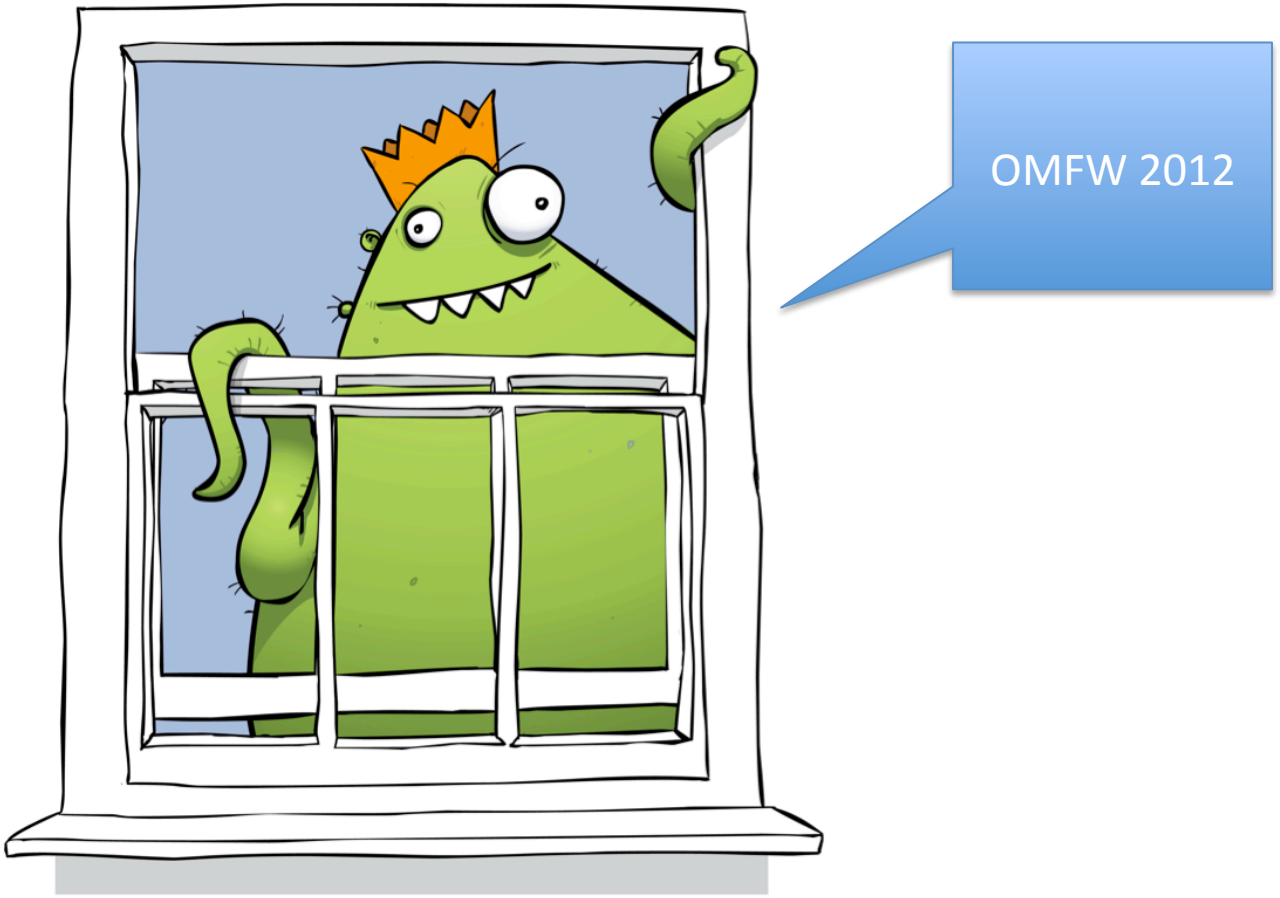
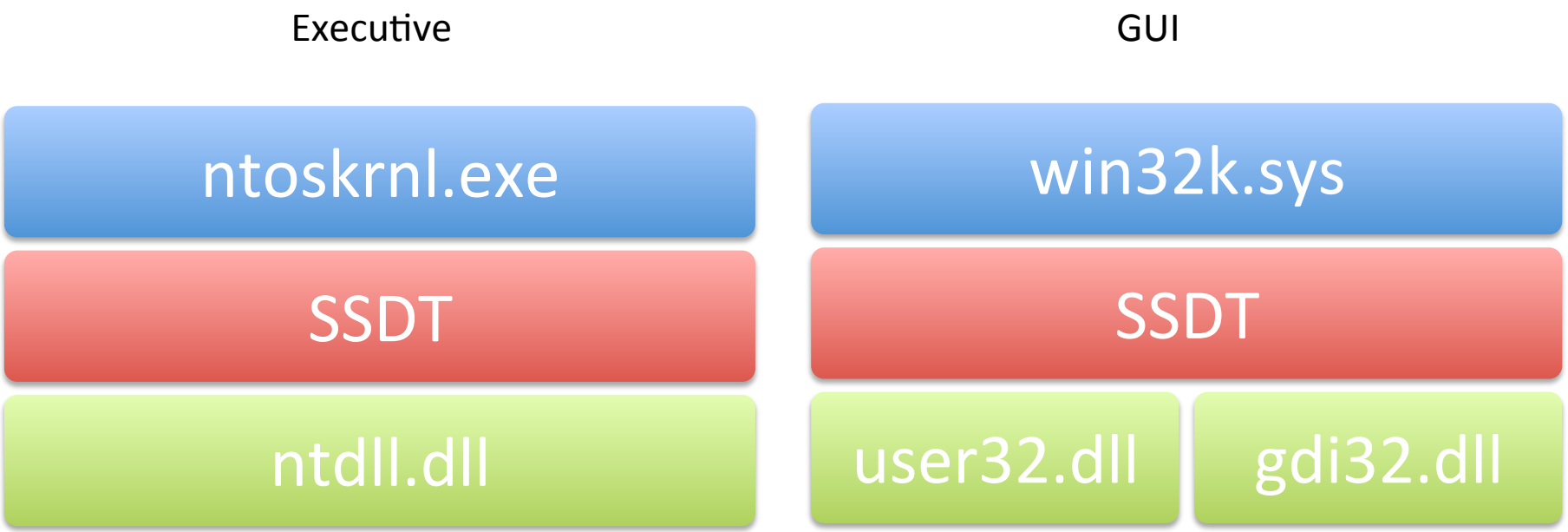


Malware in the Windows GUI Subsystem

The GUI Subsystem

- Everything you see (desktops, windows, buttons, scroll bars, edit boxes, etc)
- Everything you do *can and will be used against you* (keyboard input, mouse movements, window arrangements)

Where is it implemented



Relative ~~complexity~~?

```
$ python vol.py -f grrcon.img ssdt
Volatile Systems Volatility Framework 2.1_alpha
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at 804e26a8 with 284 entries
  Entry 0x0000: 0x8058fdf5 (NtAcceptConnectPort) owned by ntoskrnl.exe
  Entry 0x0001: 0x805790f1 (NtAccessCheck) owned by ntoskrnl.exe
  Entry 0x0002: 0x80587999 (NtAccessCheckAndAuditAlarm) owned by ntoskrnl.exe
  Entry 0x0003: 0x80591130 (NtAccessCheckByType) owned by ntoskrnl.exe
  Entry 0x0004: 0x8058da83 (NtAccessCheckByTypeAndAuditAlarm) owned by ntoskrnl.exe
[snip]
SSDT[1] at bf999b80 with 667 entries
  Entry 0x1000: 0xbf935f7e (NtGdiAbortDoc) owned by win32k.sys
  Entry 0x1001: 0xbf947b29 (NtGdiAbortPath) owned by win32k.sys
  Entry 0x1002: 0xbf88ca52 (NtGdiAddFontResourceW) owned by win32k.sys
  Entry 0x1003: 0xbf93f6f0 (NtGdiAddRemoteFontToDC) owned by win32k.sys
  Entry 0x1004: 0xbf949140 (NtGdiAddFontMemResourceEx) owned by win32k.sys
  Entry 0x1005: 0xbf936212 (NtGdiRemoveMergeFont) owned by win32k.sys
[snip]
```

Why?

- Unknown != Boring
- Undocumented by Microsoft
  - Alex Ionescu (Shatter Attacks, ReactOS, etc)
  - Tarjei Mandt (Smashing the Atom, Windows Hooks of Death, 44 vulns in win32k.sys MS11-034 & MS11-054)
  - 2 pages in Windows Internals (~1200 pages)
- Unexplored from a forensics and malware perspective

Challenges

- Type information only available for Win7
  - But we must support XP, 2003, Vista, 2008, 2008 R2
  - RE and time extensive
- Many structures are not even available in Win7
  - Cool, more RE
  - Structure names, sizes, members, member types, etc
- We want Volatility to remain cross-platform
  - No PDBs ~~dbghelp.dll~~
  - Must find global variables, list heads, with heuristics

Win32 Suite

- 14 plugins
- VTypes for all 32- and 64-bit Windows profiles
- APIs (object classes)
- ~20 new objects
- $HOME = volatility/plugins/gui
- Released as of today – see Volatility 2.2

Sessions

- Represent user logon sessions; container for processes and objects
- RDP, Terminal Services, Fast-User Switching
- Session 0 isolation since Vista
- _EPROCESS.Session
- _MM_SESSION_SPACE
  - SessionId – unique ID
  - ProcessList - _EPROCESS (*psxview 2.3-devel)
  - ImageList - _IMAGE_ENTRY_IN_LIST
  - PageDirectory - _MMPTE

Psxview += session, deskthrd

```
$ python vol.py -f ~/Desktop/memory/WinXPSP3x86/ds_fuzz_hidden_proc.img psxview
Volatile Systems Volatility Framework 2.3_alpha
Offset(P)  Name                PID pslist psscan thrdproc pspcdid csrss session deskthrd
---------- ---------------- ------ ------ ------ -------- ------- ----- ------- --------
0x01a2b100 winlogon.exe        620 True   True   True     True    True  True    True
0x01a3d360 svchost.exe         932 True   True   True     True    True  True    True
0x018a13c0 VMwareService.e    1756 True   True   True     True    True  True    True
0x018e75e8 spoolsv.exe        1648 True   True   True     True    True  True    True
0x019dbc30 lsass.exe           684 True   True   True     True    True  True    True
0x0184e3a8 wscntfy.exe         560 True   True   True     True    True  True    True
0x018af860 VMwareTray.exe     1896 True   True   True     True    True  True    True
0x01a4bc20 network_listene    1696 False  False  True     True    True  True    True
0x01843b28 wuauclt.exe        1372 True   True   True     True    True  True    True
0x01a59d70 svchost.exe         844 True   True   True     True    True  True    True
0x018af448 VMwareUser.exe     1904 True   True   True     True    True  True    True
0x019f7da0 svchost.exe        1164 True   True   True     True    True  True    True
[snip]
```

Session Memory

- Process memory is private, ~~*but all processes have the same view of kernel mode….*~~

```
Welcome to volshell! Current memory image is:
file:///Volumes/Storage/memory/Win2003SP2x86/active.001
To get help, type 'hh()'
>>> import volatility.plugins.gui.sessions as sessions
>>> import hashlib
>>> kernel_space = self.addrspace
>>> for session in sessions.SessionsMixin().session_spaces(kernel_space):
...     data = session.obj_vm.zread(0xbc6f2000, 0x1000)
...     print session.SessionId, hashlib.md5(data).hexdigest()
...
0  620f0b67a91f7f74151bc5be745b7110
1  75d653f333d2ca194e369fe4fa23878e
2  35d50526c482d625372a4561ddc326a0
```

What's in session memory?

