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

KYRUS TECHNOLOGY

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!

KYRUS TECHNOLOGY

Case Study: Poison Ivy

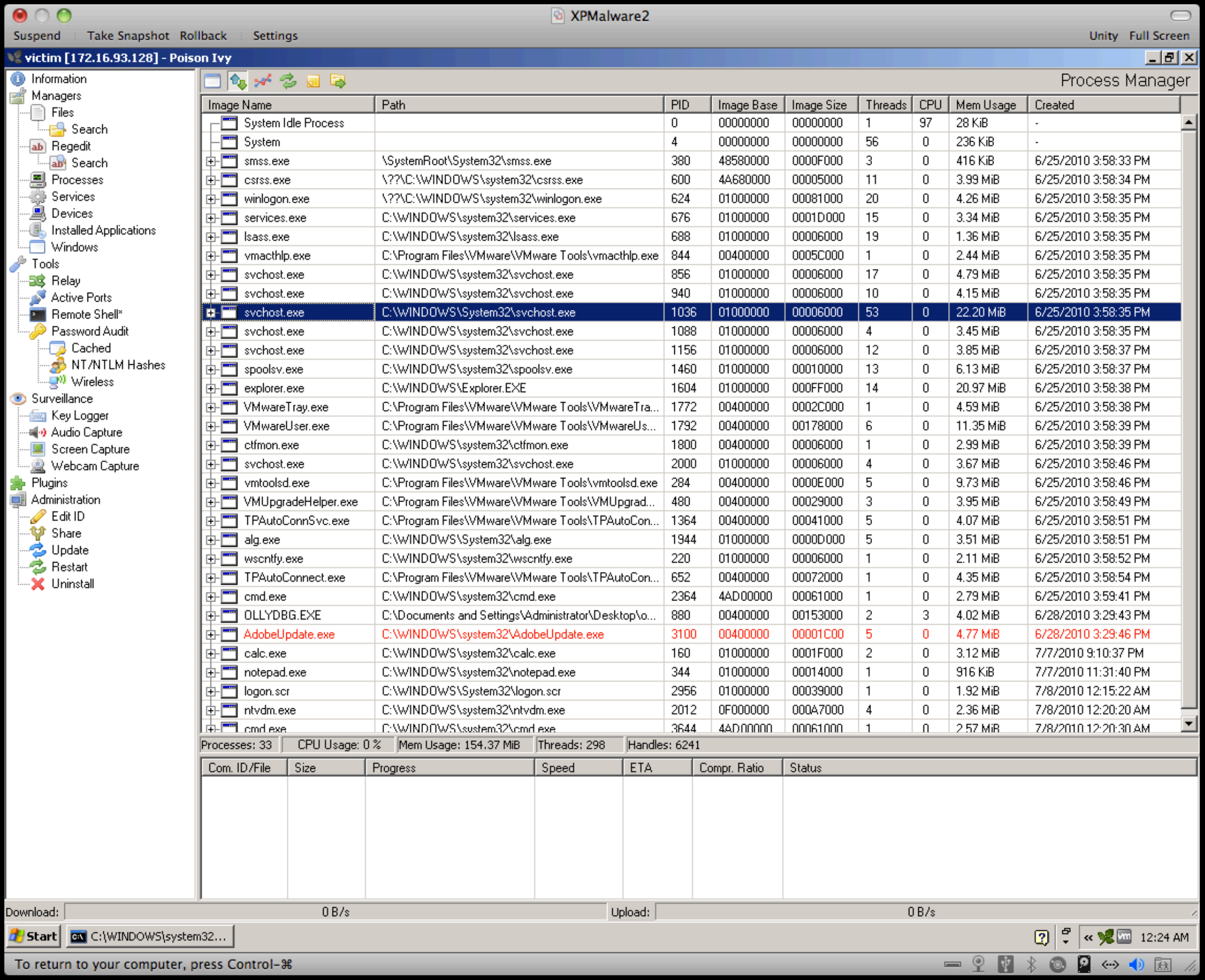
