



# Combating Malware in the age of APT

SANS Digital Forensic and Incident Response Summit  
July 2010

Jason Garman  
CTO, Kyrus Technology

# New directions for malware

---

- Malicious code used in APT attacks are usually:
  - Not “sexy” – the simple techniques work well!
  - To some extent, custom
    - Not widely disseminated = not picked up by AV
    - Not necessarily custom code but custom “packaging”
  - Highly targeted
    - Mostly a factor of the delivery mechanism, spear-phishing email, web link, etc.
  - Modular
    - Monolithic binary is risky; reveals too much about the MO, capabilities of the attacker

# Modular?

---

- Historically your neighborhood script kiddie had one of two choices for his exploitation tools:
  - The Unix way: a lot of tools, each one does a certain function very, very well
  - The Microsoft Word way: one tool to rule them all, contains all the functionality plus the kitchen sink
- However both of these techniques have drawbacks
  - The Unix way inevitably leads to tools that have vastly different interfaces, difficult learning curve
  - The Word way helps ensure a consistent interface but exposes all of your capabilities at once to the malware analyst

# Modular Implants vs. Memory Analysis

---

- These modular implants pose a significant challenge to the incident responder
  - No longer is the entire binary (or binaries) available for viewing and analysis from the disk
  - Now we must fuse together the results of traditional malware analysis with the volatile data acquisition
- Malware authors will continue to improve in this arena
  - Freeing unused memory as soon as it is no longer necessary
  - Zeroing out sensitive memory areas after use
- Will need more research and development to keep pace with the malicious code authors!

# Case Study: Poison Ivy

The screenshot displays the XPMalware2 interface for a victim machine with IP 172.16.93.120. The main window is titled "Process Manager" and lists the following processes:

Image Name	Path	PID	Image Base	Image Size	Threads	CPU	Mem Usage	Created
System Idle Process		0	00000000	00000000	1	97	28 KiB	-
System		4	00000000	00000000	56	0	236 KiB	-
smss.exe	\SystemRoot\System32\smss.exe	380	48580000	0000F000	3	0	416 KiB	6/25/2010 3:58:33 PM
csrss.exe	\?A:C:\WINDOWS\system32\csrss.exe	600	4A680000	00005000	11	0	3.99 MB	6/25/2010 3:58:34 PM
winlogon.exe	\?A:C:\WINDOWS\system32\winlogon.exe	624	01000000	00081000	20	0	4.26 MB	6/25/2010 3:58:35 PM
services.exe	C:\WINDOWS\system32\services.exe	676	01000000	0001D000	15	0	3.34 MB	6/25/2010 3:58:35 PM
lsass.exe	C:\WINDOWS\system32\lsass.exe	688	01000000	00006000	19	0	1.36 MB	6/25/2010 3:58:35 PM
vmacthlp.exe	C:\Program Files\VMware\VMware Tools\vmacthlp.exe	844	00400000	0005C000	1	0	2.44 MB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	856	01000000	00006000	17	0	4.79 MB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	940	01000000	00006000	10	0	4.15 MB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	1036	01000000	00006000	53	0	22.20 MB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	1088	01000000	00006000	4	0	3.45 MB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	1156	01000000	00006000	12	0	3.85 MB	6/25/2010 3:58:37 PM
spoolsv.exe	C:\WINDOWS\system32\spoolsv.exe	1460	01000000	00010000	13	0	6.13 MB	6/25/2010 3:58:37 PM
explorer.exe	C:\WINDOWS\Explorer.EXE	1604	01000000	000FF000	14	0	20.97 MB	6/25/2010 3:58:38 PM
VMwareTray.exe	C:\Program Files\VMware\VMware Tools\VMwareTra...	1772	00400000	0002C000	1	0	4.59 MB	6/25/2010 3:58:38 PM
VMwareUser.exe	C:\Program Files\VMware\VMware Tools\VMwareUs...	1792	00400000	00178000	6	0	11.35 MB	6/25/2010 3:58:39 PM
ctfmon.exe	C:\WINDOWS\system32\ctfmon.exe	1800	00400000	00006000	1	0	2.99 MB	6/25/2010 3:58:39 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	2000	01000000	00006000	4	0	3.67 MB	6/25/2010 3:58:46 PM
vmtoolsd.exe	C:\Program Files\VMware\VMware Tools\vmtoolsd.exe	284	00400000	0000E000	5	0	9.73 MB	6/25/2010 3:58:46 PM
VMUprgradeHelper.exe	C:\Program Files\VMware\VMware Tools\VMUprgrad...	480	00400000	00029000	3	0	3.95 MB	6/25/2010 3:58:49 PM
TPAutoConnSvc.exe	C:\Program Files\VMware\VMware Tools\TPAutoCon...	1364	00400000	00041000	5	0	4.07 MB	6/25/2010 3:58:51 PM
alg.exe	C:\WINDOWS\system32\alg.exe	1944	01000000	00000000	5	0	3.51 MB	6/25/2010 3:58:51 PM
wscntfy.exe	C:\WINDOWS\system32\wscntfy.exe	220	01000000	00006000	1	0	2.11 MB	6/25/2010 3:58:52 PM
TPAutoConnect.exe	C:\Program Files\VMware\VMware Tools\TPAutoCon...	652	00400000	00072000	1	0	4.35 MB	6/25/2010 3:58:54 PM
cmd.exe	C:\WINDOWS\system32\cmd.exe	2364	4A000000	00061000	1	0	2.79 MB	6/25/2010 3:59:41 PM
DLLYDBG.EXE	C:\Documents and Settings\Administrator\Desktop\o...	880	00400000	00153000	2	3	4.02 MB	6/28/2010 3:29:43 PM
AdobeUpdate.exe	C:\WINDOWS\system32\AdobeUpdate.exe	3100	00400000	00001C00	5	0	4.77 MB	6/28/2010 3:29:46 PM
calc.exe	C:\WINDOWS\system32\calc.exe	160	01000000	0001F000	2	0	3.12 MB	7/7/2010 9:10:37 PM
notepad.exe	C:\WINDOWS\system32\notepad.exe	344	01000000	00014000	1	0	916 KiB	7/7/2010 11:31:40 PM
logon.scr	C:\WINDOWS\system32\logon.scr	2956	01000000	00039000	1	0	1.92 MB	7/8/2010 12:15:22 AM
ntvdm.exe	C:\WINDOWS\system32\ntvdm.exe	2012	0F000000	000A7000	4	0	2.36 MB	7/8/2010 12:20:20 AM
cmd.exe	C:\WINDOWS\system32\cmd.exe	3644	4A000000	00061000	1	0	2.57 MB	7/8/2010 12:20:30 AM