- win32k!gahti – array of tagHANDLETYPEINFO
- Similar to nt!_OBJECT_TYPE


```
>>> dt("tagHANDLETYPEINFO")
'tagHANDLETYPEINFO' (16 bytes)
0x0    : fnDestroy                    ['pointer', ['void']]
0x8    : dwAllocTag                   ['String', {'length': 4}]
0xc    : bObjectCreateFlags           ['Flags', {'target':
'unsigned char', 'bitmap': {'OCF_VARIABLESIZE': 7,
'OCF_DESKTOPHEAP': 4, 'OCF_THREADOWNED': 0, 'OCF_SHAREDHEAP': 6,
'OCF_USEPOOLIFNODESKTOP': 5, 'OCF_USEPOOLQUOTA': 3,
'OCF_MARKPROCESS': 2, 'OCF_PROCESSOWNED': 1}}]
```

USER Object Types

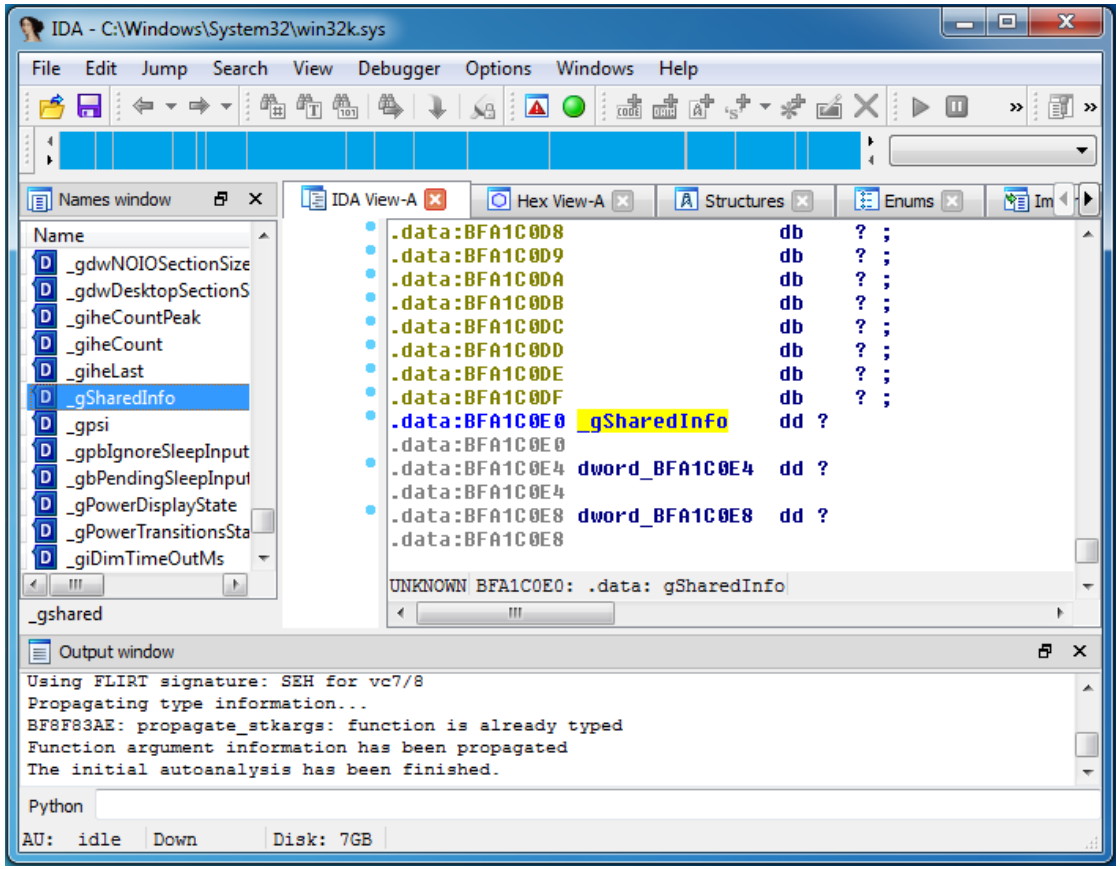
- TYPE_WINDOW => tagWND
- TYPE_HOOK => tagHOOK, etc…..

This is the gahti plugin's output

```
Session  Type                 Tag      fnDestroy          Flags
-------- -------------------- -------- ------------------ -----
   0     TYPE_FREE                     0x0000000000000000
   0     TYPE_WINDOW          Uswd     0x0000f960000df660 OCF_DESKTOPHEAP, OCF_THREADOWNED,
OCF_USEPOOLIFNODESKTOP, OCF_USEPOOLQUOTA
   0     TYPE_MENU                     0x0000f960000e15ac OCF_DESKTOPHEAP, OCF_PROCESSOWNED
   0     TYPE_CURSOR          Uscu     0x0000f960000e41a0 OCF_MARKPROCESS, OCF_PROCESSOWNED, OCF_USEPOOLQUOTA
   0     TYPE_SETWINDOWPOS    Ussw     0x0000f960000a92b4 OCF_THREADOWNED, OCF_USEPOOLQUOTA
   0     TYPE_HOOK                     0x0000f9600011e5c8 OCF_DESKTOPHEAP, OCF_THREADOWNED
   0     TYPE_CLIPDATA        Uscb     0x0000f9600010c5ac
   0     TYPE_CALLPROC                 0x0000f9600010c5cc OCF_DESKTOPHEAP, OCF_PROCESSOWNED
   0     TYPE_ACCELTABLE      Usac     0x0000f9600010c5cc OCF_PROCESSOWNED, OCF_USEPOOLQUOTA
   0     TYPE_DDEACCESS       Usd9     0x0000f9600010c5ac OCF_THREADOWNED, OCF_USEPOOLQUOTA
   0     TYPE_DDECONV         UsdA     0x0000f9600014a1fc OCF_THREADOWNED, OCF_USEPOOLQUOTA
   0     TYPE_DDEXACT         UsdB     0x0000f9600014a22c OCF_THREADOWNED, OCF_USEPOOLQUOTA
   0     TYPE_MONITOR         Usdi     0x0000f9600015a76c OCF_SHAREDHEAP
   0     TYPE_KBDLAYOUT       Uskb     0x0000f96000147c28
   0     TYPE_KBDFILE         Uskf     0x0000f960001477c8
   0     TYPE_WINEVENTHOOK    Uswe     0x0000f9600011f148 OCF_THREADOWNED
   0     TYPE_TIMER           Ustm     0x0000f960000946dc OCF_PROCESSOWNED
   0     TYPE_INPUTCONTEXT    Usim     0x0000f960000dc660 OCF_DESKTOPHEAP, OCF_THREADOWNED
   0     TYPE_HIDDATA         Usha     0x0000f96000162a34 OCF_THREADOWNED
   0     TYPE_DEVICEINFO      UsDI     0x0000f96000068cd4
   0     TYPE_TOUCH           Ustz     0x0000f9600010c5cc OCF_THREADOWNED
   0     TYPE_GESTURE         Usgi     0x0000f9600010c5cc OCF_THREADOWNED
```

USER Handle Table

- win32k!_gSharedInfo - tagSHAREDINFO



USER Handle Table

tagSHAREDINFO.psi.cbHandleTable / tagSHAREDINFO.psi.cHandleEntries == tagSHAREDINFO.HeEntrySize

```
>>> dt("tagSHAREDINFO")
'tagSHAREDINFO' (568 bytes)
0x0   : psi                    ['pointer64', ['tagSERVERINFO']]
0x8   : aheList                ['pointer64', ['_HANDLEENTRY']]
0x10  : HeEntrySize            ['unsigned long']
0x18  : pDispInfo              ['pointer64', ['tagDISPLAYINFO']]
0x20  : ulSharedDelta          ['unsigned long long']
0x28  : awmControl             ['array', 31, ['_WNDMSG']]
0x218 : DefWindowMsgs          ['_WNDMSG']
0x228 : DefWindowSpecMsgs      ['_WNDMSG']

>>> dt("tagSERVERINFO")
'tagSERVERINFO' (4640 bytes)
0x0   : dwSRVIFlags              ['unsigned long']
0x8   : cHandleEntries           ['unsigned long long']
[............]
0x350 : cbHandleTable            ['unsigned long']
```

Handle Entries

```
>>> dt("_HANDLEENTRY")
'_HANDLEENTRY' (24 bytes)
0x0    : phead                          ['pointer64', ['_HEAD']]
0x8    : pOwner                         ['pointer64', ['void']]
0x10   : bType                          ['Enumeration', {'target': 'unsigned char',
'choices': {0: 'TYPE_FREE', 1: 'TYPE_WINDOW', 2: 'TYPE_MENU'...[snip]
0x11   : bFlags                         ['unsigned char']
0x12   : wUniq                          ['unsigned short']
```

```
>>> dt("_HEAD")
'_HEAD' (16 bytes)
0x0    : h                              ['pointer64', ['void']]
0x8    : cLockObj                       ['unsigned long']
>>> dt("_THRDESKHEAD")
'_THRDESKHEAD' (40 bytes)
0x0    : h                              ['pointer64', ['void']]
0x8    : cLockObj                       ['unsigned long']
0x10   : pti                            ['pointer64', ['tagTHREADINFO']]
0x18   : rpdesk                         ['pointer64', ['tagDESKTOP']]
0x20   : pSelf                          ['pointer64', ['unsigned char']]
>>> dt("_PROCDESKHEAD")
'_PROCDESKHEAD' (40 bytes)
0x0    : h                              ['pointer64', ['void']]
0x8    : cLockObj                       ['unsigned long']
0x10   : hTaskWow                       ['unsigned long']
0x18   : rpdesk                         ['pointer64', ['tagDESKTOP']]
0x20   : pSelf                          ['pointer64', ['unsigned char']]
```

Associations

- If thread owned:
  - _HANDLEENTRY.phead.pti.pEThread (_ETHREAD)
- If process owned:
  - _HANDLEENTRY.pOwner.Process (_EPROCESS)
  - _HANDLEENTRY.phead.pti.ppi.Process

Sessions Plugin

```
$ python vol.py -f rdp.dmp --profile=Win2003SP2x86 sessions
Volatile Systems Volatility Framework 2.3_alpha
[snip]
**************************************************
Session(V): f79ff000 ID: 2 Processes: 10
PagedPoolStart: bc000000 PagedPoolEnd bc3fffff
 Process: 7888 csrss.exe 2012-05-23 02:51:43
 Process: 3272 winlogon.exe 2012-05-23 02:51:43
 Process: 6772 rdpclip.exe 2012-05-23 02:52:00
 Process: 5132 explorer.exe 2012-05-23 02:52:00
 Process: 5812 PccNTMon.exe 2012-05-23 02:52:01
 Process: 3552 VMwareTray.exe 2012-05-23 02:52:01
 Process: 5220 mbamgui.exe 2012-05-23 02:52:02
 Process: 4576 ctfmon.exe 2012-05-23 02:52:02
 Process: 5544 cmd.exe 2012-05-23 02:52:09
 Process: 6236 notepad.exe 2012-05-23 03:20:35
 Image: 0x8a2fecc0, Address bf800000, Name: win32k.sys
 Image: 0x877d0478, Address bf9d3000, Name: dxg.sys
 Image: 0x8a1bdf38, Address bff60000, Name: RDPDD.dll
 Image: 0x8771a970, Address bfa1e000, Name: ATMFD.DLL
```

RDP or console user?

What's the purpose of this?

Cmdscan/Consoles

```
$ python vol.py -f rdp.dmp --profile=Win2003SP2x86 consoles

ConsoleProcess: csrss.exe Pid: 7888
Console: 0x4c2404 CommandHistorySize: 50
AttachedProcess: cmd.exe Pid: 5544 Handle: 0x25c
Cmd #6 at 0xf41b20: ftp xxxxx.com
Cmd #7 at 0xf41948: notepad xxxxx.log
Cmd #8 at 0x4c2388: notepad xxxxx.log
Cmd #9 at 0xf43e70: ftp xxxxx.com
Cmd #10 at 0xf43fb0: dir
Cmd #11 at 0xf41550: notepad xxxxx.log

C:\WINDOWS\system32\xxxxx\sample>ftp xxxxx.com
Connected to xxxxx.com.
220 Microsoft FTP Service
User (xxxxx.com:(none)): xxxxx
331 Password required for xxxxx.
Password:
230 User xxxxx logged in.
ftp> cd logs
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
05-22-12  09:34AM       <DIR>          xxxxx
226 Transfer complete.
ftp: 51 bytes received in 0.00Seconds 51000.00Kbytes/sec.
```

