      C3                // retn
```

This string of bytes can be used as an IDS signature for detecting un-packed versions of the Torpig/Anserin/Sinowal Trojans.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"BLEEDING-EDGE VMM
Detecting Torpig/Anserin/Sinowal Trojan"; flow:to_client,established;
content:"|51 51 0F 01 4C 24 00 8B 44 24 02 59 59 C3 E8 ED FF FF FF 25
00 00 00 FF 33 C9 3D 00 00 00 80 0F 95 C1 8B C1 C3|"; classtype:trojan-
activity; sid:20060810; rev:1;)
```

Unfortunately, the malware probably won't ever be seen in un-packed form in the wild, so this rule is of limited value. To generate something a little more useful, a signature needs to be created based on the bytes in the packed file that correspond to the ones shown above. Lucky for us, UPX is natively reversible and quite simple to trace.

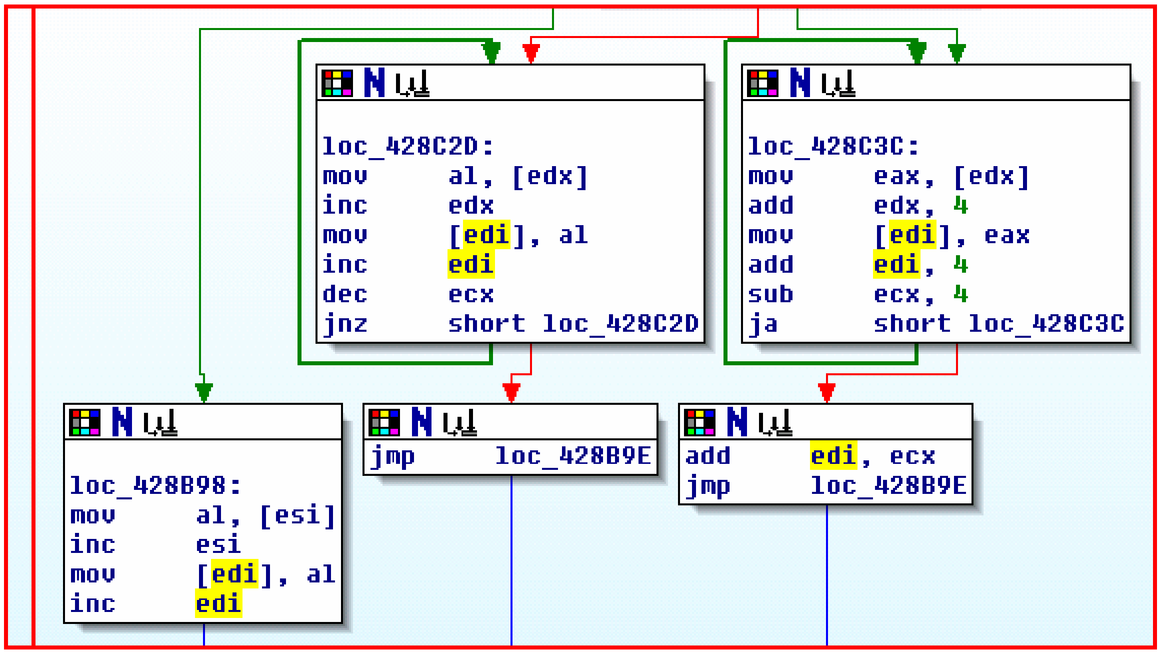
When the packed file begins execution, it does the following:

```
UPX1:00428B80 start           proc near
UPX1:00428B80
UPX1:00428B80 var_AC  = byte ptr -0ACh
UPX1:00428B80
UPX1:00428B80         pusha
UPX1:00428B81         mov     esi, offset dword_417000
UPX1:00428B86         lea     edi, [esi-16000h]
```

This loads the address to the start of the UPX1 section into esi, which is where the packed bytes reside. Then it loads the address to the start of the UPX0 section into edi, which is where the unpacked bytes will be written. After all the replacement has been completed, the code jumps to a location within the new UPX0 section and proceeds with normal execution of the program.

```
UPX1:00428D0B                 jmp     near ptr dword_401628
UPX1:00428D0B start           endp
```

So in order to find out the original values of the bytes that compose the IDT functions, first the exact locations where overwrites occur were marked in IDA. There are only three possible places, two of which move a byte from al and one that moves a dword from eax:



```
loc_428C2D:
mov     al, [edx]
inc     edx
mov     [edi], al
inc     edi
dec     ecx
jnz     short loc_428C2D
```

```
loc_428C3C:
mov     eax, [edx]
add     edx, 4
mov     [edi], eax
add     edi, 4
sub     ecx, 4
ja      short loc_428C3C
```

```
loc_428B98:
mov     al, [esi]
inc     esi
mov     [edi], al
inc     edi
```

```
jmp     loc_428B9E
```

```
add     edi, ecx
jmp     loc_428B9E
```