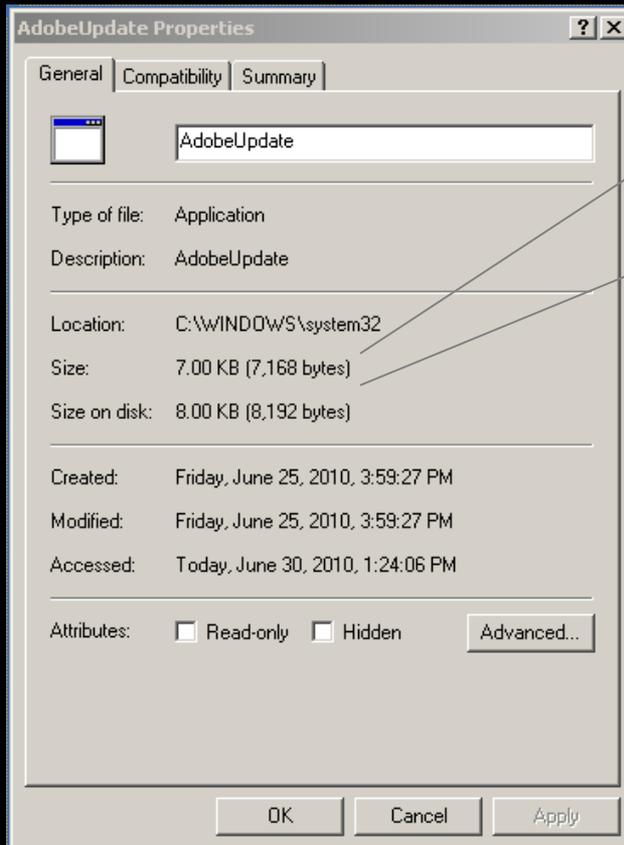
Summary: Processes: 33 | CPU Usage: 0% | Mem Usage: 154.37 MB | Threads: 298 | Handles: 6241