PIDs from RDP session…

See everything the attacker sees

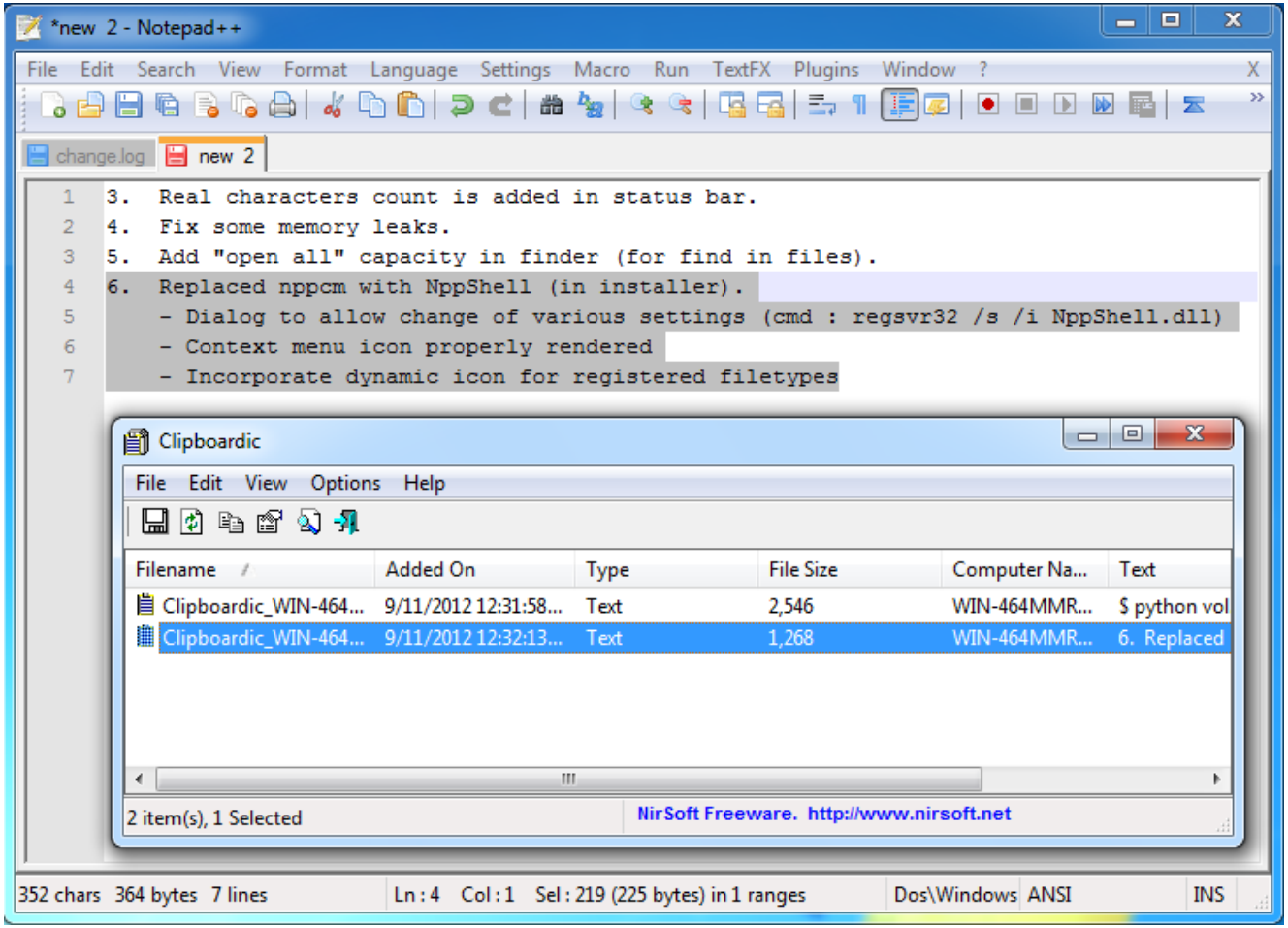
Window Stations

- Security boundary for processes and desktops
- Securable objects just like _EPROCESS, _ETHREAD, _KMUTANT, _FILE_OBJECT, etc
- WinSta0 is interactive. Name comes from _OBJECT_HEADER
- tagWINDOWSTATION
  - dwSessionId – matches session ID
  - rpwinstaNext – list of other window stations
  - rpdeskList – desktops (i.e. WinSta0\Default)
  - iClipSequenceNumber – count of copy & paste
  - pGlobalAtomTable – atom table pointer
  - pClipBase – tagCLIP array (clipboard handles)
- win32k!_grpWinStaList but we use PoolScanner (“Wind”)

Clipboard Snooping

- Hook SetClipboardData – obvious modification
- GetClipboardData loop – can lock clipboard access
- SetClipboardViewer / AddClipboardFormatListener – not no obvious but still detectable
- Example:
  - Copy & Paste from host to guest
  - Copy & Paste in host (1 app to another app)

Nirsoft Clipboardic



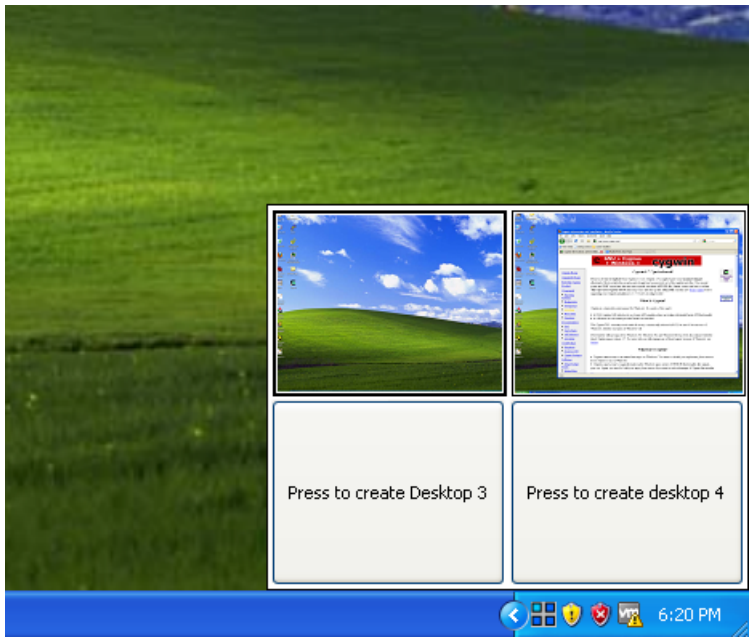
Caught

```
$ python vol.py -f memory.dmp --profile=Win7SP1x86 wndscan
**************************************************
WindowStation: 0x7ea45d00, Name: WinSta0, Next: 0x0
SessionId: 1, AtomTable: 0x93b107f0, Interactive: True
Desktops: Default, Disconnect, Winlogon
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0xfea5db80 3616 Clipboardic.ex
cNumClipFormats: 4, iClipSerialNumber: 11
pClipBase: 0xfccb2be8, Formats: CF_UNICODETEXT,Unknown choice 8192,CF_TEXT,Unknown choice 197569
```

```
$ python vol.py -f memory.dmp --profile=Win7SP1x86 wintree | grep CLIPBRDWNDCLASS
Volatile Systems Volatility Framework 2.2_alpha
.#10062  explorer.exe:372 CLIPBRDWNDCLASS
.#100f0  explorer.exe:372 CLIPBRDWNDCLASS
.#1011e  vmtoolsd.exe:2224 CLIPBRDWNDCLASS
.#1014a  SnagIt32.exe:2300 CLIPBRDWNDCLASS
.#4002c  vmtoolsd.exe:2224 CLIPBRDWNDCLASS
.#10288  notepad++.exe:3140 CLIPBRDWNDCLASS
.#1032c  Clipboardic.ex:3616 CLIPBRDWNDCLASS
```

Desktops

- Container for application windows
- Just like Linux and OSX
  - no obvious way to use
  - SysInternals Desktops
  - CreateDesktop
  - SwitchDesktop

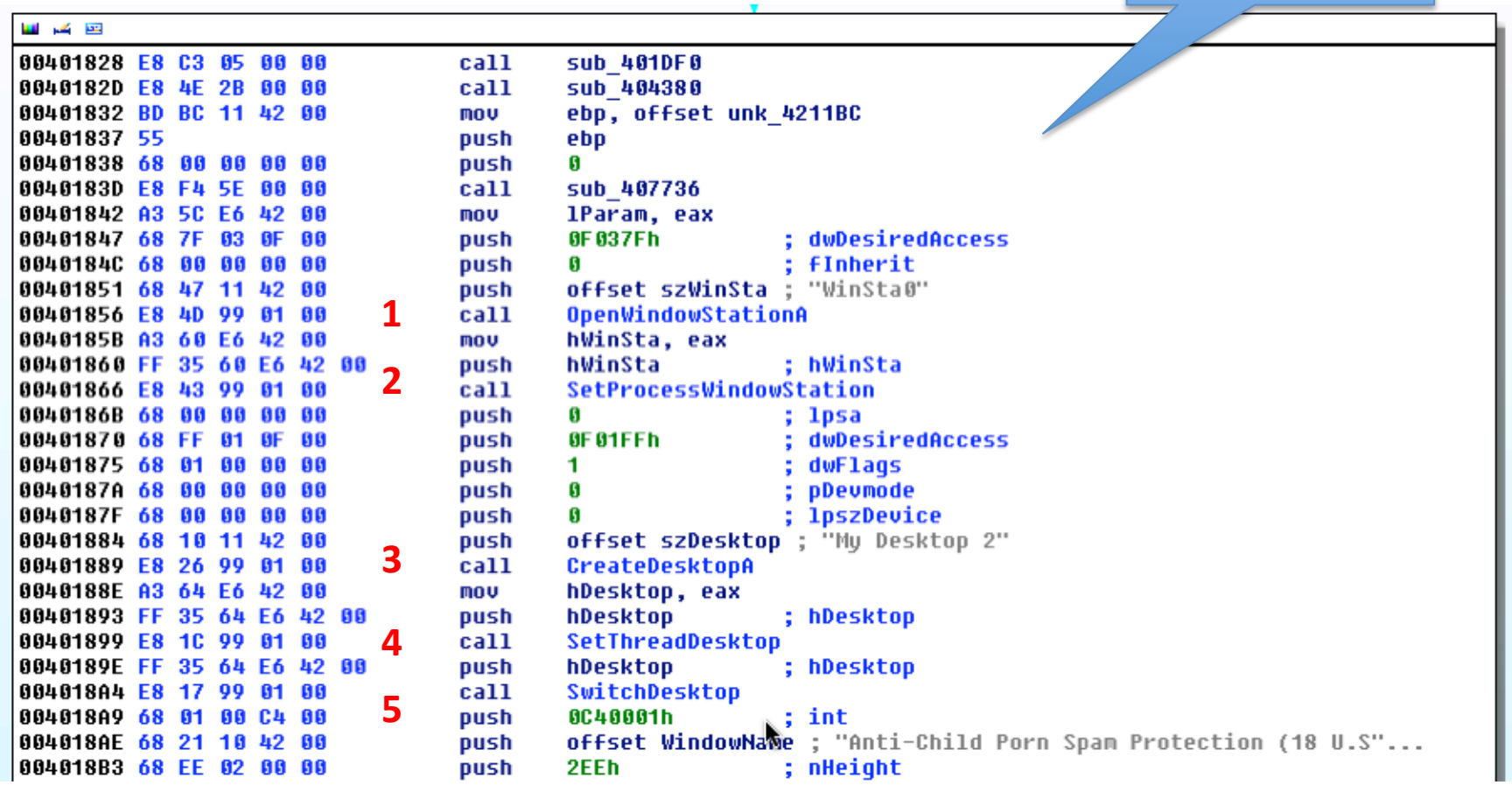


Desktops

- tagDESKTOP
  - dwSessionId – matches Session ID
  - pDeskInfo – tagDESKINFO contains global hook info and pointer to foreground window
  - rpdeskNext – desktop list
  - rpwinstaParent – owning tagWINDOWSTATION
  - pheapDesktop – the _HEAP
  - PtiList – tagTHREADINFO (derive _ETHREAD)

