What is NtPssCaptureVaSpaceBulk

The API

Just a quick post about something I found quite interesting. In Windows 10 version 2004+ (according to this) there is a new system call which caught my attention: `NtPssCaptureVaSpaceBulk`. Specifically the caputing “bulk” referenced in the name made me think it could be useful. Looking at the API in the kernel, I quickly noticed several calls using the `PsProcessType` type as the first parameter, so I looked at the prototype of `NtQueryVirtualMemory` and went from there. My Google searching skills discovered that there is no offical prototype for this function, and so my final definition looks like this:

```
NTSTATUS NtPssCaptureVaSpaceBulk(HANDLE ProcessHandle,
        PVOID BaseAddress,
        PBULK_MEMORY_INFORMATION MemoryInfo,
        SIZE_T Length,
        PSIZE_T ReturnLength);
```

The structure of BULK_MEMORY_INFORMATION is:

```
typedef struct _BULK_MEMORY_INFORMATION
{
        ULONG QueryFlags;
        ULONG NumberOfEntries;
        PVOID MaxUserAddress;
        BYTE Reserved[0x18];
        PVOID LowestAddressFound;
        BYTE Reserved2[0x10];
    MEMORY_BASIC_INFORMATION MemoryInfo[1];
} BULK_MEMORY_INFORMATION, *PBULK_MEMORY_INFORMATION;
```

As a side note: When creating this structure definition I remembered seeing similar Windows structs which has arrays as the last element but had it defined as a 1-length based, looking into this I found this post by Raymond Chen explaining why this is the case.

The `QueryFlags` member does not seem to be used except for being bitwise anded with 0xFFFFFFFC so only 1, 2, and 3 are valid. Again these are currently not used to control anything, but needs to be set or you’ll get STATUS_NOT_SUPPORTED. The main logic then follows which involves attaching to the provided process handle, re-calling itself in with the `Zw` variant of `PssCaptureVaSpaceBulk`, and then preforming a loop of `NtQueryVirtualMemory`. In the past, looping `VirtualQuery(Ex)` is essentially what developers had to do to query the memory of a process. That code would look something like this:

```
MEMORY_BASIC_INFORMATION memInfo;
for (PVOID pMem = NULL; VirtualQueryEx(GetCurrentProcess(), pMem, &memInfo,
sizeof(memInfo)) == sizeof(memInfo); (BYTE*)pMem += memInfo.RegionSize)
{
        printf("Region: %p\n", memInfo.BaseAddress);
}
```

But now you can do something like this:

```
SIZE_T nReturn;
SIZE_T nHeap = 0x1000;
PBULK_MEMORY_INFORMATION pBulk = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, nHeap);
pBulk->QueryFlags = 0x1;
while (NtPssCaptureVaSpaceBulk(GetCurrentProcess(), 0, pBulk, nHeap, &nReturn))
{
        HeapFree(GetProcessHeap(), 0, pBulk);
        nHeap += 0x1000;
        pBulk = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, nHeap);
        pBulk->QueryFlags = 0x1;
}
for (size_t i = 0; i < pBulk->NumberOfEntries; i++)
{
        printf("Mem: %p\n", pBulk->MemoryInfo[i].BaseAddress);
}
```

Is it more complicated? Yes. Why use it? I’m not sure, but maybe in the future the query flags will be able to gather different information. For now it is simply appears to be a redundent way to capture the entire virtual address space of a process.

Posted on May 14, 2021