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 Challenge

- A 7kb file? Probably not much in there… but let’s try anyway.
The 10,000 foot view

BEGIN

Am I already persistent?

NO

YES

Copy self to persistent location

Connect out to remote server

Decrypt code from remote server

Execute code from remote server

Code from remote server
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 vaddrump 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:

```plaintext
\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
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
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?
Questions?

Jason Garman
jason.garman@kyrus-tech.com