Download: 0 B/s | Upload: 0 B/s

Start | C:\WINDOWS\system32\... | 12:24 AM

To return to your computer, press Control-~

# The Challenge

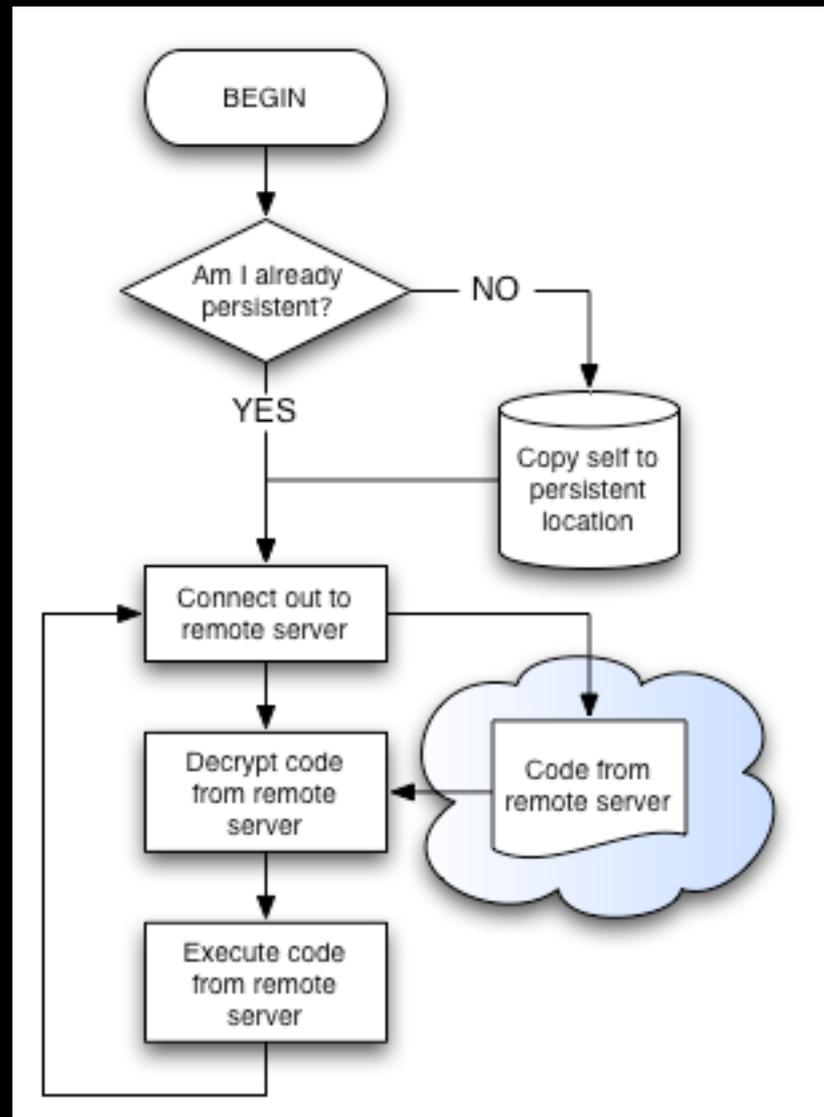


7.00 KB (7,168 bytes)

- A 7kb file? Probably not much in there... but let's try anyway.



# The 10,000 foot view



# What do we have?

---

- We know that it pulls in several useful imports:
  - Socket creation/connection
  - Registry set/query (RegSetValue, etc.)
  - File manipulation (CreateFile/WriteFile, etc.)
  - Process listing (CreateToolHelp32Snapshot...)
  - Memory manipulation (VirtualAlloc/Free)
- Also, some framework for future “modules”:
  - Most notably, a custom import resolver (to avoid using GetProcAddress)
  - Also, decryption code (Camellia block cipher)

# But... not much else

---

- The application code as it exists on disk is limited to placing itself in the run key (for persistence) and using the network functions to “call out” to a server
- No indication of “command” functionality... but instead:
  - It validates that the server has the correct key
  - Decrypts the incoming data
  - Allocates some memory, copying the decrypted data to the new memory area
  - ... and jumps to it (blindly)



# So now what?

---

- We can use the memory image of the target machine to (hopefully) reconstruct some of the capabilities loaded at run time by the attacker
- Wouldn't it be nice to have some record of the commands invoked by the attacker as well?

# Some questions we can answer

---

- What dlls were loaded into this process?
  - Use dlllist from volatility
- Are there executable code segments outside of the mapped executable image?
  - If so, can we disassemble them?
  - Use the VAD tree to find these memory mappings and dump using vaddump from volatility
- What strings exist that might indicate malicious activity?
  - Possibly including command lines, etc.
- More importantly, we want to exclude 7kb image from these analyses, so we can “diff” against a baseline

# Volatile “Diffing”

---

- Take a “baseline” of the VAD tree/DLL list/file list/etc when the binary has started up (without network connection)
- Compare with the corresponding analysis on the memory image from your incident
- This is especially useful if the original binary was packed
  - For example, the memory regions used to unpack the binary
- For example...

