Reconstructing the Scene of the Crime

METASPLOIT AUTOPSY
Who are they?

STEVE DAVIS
- Security Consultant / Researcher at MANDIANT

PETER SILBERMAN
- Engineer / Researcher at MANDIANT
Agenda

- ½ Demo
  - Pop it like its *hotttt*
- Problem / Solution
- Process Acquisition
- Metasploit
- Meterpreter Communication
- Metasploit Forensic Framework (MSFF)
- ½ Demo
  - Reconstructing it like its *hotttt*
Demo Part 1

- Box Windows XP Fresh SP3
  Same box that our slides are running from…
  Oh noes!

- MS08-067 meterpreter bind tcp
Back to our regularly scheduled slides...
Problem

- Meterpreter
  - Traditional disk forensics is helpless
    - Attack vector may never touch disk
  - No way to determine what happened

Goal

- Reconstruct attacker’s Meterpreter sessions with as much reliability as possible
Solution

- Acquire exploited processes’ address space
- Parse out meterpreter protocol from acquired memory sections
  - Reconstruct meterpreter sessions
MANDIANT Memoryze

ENUMERATION

- All running processes
  - Handle table
  - Memory sections
  - Ports
  - Strings
- Drivers
  - Including layered ones
- Certain kernel hooks

ACQUISITION

- Physical memory image
- Running process’s memory space
  - Binary
  - Loaded DLL’s
  - Stacks
  - Heaps
  - Data sections
- Drivers
MANDIANT Memoryze

- Can analyze memory live, or from image
  - Live analysis can use paging file for a more complete picture of memory

- Supported platforms
  - Beta support for Vista

- Download at
  - [http://www.mandiant.com/](http://www.mandiant.com/)
Process Acquisition
Why Process Acquisition?

- Acquisition was originally used mostly for malware analysis
  - Acquire packed binaries running in memory
    - Usually utilized debuggers
    - Can defeat most packers
- Acquisition has other uses:
  - Acquire unknown binaries for Virustotal
  - Acquire memory to look for protocol strings
    - Encrypted strings are unencrypted in memory
Classic Process Acquisition

- **Current Methodology**
  - Open handle to process, OR
  - Attach to process
    - `ReadProcessMemory(hProc, ImageBase, buffer, ImageSize, BytesRead)`

- **Current drawbacks**
  - Requires “touching” a process
  - Detecting debuggers is trivial
  - Gives an incomplete picture of memory
# Process Acquisition: Memoryze

<table>
<thead>
<tr>
<th>RELIES ON</th>
<th>DOES NOT RELY ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Physical memory access</td>
<td>- Attaching to a process with a debugger</td>
</tr>
<tr>
<td>- Virtual to physical address translation</td>
<td>- Opening handles to processes or threads</td>
</tr>
<tr>
<td></td>
<td>- API calls</td>
</tr>
<tr>
<td></td>
<td>- The OS’s Virtual Memory Manager</td>
</tr>
</tbody>
</table>
Memoryze: Process Acquisition

- Accessing Physical Memory
  - Live analysis
  - Acquisition
- \Device\PhysicalMemory
  - Section object exposed by Windows
  - Reading from handle allows application to read physical memory
  - Every virtual address must be translated to a physical offset within the section object
## Virtual Address

<table>
<thead>
<tr>
<th>Virtual Address Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Directory Index (512)</td>
</tr>
<tr>
<td>Page Table Index (512)</td>
</tr>
<tr>
<td>Byte Index (4096)</td>
</tr>
</tbody>
</table>

## Page Tables

- **Page Directory Index (512)**
- **Page Table Index (512)**
- **Byte Index (4096)**

## KPROCESS

- **Virtual Address**: Physical address

## Physical Memory

- **Physical Memory**
- **Page Frame**

## Page Dir Pointers

- **Page Dir Pointers**
- **PDP**

## PDE

- **Page Dir**
- **PDE**

## PTE

- **Individual page table**
- **PTE**
Memoryze: Process Acquisition

- Map physical memory into buffer
- Acquisition:
  - Write buffer to disk (dd)
- Analysis:
  - Scan buffer for known signatures of kernel structures, e.g. EPROCESS
New Process Acquisition