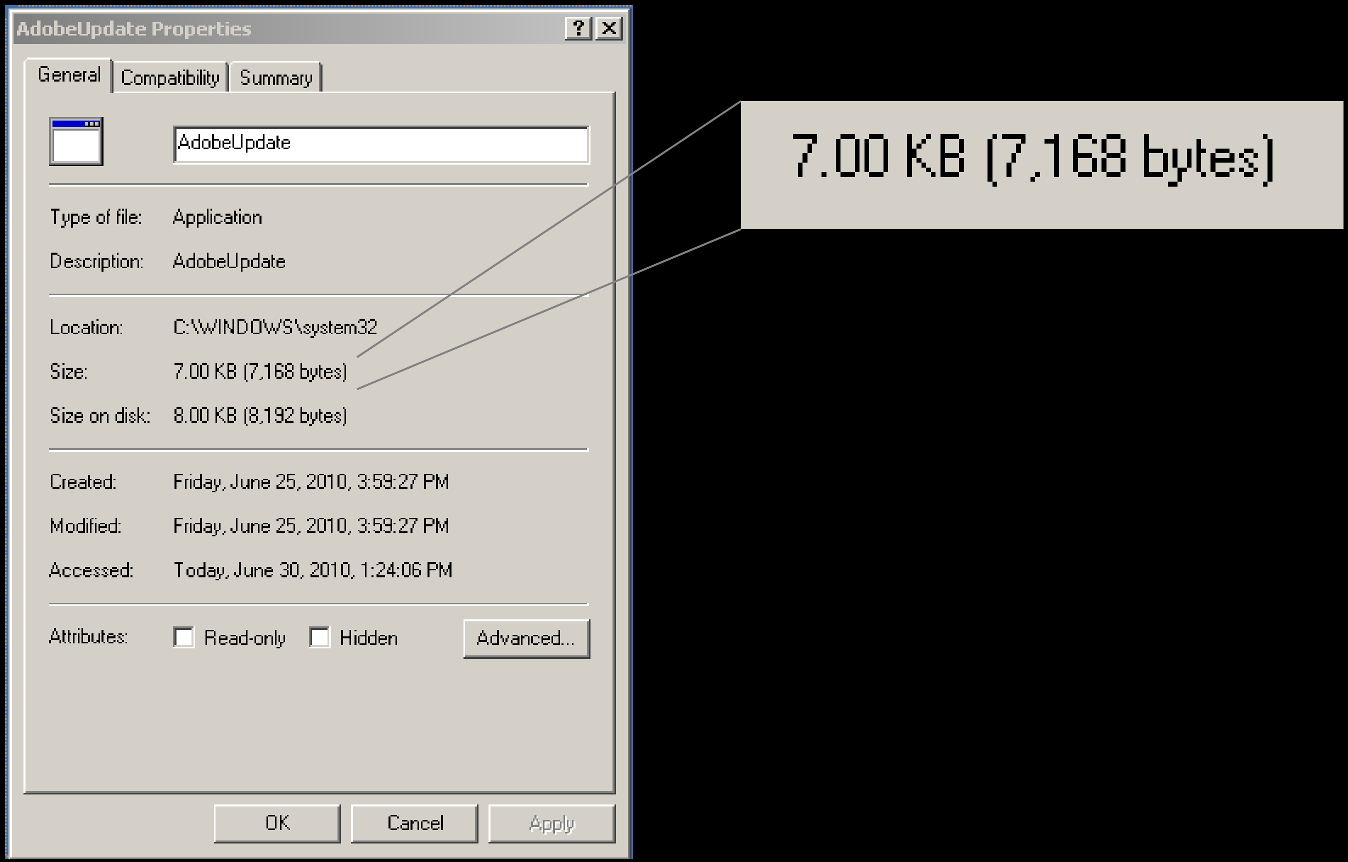
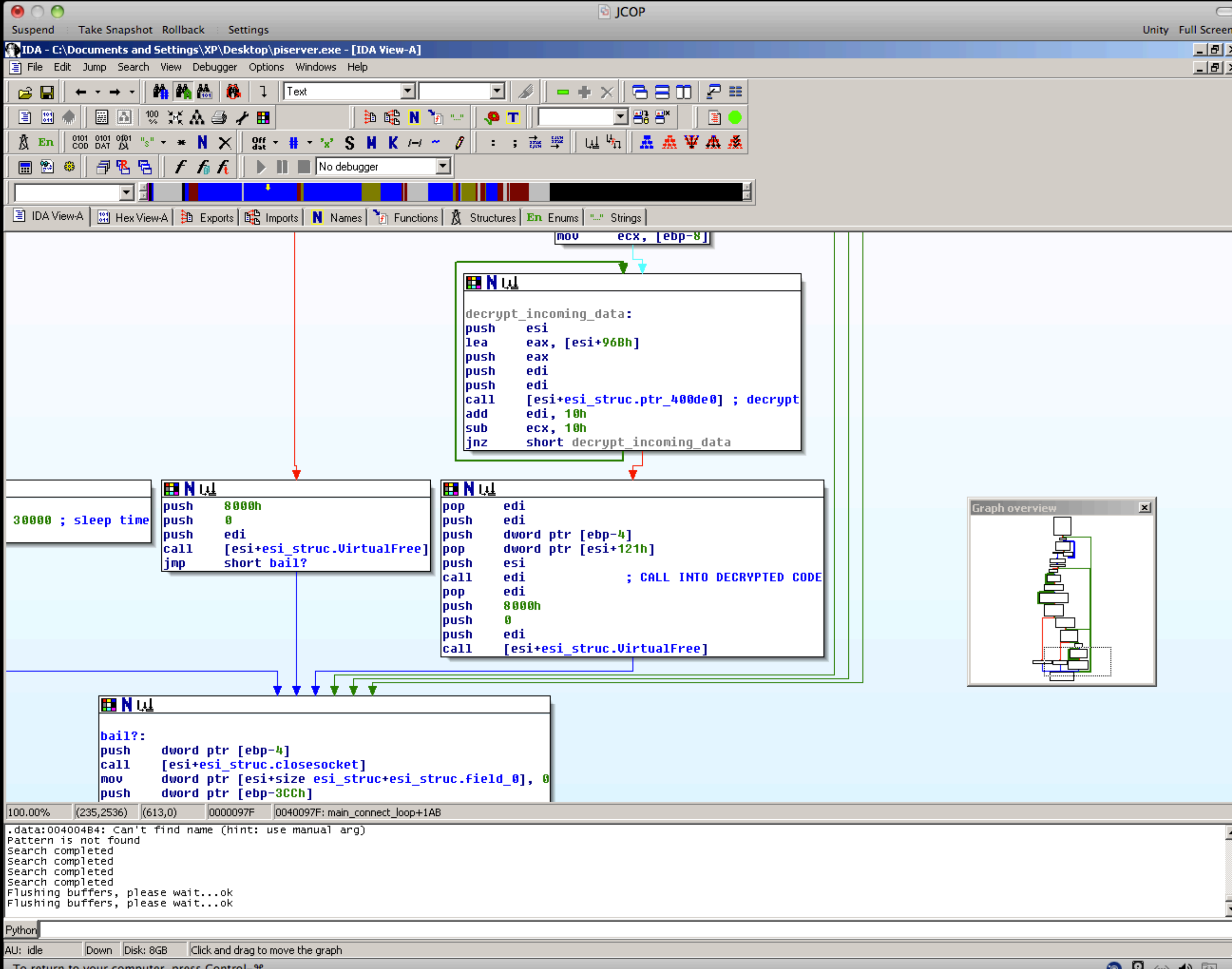