# Example

---

- Collect the DLL listing for the baseline and incident images:
  - `volatility dlllist -p [PID] -f [Baseline Memory Image] > dlllist_base.txt`
  - `volatility vadinfo -p [PID] -f [Incident Memory Image] > dlllist_incident.txt`
- Diff the two to determine what new DLLs were loaded once Poison Ivy was able to call out to the C&C server:
  - `diff -u dlllist_base.txt dlllist_incident.txt`

# Diffing the Loaded DLLs

---

- The code executed from the server loads several additional Windows DLLs:

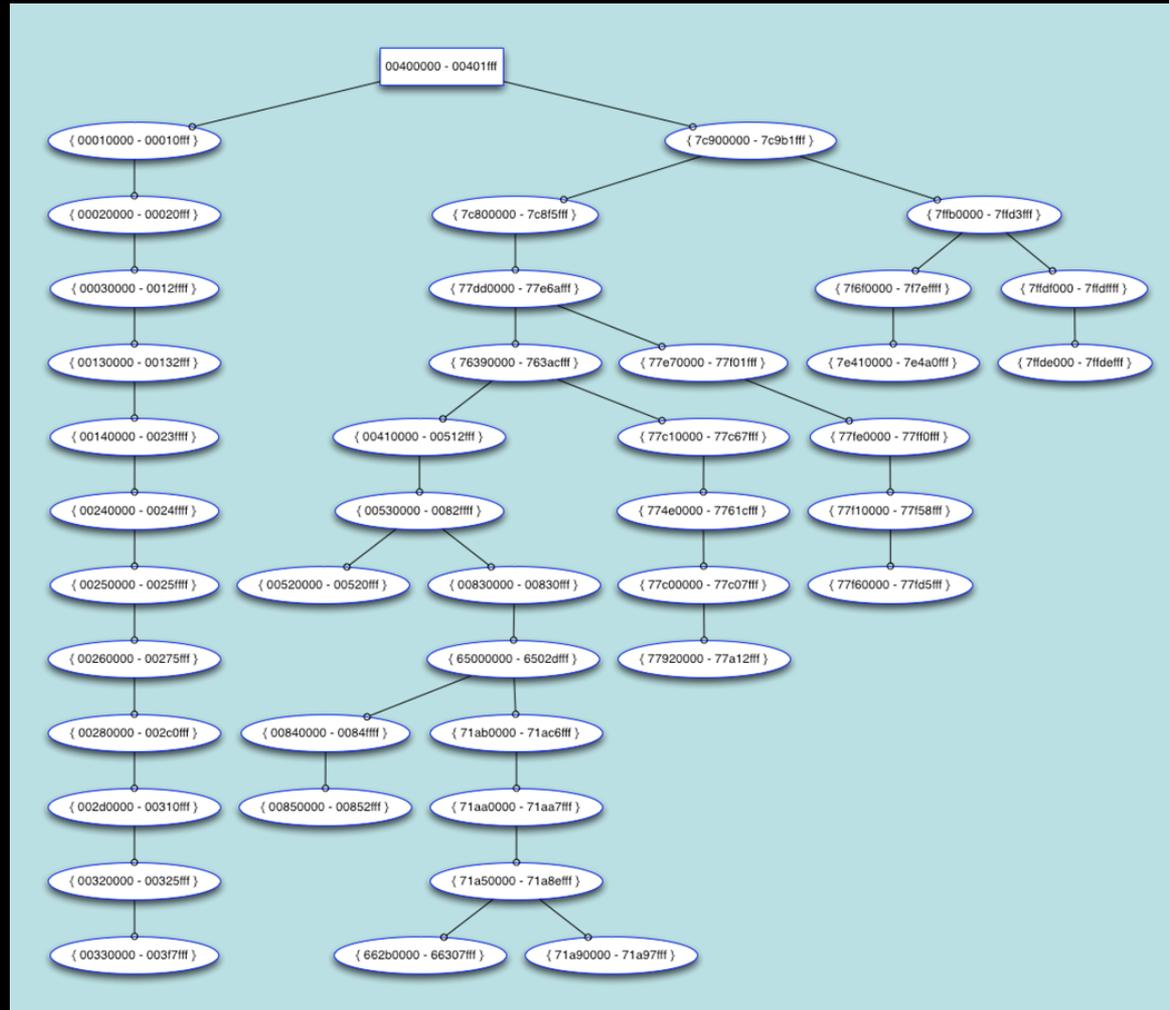
```
\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-  
  Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll  
\WINDOWS\system32\atl.dll  
\WINDOWS\system32\avicap32.dll  
\WINDOWS\system32\comctl32.dll  
\WINDOWS\system32\crypt32.dll  
\WINDOWS\system32\iphlpapi.dll  
\WINDOWS\system32\mpr.dll  
\WINDOWS\system32\msasn1.dll  
\WINDOWS\system32\msvfw32.dll  
\WINDOWS\system32\pstorec.dll  
\WINDOWS\system32\shell32.dll  
\WINDOWS\system32\winmm.dll
```