The last required piece of information is the address where the IDT function(s) begin in the unpacked file. As shown in the disassembly below, this would be 0x0040166A.

```
.text:0040166A ; ¦¦¦¦ S U B R O U T I N E ¦¦¦¦
.text:0040166A
.text:0040166A
.text:0040166A sub_40166A      proc near   ; CODE XREF: sub_401678p
.text:0040166A
.text:0040166A var_8  = qword ptr -8
.text:0040166A
.text:0040166A        push    ecx
.text:0040166B        push    ecx
.text:0040166C        sidt    [esp+8+var_8]
.text:00401671        mov     eax, dword ptr [esp+8+var_8+2]
.text:00401675        pop     ecx
.text:00401676        pop     ecx
.text:00401677        retn
.text:00401677 sub_40166A      endp
.text:00401677
.text:00401678
.text:00401678 ; ¦¦¦¦ S U B R O U T I N E ¦¦¦¦
.text:00401678
.text:00401678
.text:00401678 sub_401678 proc near  ; CODE XREF: _func+11p
.text:00401678        call    sub_40166A
.text:0040167D        and     eax, 0FF000000h
.text:00401682        xor     ecx, ecx
.text:00401684        cmp     eax, 80000000h
.text:00401689        setnz   cl
.text:0040168C        mov     eax, ecx
.text:0040168E        retn
.text:0040168E sub_401678 endp
```

Now the program can be executed until this start address is in edi, and we can take the byte pointed to by esi-1 (its incremented before overwriting) as the start of the packed signature. Once the retn instruction at 0x0040168E is written, the byte pointed to by esi-1 again will be the end of the signature.

This allows for creation of the following IDS signature based on the packed version of the file:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"BLEEDING-EDGE (UPX)
VMM Detecting Torpig/Anserin/Sinowal Trojan";
flow:to_client,established; content:"|51 51 0F 01 27 00 C1 FB B5 D5  35
02 E2 C3 D1 66 25 32 BD 83 7F B7 4E 3D 06 80 0F 95 C1 8B C1 C3|";
classtype:trojan-activity; sid:20060810; rev:1;)
```

Caveats and results of signature QA testing

There are countless variations of the VMM detecting code, so bypassing the rules are trivial. For example, the malware author could use different registers or insert trash instructions to break the contiguity of the signature bytes. Alas, the proposed signatures are good for detecting the samples found in the wild across a period of several months. For QA testing, the signatures were tested against the original packed file, an unpacked version of the file, and a version of the file after having re-packed it with UPX. Since the main executable is essentially a dropper, the signatures were also tested successfully against two of the executables that are extracted from its .data section during runtime, both of which detect the presence of VMMs in the same manner.

Anti-virus vendors refer to the code by several different names:

```
ANTIVIR     7.1.1.16/20060909 FOUND [TR/PSW.SINOWAL.AQ.30]
AVAST 4.7.844.0/20060908      FOUND [WIN32:TROJANO-P]
AVG   386/20060908      FOUND [PSW.GENERIC2.GEU]
BITDEFENDER 7.2/20060910      FOUND [TROJAN.PWS.SINOWAL.AK]
DRWEB  4.33/20060910    FOUND [TROJAN.PWS.SNAP]
ETRUST-VET  30.3.3070/20060909      FOUND [WIN32/ANSERIN!GENERIC]
EWIDO 4.0/20060910      FOUND [TROJAN.SINOWAL.AQ]
FORTINET    2.77.0.0/20060909 FOUND [W32/TORPIG.AQ!TR.PWS]
KASPERSKY   4.0.2.24/20060910 FOUND [TROJAN-PSW.WIN32.SINOWAL.AQ]
MICROSOFT   1.1560/20060909   FOUND [PWS:WIN32/SINOWAL!FEE9]
NOD32V2     1.1746/20060908   FOUND [A VARIANT OF WIN32/PSW.SINOWAL]
NORMAN      5.90.23/20060908  FOUND [W32/SINOWAL.TH]
PANDA 9.0.0.4/20060909 FOUND [SUSPICIOUS FILE]
SOPHOS      4.09.0/20060910   FOUND [TROJ/TORPIG-BH]
VBA32 3.11.1/20060910   FOUND [TROJAN-PSW.WIN32.SINOWAL.AQ]
VIRUSBUSTER 4.3.7:9/20060910  FOUND [TROJAN.DR.SINOWAL.GEN.8]
```