- Find all processes (EPROCESS) in physical memory
  - VadRoot within the EPROCESS structure
  - The VadRoot is the top node of a tree of Memory Manager Virtual Address Descriptor (MMVAD) entries
  - MMVAD entries contain the virtual start address and size of each memory section within a process
  - MMVAD entries containing mapped DLL’s or EXE’s will have a pointer to the path of the binary
    - Helps manage process’ virtual address space
Memoryze: Process Acquisition

- OllyDbg’s memory map view shows the different sections

<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Owner</th>
<th>Section</th>
<th>Contains</th>
<th>Type</th>
<th>Access</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010000</td>
<td>00001000</td>
<td></td>
<td></td>
<td></td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>00020000</td>
<td>00001000</td>
<td></td>
<td></td>
<td></td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>00030000</td>
<td>00001000</td>
<td></td>
<td></td>
<td></td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>0007E000</td>
<td>00001000</td>
<td></td>
<td></td>
<td>stack of ma</td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>0007C000</td>
<td>00004000</td>
<td></td>
<td></td>
<td></td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>00080000</td>
<td>00003000</td>
<td></td>
<td></td>
<td></td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>00090000</td>
<td>00002000</td>
<td></td>
<td></td>
<td>Map</td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>000A0000</td>
<td>00010000</td>
<td></td>
<td></td>
<td>Map</td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>001A0000</td>
<td>00006000</td>
<td></td>
<td></td>
<td>Map</td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
<tr>
<td>001B0000</td>
<td>00003000</td>
<td></td>
<td></td>
<td>Map</td>
<td>Priv</td>
<td>RW</td>
<td>RW</td>
</tr>
</tbody>
</table>

- Each address range is an entry in VadRoot, represented by a MMVAD structure
- Enumeration of VadRoot allows access to heaps, stacks, and binary images
Finding Processes

Kernel Address Space

0x8aadb830
0x8aadb834
0x8aadb838
0x8aadb83C
0x8aadb840
0x8aadb844
0x8aadb848
0x8aadb84C
0x8aadb850

...
Finding Processes

Kernel Address Space

0x8aadb830
0x8aadb834
0x8aadb838
0x8aadb83C
0x8aadb840
0x8aadb844
0x8aadb848
0x8aadb84C
0x8aadb850

...
Finding Processes

Indicates EPROCESS, DISPATCH_HEADER, further checks are needed
Finding Processes

Kernel Address Space

... 0x8aadb830
0x8aadb88
0x8aadb8c
0x8aadb90
0x8aadb94
...

EPROCESS

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>PCB</td>
</tr>
<tr>
<td>0xC4</td>
<td>ObjectTable</td>
</tr>
<tr>
<td>0x11C</td>
<td>VadRoot</td>
</tr>
<tr>
<td>0x174</td>
<td>ImageFileName</td>
</tr>
<tr>
<td>0x258</td>
<td>Cookie</td>
</tr>
</tbody>
</table>

Found an EPROCESS structure
HANDLE_TABLE
_MMVAD
“calc.exe”
Parsing MMVAD

- EPROCESS
  - 0x00: PCB
  - 0xC4: ObjectTable
  - 0x11C: VadRoot
  - 0x174: ImageFileName
  - 0x258: Cookie

- MMVAD
  - Parent
  - StartingVPN
  - EndingVPN
  - LeftChild
  - RightChild

- MMVAD
  - Parent
  - StartingVPN
  - EndingVPN
  - LeftChild
  - RightChild

- MMVAD
  - Parent
  - StartingVPN
  - EndingVPN
  - LeftChild
  - RightChild
Writing VADs to disk

For each VAD write to disk:
StartVPN to StartVPN+EndingVPN
New Process Acquisition

- Allows dumping of full address space
- Overcomes most binary packing
- Captures communication protocol strings
- Bypasses any anti-debugging techniques

Acquire(s):
- DLL’s that are only in memory
- Code corresponding to injected threads or shellcode
Metasploit