# Getting to Executable Code...

---

- We could dump the entire process space, but that includes a lot of code & data we're not interested in (or have already analyzed)...
- So let's use "VAD Diffing" to narrow down to the new code downloaded by the tool from the network
- But first... what is the VAD?
  - Virtual Address Descriptor
  - Forensic application first discussed in a 2007 paper by Brendan Dolan-Gavitt
  - Essentially, metadata about allocated memory regions in a process
    - Is the region backed by disk?
    - What are the page protections?

# VAD Tree for Poison Ivy



# The VAD info list

- Each loaded executable or DLL image will have its own entry in the VAD info list

```
VAD node @8221ec40 Start 65000000 End 6502dfff Tag Vad
Flags: ImageMap
Commit Charge: 15 Protection: 7
ControlArea @820db218 Segment e1835300
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences:          0 NumberOfPfnReferences:          32
NumberOfMappedViews:                1 NumberOfSubsections:          5
FlushInProgressCount:               0 NumberOfUserReferences:       1
Flags: Accessed, HadUserReference, Image, File
FileObject @822c6028 (024c6028), Name: \WINDOWS\system32\advpack.dll
WaitingForDeletion Event: 00000000
ModifiedWriteCount:                  0 NumberOfSystemCacheViews:      0
First prototype PTE: e1835340 Last contiguous PTE: ffffffff
Flags2: Inherit
File offset: 00000000
```

# The VAD info list

---

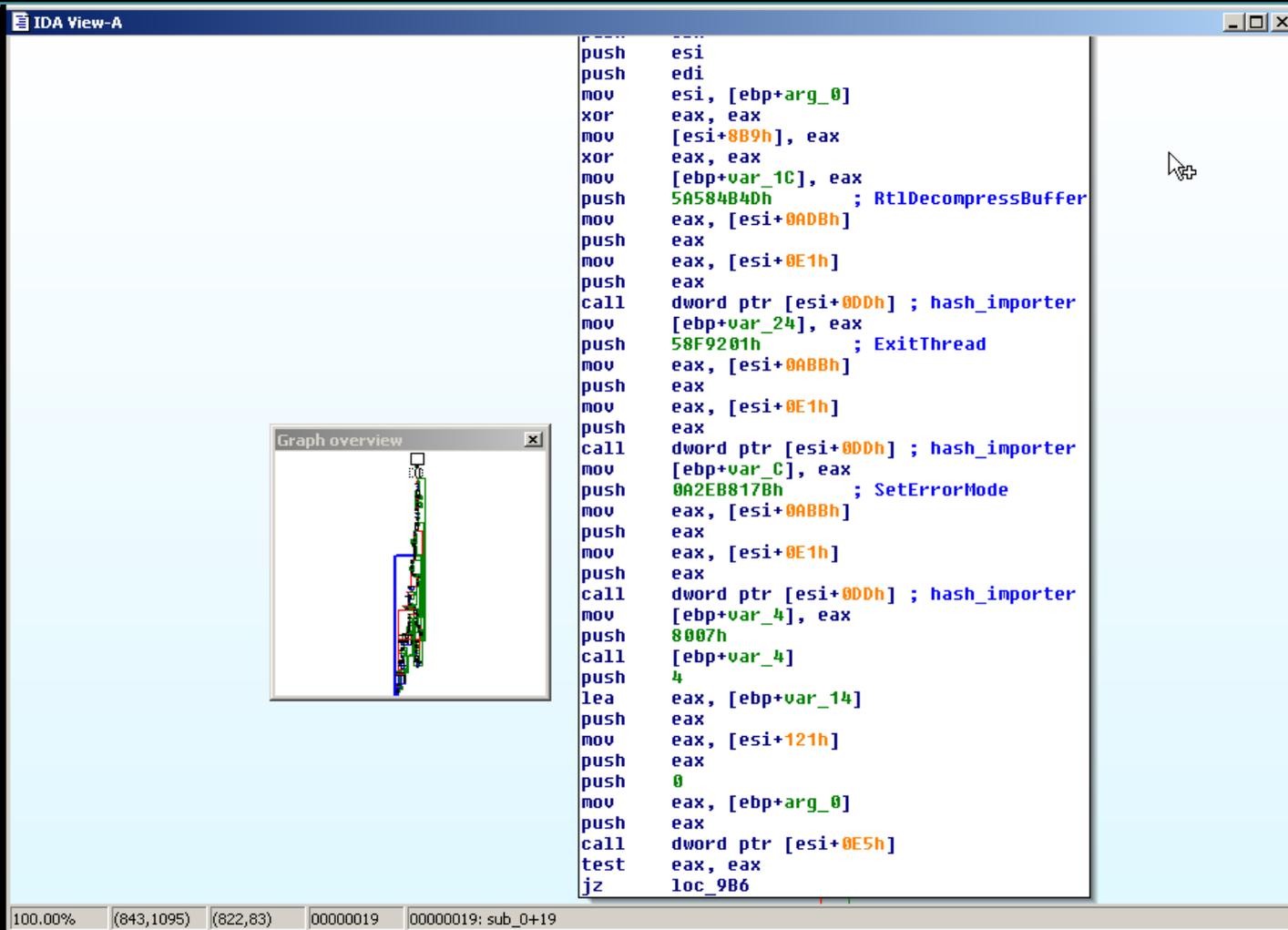
- Dynamically allocated memory looks a bit different:

```
VAD node @81de8288 Start 00aa0000 End 00aa0fff Tag VadS  
Flags: MemCommit, PrivateMemory  
Commit Charge: 1 Protection: 6
```

```
VAD node @81d68330 Start 00ac0000 End 00ac0fff Tag VadS  
Flags: MemCommit, PrivateMemory  
Commit Charge: 1 Protection: 6
```

