

# Administrators Intro to Debugging

Michael Morales

(Reprinted From WindowsItPro Magazine)

## Executive Summary

As a Microsoft Windows administrator, in certain situations the tools you usually use to diagnose Windows system problems won't work. For those cases, the Debugging Tools for Windows can be a useful alternative troubleshooting tool. Get acquainted with the debugger by walking through this real-life case of troubleshooting the cause of bloating heap memory on a system.

As a Windows administrator, you realize the crucial role that tools play in helping you diagnose and resolve system problems. Previously, I've discussed two essential tools that help identify process leaks: user-mode dump heap (UMDH) and DebugDiag. (See this article's Learning Path for links to these and my other columns.) However, sometimes these tools add too much overhead to the system, causing high CPU usage and creating a situation that makes running UMDH or DebugDiag impractical. When your diagnostic tools fail or become unavailable, the problem becomes much more difficult to solve, so having a backup troubleshooting plan is vital.

Do you have an alternative approach when you can't use your regular set of troubleshooting tools? My aim in this article is to provide you with at least one backup plan and also demonstrate the importance of learning about Windows internals and using the Windows debugger to find helpful nuggets of diagnostic information.

## Debugging a Bloated Process

Recently we resolved a customer issue where `wmiprvse.exe` (the WMI Provider Host process) was consuming increasing amounts of memory. The customer provided a dump file to help determine the root cause. We started by enabling DebugDiag and UMDH separately. However, when each tool ran, we saw that the CPU was constantly spiked at 100 percent, thus freezing the system. Since our regular troubleshooting tools overstressed the system, we needed to turn to our backup plan.

We had only the 185MB `wmiprvse.exe` process dump file to help us identify why the process had bloated to such a large size and what, if anything, the customer could do to decrease the memory consumption. First we opened the dump file using the `windbg.exe` debugger included in the Debugging Tools for Windows. The debugging toolset is a must-have for any administrator who wants to dig a little deeper into system problems. You can retrieve a lot of information from a dump file using the debuggers without being a debugging expert or having a lot of code knowledge.

To debug the dump file, we followed these steps:

1. Start `windbg.exe`
2. Ensure that the Symbol path is set correctly by clicking File, Symbol File Path, then typing the following path into the box provided:

```
srv*c:\symbols*http://msdl.microsoft.com/download/symbols
Symbols are files that translate machine code into easy-to-read function
calls.)
```

## Administrators Intro to Debugging

Michael Morales

(Reprinted From WindowsItPro Magazine)


3. Open the wmiprvse.dmp file in the Windbg debugger by selecting File, then clicking Open Crash Dump.

We then entered this command in the debugger:

```
!address -summary
```

This is a handy command because it provides a summary of the type of memory consumed within the process, as you can see in Figure 1.

```
----- Usage SUMMARY -----
TotSize (      KB) Pct(Tots) Pct(Busy)  Usage
140b000 (   20524) : 00.98%  08.14%  : RegionUsageIsVAD
709b8000 ( 1844960) : 87.98%  00.00%  : RegionUsageFree
2636000 (   39128) : 01.87%  15.52%  : RegionUsageImage
4c0000 (    4864) : 00.23%  01.93%  : RegionUsageStack
 13000 (      76) : 00.00%  00.03%  : RegionUsageTeb
b720000 (  187520) : 08.94%  74.37%  : RegionUsageHeap
   0 (      0) : 00.00%  00.00%  : RegionUsagePageHeap
 1000 (      4) : 00.00%  00.00%  : RegionUsagePeb
 1000 (      4) : 00.00%  00.00%  : RegionUsageProcessParametrs
 2000 (      8) : 00.00%  00.00%  : RegionUsageEnvironmentBlock
Tot: 7fff0000 (2097088 KB) Busy: 0f638000 (252128 KB)
```



**Figure 1**  
**Output Of !Address -Summary Command**

The most important column in the output is the Pct(Busy) column, which represents the type of memory consumed on a percentage basis. We quickly scanned this column and identified that the RegionUsageHeap memory was the highest consumer at 74 percent. RegionUsageHeap is the label for heap memory, an area of memory reserved for processes to store data. Every process has heap memory, which is initially 1MB by default. However, this area can grow, and more memory will be allocated for the heap as the process requires.

Understanding that heap memory is where a program stores its data was a key component in our investigation. You can think of heap memory as a bucket represented as an address of memory where applications store process-specific data. Since a process can have several heap "buckets," it's important to know which bucket you want to explore because not every heap will be filled with data and high in memory usage. Our goal was to dump out the heap bucket consuming the highest amount of memory to try to identify some clues about why the process was consuming so much memory.

# Administrators Intro to Debugging

Michael Morales

(Reprinted From WindowsItPro Magazine)

## Consulting the Debugger Help File

As I mentioned, a process will have several heap buckets available to explore, so we needed a way to conduct a more targeted investigation. This is where the debugger Help file (debugger.chm) saved the day. I knew that we needed to use the !heap debugger command to investigate a heap-related problem in a process. However, when you aren't sure what debugger command to use, try searching in debugger.chm for a term relevant to your investigation. For example, if you search for the term "heap," the first hit you receive is the !heap command and all the !heap parameters.