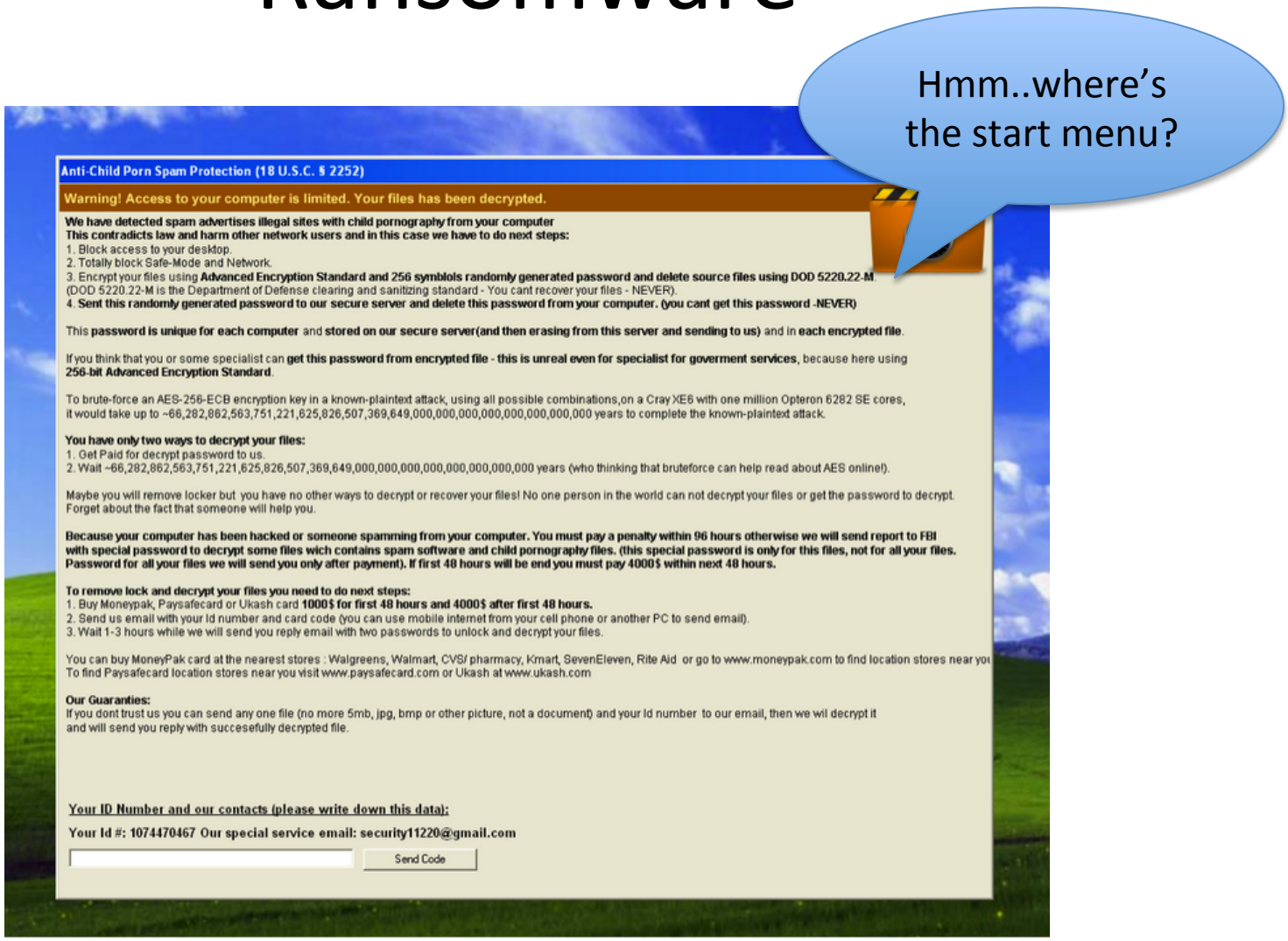
Ransomware

Starts at boot w/ run key



```
00401828 E8 C3 05 00 00          call    sub_401DF0
0040182D E8 4E 2B 00 00          call    sub_404380
00401832 BD BC 11 42 00          mov     ebp, offset unk_4211BC
00401837 55                      push    ebp
00401838 68 00 00 00 00          push    0
0040183D E8 F4 5E 00 00          call    sub_407736
00401842 A3 5C E6 42 00          mov     lParam, eax
00401847 68 7F 03 0F 00          push    0F037Fh         ; dwDesiredAccess
0040184C 68 00 00 00 00          push    0               ; fInherit
00401851 68 47 11 42 00          push    offset szWinSta ; "WinSta0"
00401856 E8 4D 99 01 00    1     call    OpenWindowStationA
0040185B A3 60 E6 42 00          mov     hWinSta, eax
00401860 FF 35 60 E6 42 00       push    hWinSta         ; hWinSta
00401866 E8 43 99 01 00    2     call    SetProcessWindowStation
0040186B 68 00 00 00 00          push    0               ; lpsa
00401870 68 FF 01 0F 00          push    0F01FFh         ; dwDesiredAccess
00401875 68 01 00 00 00          push    1               ; dwFlags
0040187A 68 00 00 00 00          push    0               ; pDevmode
0040187F 68 00 00 00 00          push    0               ; lpszDevice
00401884 68 10 11 42 00          push    offset szDesktop ; "My Desktop 2"
00401889 E8 26 99 01 00    3     call    CreateDesktopA
0040188E A3 64 E6 42 00          mov     hDesktop, eax
00401893 FF 35 64 E6 42 00       push    hDesktop        ; hDesktop
00401899 E8 1C 99 01 00    4     call    SetThreadDesktop
0040189E FF 35 64 E6 42 00       push    hDesktop        ; hDesktop
004018A4 E8 17 99 01 00          call    SwitchDesktop
004018A9 68 01 00 C4 00    5     push    0C40001h        ; int
004018AE 68 21 10 42 00          push    offset WindowName ; "Anti-Child Porn Spam Protection (18 U.S"...
004018B3 68 EE 02 00 00          push    2EEh            ; nHeight
```

Ransomware


Hmm..where's the start menu?
Anti-Child Porn Spam Protection (18 U.S.C. § 2252)
Warning! Access to your computer is limited. Your files has been decrypted.
We have detected spam advertises illegal sites with child pornography from your computer
This contradicts law and harm other network users and in this case we have to do next steps:
1. Block access to your desktop.
2. Totally block Safe-Mode and Network.
3. Encrypt your files using Advanced Encryption Standard and 256 symbols randomly generated password and delete source files using DOD 5220.22-M.
(DOD 5220.22-M is the Department of Defense clearing and sanitizing standard - You cant recover your files - NEVER).
4. Sent this randomly generated password to our secure server and delete this password from your computer. (you cant get this password -NEVER)
This password is unique for each computer and stored on our secure server(and then erasing from this server and sending to us) and in each encrypted file.
If you think that you or some specialist can get this password from encrypted file - this is unreal even for specialist for goverment services, because here using 256-bit Advanced Encryption Standard.
To brute-force an AES-256-ECB encryption key in a known-plaintext attack, using all possible combinations,on a Cray XE6 with one million Opteron 6282 SE cores,
it would take up to ~66,282,862,563,751,221,625,826,507,369,649,000,000,000,000,000,000,000,000,000 years to complete the known-plaintext attack.
You have only two ways to decrypt your files:
1. Get Paid for decrypt password to us.
2. Wait ~66,282,862,563,751,221,625,826,507,369,649,000,000,000,000,000,000,000,000,000 years (who thinking that bruteforce can help read about AES online!).
Maybe you will remove locker but you have no other ways to decrypt or recover your files! No one person in the world can not decrypt your files or get the password to decrypt.
Forget about the fact that someone will help you.
Because your computer has been hacked or someone spamming from your computer. You must pay a penalty within 96 hours otherwise we will send report to FBI
with special password to decrypt some files wich contains spam software and child pornography files. (this special password is only for this files, not for all your files.
Password for all your files we will send you only after payment). If first 48 hours will be end you must pay 4000$ within next 48 hours.
To remove lock and decrypt your files you need to do next steps:
1. Buy Moneypak, Paysafecard or Ukash card 1000$ for first 48 hours and 4000$ after first 48 hours.
2. Send us email with your Id number and card code (you can use mobile internet from your cell phone or another PC to send email).
3. Wait 1-3 hours while we will send you reply email with two passwords to unlock and decrypt your files.
You can buy MoneyPak card at the nearest stores : Walgreens, Walmart, CVS/ pharmacy, Kmart, SevenEleven, Rite Aid or go to www.moneypak.com to find location stores near you
To find Paysafecard location stores near you visit www.paysafecard.com or Ukash at www.ukash.com
Our Guaranties:
If you dont trust us you can send any one file (no more 5mb, jpg, bmp or other picture, not a document) and your Id number to our email, then we wil decrypt it
and will send you reply with succesefully decrypted file.
Your ID Number and our contacts (please write down this data):
Your Id #: 1074470467 Our special service email: security11220@gmail.com
Send Code

Deskscan Plugin

$ **python vol.py –f ACCFISA.vmem deskscan**

Volatile Systems Volatility Framework 2.1_alpha

[snip]

****

Desktop: 0x24675c0, Name: **WinSta0\My Desktop 2**, Next: 0x820a47d8

SessionId: 0, DesktopInfo: 0xbc310650, fsHooks: 0

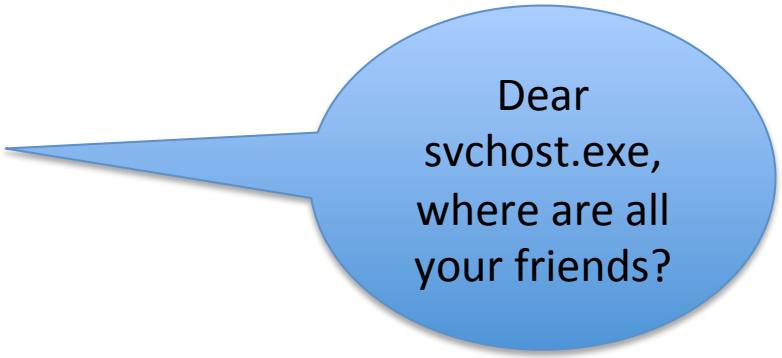
spwnd: 0xbc3106e8, Windows: 111

Heap: 0xbc310000, Size: 0x300000, Base: 0xbc310000, Limit: 0xbc610000

652 (csrss.exe 612 parent 564)

648 (csrss.exe 612 parent 564)

308 (svchost.exe 300 parent 240)

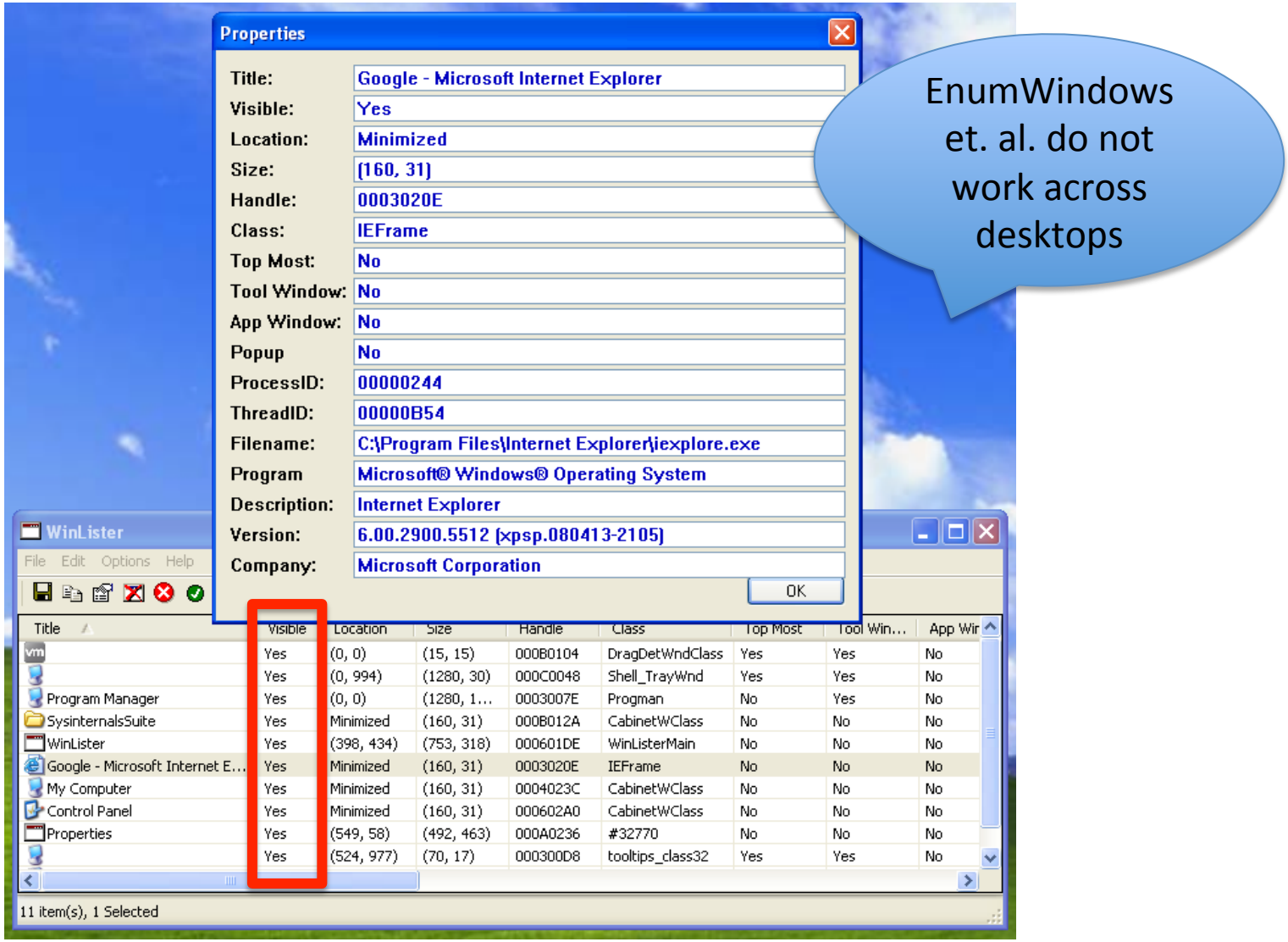


ACCDFISA

$ **python vol.py -f ACCFISA.vmem wintree**
Volatile Systems Volatility Framework 2.1_alpha
[snip]
****

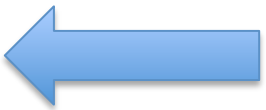
Window context: **0\WinSta0\My Desktop 2**
[snip]
.#100e2 csrss.exe:612 -
.#100e4 csrss.exe:612 -
#100de (visible) csrss.exe:612 -
.**Anti-Child Porn Spam Protection** (18 U.S.C. ? 2252) (visible) svchost.exe:300 WindowClass_0
..Send Code (visible) svchost.exe:300 Button
..#100ee (visible) svchost.exe:300 Edit
..Your Id #: 1074470467 Our special service email: **security11220@gmail.com** (visible) svchost.exe:300 Static
..Your ID Number and our contacts (please write down this data): (visible) svchost.exe:300 Static
..#100e8 (visible) svchost.exe:300 Static

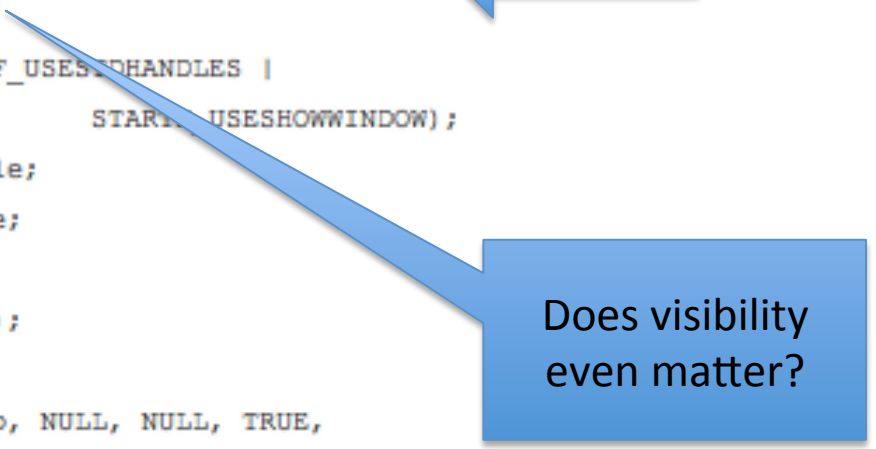
Window Scope / Visibility



Out of Scope?

```
memset(&StartupInfo, 0, sizeof(StartupInfo));
GetStartupInfo(&StartupInfo);

StartupInfo.cb = sizeof(StartupInfo);
1  StartupInfo.lpDesktop = _T("Winsta0\\system_temp_");
StartupInfo.wShowWindow = 1;
StartupInfo.dwFlags = (STARTF_USESTDHANDLES |
                                    STARTF_USESHOWWINDOW);
StartupInfo.hStdOutput = hFile;
StartupInfo.hStdError = hFile;

LPTSTR szDup = _tcsdup(szCmd);

2  if (CreateProcess(NULL, szDup, NULL, NULL, TRUE,
           CREATE_NEW_CONSOLE, NULL, NULL,
           &StartupInfo, &ProcessInfo))
{
 if (bWait)
    WaitForSingleObject(ProcessInfo.hProcess, INFINITE);
}
```