- We are most interested in these segments!
- As long as the system patchlevels match between the two machines and the program's allocation pattern doesn't change wildly between runs, you can get meaningful results from this (crude) method

# IDA Pro with Dynamically Loaded Modules



The screenshot displays the IDA Pro interface. The main window, titled "IDA View-A", shows assembly code for a function. The code includes various instructions such as `push esi`, `push edi`, `mov esi, [ebp+arg_0]`, `xor eax, eax`, `mov [esi+8B9h], eax`, `xor eax, eax`, `mov [ebp+var_1C], eax`, `push 5A584B4Dh ; RtlDecompressBuffer`, `mov eax, [esi+0ADBh]`, `push eax`, `mov eax, [esi+0E1h]`, `push eax`, `call dword ptr [esi+0DDh] ; hash_importer`, `mov [ebp+var_24], eax`, `push 58F9201h ; ExitThread`, `mov eax, [esi+0ABBh]`, `push eax`, `mov eax, [esi+0E1h]`, `push eax`, `call dword ptr [esi+0DDh] ; hash_importer`, `mov [ebp+var_C], eax`, `push 0A2EB817Bh ; SetErrorMode`, `mov eax, [esi+0ABBh]`, `push eax`, `mov eax, [esi+0E1h]`, `push eax`, `call dword ptr [esi+0DDh] ; hash_importer`, `mov [ebp+var_4], eax`, `push 8007h`, `call [ebp+var_4]`, `push 4`, `lea eax, [ebp+var_14]`, `push eax`, `mov eax, [esi+121h]`, `push eax`, `push 0`, `mov eax, [ebp+arg_0]`, `push eax`, `call dword ptr [esi+0E5h]`, `test eax, eax`, and `jz loc_9B6`. A "Graph overview" window is open on the left, showing a vertical flow graph with a single node. The status bar at the bottom indicates 100.00% zoom, address ranges (843,1095) and (822,83), and the current instruction address 00000019, which is `sub_0+19`.

```
push esi
push edi
mov esi, [ebp+arg_0]
xor eax, eax
mov [esi+8B9h], eax
xor eax, eax
mov [ebp+var_1C], eax
push 5A584B4Dh ; RtlDecompressBuffer
mov eax, [esi+0ADBh]
push eax
mov eax, [esi+0E1h]
push eax
call dword ptr [esi+0DDh] ; hash_importer
mov [ebp+var_24], eax
push 58F9201h ; ExitThread
mov eax, [esi+0ABBh]
push eax
mov eax, [esi+0E1h]
push eax
call dword ptr [esi+0DDh] ; hash_importer
mov [ebp+var_C], eax
push 0A2EB817Bh ; SetErrorMode
mov eax, [esi+0ABBh]
push eax
mov eax, [esi+0E1h]
push eax
call dword ptr [esi+0DDh] ; hash_importer
mov [ebp+var_4], eax
push 8007h
call [ebp+var_4]
push 4
lea eax, [ebp+var_14]
push eax
mov eax, [esi+121h]
push eax
push 0
mov eax, [ebp+arg_0]
push eax
call dword ptr [esi+0E5h]
test eax, eax
jz loc_9B6
```

# What are we missing?

---

- How do the pieces fit together? Not clear...
  - Perhaps with interpretation of the thread state and stack we could determine a code flow
  - Would need to be semi-automated to be useful
- Everything in Poison Ivy is PIC, so lots of tables of imports and local functions are used vtable-style
  - Requires some significant effort on the part of the reverse engineer, but can be automated
- Once a module is no longer needed, the memory is VirtualFree()'d
  - Unlinks the memory region from the VAD tree and makes it very difficult to find and associate back with the process
  - Means we lose not only modules but also the associated data (commands, search strings, etc.)