Image Name	Path	PID	Image Base	Image Size	Threads	CPU	Mem Usage	Created
System Idle Process		0	00000000	00000000	1	97	28 KiB	-
System		4	00000000	00000000	56	0	236 KiB	-
smss.exe	\SystemRoot\System32\smss.exe	380	48580000	0000F000	3	0	416 KiB	6/25/2010 3:58:33 PM
csrss.exe	\??\C:\WINDOWS\system32\csrss.exe	600	4A680000	00005000	11	0	3.99 MiB	6/25/2010 3:58:34 PM
winlogon.exe	\??\C:\WINDOWS\system32\winlogon.exe	624	01000000	00081000	20	0	4.26 MiB	6/25/2010 3:58:35 PM
services.exe	C:\WINDOWS\system32\services.exe	676	01000000	0001D000	15	0	3.34 MiB	6/25/2010 3:58:35 PM
lsass.exe	C:\WINDOWS\system32\lsass.exe	688	01000000	00006000	19	0	1.36 MiB	6/25/2010 3:58:35 PM
vmacthlp.exe	C:\Program Files\VMware\VMware Tools\vmacthlp.exe	844	00400000	0005C000	1	0	2.44 MiB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	856	01000000	00006000	17	0	4.79 MiB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	940	01000000	00006000	10	0	4.15 MiB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\System32\svchost.exe	1036	01000000	00006000	53	0	22.20 MiB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	1088	01000000	00006000	4	0	3.45 MiB	6/25/2010 3:58:35 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	1156	01000000	00006000	12	0	3.85 MiB	6/25/2010 3:58:37 PM
spoolsv.exe	C:\WINDOWS\system32\spoolsv.exe	1460	01000000	00010000	13	0	6.13 MiB	6/25/2010 3:58:37 PM
explorer.exe	C:\WINDOWS\Explorer.EXE	1604	01000000	000FF000	14	0	20.97 MiB	6/25/2010 3:58:38 PM
VMwareTray.exe	C:\Program Files\VMware\VMware Tools\VMwareTra...	1772	00400000	0002C000	1	0	4.59 MiB	6/25/2010 3:58:38 PM
VMwareUser.exe	C:\Program Files\VMware\VMware Tools\VMwareUs...	1792	00400000	00178000	6	0	11.35 MiB	6/25/2010 3:58:39 PM
ctfmon.exe	C:\WINDOWS\system32\ctfmon.exe	1800	00400000	00006000	1	0	2.99 MiB	6/25/2010 3:58:39 PM
svchost.exe	C:\WINDOWS\system32\svchost.exe	2000	01000000	00006000	4	0	3.67 MiB	6/25/2010 3:58:46 PM
vmtoolsd.exe	C:\Program Files\VMware\VMware Tools\vmtoolsd.exe	284	00400000	0000E000	5	0	9.73 MiB	6/25/2010 3:58:46 PM
VMUpgradeHelper.exe	C:\Program Files\VMware\VMware Tools\VMUpgrad...	480	00400000	00029000	3	0	3.95 MiB	6/25/2010 3:58:49 PM
TPAutoConnSvc.exe	C:\Program Files\VMware\VMware Tools\TPAutoCon...	1364	00400000	00041000	5	0	4.07 MiB	6/25/2010 3:58:51 PM
alg.exe	C:\WINDOWS\System32\alg.exe	1944	01000000	0000D000	5	0	3.51 MiB	6/25/2010 3:58:51 PM
wscntfy.exe	C:\WINDOWS\system32\wscntfy.exe	220	01000000	00006000	1	0	2.11 MiB	6/25/2010 3:58:52 PM
TPAutoConnect.exe	C:\Program Files\VMware\VMware Tools\TPAutoCon...	652	00400000	00072000	1	0	4.35 MiB	6/25/2010 3:58:54 PM
cmd.exe	C:\WINDOWS\system32\cmd.exe	2364	4AD00000	00061000	1	0	2.79 MiB	6/25/2010 3:59:41 PM
OLLYDBG.EXE	C:\Documents and Settings\Administrator\Desktop\o...	880	00400000	00153000	2	3	4.02 MiB	6/28/2010 3:29:43 PM
AdobeUpdate.exe	C:\WINDOWS\system32\AdobeUpdate.exe	3100	00400000	00001C00	5	0	4.77 MiB	6/28/2010 3:29:46 PM
calc.exe	C:\WINDOWS\system32\calc.exe	160	01000000	0001F000	2	0	3.12 MiB	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 MiB	7/8/2010 12:15:22 AM
ntvdm.exe	C:\WINDOWS\system32\ntvdm.exe	2012	0F000000	000A7000	4	0	2.36 MiB	7/8/2010 12:20:20 AM
cmd.exe	C:\WINDOWS\system32\cmd.exe	3644	4AD00000	00061000	1	0	2.57 MiB	7/8/2010 12:20:30 AM