Atoms

- Tables of strings shared between processes
  - Direct
    - AddAtom and GlobalAddAtom
  - Indirect
    - Window class names (RegisterClassEx)
    - Injected DLL paths (SetWindowsHookEx, SetWinEventHook)
    - Registered window messages (RegisterWindowMessage)
- You can’t cover up what you don’t know about

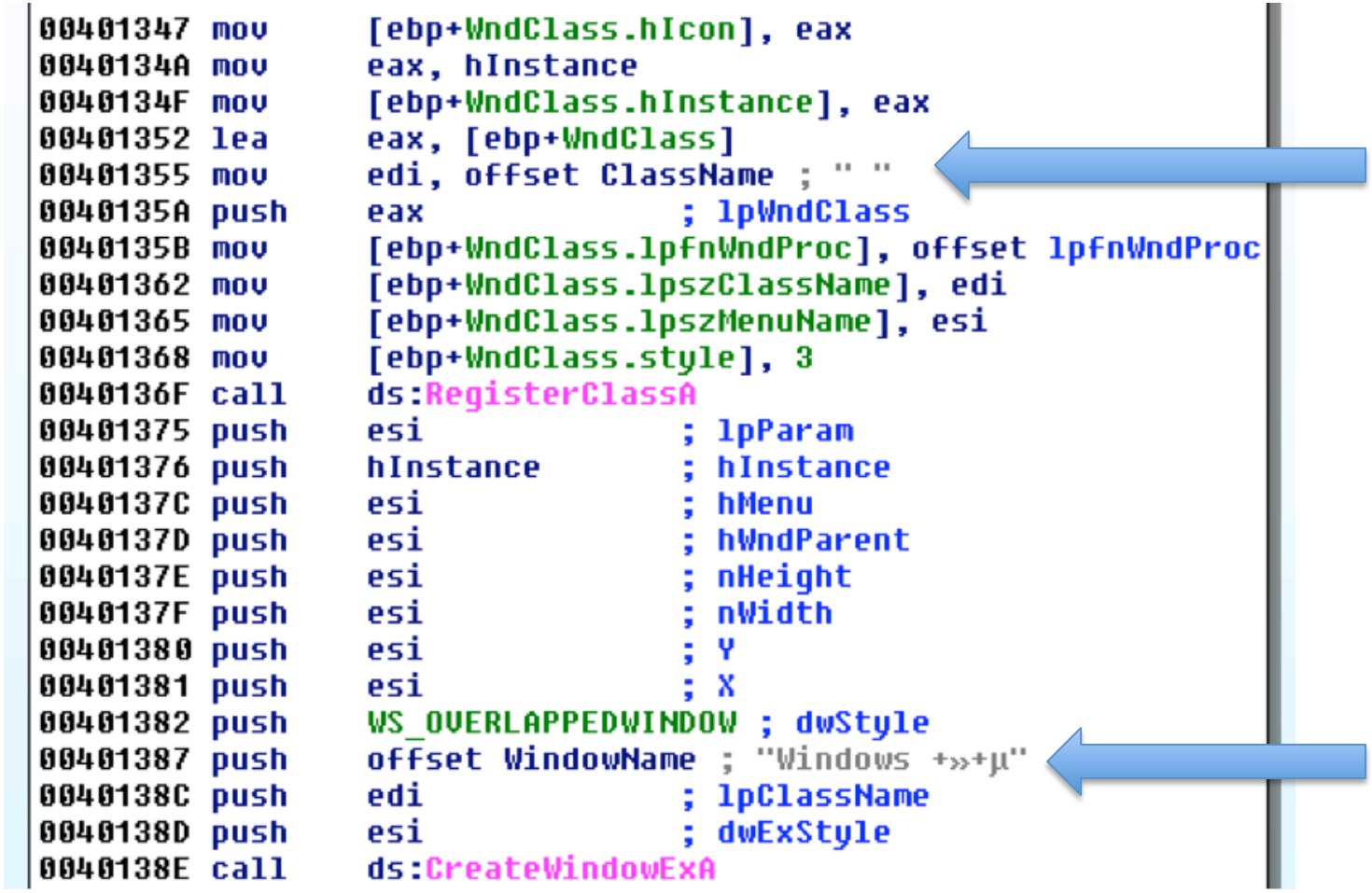
Atoms

```
>>> dt("_RTL_ATOM_TABLE")
'_RTL_ATOM_TABLE' (68 bytes)
0x0   : Signature                          ['unsigned long']
0x4   : CriticalSection                    ['_RTL_CRITICAL_SECTION']
0xc   : NumBuckets                         ['unsigned long']
0x10  : Buckets                            ['array', <function
<lambda> at 0x1046fe938>, ['pointer', ['_RTL_ATOM_TABLE_ENTRY']]]

>>> dt("_RTL_ATOM_TABLE_ENTRY")
'_RTL_ATOM_TABLE_ENTRY' (16 bytes)
0x0   : HashLink                           ['pointer',
['_RTL_ATOM_TABLE_ENTRY']]
0x4   : HandleIndex                        ['unsigned short']
0x6   : Atom                               ['unsigned short']
0x8   : ReferenceCount                     ['unsigned short']
0xa   : Flags                              ['unsigned char']
0xb   : NameLength                         ['unsigned char']
0xc   : Name                               ['String', {'length':
<function <lambda> at 0x1046fe9b0>, 'encoding': 'utf16'}]
```

Finding Atoms

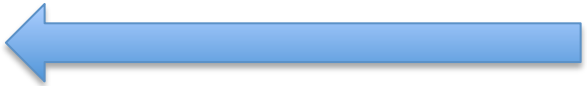
- Symbols: win32k!UserAtomHandleTable
- Members: tagWINDOWSTATION.pGlobalAtomTable
- Pool Scanning
  - PoolTag “AtmT”
  - Signature “Atom”
- FYI ReferenceCount – atoms remain until ref count == 0 even if creating process terminates

Mutihack


00401347 mov [ebp+WndClass.hIcon], eax
0040134A mov eax, hInstance
0040134F mov [ebp+WndClass.hInstance], eax
00401352 lea eax, [ebp+WndClass]
00401355 mov edi, offset ClassName ; " "
0040135A push eax ; lpWndClass
0040135B mov [ebp+WndClass.lpfnWndProc], offset lpfnWndProc
00401362 mov [ebp+WndClass.lpszClassName], edi
00401365 mov [ebp+WndClass.lpszMenuName], esi
00401368 mov [ebp+WndClass.style], 3
0040136F call ds:RegisterClassA
00401375 push esi ; lpParam
00401376 push hInstance ; hInstance
0040137C push esi ; hMenu
0040137D push esi ; hWndParent
0040137E push esi ; nHeight
0040137F push esi ; nWidth
00401380 push esi ; Y
00401381 push esi ; X
00401382 push WS_OVERLAPPEDWINDOW ; dwStyle
00401387 push offset WindowName ; "Windows +»+µ"
0040138C push edi ; lpClassName
0040138D push esi ; dwExStyle
0040138E call ds:CreateWindowExA

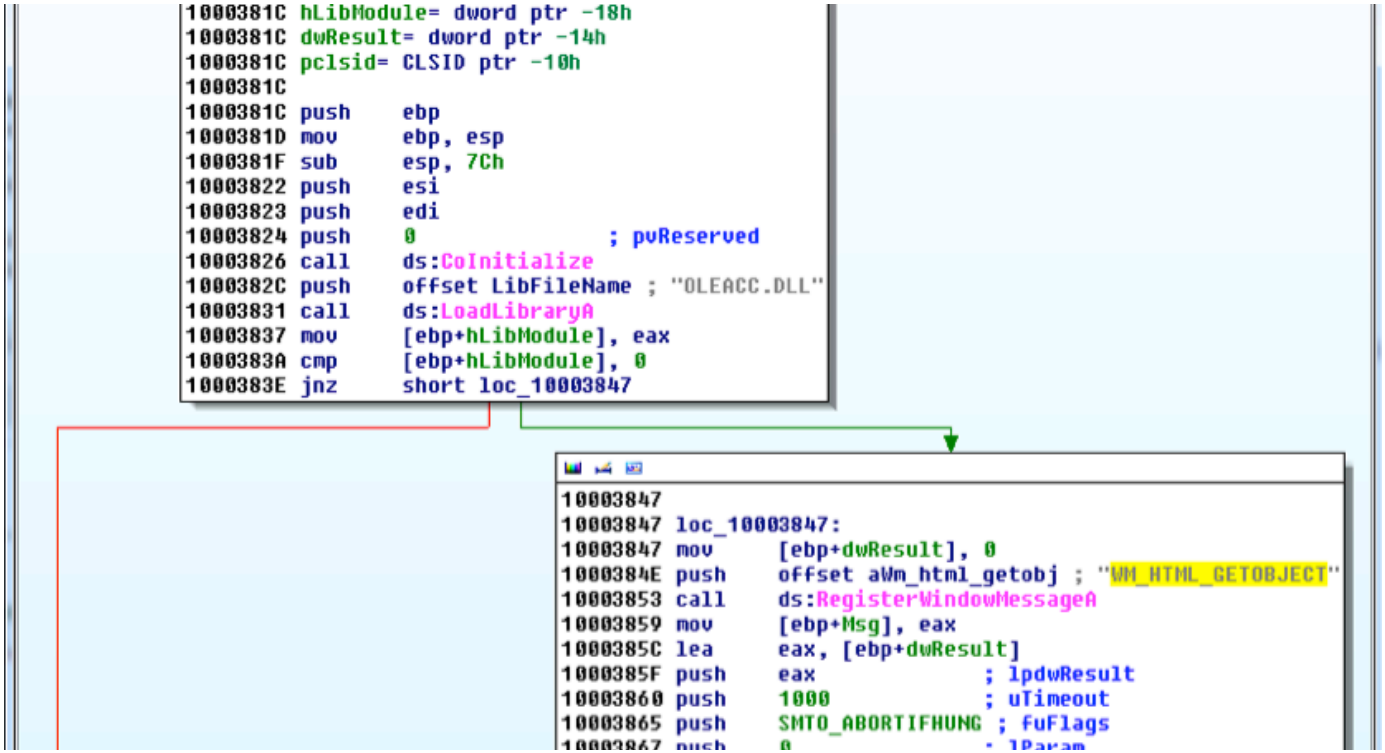
Detecting Class Names

```
$ python vol.py -f mutihack.vmem atomscan
AtomOfs(V)        Atom Refs    Pinned Name
---------- ---------- ------ ------ ----
[snip]
0xe179d850     0xc038      1      1 OleMainThreadWndClass
0xe17a7e40     0xc094      2      0 Shell_TrayWnd
0xe17c34b8     0xc0c4      2      0 UnityAppbarWindowClass
0xe17c7678     0xc006      1      1 FileName
0xe17d40a0     0xc0ff      2      0
0xe17d4128     0xc027      1      1 SysCH
0xe17e78f0     0xc01c      1      1 ComboBox
0xe17e9070     0xc065     26      0 6.0.2600.6028!Combobox
0xe17ec350     0xc13e      1      0 Xaml
0xe18119c0     0xc08c      5      0 OM_POST_WM_COMMAND
[snip]
```