# There be Nuggets

- Fragments of data before decompression:
  - “confidential information.txt”
  - Not reliable as it gets overwritten pretty quickly

The screenshot shows a hex editor window titled 'XPMalware1-Snapshot3.vmem'. The search bar contains 'confid' and the 'Find' button is highlighted. The main display area shows a memory dump with hexadecimal addresses on the left and corresponding data on the right. The data is mostly zeros, with some non-zero values starting at address 186DA000. The search results are highlighted in blue. At the bottom, there is a status bar showing '409837995 out of 536870912 bytes'.

```
186D9FA0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
186D9FC0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
186D9FE0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
186DA000 6FD2 3290 1CD3 326D 201C 8D45 606A 811F E201 8B45 1400 508A 4510 3000 0C30 00E0 o"2ê."2m .çE'jÅ.,.âE.-PGE.0..0.†
186DA020 2910 40D8 8B55 9024 E0FF 530E 2C5B 21AF 14A6 1412 FFFF FF67 A714 6603 003B 5625 ).MydUe$†$.S.,[I0.†. "" gS.f..;V%
186DA040 784F A0AD 8E22 C030 4683 C4C8 0046 8BFA F88B F089 F095 8096 32AD 308C 6BAD F828 xD†=â"¿0FÉ†.Fâ"†âââiââ2=0âk=*(
186DA060 05ED 3018 CF45 6FAD C245 D162 80E3 5057 56FF 93D0 F731 8C33 C080 B530 278B 45CC .I0.eEo=-E-bâ,,PWV"i-"1â3¿âµ0'âEâ
186DA080 42AE 7EDC 0301 9132 9602 D095 7F03 7C03 D8CD 7503 D817 067F 0300 0063 0251 7C55 BE<-<..e2ñ.-i..|.ÿ0u.ÿ.....c.QIU
186DA0A0 F306 D475 03D4 7A03 D3DA 00D2 4DDA 00D1 D600 E039 C680 0148 0151 C115 8A45 D310 0.'u.'t.z."/."M/.-..†9âââH.Qj.âE".
186DA0C0 1AD2 3000 D1C4 508B 511A DC8B C640 1A46 0248 00B9 2AB0 2233 D251 0108 7F36 1882 ."0.-†PâQ.âââ.F.H.m*"3"Q...6.ç
186DA0E0 5300 BB06 0131 6C80 0979 19C3 29A1 2E00 8AD0 0300 7019 81C4 02B8 9131 F180 7C24 S.¿.1l..y.ÿ)*.â..p.âf.†âââçL$
186DA100 08B9 1250 4001 F3A5 A01A D888 C304 8B90 A2C7 7234 8D4C 2400 3488 D78B C3FF 566C .m.Pe.O.*.ÿ.âf.âââçr4çL$.ââââ'vI
186DA120 0180 7C44 240C 5057 53FF 0496 CC51 1708 8D44 2420 21E6 008B 4424 2400 0900 3308 .â|D$.PWS"ââQ.çD$!E.âD$$.--3.
186DA140 C052 5080 0030 33D2 0300 0424 1354 2404 83C4 6008 8904 2489 9000 2079 44C4 2404 çRP..03"...$.T$.Éf'.â.çââ. yDf$.
186DA160 1603 81C4 4860 0480 6954 C30A E108 94C0 0001 1F23 C0C6 8B1F 2381 D028 0589 0003 ..âfH\âITÿ...i¿..#âââ.â.#.(â..
186DA180 1F23 DF1F 231F 2385 6710 1C1F 2300 9F26 90D4 D770 031F 237F 0300 7F03 451F 2372 .#ñ.#.#0g...#.ûââ?)...#....E.#.
186DA1A0 036F 1F23 742E 5028 1F23 001F 231F 2300 1F1F 2381 2B1F 2320 5C1D 2303 433A 005C .o.#.†P(.#.#.#.#.#â+.#\#.C:\.
186DA1C0 1C63 6F6E 6669 6400 656E 7469 616C 5F69 006E 666F 726D 6174 6940 6F6E 2E74 7874 .confidential_i.informat@on.txt
186DA1E0 20E6 0000 0000 0000 0000 0000 0000 0000 00A4 986A 7CE3 A618 0387 CCA5 00F5 Ê.....Söjl,†.ââ*.†
