Memory Forensics

Always Test Your Forensics Tools

A Rendition InfoSec White Paper

Jake Williams, Principal Consultant, @Malware Jake

Brandon McCrillis, Senior Security Analyst, @13M4C

October 2015

Purpose

Always test your forensics tools. This applies to new versions of those that you've used for years as well as those that you are just adopting. Just because the tool is the new hotness and "everyone is using it" doesn't mean that it doesn't have bugs that can impact your investigation (and your employment).

Why Memory Forensics

Rendition InfoSec (RIS) stresses the need for memory forensics in our investigations. We routinely are able to find activity (or get additional data around) malicious activity when we examine memory. One of the RIS founders is the co-author of the SANS Memory Forensics class – you can't work here and ignore memory. Memory isn't just for malware, insider cases benefit from a thorough memory analysis as well. If you're not checking memory, you're leaving evidence at the crime scene.

But before we can start memory analysis, memory acquisition has to occur. How can we acquire memory using the least intrusive methods possible? Ideally, the tool used to acquire memory will have a minimally invasive footprint for itself, leaving as much memory available for analysis as possible.

WinPmem 2.x Appears Buggy

At RIS, we had not yet updated to WinPmem 2.x for memory acquisition, mostly due to the output format being in AFF4. This required a conversion step to view the memory in any tool other than Rekall, which while admittedly great, doesn't cover all of our needs. After reading Brent Muir's slide show on Windows 10 forensic artifacts (http://www.slideshare.net/bsmuir/windows-10-forensics-os-evidentiary-artefacts), we decided to take a look at WinPmem if for no other reason than to confirm that version 2.x was actually leaving an 80MB footprint in memory on his machine. The 10MB footprint claimed for version 1.6 also seemed high, so we were skeptical as testing began. RIS also wanted to know that if 80MB were correct, was it specific to Windows 10 or was there some issue in WinPmem 2.x that impacted all version of Window?

Testing

For testing, we ran WinPmem 1.6.2 and 2.0.1 on a variety of different Windows versions to observe the memory used. For this particular issue, the most important number is the 'Peak Private Bytes' since this is the highest amount of memory actually used by the application (and hence overwritten) vs. simply being reserved for potential future use or including objects shared between processes. Note that because WinPmem 2.x supports compression, we tested with and without compression in case that contributed to the high memory usage. WinPmem 1.6.2 does not support compression so that option could not be evaluated.

Windows 10 (x64, 32GB RAM)

The Peak Virtual Bytes for this operation were 1.8MB.

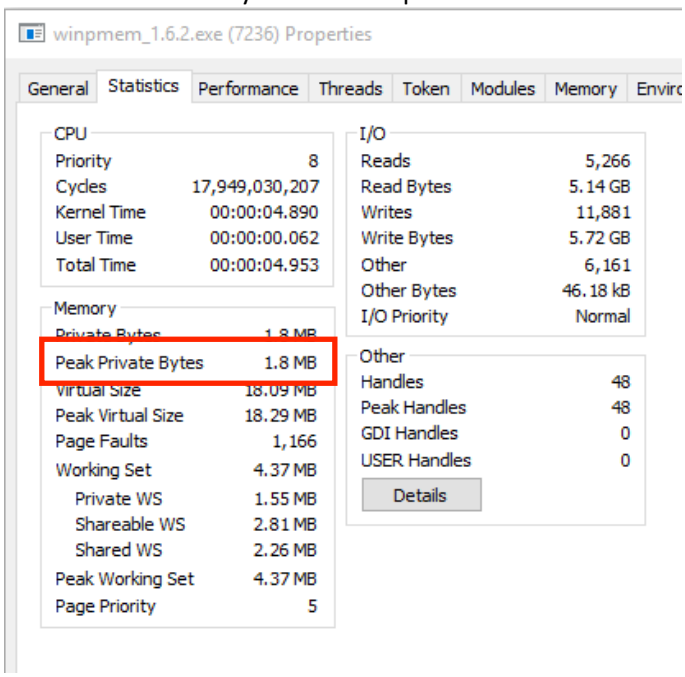

Figure 1-WinPmem 1.6.2

The Peak Virtual Bytes for this operation were 94.81 MB.