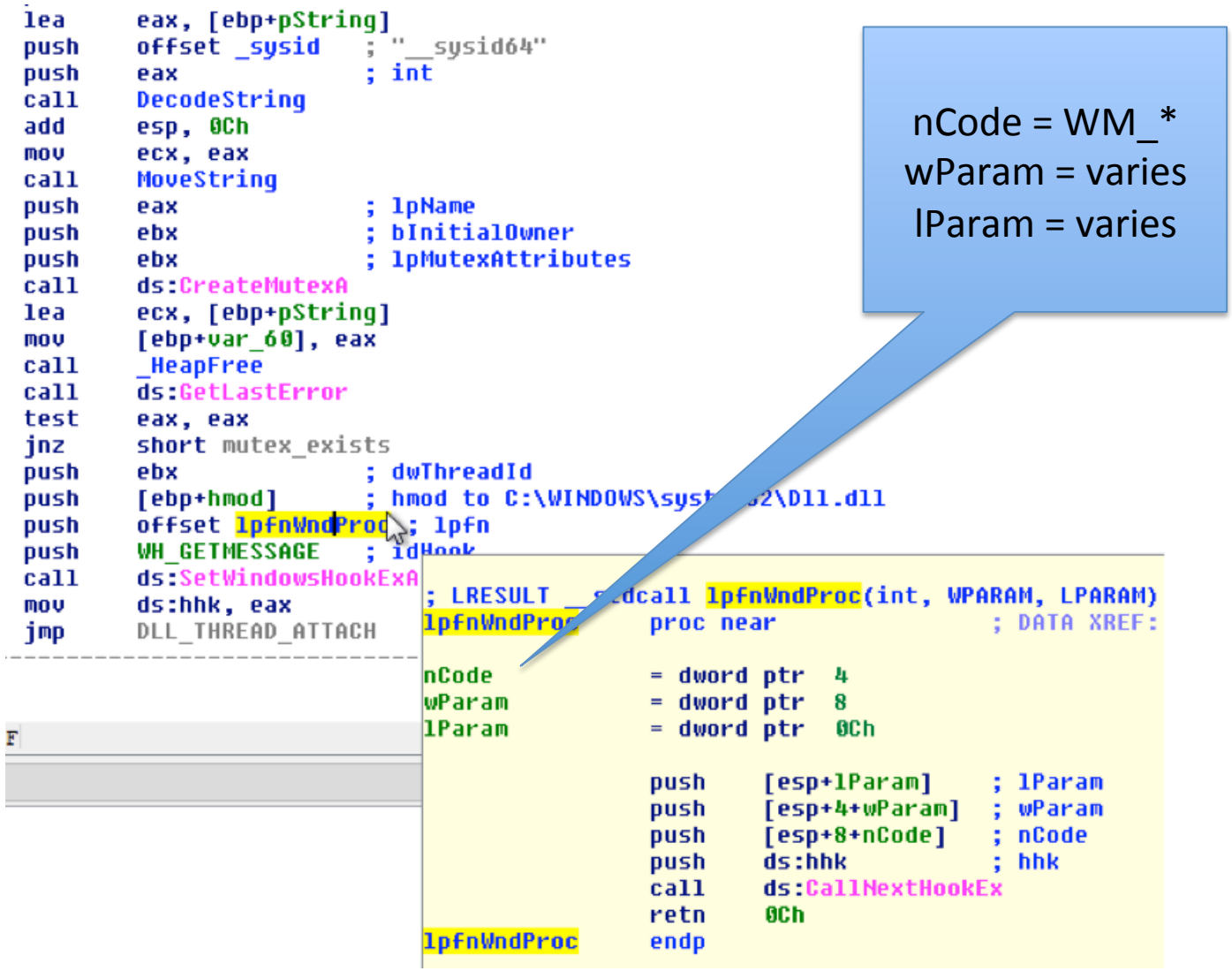


Clod

- IHTMLDocument2 from an HWND
- Scriptable IE for TAN-grabbing, HTMLi, etc

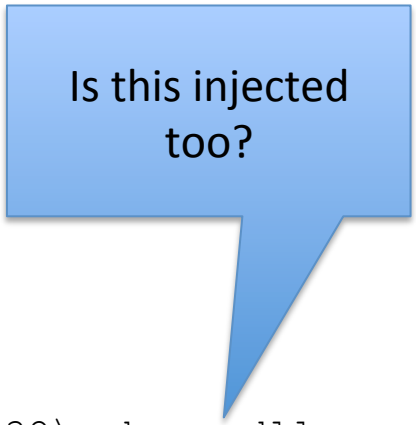


DLL Injection

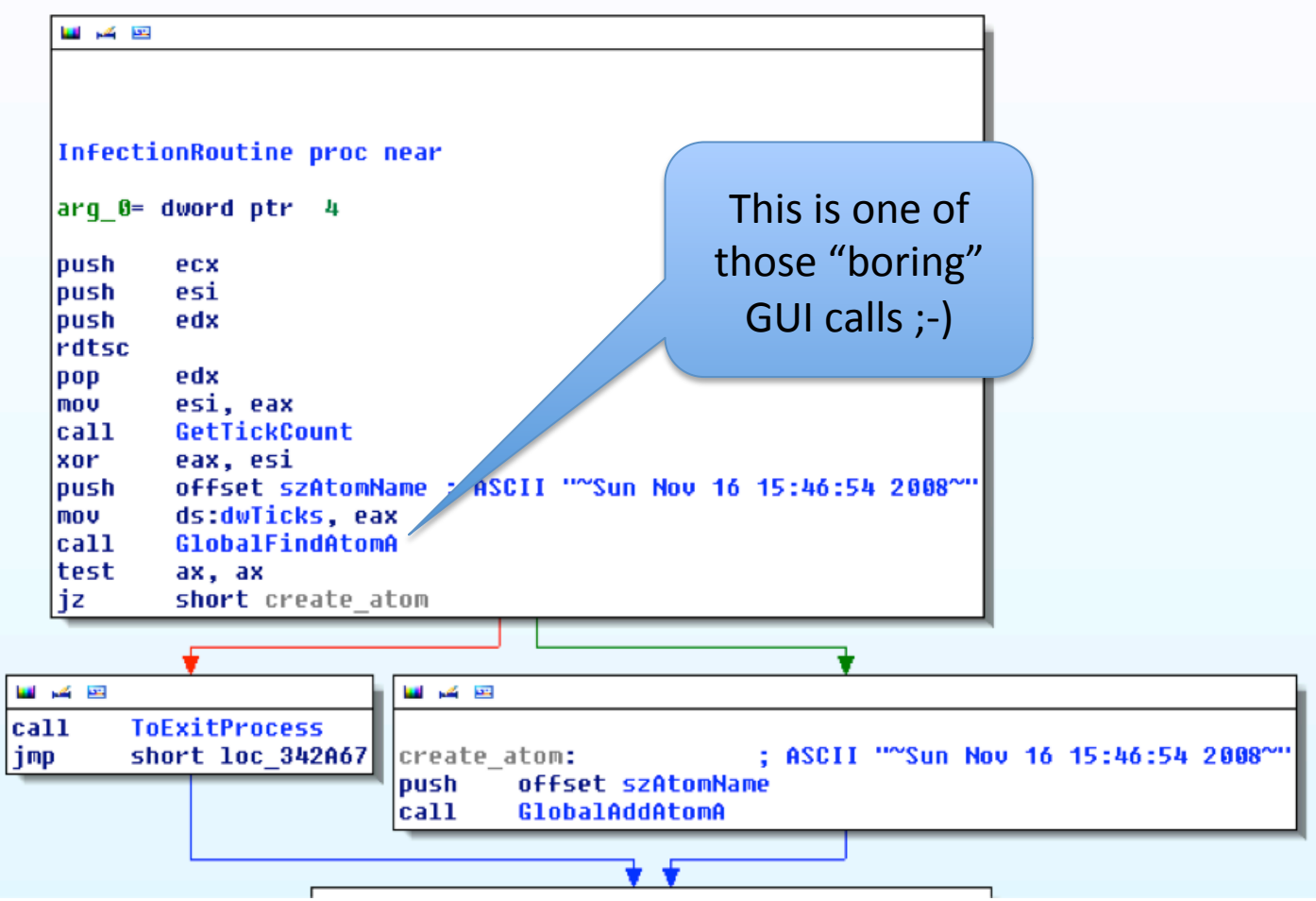


Detecting DLL Paths

```
$ python vol.py -f laqma.vmem atomscan
AtomOfs(V)        Atom Refs    Pinned Name
---------- ---------- ------ ------ ----
0xe1000d10     0xc001      1      1 USER32
0xe155e958     0xc002      1      1 ObjectLink
0xe100a308     0xc003      1      1 OwnerLink
0xe1518c00     0xc004      1      1 Native
0xe1b2aa88     0xc1b2      2      0 __axelsvc
0xe4bcb888     0xc1be      2      0 ShImgVw:CPreview
0xe11b0250     0xc1c1      2      0 __srvmgr32
0xe1f8bc30     0xc1c3      1      0 C:\WINDOWS\system32\psbase.dll
0xe28ed818     0xc1c7      1      0 BCGP_TEXT
0xe2950c98     0xc19f      1      0 ControlOfs0142000000000FC
0xe11d6290     0xc1a0      1      0 C:\WINDOWS\system32\Dll.dll
0xe1106380     0xc1a1      1      0 BCGM_ONCHANGE_ACTIVE_TAB
0xe11a5090     0xc1a2      1      0 ControlOfs01EE0000000003C8
[snip]
```



The New Mutex



The New Mutex

```
int __stdcall Putas_Global_Atom_Hook_Thread(int a1)
{
  while ( !GlobalFindAtomA("putas38") )
    Sleep(0xC8u);
  if ( Get_Netscape_DLL() )
  {
    Hook_Netscape_PRWrite();
  }
  else
  {
    if ( Null_Function() )
      Hook_Wininet_WSASend();
  }
  Hook_Wininet_Advapi_Functions();
  return 0;
}
```

Windows

- Containers for buttons, scroll bars, text areas
- Visibility, coordinates, Z-Order
- tagWND
  - spwnd[Parent|Child|Next] – pointers to related windows
  - rcWindow and rcClient – coordinates
  - lpfnWndProc – window procedure function
  - pcls – tagCLS window class
  - strName – window name passed to CreateWindowEx

Window Name Detection

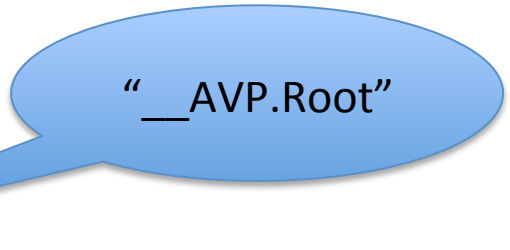
- FindWindow("Wireshark") or EnumWindows
- GetWindowThreadProcessID
- TerminateProcess
  - Either wireshark or itself

- Smarter malware will use EnumWindowStations and EnumDesktops also

KAV Shatter Attack

- Disables KAV with PostMessage

```
__int32 __cdecl Disable_Kaspersky_Antivirus()
{
  __int32 result; // eax@1
  HWND pwnd; // esi@1
  DWORD dwTicks; // eax@2
  HKEY hKey; // [sp+3Ch] [bp-10h]@2
  BYTE Data[4]; // [sp+40h] [bp-Ch]@1
  int v5; // [sp+44h] [bp-8h]@1
  char v6; // [sp+48h] [bp-4h]@1

  *Data = dword_3447A4;
  v5 = dword_3447A8;
  v6 = byte_3447AC;
  result = FindWindowA(ClassName, 0);
  pwnd = result;
  if ( result )
  {
    dwTicks = GetTickCount();
    PostMessageA(pwnd, 0x466u, 0x10001u, dwTicks);
    result = RegCreateKeyA(
               HKEY_LOCAL_MACHINE,
               "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\avp.exe",
               &hKey);
    if ( !result )
    {
      RegSetValueExA(hKey, "Debugger", 0, 1u, Data, 8u);
      result = RegCloseKey(hKey);
    }
  }
  return result;
}
```



Simulation

```
void __stdcall Send_Alt_Ctrl_Delete(int a1)
{
  DWORD dwTid; // eax@1
  HDESK hDesk; // esi@1

  dwTid = GetCurrentThreadId();
  hDesk = GetThreadDesktop(dwTid);
  if ( OpenDesktop("Winlogon") )
  {
    PostMessageA(HWND_BROADCAST, WM_HOTKEY, 0, 0x2E0003u);
    if ( hDesk )
      Set_Thread_Desktop(hDesk);
  }
  ExitThread(0);
}
```

MOD_ALT | MOD_CTRL | MOD_DEL

```
if ( stricmp(*(a3 + 4), "EVENTMOUSE") )
  return 0;
*buf = 0;
X = 0;
dy = 0;
dwData = 0;
v9 = Recv_Data(buf, 16, 0);
if ( v9 == 16 )
{
  SetCursorPos(X, dy);
  mouse_event(*buf, X, dy, dwData, 0);
  return 1;
}
```

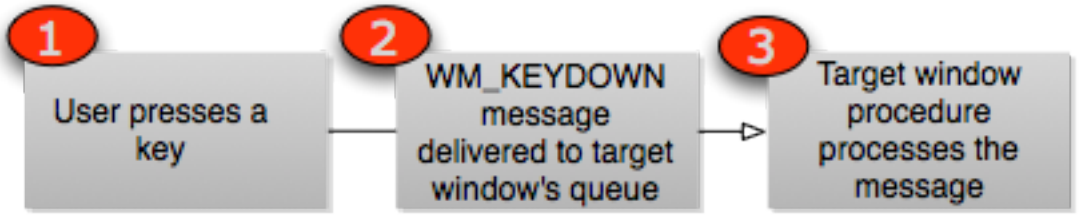
Moves the mouse on screen

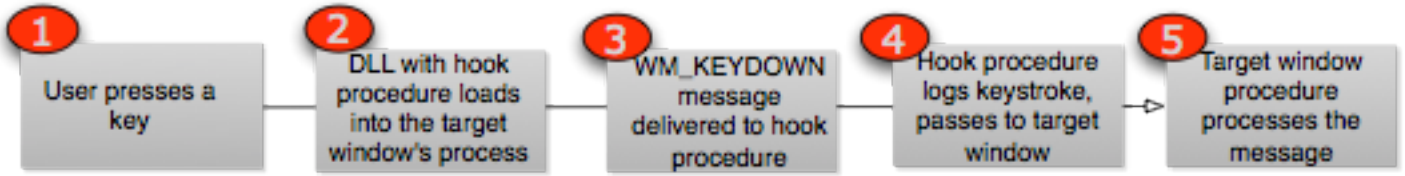
Detecting USB

```
$ python vol.py -f stuxnet.vmem windows

Window Handle: #e00e8 at 0xbc940720, Name: AFX64c313
ClassAtom: 0xc118, Class: AFX64c313
SuperClassAtom: 0xc118, SuperClass: AFX64c313
pti: 0xe1e81380, Tid: 1420 at 0x82126bf0
ppi: 0xe163f008, Process: services.exe, Pid: 668
Visible: No
Left: 92, Top: 146, Bottom: 923, Right: 695
Style Flags:
WS_MINIMIZEBOX,WS_TABSTOP,WS_DLGFRAME,WS_BORDER,WS_THICKFRAME,WS_CAPTION,WS_SYSMENU,WS_MAXIMIZEB
OX,WS_GROUP,WS_OVERLAPPED,WS_CLIPSIBLINGS
ExStyle Flags: WS_EX_LTRREADING,WS_EX_RIGHTSCROLLBAR,WS_EX_WINDOWEDGE,WS_EX_LEFT
Window procedure: 0x13fe695

$ echo "cc(pid = 668); dis(0x13fe695)" | ./vol.py -f stuxnet.vmem volshell

0x13fe695 55              PUSH EBP
0x13fe696 8bec            MOV EBP, ESP
0x13fe698 817d0c19020000  CMP DWORD [EBP+0xc], 0x219; WM_DEVICE_CHANGE
0x13fe69f 7514            JNZ 0x13fe6b5
0x13fe6a1 ff7514          PUSH DWORD [EBP+0x14]
0x13fe6a4 ff7510          PUSH DWORD [EBP+0x10]
0x13fe6a7 e810000000      CALL 0x13fe6bc
```