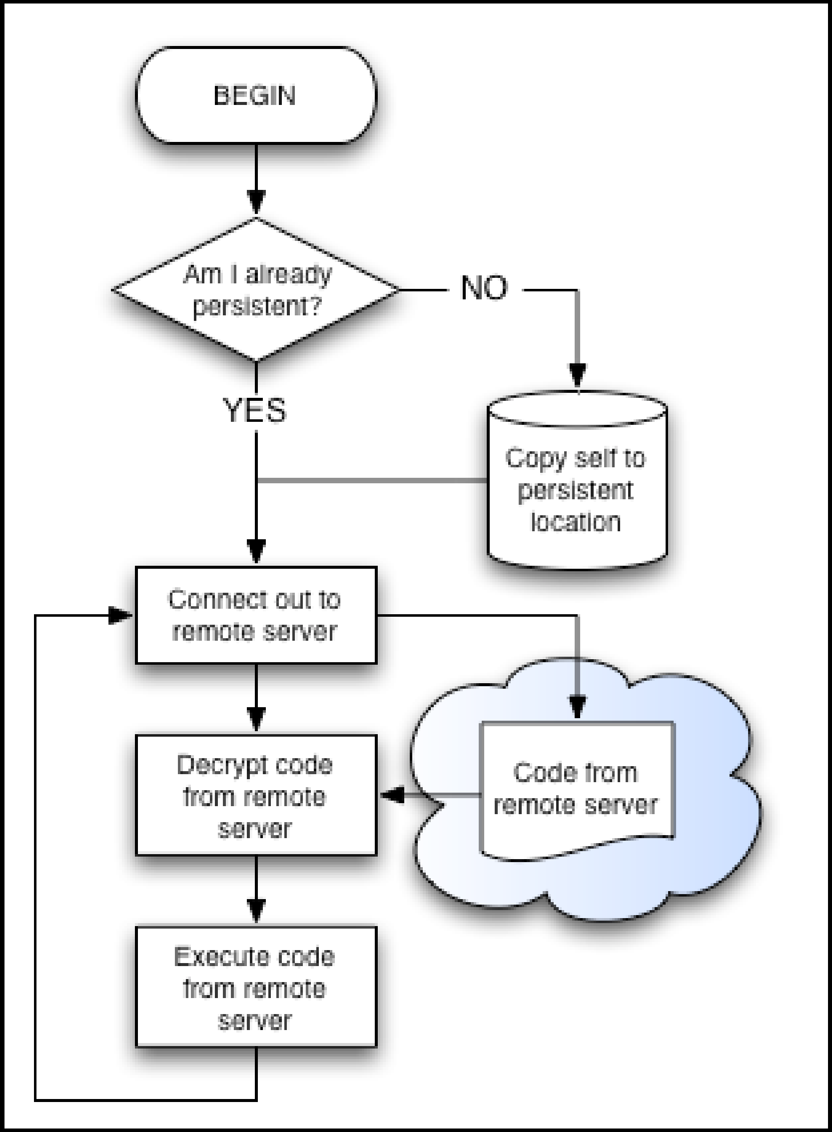
The Challenge



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


JCOP
Suspend Take Snapshot Rollback Settings
Unity Full Screen
IDA - C:\Documents and Settings\XP\Desktop\piserver.exe - [IDA View-A]
File Edit Jump Search View Debugger Options Windows Help
Text
No debugger
IDA View-A Hex View-A Exports Imports Names Functions Structures Enums Strings
mov ecx, [ebp-8]
decrypt_incoming_data:
push esi
lea eax, [esi+96Bh]
push eax
push edi
push edi
call [esi+esi_struc.ptr_400de0] ; decrypt
add edi, 10h
sub ecx, 10h
jnz short decrypt_incoming_data
30000 ; sleep time
push 8000h
push 0
push edi
call [esi+esi_struc.VirtualFree]
jmp short bail?
pop edi
push edi
push dword ptr [ebp-4]
pop dword ptr [esi+121h]
push esi
call edi ; CALL INTO DECRYPTED CODE
pop edi
push 8000h
push 0
push edi
call [esi+esi_struc.VirtualFree]
Graph overview
bail?:
push dword ptr [ebp-4]
call [esi+esi_struc.closesocket]
mov dword ptr [esi+size esi_struc+esi_struc.field_0], 0
push dword ptr [ebp-3CCh]
100.00% (235,2536) (613,0) 0000097F 0040097F: main_connect_loop+1AB
.data:004004B4: Can't find name (hint: use manual arg)
Pattern is not found
Search completed
Search completed
Search completed
Search completed
Flushing buffers, please wait...ok
Flushing buffers, please wait...ok
Python
AU: idle Down Disk: 8GB Click and drag to move the graph
To return to your computer, press Control-⌘

The 10,000 foot view



KYRUS TECHNOLOGY

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?