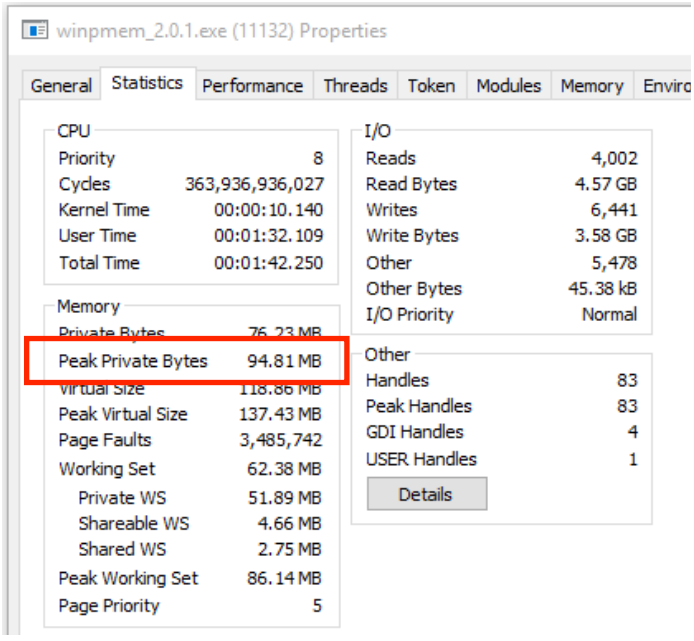

Figure 2-WinPmem 2.0.1 with Default Compression (zlib)

The Peak Virtual Bytes for this operation were 96.96 MB.

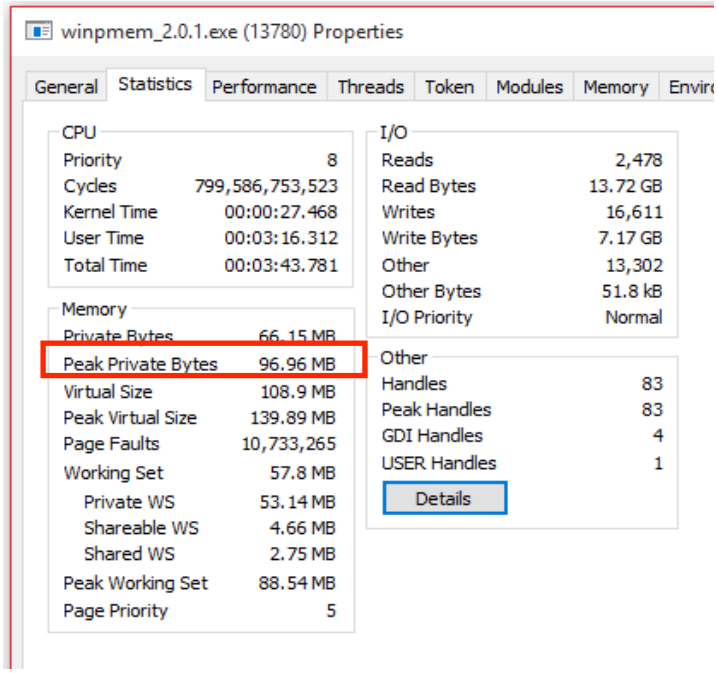

Figure 3-WinPmem 2.0.1 with No Compression

The Peak Virtual Bytes for this operation were 96.87 MB.