Hooks

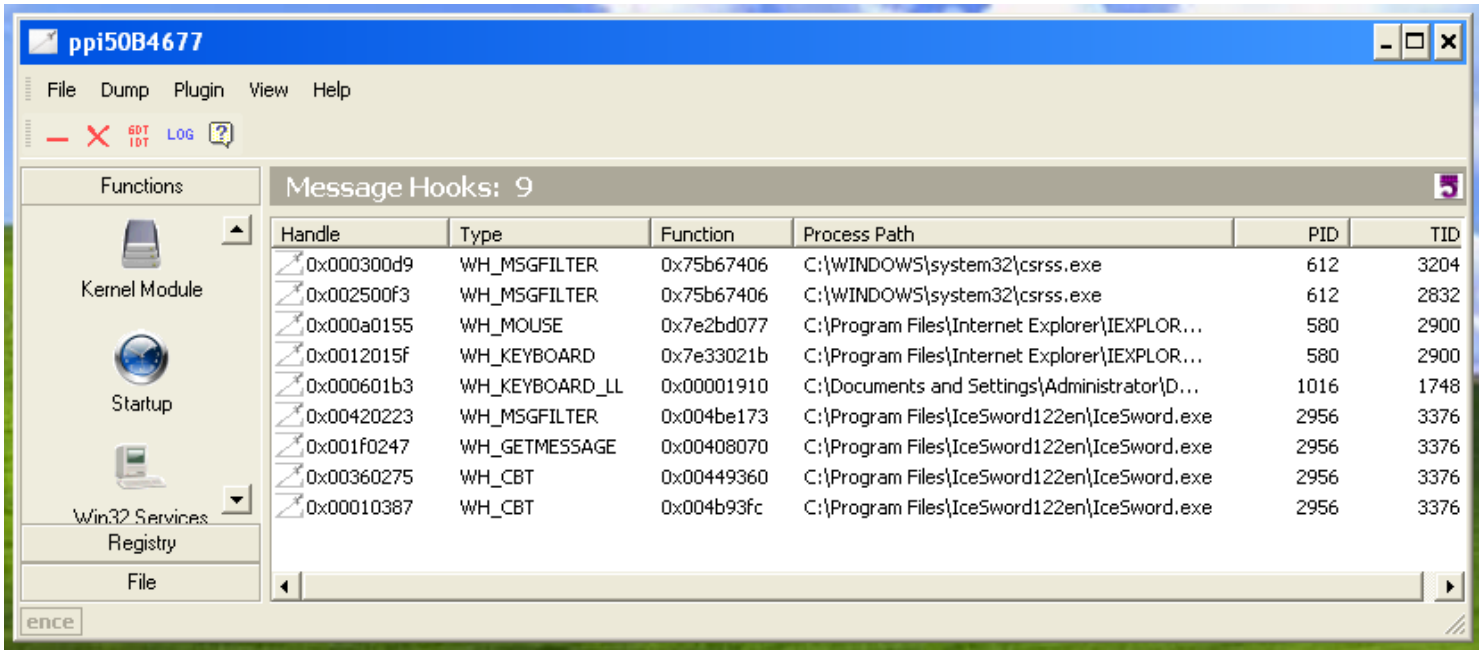
- Customize user experience
- Receive notifications of actions
- Record/Playback for CBT training





Message Hooks

- SetWindowsHookEx
- Parameters: thread ID (scope), DLL handle, function pointer, filter

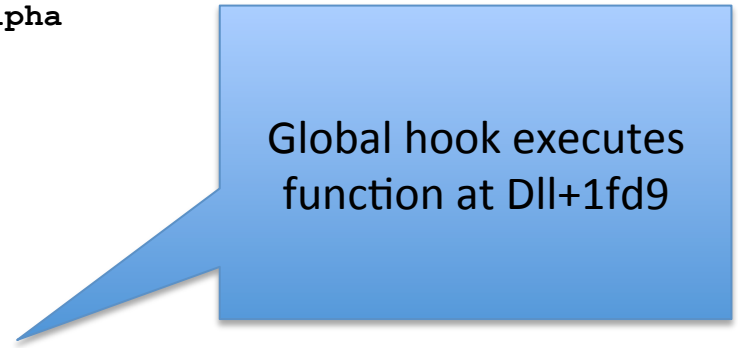


Message Hooks

- tagHOOK
  - phkNext – pointer to the next hook in the chain
  - offPfn – RVA or absolute depending on global hook
  - ptiHooked – identify hooked thread
  - rpdesk – identify the desktop in which the hook is set

MessageHooks Plugin

```
$ python vol.py -f laqma.vmem messagehooks --output=block
Volatile Systems Volatility Framework 2.1_alpha
Offset(V)   : 0xbc693988
Session     : 0
Desktop     : WinSta0\Default
Thread      : <any>
Filter      : WH_GETMESSAGE
Flags       : HF_ANSI, HF_GLOBAL
Procedure   : 0x1fd9
ihmod       : 1
Module      : C:\WINDOWS\system32\Dll.dll

Offset(V)   : 0xbc693988
Session     : 0
Desktop     : WinSta0\Default
Thread      : 1584 (explorer.exe 1624)
Filter      : WH_GETMESSAGE
Flags       : HF_ANSI, HF_GLOBAL
Procedure   : 0x1fd9
ihmod       : 1
Module      : C:\WINDOWS\system32\Dll.dll
```



Clipboard

tagWINDOWSTATION.pClipBase points to an array of these

```
>>> dt("tagCLIP")
'tagCLIP' (24 bytes)
0x0    : fmt                           ['Enumeration', {'target': 'unsigned long',
'choices': {128: 'CF_OWNERDISPLAY', 1: 'CF_TEXT', 2: 'CF_BITMAP', 3: 'CF_METAFILEPICT',
4: 'CF_SYLK', 5: 'CF_DIF', 6: 'CF_TIFF', 7: 'CF_OEMTEXT', 8: 'CF_DIB', 9: 'CF_PALETTE',
10: 'CF_PENDATA', 11: 'CF_RIFF', 12: 'CF_WAVE', 13: 'CF_UNICODETEXT', 14:
'CF_ENHMETAFILE', 15: 'CF_HDROP', 16: 'CF_LOCALE', 17: 'CF_DIBV5', 131:
'CF_DSPMETAFILEPICT', 129: 'CF_DSPTEXT', 130: 'CF_DSPBITMAP', 142:
'CF_DSPENHMETAFILE'}}]
0x8    : hData                         ['pointer64', ['void']]
0x10   : fGlobalHandle                 ['long']

>>> dt("tagCLIPDATA")
'tagCLIPDATA' (None bytes)
0x10   : cbData                        ['unsigned int']
0x14   : abData                        ['array', <function <lambda> at 0x1048e5500>,
['unsigned char']]

$ python vol.py -f rdp.dmp --profile=Win2003SP2x86 userhandles -t TYPE_CLIPDATA
Volatile Systems Volatility Framework 2.3_alpha
Object(V)      Handle bType                 Flags    Thread   Process
---------- ---------- -------------------- -------- -------- -------
0xe807fe78    0x80073 TYPE_CLIPDATA           0     -------- -
0xe834a508    0x40075 TYPE_CLIPDATA           0     -------- -
```

Clipboard Plugin

```
$ python vol.py -f dfrws2008-rodeo-memory.img clipboard
Volatile Systems Volatility Framework 2.1_alpha
Session  WindowStation Format                Handle Object     Data
-------- ------------- ---------------- ---------- ---------- ------------
    0    WinSta0       CF_UNICODETEXT     0x4900c3 0xe12a7c98 pp -B -p -o out.pl file
    0    WinSta0       CF_LOCALE           0x80043 0xe12362d0
    0    WinSta0       CF_TEXT                0x1 ----------
    0    WinSta0       CF_OEMTEXT             0x1 ----------

$ python vol.py -f system.dmp clipboard
[snip]
    0    WinSta0       CF_HDROP          0x10230131 0xe1fa6590
0xe1fa659c  14 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0xe1fa65ac  01 00 00 00 43 00 3a 00 5c 00 44 00 6f 00 63 00 ....C.:.\.D.o.c.
0xe1fa65bc  75 00 6d 00 65 00 6e 00 74 00 73 00 20 00 61 00 u.m.e.n.t.s...a.
0xe1fa65cc  6e 00 64 00 20 00 53 00 65 00 74 00 74 00 69 00 n.d...S.e.t.t.i.
0xe1fa65dc  6e 00 67 00 73 00 5c 00 41 00 64 00 6d 00 69 00 n.g.s.\.A.d.m.i.
0xe1fa65ec  6e 00 69 00 73 00 74 00 72 00 61 00 74 00 6f 00 n.i.s.t.r.a.t.o.
0xe1fa65fc  72 00 5c 00 44 00 65 00 73 00 6b 00 74 00 6f 00 r.\.D.e.s.k.t.o.
0xe1fa660c  70 00 5c 00 6e 00 6f 00 74 00 65 00 2e 00 74 00 p.\.n.o.t.e...t.
0xe1fa661c  78 00 74 00 00 00 00 00                         x.t.....
```

GDI Timers

- Like _ETIMER/_KTIMER (see “timers” plugin)
- User-mode GUI threads call SetTimer
- Callback function runs when timer expires
  How is this different than:

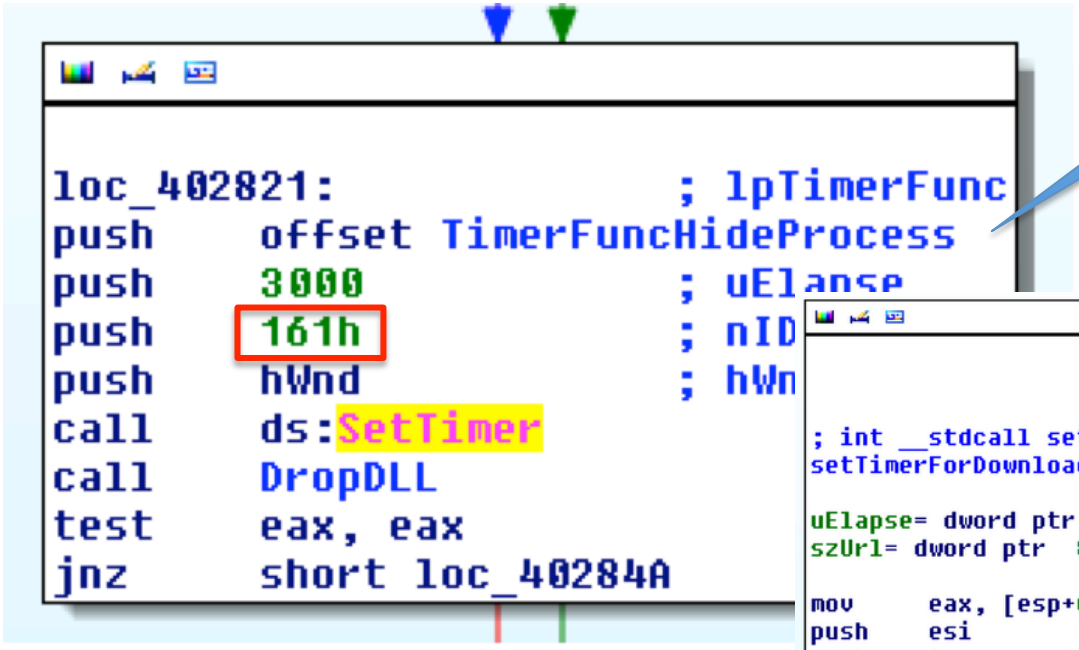
```
while(1) {
        do_something();
        Sleep(amount);
};
```