KYRUS TECHNOLOGY

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

KYRUS TECHNOLOGY

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…

KYRUS TECHNOLOGY

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`

KYRUS TECHNOLOGY

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
```

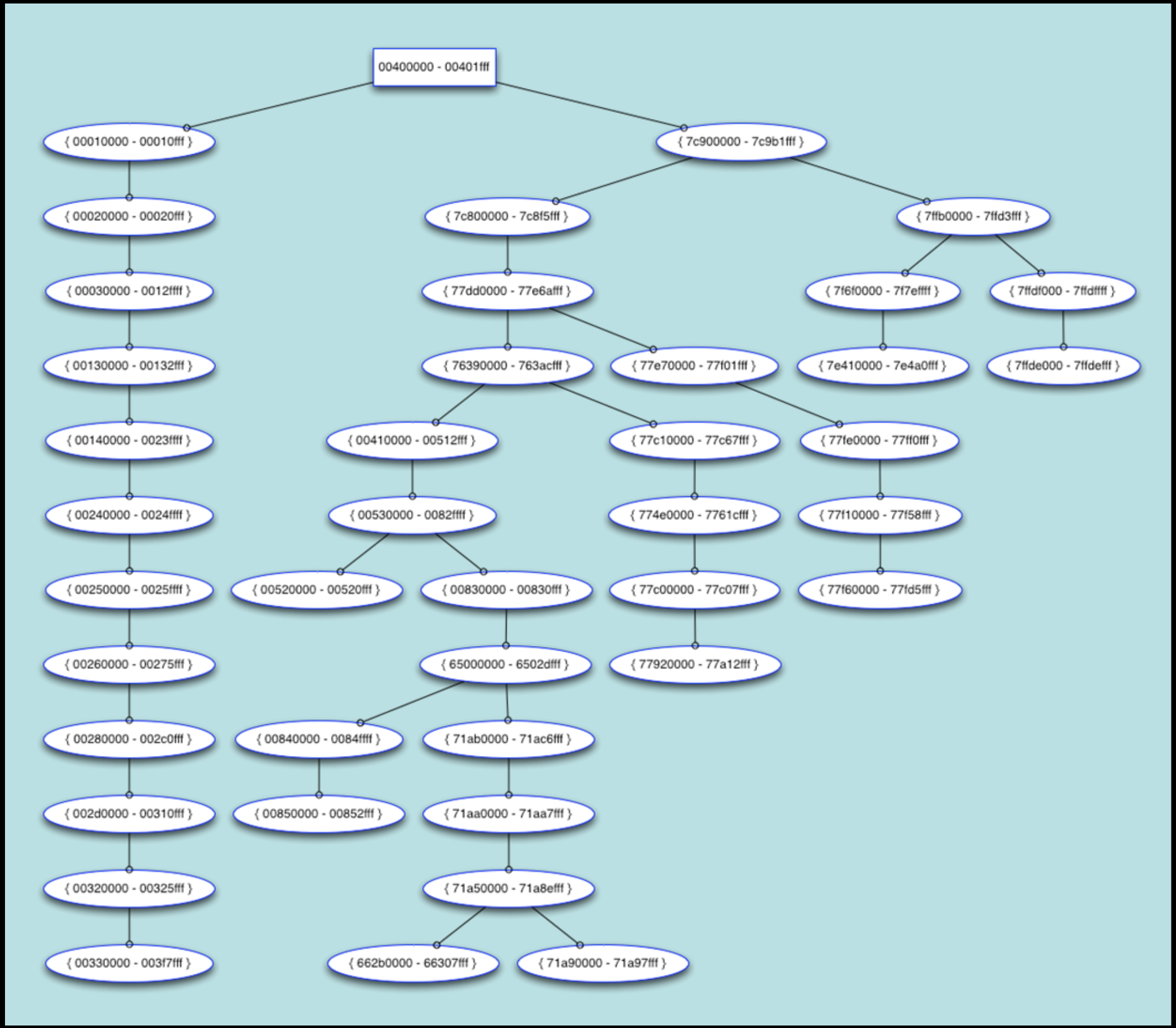
Getting to Executable Code…

- 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



KYRUS TECHNOLOGY

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: fffffffc
Flags2: Inherit
File offset: 00000000
```