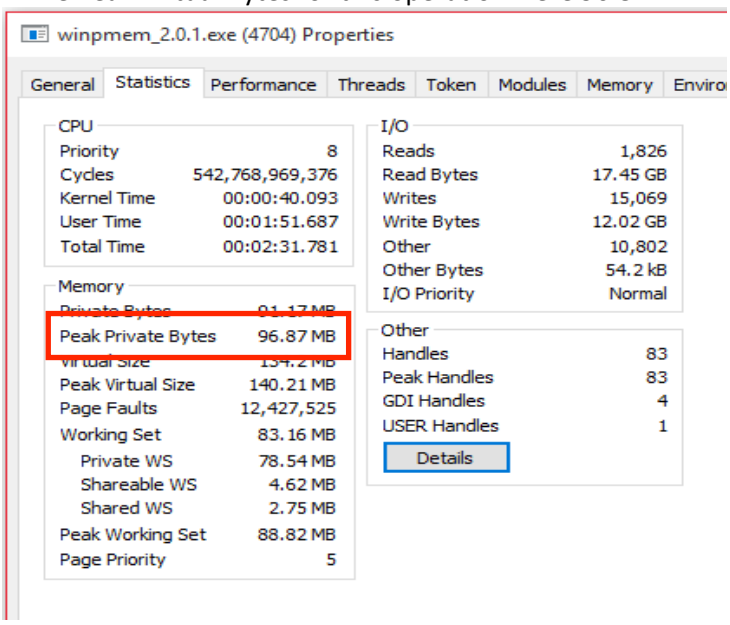

Figure 4-WinPmem 2.0.1 with Snappy Compression

Windows 7 (x86, 4GB RAM)

Note that the Windows 7 machine was x86 and had only 4G of RAM. It was a VMWare virtual machine.

The Peak Virtual Bytes for this operation were 1.35 MB.

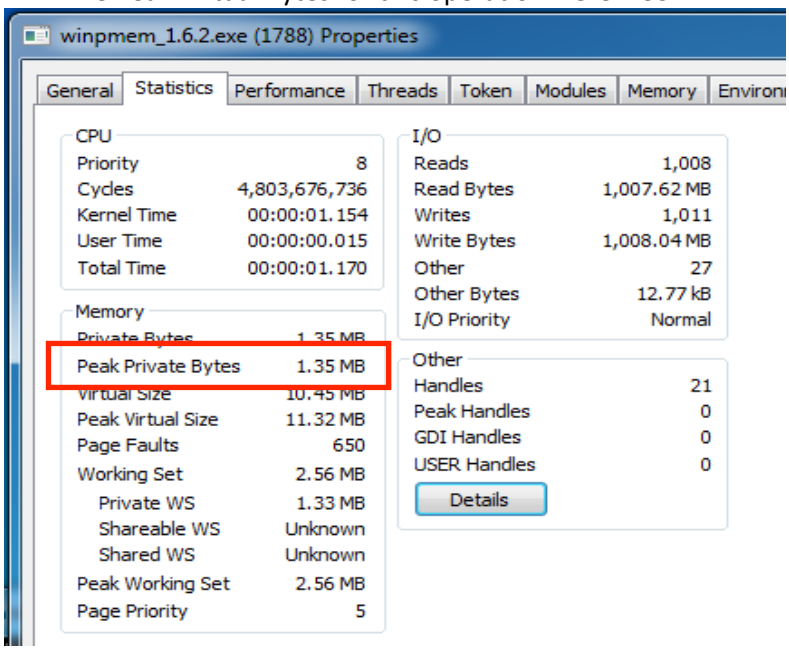

Figure 5-WinPmem 1.6.2

The Peak Virtual Bytes for this operation were 96.63 MB.