186DA200 6EED 4D92 1272 D0F0 03BE 39A0 496F 4B30 8283 F2EC 10B2 05CD 2003 4F2F 4F2F 6382 nM†.r.-.e9†toK0çEUI.ç.0 .0/0/çç
186DA220 1455 D377 3908 40B5 7088 45CA E301 4CE3 30D3 D350 A027 041F 1445 E370 4108 4901 .U"†9.çupâE ,L,0"†††'....Ejpa.I.
186DA240 837D 0C00 0074 1983 7D10 0076 5013 8B45 10E0 010C E901 8D18 55D3 89A1 9142 3DF6 É)...t.É)...v.P.âE.†.Ê.ç.U"††Bâ†
186DA260 D81B CEFF 0001 A106 2628 8BC7 6611 B000 9881 C494 30AF E000 8B09 2391 00C6 45DB ÿ.E"...*(ââf...ââf100†.âÿ#e.âEe
186DA280 008B 7DE0 8B02 87D2 5940 3489 45CC 8BAD 2000 8035 5AB1 00F0 F10E BDD0 1157 FF0E .â)†â.âÿ†ââââââ.âSZe.#0.ñ..W".
186DA2A0 FF0E F20E 6AA0 6FDC 600A 8C08 50FF 9781 8166 817D 8C20 4D5A 0F85 5F20 1D68 F83F .Û.jto<.â.P"ââââââââ MZ.Û..h"?
186DA2C0 9013 A038 8079 F0A7 E108 4402 81BD 29C2 0045 0070 DF36 8002 6A04 4868 0020 008A ê.†âÿâââ.D.ââ).-E.pâââ.j.Hh..â
186DA2E0 85E4 2008 5008 8805 C861 00FF 5721 8BE0 F085 F675 150B 0130 8F83 0144 0F84 B11C 0% .P.â0âa."W!â†â0"u.e.0âE..-3.
186DA300 8973 0800 8143 8210 4100 00C6 4314 00D1 02D8 21C9 B304 5650 0416 01E8 6105 1101 âs..âçç.A..âC.-.-.€!...VP...âa..
186DA320 F3F0 9240 6203 8502 0100 0A0A 16C0 9227 F20B 7057 0023 45D4 300B 8943 0004 8BFE 0â†ââ.0...†.¿i'Û.pW.#E'0.âC..â.
186DA340 8978 3453 8D16 85E2 0A83 02E0 C002 C0FF 9022 F490 128B C72B A20A 85C0 D874 0B50 âx4Sç.0.,E.†.â"âe"Ue.âââ†.0ÿ†.P
186DA360 7072 0001 F860 0190 00AD 4402 0430 53C0 893F 1701 0810 0112 8B70 0540 2840 0321 pr".†.â.âD..0S¿â†)...âp.@(â!
186DA380 8B00 0003 D789 55D0 85D2 7402 1C90 4A01 57FF 55D0 83A0 F801 18C0 4040 030A C00C â..âÛU-0†t..âj.W"U.É†...¿ââ.¿.
186DA3A0 6201 301B 01EB 1791 1B27 040C 2320 0462 308A 45D0 031F C3FA A970 1600 1E80 1401 b.0..I.ê'..#.b00Ee..ÿ†op...â..
186DA3C0 821F D070 0E01 801F 8955 E088 D88B F30C 8886 011F 2168 7834 8887 01D5 1E87 E803 Ç.-p..â.âÛ†âÿâ0.âÛ..†xâââ'.âE.
186DA3E0 0000 003E 2F5F B7F0 7025 48FF 9CB3 EB5A 02C8 A000 0000 0000 0000 0000 0000 0000 ...¿_¿âpNH"âE¿Z.†
186DA400 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
186DA420 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
```



# Which leaves us with...

---

- Some answers...
  - We can quickly focus in on code loaded/injected at runtime
  - That code can be analyzed just as if it were sitting on disk
- But in general, more questions ...
  - How do we (or can we) get that list of commands we were promised?
  - What new tools & techniques are required (or even possible) against this class of malicious code?
  - How best to integrate more “context” available from the memory dump into the reverse engineering analysis?



the cake is a lie!

# Questions?

---

Jason Garman

[jason.garman@kyrus-tech.com](mailto:jason.garman@kyrus-tech.com)