The VAD info list

- Dynamically allocated memory looks a bit different:

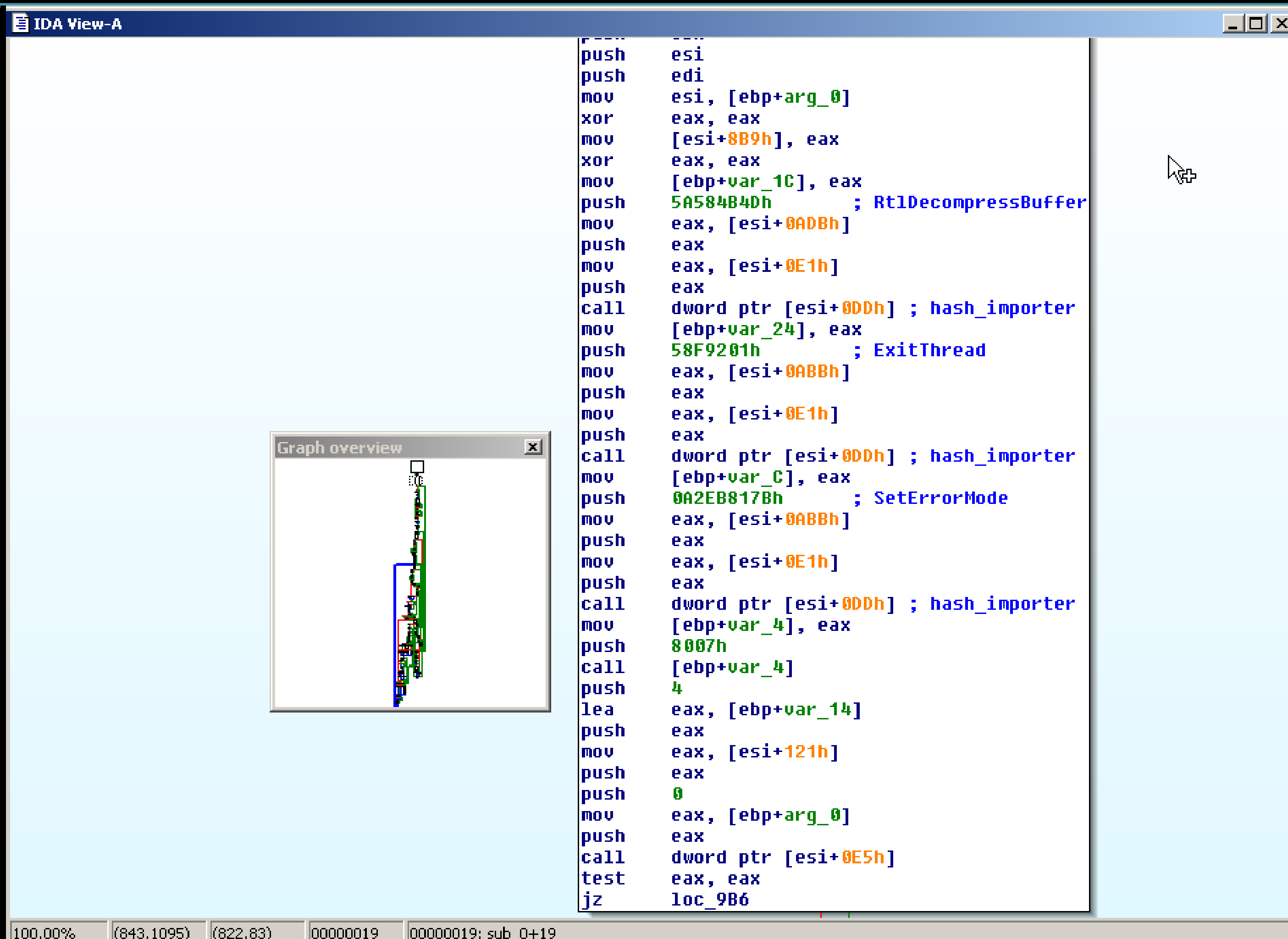
```
VAD node @81de8288 Start 00aa0000 End 00aa0fff Tag VadS
Flags: MemCommit, PrivateMemory
Commit Charge: 1 Protection: 6

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

KYRUS TECHNOLOGY