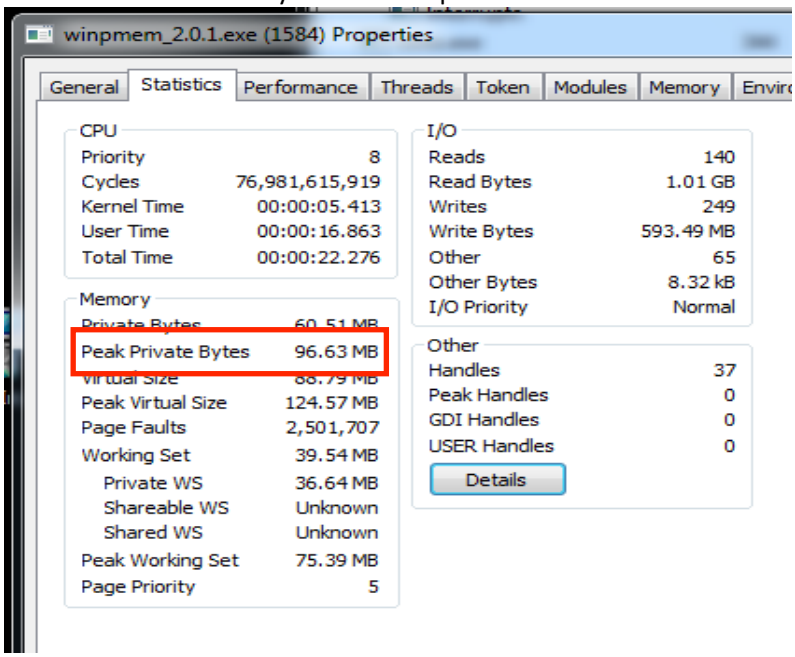

Figure 6-WinPmem 2.0.1 with Default Compression (zlib)

The Peak Virtual Bytes for this operation were 96.14 MB. Note that on Windows 7, snappy compression caused winPmem to crash on exit. RIS does not recommend snappy compression for this reason.

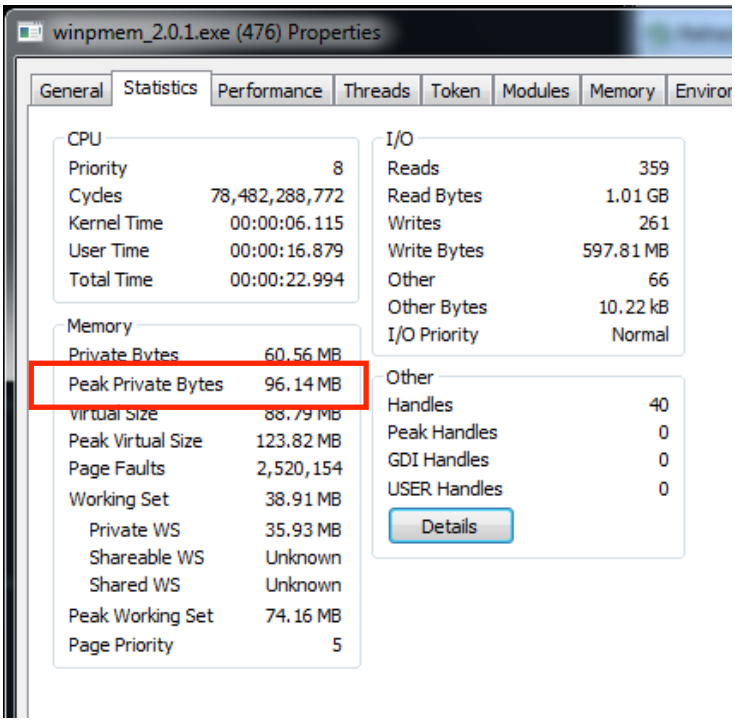

Figure 7-WinPmem 2.0.1 with Snappy Compression

Windows 8.1 x64

RIS also tested Windows 8.1 x64 and found similar results.