GDI Timers

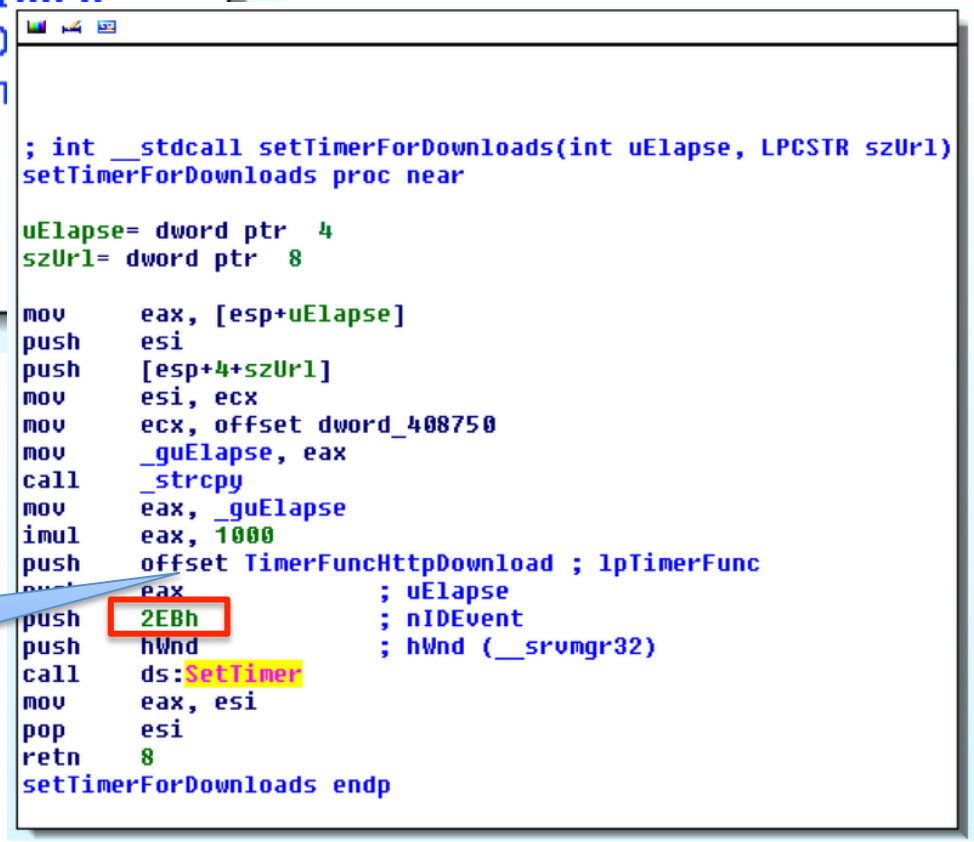
- tagTIMER
  - pti – tagTHREADINFO identifies owner
  - spwnd – pointer to window
  - nID – unique ID returned by SetTimer
  - cmsCountdown – time left (ms)
  - cmsRate – initial time (ms)
  - pfn – pointer to callback

Setting Timers

If a process becomes unhidden, hide it again



```
loc_402821:                     ; lpTimerFunc
push    offset TimerFuncHideProcess
push    3000                    ; uElapse
push    161h                    ; nID
push    hWnd                    ; hWn
call    ds:SetTimer
call    DropDLL
test    eax, eax
jnz     short loc_40284A
```



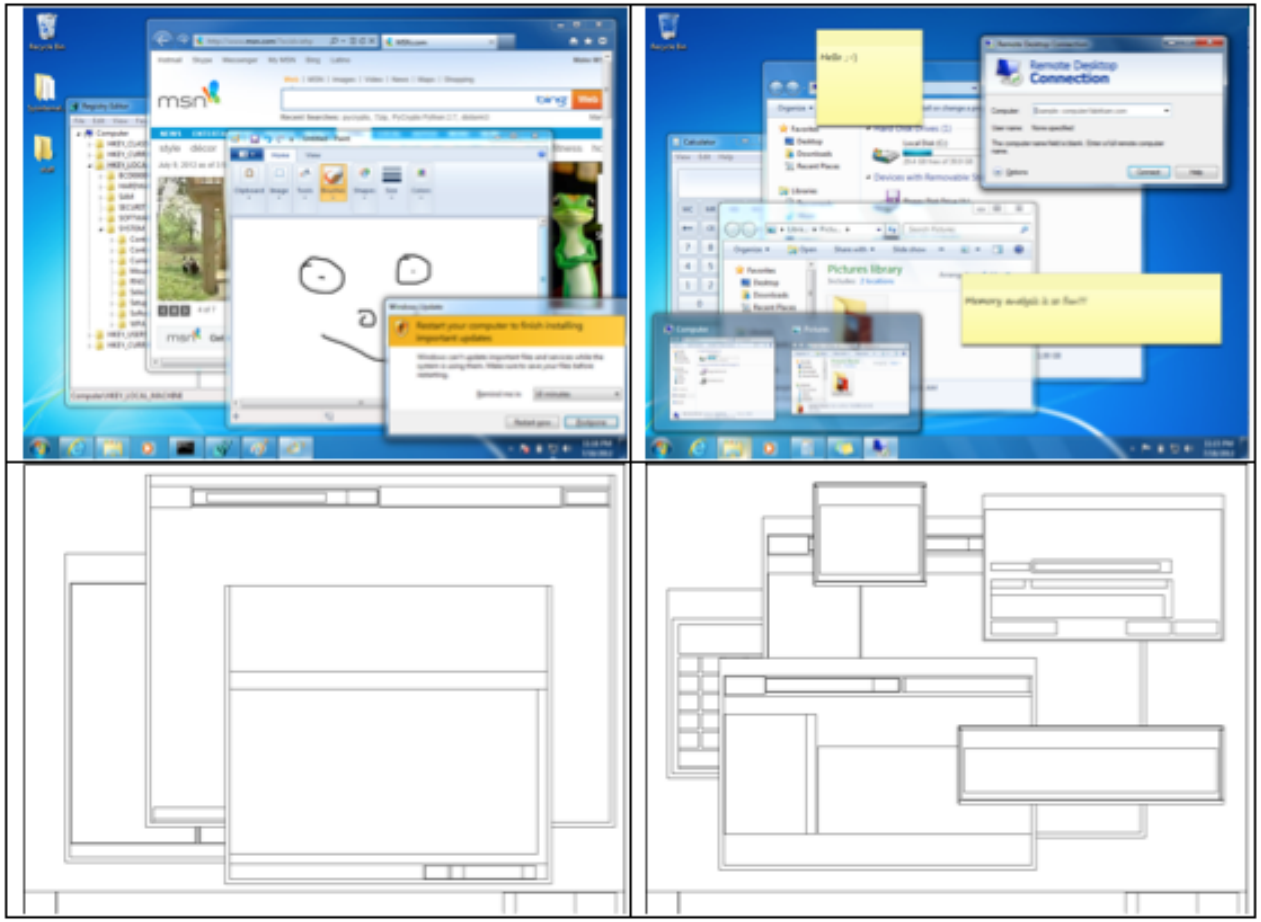
```
; int __stdcall setTimerForDownloads(int uElapse, LPCSTR szUrl)
setTimerForDownloads proc near

uElapse= dword ptr  4
szUrl= dword ptr  8

mov     eax, [esp+uElapse]
push    esi
push    [esp+4+szUrl]
mov     esi, ecx
mov     ecx, offset dword_408750
mov     _guElapse, eax
call    _strcpy
mov     eax, _guElapse
imul    eax, 1000
push    offset TimerFuncHttpDownload ; lpTimerFunc
push    eax                     ; uElapse
push    2EBh                    ; nIDEvent
push    hWnd                    ; hWnd (__srvmgr32)
call    ds:SetTimer
mov     eax, esi
pop     esi
retn    8
setTimerForDownloads endp
```

If the C2 is offline, keep trying

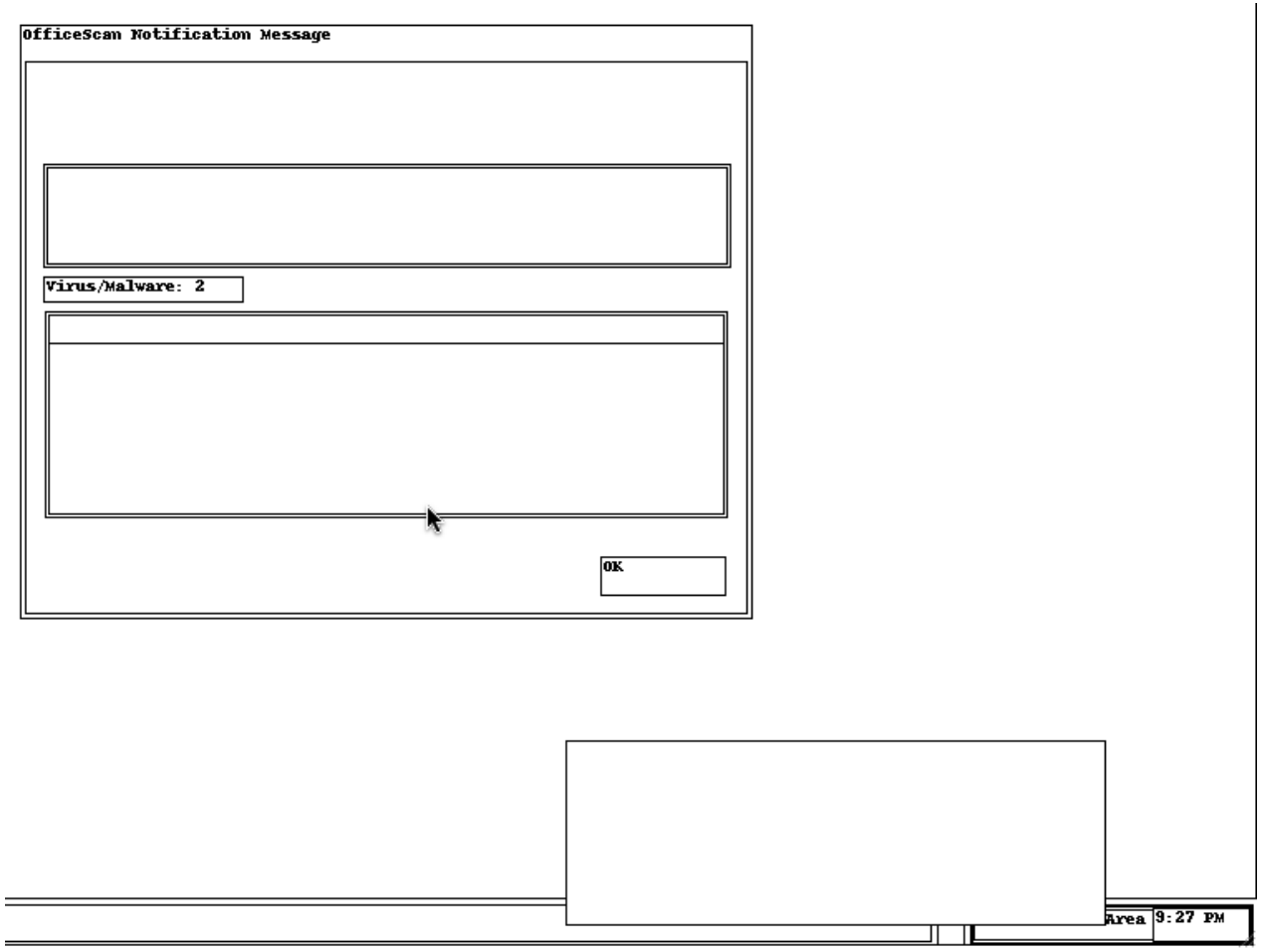
Timers Plugin

```
$ python vol.py -f laqma.vmem gditimers
Volatile Systems Volatility Framework 2.1_alpha
Thread   Process                    nID Rate(ms)    Countdown(ms) Func
-------- ------------------- ---------- ---------- ------------- ----------
     696 csrss.exe:660             0x7ffe       1000           734 0xbf8012b8
    1648 explorer.exe:1624          0x15      60000         45109 0x00000000
    1480 svchost.exe:1064         0x7476      60000         16234 0x74f51070
     696 csrss.exe:660            0x7ffd      35000         25625 0xbf8f4d9a
    1648 explorer.exe:1624          0x19   86400000      70004672 0x00000000
    1764 VMwareTray.exe:1760         0x0       5000          4859 0x00000000
    1648 explorer.exe:1624           0xe   43200000      26805359 0x00000000
    1764 VMwareTray.exe:1760        0x11      60000         45859 0x00000000
     700 csrss.exe:660            0xfff5        100           100 0xbf807d00
     356 svchost.exe:1064            0x0     300000        131234 0x77532ebb
    2024 lanmanwrk.exe:920         0x2eb     600000        589578 0x00401fc8
    2024 lanmanwrk.exe:920         0x161       3000          1578 0x004010aa
    1648 explorer.exe:1624          0x0      60000         11922 0x00000000
     384 KernelDrv.exe:352          0x8b     600000        595359 0x00404c2b
     384 KernelDrv.exe:352          0x8c       3000          1359 0x004010aa
     384 KernelDrv.exe:352           0xd       2000          1359 0x00410850
```

Screenshots



Screenshots



Conclusions

- win32k.sys is a great place to spend your afternoon
- DKOM is not impossible but unlikely
- The sophistication and capabilities of your tools must match or exceed that of the attackers; else you #fail
- GUIs are pretty, but pretty limited

The end

- Volatility
  - http://volatility-labs.blogspot.com
  - http://code.google.com/p/volatility
  - @volatility
- Me
  - http://mnin.blogspot
  - http://www.mnin.org
  - @iMHLv2