IDA Pro with Dynamically Loaded Modules

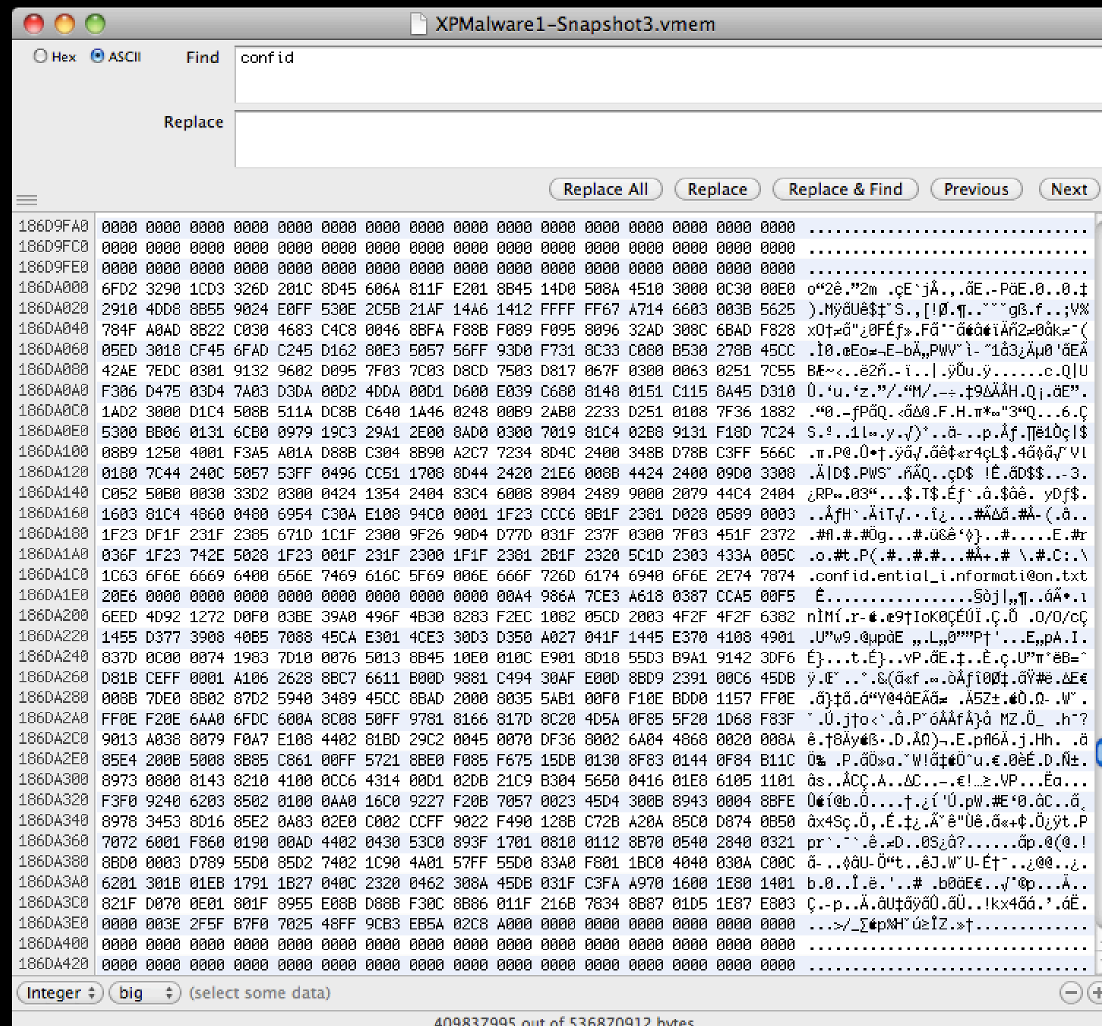


What are we missing?

- 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 vftable-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



KYRUS TECHNOLOGY

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