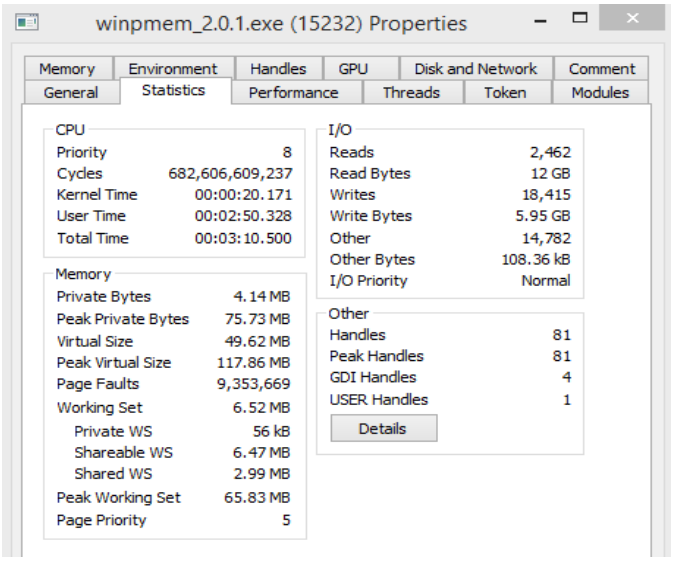

Figure 8 - WinPmem 2.0.1 on Win 8.1

WinPmem Author Comments

After testing, we confirmed that all versions of Windows are impacted and the memory usage is rather extreme in version 2.x. RIS contacted the author, Michael Cohen, who informed us that WinPmem 2.x has largely been deprecated in favor of the new aff4acquire plugin built into Rekall. He also explained that the memory use in winpmem 2.x is mainly due to the AFF4 cache, which was originally built as a purely ready-only tool.

RIS tested the memory use of the aff4acquire plugin in Rekall and found the memory usage depended on the compression algorithm used. If the default compression mechanism (snappy) was used, memory usage was 588 MB. When zlib compression was used, memory consumption peaked at 382MB on our test machine. In either case, the 93.8MB used by WinPmem 2.x looked good by comparison.

Additional Notes

Alissa Torres suggested that ELF output might result in lower memory usage from WinPmem 2.x. RIS tested the ELF output formats (nonu default) and found that indeed memory usage was substantially lower with ELF than AFF4. However, 22.6MB is still considerably higher than the 1.8MB used with WinPmem 1.6.2.

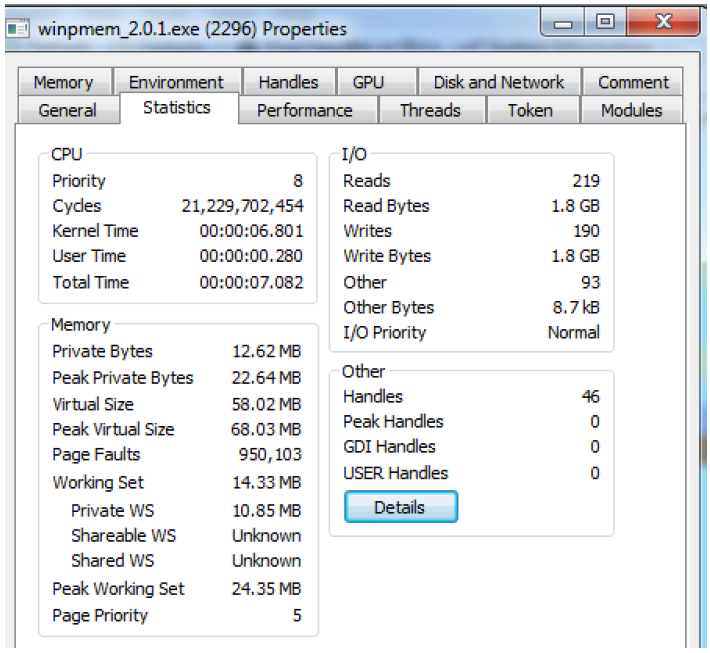

Figure 9 - WinPmem 2.0.1 with ELF output

Note that WinPmem 2.x does not support outputting to raw format.

Conclusion

Note that if your investigation only focuses on allocated memory structures, the memory use doesn't matter. If, however, the investigation may make use of unallocated structures (e.g. terminated processes, terminated network connections, closed file handles, etc.) then all things being equal, you'll want the tool with the lowest memory overhead. Every 4k page that the memory acquisition tool allocates is one more 4k page of unallocated memory that is gone forever.

RIS recommends carefully testing your memory acquisition tools in the environment where they will be used. RIS has not experienced issues with WinPmem 1.6.2. and will likely continue to use that due to its low memory consumption unless problems are encountered.

Credits:

Brent Muir for his Windows 10 Forensics Artifacts presentation

Michael Cohen (@scudette) for writing WinPmem

Alissa Torres (@sibertor) for technical review

Travis Sexson (@1nfoSex) for technical editing

Cindy Westman (@cindy_westman) for technical editing