Have YOU read the developer docs?
Metasploit

- Open source exploit framework originally developed in Perl (1.x, 2.x) by HD Moore et al.
  - Currently Ruby (3.x)
- Platform independent
- Multiple payloads
Meterpreter

- The next generation of post-exploitation payloads
  - Forget /bin/sh and cmd.exe
    - Limited to stdin, stderr, stdout
    - Non-interactive
  - Full functioning client → server interpreter
    - File upload / download
    - Key logging
    - Simple extension addition
- Can be completely memory resident
Under the Meterpreter Hood

- DLL gets injected into exploited process
- Hooks LoadLibrary (on Windows)
  - Applies hook to Win32 API LoadLibrary
  - Changes lower level API’s behavior to allow LoadLibrary to load a DLL from memory
- Hooked API’s to allow loading of metsrv.dll from memory
  - NtOpenSection, NtCreateSection
  - NtQueryAttributesFile
  - NtOpenFile, NtMapViewOfSection
Meterpreter Communication

- TLV (really LTV) Structures
  - Provide communication protocol for meterpreter server and client
  - 32 bit Length and Type Fields
  - $n$ bits Value Field
Meterpreter Communication

Sends Exploit
Payload Meterpreter bind_tcp
Meterpreter Communication

Attacker executes "getpid"
Meterpreter Communication

Request sent when attacker executes getpid

TLV Packet

<table>
<thead>
<tr>
<th>Type</th>
<th>PACKET_TYPE_REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Sizeof(TLV Packet)</td>
</tr>
<tr>
<td>Value</td>
<td>stdapi_sys_process_getpid</td>
</tr>
</tbody>
</table>

Attacker

Victim

Meterpreter
Meterpreter Communication

Meterpreter does an internal lookup for the method requested: `stdapi_sys_process_getpid`
Meterpreter builds a response on the heap; response includes the result of GetCurrentProcessId.
Meterpreter Communication

Response is sent back to the attacker
Meterpreter Communication

Attacker

Response

Victim

Meterpreter

Response packet is freed by meterpreter
### Response Packet Structure (1 of 4)

<table>
<thead>
<tr>
<th>Response Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Length</strong></td>
</tr>
<tr>
<td>sizeof(Response Packet)</td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>PACKET_TLV_TYPE_PLAIN_RESPONSE</td>
</tr>
<tr>
<td><strong>Value</strong></td>
</tr>
<tr>
<td><strong>Length</strong></td>
</tr>
<tr>
<td>sizeof(this tlv)</td>
</tr>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TLV_TYPE_METHOD</td>
</tr>
<tr>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>stdapi_sys_process_getpid</td>
</tr>
</tbody>
</table>
### Response Packet Structure

<table>
<thead>
<tr>
<th>Response Packet</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td><code>sizeof(Response Packet)</code></td>
</tr>
<tr>
<td>Type</td>
<td><code>PACKET_TLV_TYPE_PLAIN_RESPONSE</code></td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLV_TYPE_REQUEST_ID</td>
<td>3164813846702899128916537536399</td>
</tr>
<tr>
<td>Response Packet</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>--</td>
</tr>
<tr>
<td><strong>Length</strong></td>
<td><code>sizeof(Response Packet)</code></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><code>PACKET_TLV_TYPE_PLAIN_RESPONSE</code></td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th><code>sizeof(this tlv)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td><code>TLV_TYPE_PID</code></td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><code>0x000003EC</code></td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Length</td>
<td><code>sizeof(Response Packet)</code></td>
</tr>
<tr>
<td>Type</td>
<td><code>PACKET_TLV_TYPE_PLAIN_RESPONSE</code></td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td><code>sizeof(this tlv)</code></td>
</tr>
<tr>
<td>Type</td>
<td><code>TLV_TYPE_RESULT</code></td>
</tr>
<tr>
<td>Value</td>
<td><code>0x00000000</code></td>
</tr>
</tbody>
</table>
## Response Packet Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Response Packet</strong></td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>sizeof(Response Packet)</td>
</tr>
<tr>
<td>Type</td>
<td>PACKET_TLV_TYPEPLAIN_RESPONSE</td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>sizeof(this tlv)</td>
</tr>
<tr>
<td>Type</td>
<td>TLV_TYPE_METHOD</td>
</tr>
<tr>
<td>Value</td>
<td>stdapi_sys_process getpid</td>
</tr>
<tr>
<td>Length</td>
<td>sizeof(this tlv)</td>
</tr>
<tr>
<td>Type</td>
<td>TLV_TYPE_REQUEST_ID</td>
</tr>
<tr>
<td>Value</td>
<td>3164813846702899128916537536399</td>
</tr>
<tr>
<td>Length</td>
<td>sizeof(this tlv)</td>
</tr>
<tr>
<td>Type</td>
<td>TLV_TYPE_PID</td>
</tr>
<tr>
<td>Value</td>
<td>0x000003EC</td>
</tr>
<tr>
<td>Length</td>
<td>sizeof(this tlv)</td>
</tr>
<tr>
<td>Type</td>
<td>TLV_TYPE_RESULT</td>
</tr>
<tr>
<td>Value</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
Response Packet from Memory