Running the !heap command displays a list of heap memory addresses, but this list alone wouldn't provide us enough information to find the heap bucket containing the highest amount of memory usage. As I scrolled down through the !heap parameters, I noticed the -stat parameter, which displays heap usage statistics. Remembering our goal was to find the heap (or bucket) containing the most memory, I ran this command:

```
!heap -stat
```


As Figure 2 shows, running this command displays each heap in descending order, from highest to lowest consumer. So the first heap displayed is the heap address we need to investigate.

## Administrators Intro to Debugging

Michael Morales

(Reprinted From WindowsItPro Magazine)

```
0:000> !heap -stat
_HEAP 00700000
  Segments      00000008
  Reserved bytes 08000000
  Committed bytes 06c03000
  VirtAllocBlocks 00000000
  VirtAlloc bytes 00000000
_HEAP 00090000
  Segments      00000006
  Reserved bytes 02000000
  Committed bytes 00881000
  VirtAllocBlocks 00000000
  VirtAlloc bytes 00000000
_HEAP 00030000
  Segments      00000003
  Reserved bytes 00310000
  Committed bytes 002a1000
  VirtAllocBlocks 00000000
  VirtAlloc bytes 00000000
```



**Figure 2**  
**Running !Heap -Stat To View Highest-To-Lowest Heap Address**

Notice in Figure 2, each heap bucket is designated with a specific address. The first heap bucket is designated by the 00700000 address; the next heap is located at address 00090000. The important row is Committed bytes—the amount of memory committed in this particular heap. And we know that heap 00700000 is the highest-memory-consuming heap because the -stat parameter displays each heap in highest-to-lowest order based on memory consumption. For further confirmation, we can convert the committed bytes hexadecimal number 06c03000 to decimal either using calc.exe (i.e., the Windows calculator) or within the debugger by using the ? (Evaluate Expression) command, as follows:

```
0:000> ?06c03000
Evaluate expression: 113258496 = 06c03000
```

# Administrators Intro to Debugging

Michael Morales

(Reprinted From WindowsItPro Magazine)

(The second line is the command's output.) 113258496 is 06c03000 converted into decimal, so this output confirms for us that this heap address contains 113MB of memory. Considering the entire process consumed 185MB of memory, we can be fairly certain that we're on the right track. We know our process is bloating because of heap usage, and we know that our highest-consuming heap contains 113MB. Now we can enter debugger commands that will display the contents of this heap, which may point to hints about why the process is utilizing so much memory. Simply displaying the heap might not reveal a smoking gun, but the information you find could point you in the right direction, as it did for us.

To dump out the heap, we use the dc command (which displays the values as ASCII characters) followed by the heap address. So in our case, the command and partial output would look like that in Figure 3.

The output in Figure 3 reveals the beginning part of the heap.

**Figure 3**  
**Using The Dc Command To Display A Heap's Contents**

```
0:000> dc 00700000
00700000 000000c8 0000018d eeffe0ff 00001002 .....
00700010 00000000 0000fe00 08000000 00002000 .....
00700020 00000200 00002000 000575c7 7ffdefff .....u.....
00700030 06080006 00000000 00000000 00000000 .....
00700040 01d10000 01d100e0 0000000f ffffffff8 .....
00700050 00700050 00700050 00700640 01800000 P.p.P.p.@.p.....
00700060 02610000 029d0000 04380000 05830000 ..a.....8.....
```

Usually you'll need to keep dumping out more and more heap (by pressing Enter after the initial dc command output is displayed), to reveal interesting information. Figure 4 shows the section of the dump containing information that helped us deduce that our heap memory was consumed with information generated by Event Tracing for Windows (ETW).

**Administrators Intro to Debugging**  
Michael Morales  
(Reprinted From WindowsItPro Magazine)

**Figure 4**  
**Section Of Dump Revealing High Memory Consumption By ETW**

```
0:000>
00701780  00000b00  00000000  0000c000  00000000  .....
00701790  00000000  0015b800  00000400  00b00100  .....
007017a0  54008000  65636172  67676f4c  00007265  ...TraceLogger..
007017b0  5453534d  65636172  00000100  00001f00  MSTrace.....
007017c0  387b0000  32384533  2d393832  41393231  ..{93E82289-129A
007017d0  4544342d  33422d30  412d4335  30463730  -4DF0-B35C-A07F0
007017e0  30303541  7d303532  00000100  00000000  A500250}.....0:000>
00701800  6c616974  65500000  00006672  535c3a64  tial..Perf..D:\App
00701810  63656570  74614468  6f4c5c61  4d5c7367  Data\Logs\M
00701820  72545353  5f656361  39303032  31313230  STrace_20080615
00701830  3132325f  5f313035  43646172  32464332  _221501_SERVER1
00701840  6c74652e  72540000  4c656361  6567676f  .etl..TraceLogger
```

Notice we even found the filename and location in the dump:

```
D:\AppData\Logs\MSTrace_20080615_221501_SERVER1.etl.
```

### Problem Solved

Our biggest clue that the data contained in the heap was indeed ETW tracing-related information was the file extension .etl, which is associated with ETW. As it turned out, the customer hadn't realized ETW tracing was still enabled for a previous problem resolved months before. Turning off ETW tracing resolved the customer's problem, and the wmiprvse.exe process's memory consumption decreased immediately. By knowing a few debugger commands and a little about OS internals, you can successfully troubleshoot when your usual tools aren't available.

Special thanks to Venkatesh Ganga, a Microsoft senior escalation engineer, who contributed to this article.