TLV Packet

<table>
<thead>
<tr>
<th>Length</th>
<th>Doesn’t exist do to free()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>TLV_TYPE_METHOD 0x00010001</td>
</tr>
<tr>
<td>Value:</td>
<td>stdapi_sys_process_getpid</td>
</tr>
</tbody>
</table>
Response Packet from Memory

TLV Packet

<table>
<thead>
<tr>
<th>Length</th>
<th>0x29</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>TLV_TYPE_REQUEST_ID 0x00010002</td>
</tr>
<tr>
<td>Value:</td>
<td>3164813846702899128916537536399</td>
</tr>
</tbody>
</table>
### TLV Packet

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>0x0C</td>
</tr>
<tr>
<td>Type:</td>
<td>TLV_TYPE_PID</td>
</tr>
<tr>
<td>Value:</td>
<td>0x000003EC</td>
</tr>
</tbody>
</table>

- **Response Packet from Memory**

```
08 74 04 06 00 01 00 01 73 74 64 61 70 69 5F 73 ; .t.....stdapi_s
79 73 5F 70 72 6F 63 65 73 73 5F 67 65 74 70 69 ; ys_process_getpi
64 00 00 00 00 29 00 01 00 02 33 31 36 34 38 31 ; d....}....316481
33 38 34 36 37 30 32 38 39 39 31 32 38 39 31 36 ; 3846702899128916
35 33 37 35 33 36 33 39 39 34 00 00 00 00 DC 00 ; 5375363994.......02 08 FC 00 00 03 EC 00 00 00 0C 00 02 00 04 00 ; ....ü.........00 00 00 00 01 48 05 98 01 0B 00 0E 00 C7 01 0E 00 ; .....H.~.....Ç...
```
Response Packet from Memory

<table>
<thead>
<tr>
<th>TLV Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Length</strong></td>
</tr>
<tr>
<td><strong>Type:</strong></td>
</tr>
<tr>
<td><strong>Value:</strong></td>
</tr>
</tbody>
</table>
Meterpreter Communication

- The response packet is freed by meterpreter
- However...
- When Windows’ memory manager frees memory, it is not *immediately* reused.
  - It can take hours for memory to be reclaimed after it has been freed.
Metasploit Forensic Framework

Finding one pwned system at a time
Metasploit Forensic Framework

- Scan acquired VADs looking for:
  - Strings containing meterpreter methods
    - This indicates a TLV response to a specific method
    - Parsing out the response TLV gives analysts the data attackers received
      - Also indicates what commands were executed on the machine
Conclusion

- Windows memory manager gives analysts a chance to see artifact memory
- Large impact for forensics
  - Not so large on Metasploit project
- Combining memory analysis with further research will lead to better and more effective projects
Demo Part 3

- Acquire svchost.exe
  - Remember attacker terminated connection *roughly* 30 minutes ago
  - Run Metasploit Forensic Framework (msff)
Questions???

- stephen.davis@mandiant.com
- peter.silberman@mandiant.com